

ARTIST2

Graduate Course on Embedded Control Systems

April 3rd-7th, 2006
Prague, Czech Republic

Student **handouts**

Organized by Zdeněk Hanzálek
Department of Control Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague



CAK ARTIST2



Program of Graduate Course on Embedded Control Systems

Department of Control Engineering, FEE, CTU Prague

Karlovo náměstí 13, Building E, Room K112

Monday 3rd of April

- 8:00 Registration
- 8:30 M1 Motivation and examples, Bengt Eriksson and Martin Torngren, 1.5 hour (KTH)
- 10:00 Coffee
- 10:15 M2 Control issues, Pedro Albertos, 2 hours (UPVLC)
- 12:15 Lunch
- 13:45 M3 RT issues, Alfons Crespo, 2 hours (UPVLC)

Tuesday 4th of April

- 9:00 T1 Kernels and safe (back-up) operation, Pedro Albertos and Alfons Crespo, 1 hour (UPVLC)
- 10:00 Coffee
- 10:15 T2a Control design practical issues - principles, Bengt Eriksson, 1 hour (KTH)
- 11:15 T2b Control design practical issues - models, Jindrich Fuka, Jiri Roubal, 1 hour (laboratories K23 and K26 - CTU)
- 12:15 Lunch
- 13:45 T3 Integrated control design and implementation, Karl-Erik Arzen and Anton Cervin, 2 hours (LTH)

Wednesday 5th of April

- 8:00 W1 Control of Computing Systems, Karl-Erik Arzen and Anton Cervin, 2 hours (LTH)
- 10:00 Coffee
- 10:15 W2 Jitterbug and Truetime, Karl-Erik Arzen and Anton Cervin, 2 hours (laboratory K2 – LTH)
- 12:15 Lunch
- 13:45 W3 ECS Deployment, Bengt Eriksson and Martin Torngren, 2 hours (KTH)

Thursday 6th of April

- 8:00 Th1 Off-line scheduling, Zdenek Hanzalek, 2 hours (CTU)
- 10:00 Coffee
- 10:15 Th2 Platform for Advanced Process Control and Real Time Optimization, Vladimir Havlena and Jiri Findejs, 2 hours (Honeywell Laboratory Prague)
- 12:15 Lunch
- 13:45 Th3, RT practical issues, Michal Sojka and Ondrej Spinka, 2 hours (laboratory K09 - CTU)

Friday 7th of April

- 8:00 F1, Torsche – Matlab scheduling toolbox, Premysl Sucha and Michal Kutil, 2 hours (laboratory K2 - CTU)
- 10:00 Coffee
- 10:15 F2, Implementing Floating-Point DSP and Control with PicoBlaze Processors, Jiri Kadlec, 2 hours (CTU)
- 12:15 Closing remarks and discussion

M1 Motivation and examples, Bengt Eriksson and Martin Torngren, 1.5 hour (KTH)

In this introductory session, the general problem of the course will be presented and motivated. What Embedded systems (ES) are? What Embedded control systems (ECS) are? Why? Motivating examples: inverted pendulum, mobile robot, car safety control. Main issues in the design of ECS: typical requirements, conflicting requirements, design trade-offs, typical architectures, design parameters.

M2 Control issues, Pedro Albertos, 2 hours (UPVLC)

Real-time implementation of control algorithms in a multitasking environment involves a number of issues that should be taken into account. The unavoidable delays, both in computation and in data handling, the lost of data, the change of operation mode, the changes in sampling periods and the performance degrading are among the main issues to be considered. In this session, a review of these concepts for a general audience will be presented. The goal of this session would be to emphasize the relevance of these control design issues, to be strongly connected to the actual implementation of the control, to be discussed in the next sessions.

M3 RT issues, Alfons Crespo, 2 hours (UPVLC)

The aim of this session is to introduce the most important concepts of ECS from the real-time (RT) systems perspective. The different types of RT tasks are introduced, and the importance of RT constraints is emphasized, especially in the context of control systems design. The central role of processor scheduling for guaranteeing RT constraints is motivated, and the main paradigms of RT scheduling are introduced. Fixed and dynamic priority scheduling methods are described, including temporal analysis methods. Resource usage and jitter control are also introduced. Finally, implementation approaches in view of the existing RT operating systems and programming languages technologies are discussed. The level of presentation of the topics is introductory, but a basic knowledge of operating systems, computer architecture, and programming in a high-level language is assumed.

T1 Kernels and safe (back-up) operation, Pedro Albertos and Alfons Crespo, 1 hour (UPVLC)

ECS require to work in a variety of (unexpected) circumstances. The operating system (OS) should provide a number of basic options to guarantee the safe behaviour of ECS. In this session, a new set of operating services to provide the applications a higher control of faults and temporal constraints will be described. Some examples of this functionalities are: Execution timers, application defined scheduling, fault tolerant monitors, etc. From the control viewpoint, a hierarchical sorting of activities should be scheduled in agreement with the OS kernel to get the best, among the possible, control options. Safe (back-up) operation, basic control actions, optional and supervision are among the main issues to be discussed.

T2 Control design practical issues – principles and models, Bengt Eriksson, Jindrich Fuka, Jiri Roubal, 2 hours (KTH, CTU)

Introductory and simple exercises about control design using CACD (computer aided control design) packages will allow a better insight into the RT control design algorithms. Moreover, using some simple rigs, the participants will get some hands-on control design approaches. Some principles will be demonstrated on laboratory models.

T3 Integrated control design and implementation, Karl-Erik Arzen and Anton Cervin, 2 hours (LTH)

This session will focus on the interaction between the control design and control implementation. In embedded systems, floating point arithmetic is sometimes too costly. The problems associated with fixed point arithmetic are discussed. The implementation platform normally introduces input-output latencies due to computation and communication delays. The effects of this on control performance and how it can be compensated for will be discussed. Special emphasis will be given to the recent jitter margin concept. The implementation platform also introduces jitter in sampling intervals. This will also be discussed. The control server is a computational model for controller tasks that combines the benefits of static scheduling and dynamic event-based scheduling. Changing controller task parameters such as sampling periods on-line could sometimes be useful in order to adapt to changing conditions. The problems associated with this and the risk of switching induced instabilities will be discussed.

W1 Control of Computing Systems, Karl-Erik Arzen and Anton Cervin, 2 hours (LTH)

Using control-based approaches for modeling, analysis, and design of embedded computer and communications systems is currently receiving increased attention from the real-time systems community, as a promising foundation for controlling the uncertainty in large and complex real-time systems. The control-based approach has the potential to increase flexibility, while preserving dependability and efficiency. In this session we will give an overview of the work that is being done within the area with a special emphasis on two areas: Control of Web-servers and feedback scheduling of controller tasks. An inverted pendulum control example will illustrate some of the issues.

W2 Jitterbug and Truetime, Karl-Erik Arzen and Anton Cervin, 2 hours (laboratory K2 – LTH)

A hands-on session/exercise where the users will become familiar with the two co-design tools Jitterbug and TrueTime. Jitterbug is a MATLAB-based toolbox that computes a quadratic performance criterion for a linear control system under various timing conditions. Using the toolbox, one can easily and quickly assert how sensitive a control system is to delay, jitter, lost samples, etc., without resorting to simulation. The tool is quite general and can also be used to investigate jitter-compensating controllers, aperiodic controllers, and multi-rate controllers. TrueTime is a MATLAB/Simulink-based tool that facilitates simulation of the temporal behavior of a multitasking real-time kernel executing controller tasks. The tasks are controlling processes that are modeled as ordinary continuous-time Simulink blocks. TrueTime also makes it possible to simulate simple models of communication networks and their influence on networked control loops.

W3 ECS Deployment, Bengt Eriksson and Martin Torngren, 2 hours (KTH)

The practical issues of ECS deployment will be discussed in this session, including: ECS implementation and platform selection (e.g. which type of OS?, which hardware?); OS configuration, components selection and loading (static vs dynamic OS types); Cross-compiling; Code generation; Verification and validation. A case study will illustrate the approach.

Th1 Off-line scheduling, Zdenek Hanzalek, 2 hours (CTU)

The objective of this course is to provide an overview of different off-line scheduling problems found in embedded systems. In order to classify the scheduling problems, we show

alpha|beta|gamma notation first. Then we develop several algorithms for real-time monoprocessor applications. Namely we show Bratley's branch&bound algorithm for Cmax optimization with release dates and deadlines and we underline main ideas of 0/1 programming solution for weighted completion time optimization with precedence constraints. The class of monoprocessor problems is concluded by minimization of maximum latency, i.e. Earliest Due-Date First algorithm and Earliest Deadline First algorithm. Finally we give an insight into the scheduling on dedicated processors and we provide examples on code synthesis for FPGA.

Th2 Platform for Advanced Process Control and Real Time Optimization, Vladimir Havlena, 2 hours (Honeywell Prague)

The talk will demonstrate componentised architecture for Advanced Process Control and Real Time Optimization. The concept will be illustrated by the Unified Energy Solutions (UES) package developed by the Honeywell Laboratory in Prague, a portfolio of advanced control and optimization components for utilities and industrial energy, with the objective to operate the plant with maximum achievable profit (maximum efficiency) under the constraints imposed by technology and environmental impacts.

Th3, RT practical issues, Michal Sojka and Ondrej Spinka, 2 hours (laboratory K09 - CTU)

In this laboratory exercise the students will learn, how to use the Linux for low level control of a laboratory model. The main goal of this session will be to control the velocity of a DC motor. The motor is actuated by a PWM signal realized via two bit outputs as one periodic thread. The measured velocity is derived from two phase-shifted signals while implementing IRC (Incremental Radial Counter) sensor as an aperiodic thread. The motor is connected to a PC using printer port through a simple electronics consisting of a motor driver and basic logic circuits. The organization of the session will be as follows (it is assumed the students know to write a simple RT Linux program, Session T3): First, the students will be provided with information on how to control parallel port circuits through the configuration registers. Second, the students will try to generate the PWM signal for motor control. Third, they will write the code to measure the rotation velocity and they will program a simple PID controller for velocity control. Finally the use of RT Linux will be discussed.

F1, Torsche – Matlab scheduling toolbox, Premysl Sucha and Michal Kutil, 2 hours (laboratory K2 - CTU)

The aim of the seminar is to present a Matlab based Scheduling toolbox TORSCHE (Time Optimization of Resources, SCHEduling). The toolbox is intended mainly as a research tool to handle control and scheduling co-design problems. It offers a collection of data structures that allow the user to formalize various off-line and on-line scheduling problems. Potential of the toolbox will be shown on a high level synthesis of parallel algorithms.

F2, Implementing Floating-Point DSP and Control with PicoBlaze Processors, Jiri Kadlec, 2 hours (CTU)

For developers using reconfigurable HW for the implementation of floating-point DSP and Control algorithms, one key challenge is how to decompose the computation algorithm into sequences of parallel hardware processes while efficiently managing data flow through the parallel pipelines of these processes. Lecture, will summarize our current experiences with architecture based on network of Xilinx PicoBlaze controllers on a single chip. Complete design path from model-based (Simulink) and C-based designs (Handel-C) to the concrete reconfigurable HW will be demonstrated.

Monday 3rd of April



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

ARTIST2

Embedded Control Systems: Motivation and Examples

M. Törngren and B. Eriksson
Division of Mechatronics, Dept. of Machine Design
KTH - Royal Institute of Technology, Stockholm
www.md.kth.se e-mail: martin@md.kth.se



KTH Industrial Engineering
and Management

3/21/2006
©M. Törngren 2006

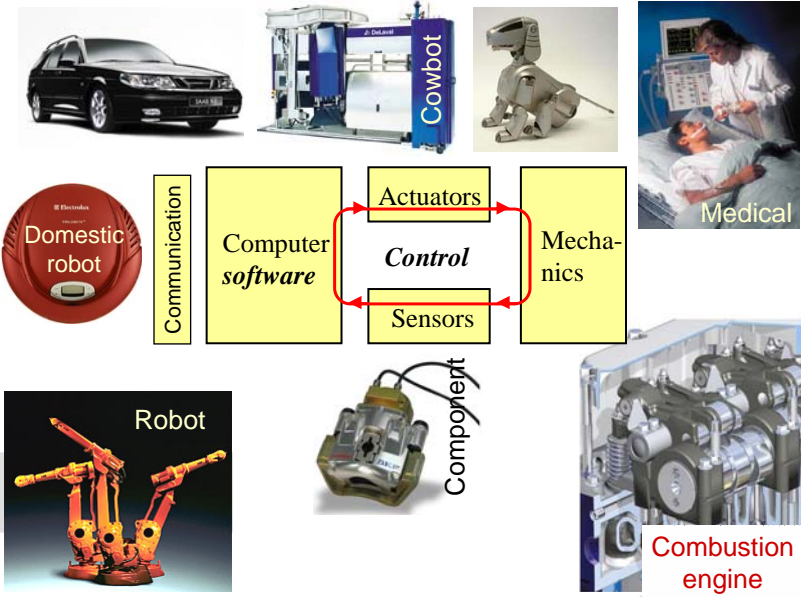
ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- **Background: evolution of electronics and software**
- **Basic concepts and characteristics**
 - Embedded vs. general purpose computing systems
 - Concepts in real-time control
 - Characteristics
- **Technical issues in ECS design**
- **Application examples**
- **Concluding Remarks**

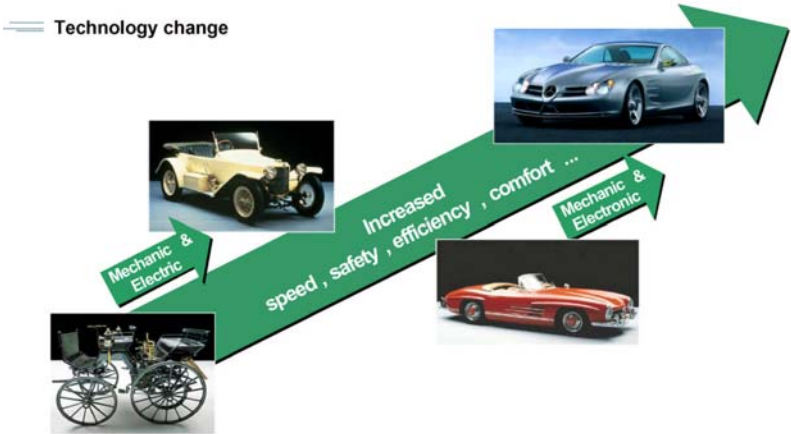
3/21/2006
©M. Törngren 2006

Products relying on embedded control



ARTIST2 Graduate Course on Embedded Control Systems Prague, Czech Republic, April 3-7, 2006

The role of software and electronics





Electronic components and software will, to a large extent, shape tomorrow's vehicles (90% of vehicle innovations)

Source: The SEA consortia

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 2-7, 2006

Mechanics → Mechatronics

Hy-Wire från GM
Skateboard concept
(Autonomy 2)

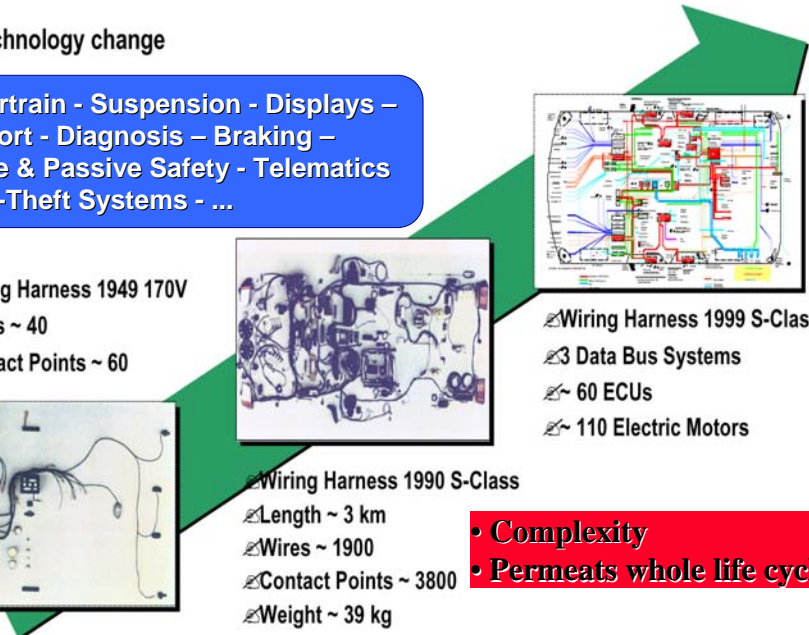
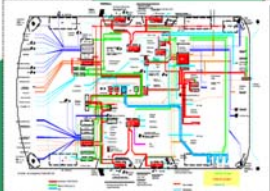
Fuel-cell
Distributed control
Electrical actuators

3/21/2006
Source: GM

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Technology change

Powertrain - Suspension - Displays -
Comfort - Diagnosis - Braking -
Active & Passive Safety - Telematics
- Anti-Theft Systems - ...

- Wiring Harness 1949 170V
 - Wires ~ 40
 - Contact Points ~ 60
- Wiring Harness 1990 S-Class
 - Length ~ 3 km
 - Wires ~ 1900
 - Contact Points ~ 3800
 - Weight ~ 39 kg
- Wiring Harness 1999 S-Class
 - 3 Data Bus Systems
 - 60 ECUs
 - 110 Electric Motors

Complexity
Permeates whole life cycle

Source: The SEA consortia

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Historical perspectives

- "I think there is a world market for about five computers",
Tomas J Watson Sr, IBM 1943
- "There are no reasons for any individuals to have a computer
in their home", Ken Olson, Digital Equipment 1977
- "The current rate of progress cannot continue much longer",
various computer technologists, 1950

- 'Moore's law' (Intel, 1965): Microelectronics performance is
~doubled every 18 months and chip size is reduced by 50%

- Compare: Intel 4004/1971 vs. Intel Pentium/1996
→ from 2300 to 5.5 million transistors

3/21/2006
©M. Törngren 2006

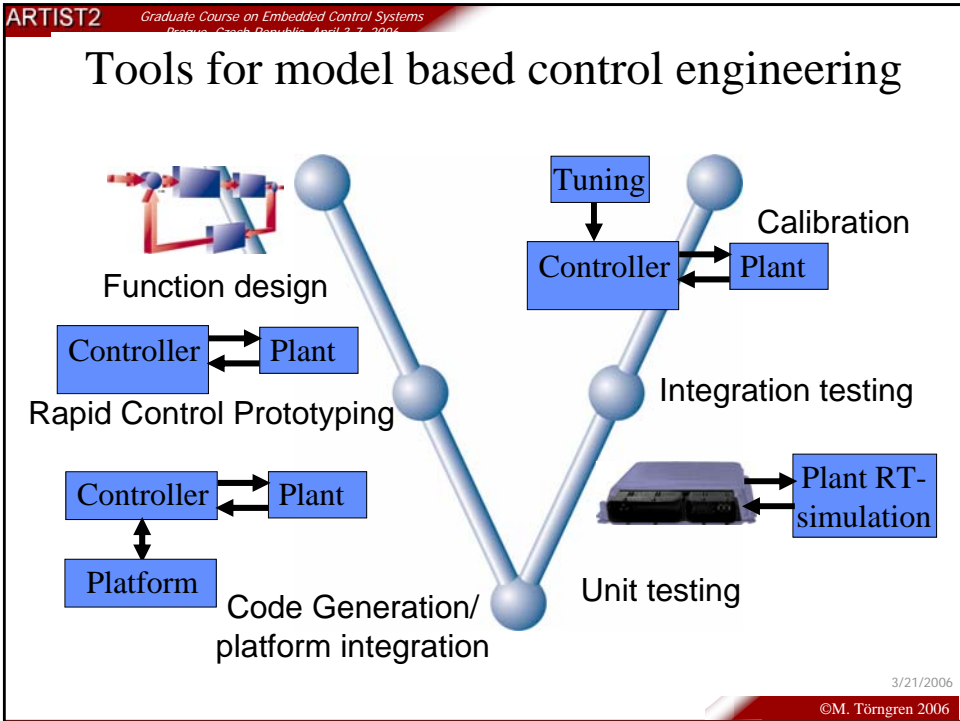
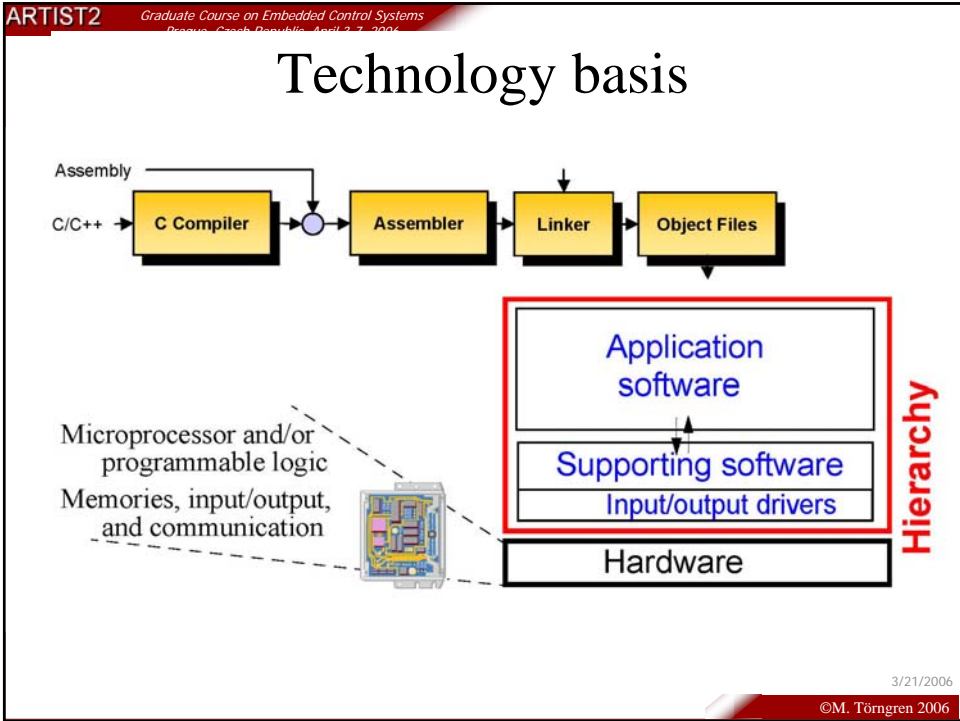
ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Technology evolution

The diagram illustrates the evolution of control technology through several stages:

- Mechanical, hydraulic and pneumatic controllers** (e.g. centrifugal regulator, 18th century)
- Combustion engine** (19th century)
- (Pneumatic controllers early 20th century)**
- Electrical relays**
- Analogue electronic controllers** (1940-50) (transistor, 1947)
- Direct digital control** (early 1960:ies)
- Distributed computer control** (~1970:ies) (4004 microprocessor, 1971)

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- Background: evolution of electronics and software
- **Basic concepts and characteristics**
 - **Embedded vs. general purpose computing systems**
 - **Concepts in real-time control**
 - **Characteristics**
- Technical issues in ECS design
- Application examples
- Concluding Remarks

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Definition of embedded computer system


Embedded computer system (IEEE)

A computer system that is part of a larger system and performs some of the requirements of that system; for example, a computer system used in an aircraft or rapid transit system.

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Embedded systems (ES)



- An enabling technology
- ES themselves as products (e.g. OS)
- Very broad variety of applications & different types of requirements:
 - from critical to non-critical
 - long to short life time etc.

→ Room for many methodologies and technologies

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Computer Architecture Trends

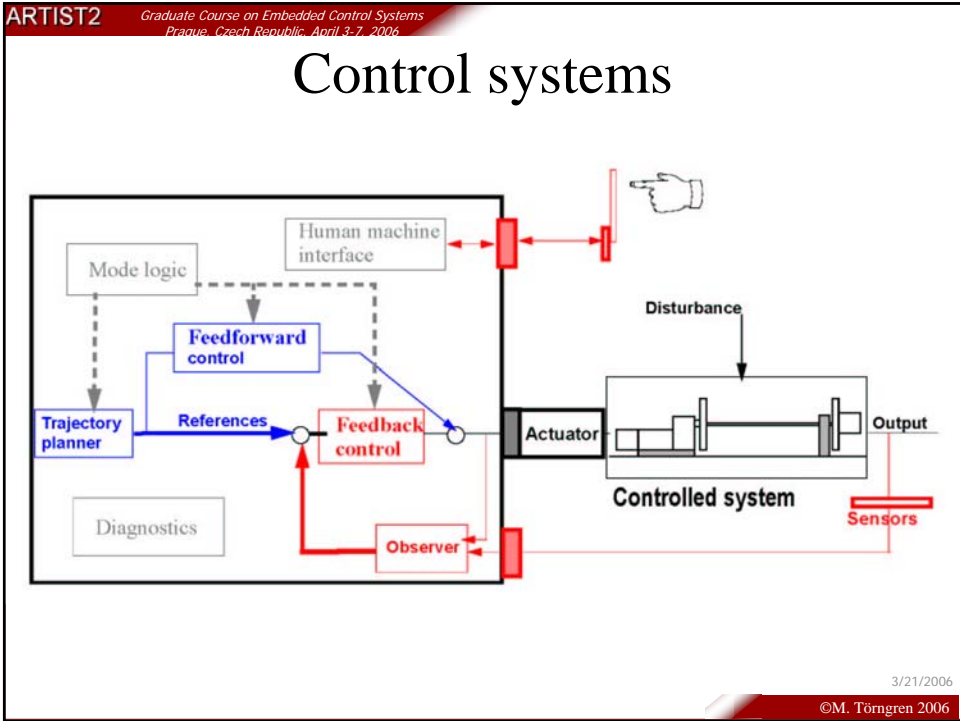
Present proposals for future billion-transistor computers:

- Desktop uniprocessors for technical applications
- Multiprocessor servers for transaction processing
- Large continuous data-processing capability

Future embedded system computers:

- Harsh environment tolerance; temperature, vibration, radiation
- Low power dissipation, power down modes
- On-chip input-output units, communication and memory
- Predictable behavior, support for concurrency

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Definition of real-time

A system, where correct timing behavior is strongly related to functionality, performance and reliability

Common definitions:

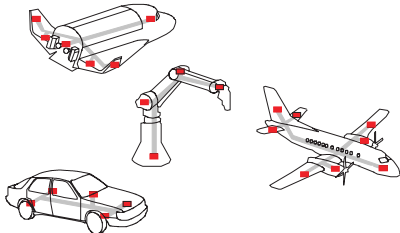
- *A computer system is a real-time one if it explicitly manages resources in order to meet timing constraints (Douglas Jensen, 1992)*
- *A real-time system is a system where the correctness depends not only on the logical result of computation but also on the time at which the results are produced". (Jack Stankovic, 1988).*

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Characteristics of ECS

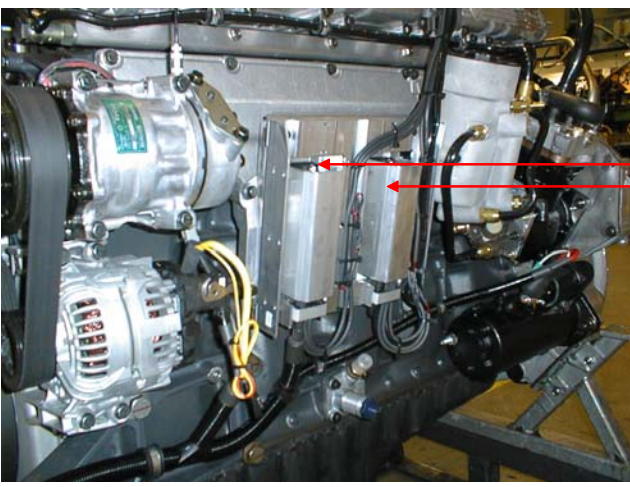
- Rich functionality
- Resource constraints
- Increasing connectivity
- Tight process relation
 - RT constraints
 - ET and TT, parallelism
 - Roughness
- Dependability: safety, reliability/availability, security
- Multidisciplinarity



3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Scania diesel engine and controller



ECU connectors
on top of the ECU

Engine ECU tasks: control of engine, fan, alternator, engine brake, turbo, and the EGR valve + CAN communication, + diagnostics, ...

3/21/2006
Courtesy of Scania

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Requirements and conflicts

Control Functionality, Stability, and Performance

Reliability
Availability
Security
Safety

Performance
Functionality

Dependability

Reusability
Usability
Testability
Cost-effectiveness
Time-to-market

Produceability

Complexity

Integrability
Interoperability

Flexibility

- Portability
- Extensibility
- Changeability
- Reconfigurability
- Comprehensibility
- Communicability
- Analyzability

Can you give an example of conflicting requirements?
How can these conflicts affect the design?

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Conflicts?

- Cost vs. Quality (in general) vs. Time
- Testability vs. performance
- Performance vs. flexibility
- Reliability vs. cost
- Safety vs. availability
- Control robustness vs. performance
- ...

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 2-7, 2006

Models of computation

	Continuous data	Discrete data
Continuous time	$dx/dt(t) = Ax(t)+Bu(t)$	$dx_q/dt(t) = A_q x_q(t)+B_q u_q(t)$
Discrete time	$x(kh+h) =$ $x(k+1) = \Gamma(k)x(k) + \Phi(k)u(k)$	$x := 0$ $b, (x < 2)$

Reality: A little of everything!

3/21/2006

©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Multidisciplinary ECS development

Specifications

Architectural design

Mechanics Control Software Electronics

Integration

Different traditions

Big bang effects

3/21/2006

©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- Background: evolution of electronics and software
- Basic concepts and characteristics
 - Embedded vs. general purpose computing systems
 - Concepts in real-time control
 - Characteristics
- **Technical issues in ECS design**
- Application examples
- Concluding Remarks

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

A sample of technical issues in ECS design and implementation

- Discretization
- Quantization
- Delays
- Jitter in delays and periods
- Aliasing
- Triggering and tasking partitioning, scheduling
- Code implementation
- Sensor and actuator limitations
- Calibration/diagnostics
- Error detection and error handling

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Codesign, aspects and needs

Control performance,
Stability, control effort,
Sensor accuracy, etc.

Where do they meet??

Structuring,
scheduling,
triggering,
communication

Platform
SW
HW

Drivers:
• Performance
• Cost
• Dependability
• Flexibility

Trade-offs/optimality requires considering both domains

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Control system timing behavior

Requirements $\tau(k), h(k)$ Constraints

Platform
SW
HW

end to end delay
TDMA period
Sensor Bus Control Actuate
offset

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Examples of timing properties caused by a particular implementation

sample z
actuate u

time t

Period: $h(k) = t_k - t_{k-1}$ and jitter
E2e delay, $\tau(k)$, and jitter, e.g. $\tau_{\min}(k) \leq \tau(k) \leq \tau_{\max}(k)$

How does the given timing scenario affect control performance?
How does choices in computer system design parameters affect the timing behavior and thereby the control performance?

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Control analysis/compensation

- How do the timing properties affect the closed loop?

- Analysis of steady state and transient behavior

$$\mathbf{x}(t_{k+1}) = \Phi(h_k)\mathbf{x}(t_k) + \sum \Gamma(h_k)u(t_{k-(nd-g)})$$
- Potential compensation for jitter
 - Run-time information example: Actual data delay

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Co-design with respect to timing

Has to be defined to ensure proper operation

Typical solutions:

- $\tau \ll T$ enforced (costly)
- $\tau < T$ assumed (jitter neglected)
- $\tau = T$, with deterministic solution, however causing some performance degradation and reduced flexibility (trade-offs!)

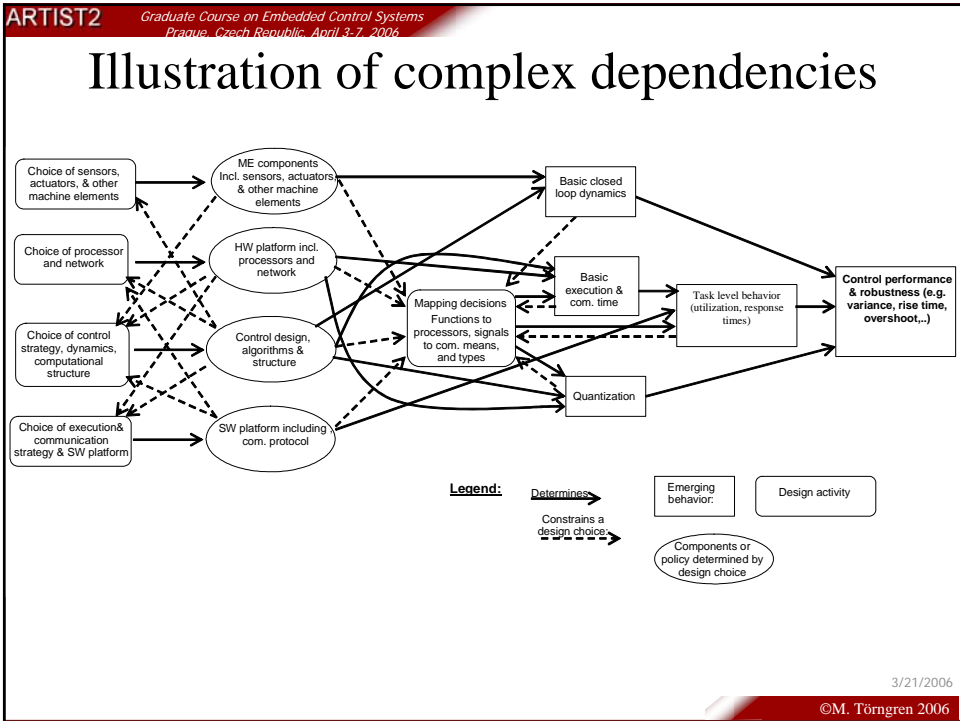
Computer system options: Design for predictability if possible

Control system options: Analyse, and compensate if possible

Co-design options:

- May leave some unpredictability in computer system design
- May be able to handle non-perfect existing computer system

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 2-7, 2006

"Fault-tolerant control" - Codesign

$$x(kh+h) = \Gamma x(kh) + \Phi u(kh) + \delta(kh)$$

Fault and failure models
Diagnostics
Analytic redundancy

Fault and failure models
Fault-injection
Error masking
Error detection
Error handling

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 2-7, 2006

Control for resource management in embedded systems

- Implementation issues similar to RTC, Constraints may differ depending on application
- Controller implemented as part of the plant
- Plant modeling and characteristics differ

3/21/2006
©M. Törngren 2006

ARTIST2 *Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006*

Outline

- Background: evolution of electronics and software
- Basic concepts and characteristics
 - Embedded vs. general purpose computing systems
 - Concepts in real-time control
 - Characteristics
- Technical issues in ECS design
- **Application examples**
- Concluding Remarks

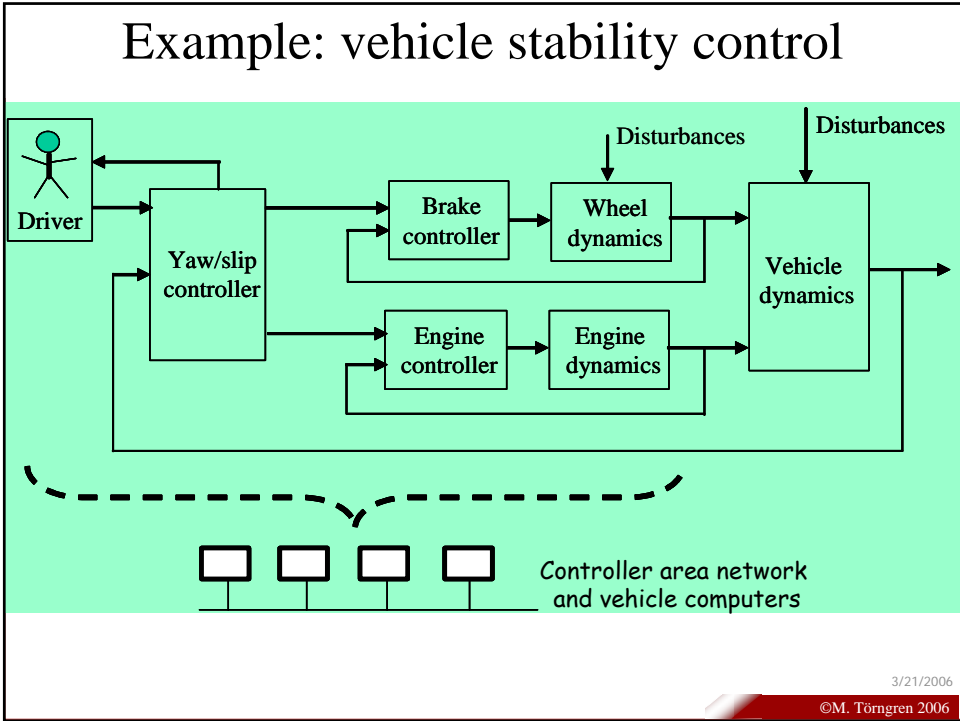
3/21/2006
©M. Törngren 2006

Example application: stability control

The diagram illustrates a vehicle's path through a curve. The top part, labeled 'With ESC', shows a car following a smooth, orange-colored path that stays within the lane boundaries. The bottom part, labeled 'Without ESC', shows a car following a red-colored path that drifts out of the lane. Three labels with arrows point to specific parameters: 'Vehicle speed' points to the car's forward motion; 'Vehicle slip angle' points to the angle between the car's longitudinal axis and its actual path; and 'Vehicle yaw rate' points to the rotation of the car around its vertical axis.

Source: ESC education

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Vehicle stability control - characteristics

- Several modes of operation
- Hierarchical and cascaded control
- Multiple input, multiple output
- Control loop closed over in-vehicle network(s)
- Availability critical (graceful degradation)
- Safety critical, real-time operation
- Redundancy in hardware (sensors, actuators, processors), information and algorithms

Some challenges:

- Software upgrades (and security)?
- Who is in charge (driver vs. computer control)?
- How to define a suitable scalable architecture?

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Many high-levels functions

Examples:

- Driver assistance; e.g. adaptive cruise control
- Active safety functions; e.g. collision mitigation by braking
- Telematics; e.g. dynamic and external road info
- Electrical power management

The diagram illustrates an adaptive cruise control scenario. On the left is a red car, and on the right is a white truck. A double-headed arrow between them is labeled d_{des} (desired distance). A single-headed arrow pointing left from the truck is labeled v (velocity). A single-headed arrow pointing left from the car is labeled v_{lead} (lead velocity). A double-headed arrow between them is labeled d_{act} (actual distance).

Adaptive cruise controller example

3/21/2006
©M. Törngren 2006


ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Resource sharing; sensors, actuators and embedded system

The block diagram shows the resource sharing architecture for an adaptive cruise control system. On the left, three input blocks are shown: 'HMI inputs', 'Internal Sensors', and 'Radar'. On the right, three output blocks are shown: 'HMI outputs', 'Selector', and 'Actuators'. In the center, there are three main processing blocks: 'Object recognition', 'ACC Mode logic', and 'ACC controllers'. Arrows indicate the flow of data: 'HMI inputs' and 'Internal Sensors' feed into 'ACC Mode logic'. 'Radar' feeds into 'Object recognition'. 'Object recognition' feeds into 'ACC Mode logic' and 'ACC controllers'. 'ACC Mode logic' feeds into 'ACC controllers' and 'HMI outputs'. 'ACC controllers' feeds into 'Selector'. 'Selector' feeds into 'Actuators'. A dashed arrow labeled 'Set points from other functions' also feeds into the 'Selector' block.

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems



Hierarchy of functions

Sensors	Total Vehicle Control										Vehicle System	Driver
	
	Suspension - Powertrain										Inter-Domain 0	
	Suspension			Powertrain				Cabin/Body			Domain Level	
	Variable Transmis.	ABS	„Sport“	„Eco“		„Winter“	Memory	Delay	Extended Basic Syst		
Steering System	Braking System	Spring, Damping	Engine	Starter Generator	Gear Box	Seat Mirror Window ..	Light	Display Telematic	Basic System			

- Definition of hierarchy
- Specification of interfaces
- Guarantee function interoperability
- Partitioning in components, modules, ECUs, software, networks

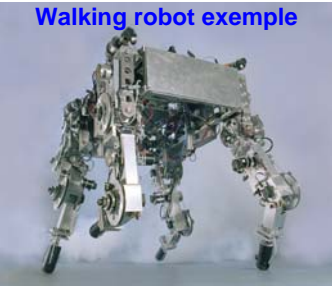
Source: EAST-EAA project

02.10.2002 ITEA Project.00009: First project review 16

©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems

Robot system hierarchy



Optimization

↕

Planning

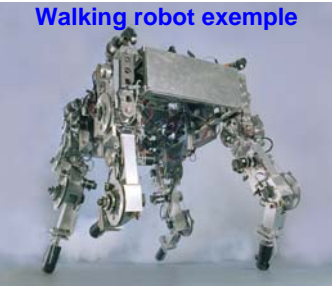
↕

Coordination MIMO control

↕

Actuator control

Walking robot exemple



Planning

↕

Navigation

↕

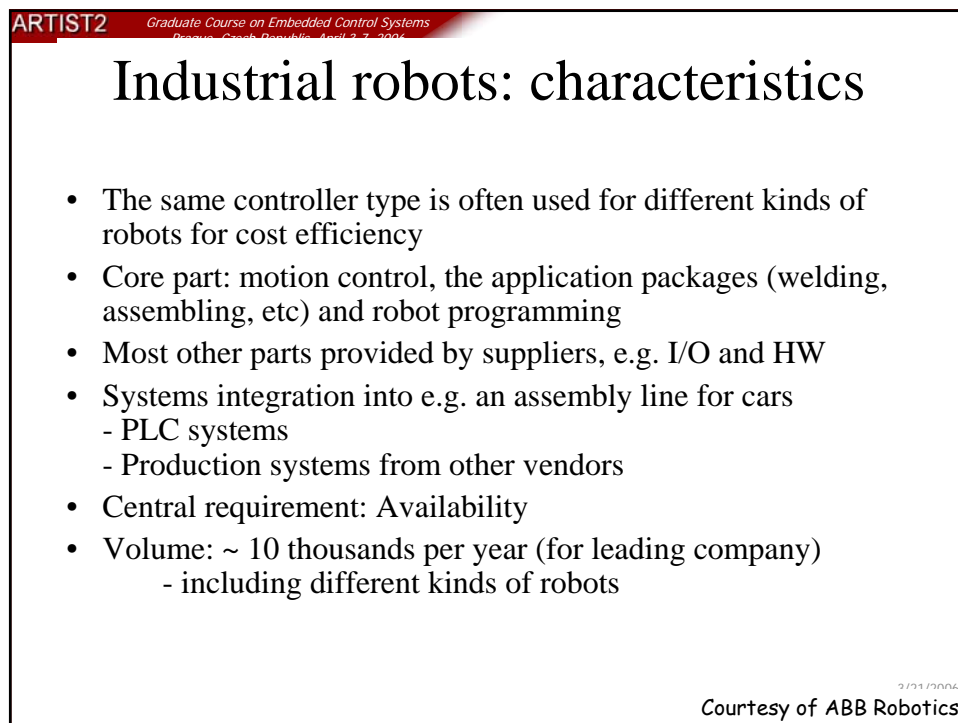
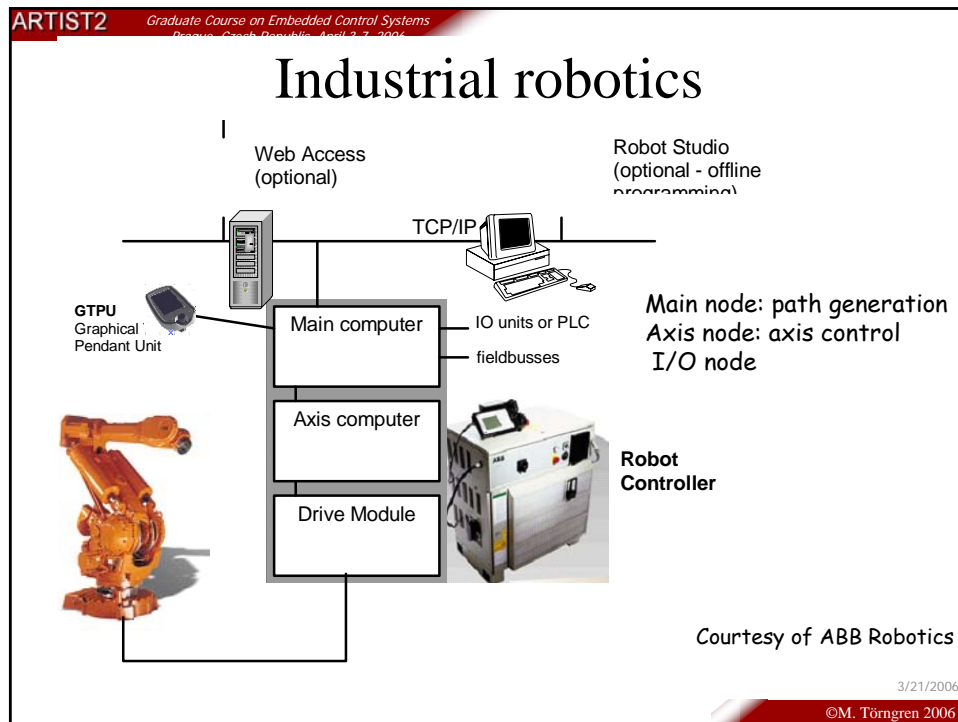
Coordination Balance, gait (e.g. galop)

↕

Joint/leg-control

3/21/2006

©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Dennis Gries, Berkeley, April 27, 2006

Industrial robotics; architecture requirements

- Openness, to facilitate integration
- Integration with different communication protocols (Profibus, Interbus, Foundation fieldbus, FIP, ...)
- The control system to be configurable to facilitate reuse
- SW has to be easy to port, 'lives' longer than HW
- Typical reliability requirement: MTBF > 60000 h
→ ~ production line with 800 robots with one robot failing per week in average

3/21/2006
©M. Törngren 2006

Hw&Sw cost proportion for industrial robotics

Total Costs %

Year	Hardware Costs (%)	Software Costs (%)
1990	20	0
2000	60	40
2010	80	20

Note: Software for the first control system developed in three months by one single person.

3/21/2006
Courtesy of ABB Robotics

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Software architecture: Industrial robotics

- Object oriented approach,
- Code written in C
- Code size: ~ 2 500 KLOC divided into 400-500 components

3/21/2006
Courtesy of ABB Robotics

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- Background: evolution of electronics and software
- Basic concepts and characteristics
 - Embedded vs. general purpose computing systems
 - Concepts in real-time control
 - Characteristics
- Technical issues in ECS design
- Application examples
- **Concluding Remarks**

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

The product life cycle and course scope

Embedded control systems touch upon all phases of a product's life cycle!

The emphasis in this course is on technical issues part of the product development phase!

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Integration challenges

Process X Process Y

Discipline X Discipline Y

People/organization
e.g. Competence, roles
integration

Technology
e.g. Tools, components

Knowledge, information Knowledge, information

Differences not only in technology, but also in processes, traditions and organizational roles

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Concluding remarks

- A very broad spectra of ECS applications!
- Multidisciplinary development – a key challenge
- Needs for co-design relates to conflicting requirements
- Closed-loop control of increasing importance
- Many other issues such as reliability/safety, diagnostics, reuse, upgrades, ...

3/21/2006
©M. Törngren 2006

ARTIST2

Embedded Control Systems: Control Issues

P. Albertos

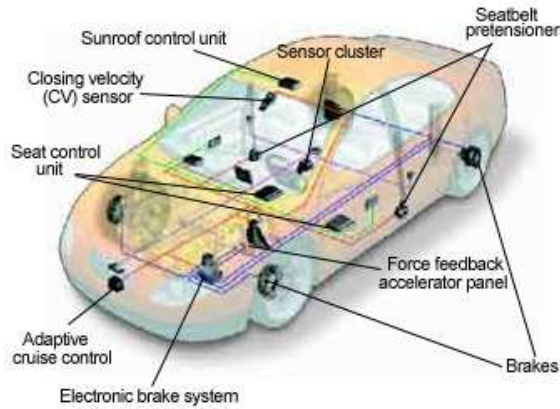
Universidad Politécnica de Valencia
Dept. of Systems Engineering and Control
POB. 22012 E-46071 Valencia, Spain.
Fax: +34 96 3879579 e-mail: pedro@aii.upv.es

ARTIST2 NoE on Embedded Systems Design – ECS Graduate Course
CTU, Prague, April 3-7, 2006

Outline

- **Embedded Systems**
 - RT constraints
- **Embedded Control Systems**
 - Control issues
- **Non-uniform sampling**
- **Missing data**
- **Changes in sampling period**
- **Performance degrading**
- **Concluding Remarks**

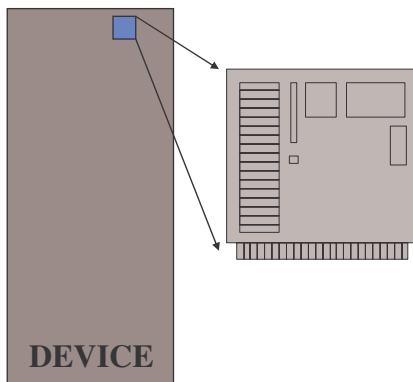
Embedded systems



Diagnosis

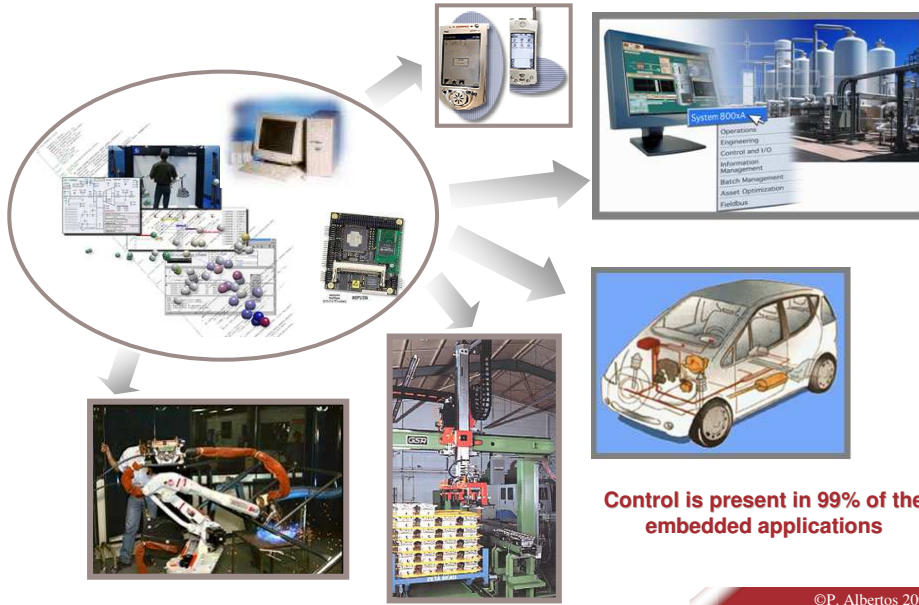


Embedded systems

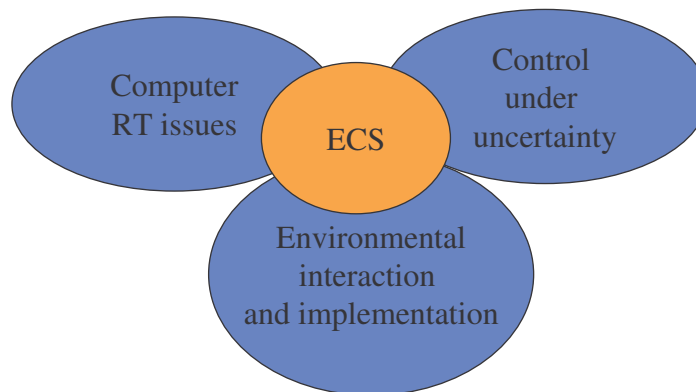


- Device:
 - Stand-alone
 - Networked
 - RT operation
- ES:
 - Compact and reduced size
 - Autonomy
 - Missing data operation
 - Fault-tolerant
 - Reconfigurability
 - Safety

Embedded systems



Embedded Control Systems

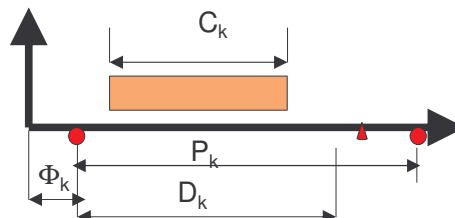


Embedded systems: RT Issues

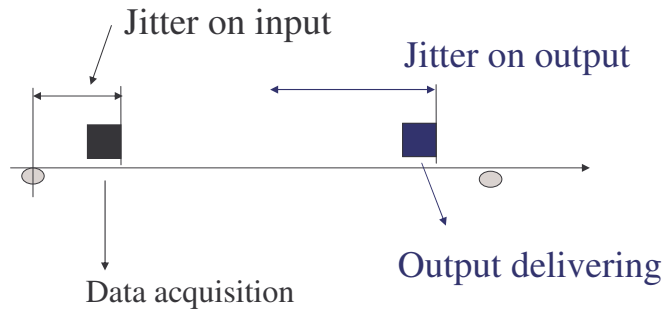
- Resource constraints
- Power aware
- Task Management: Critical and soft real-time activities
 - Task definition
 - Priority assignment
 - Time units (periods)
- Full range of communication devices
- Changeable operating conditions
- CPU utilization control
 - Adaptable to the changing conditions
 - Self-organizing
 - On-line scheduling

Real-Time Task Model

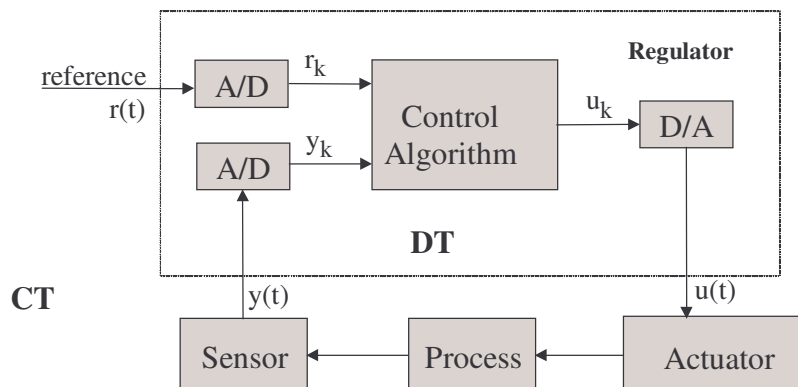
- A task (T_k) is defined by four parameters:
 - C_k : Worst Case Execution Time (WCET)
 - D_k : Deadline
 - P_k : Period
 - Φ_k : Phase



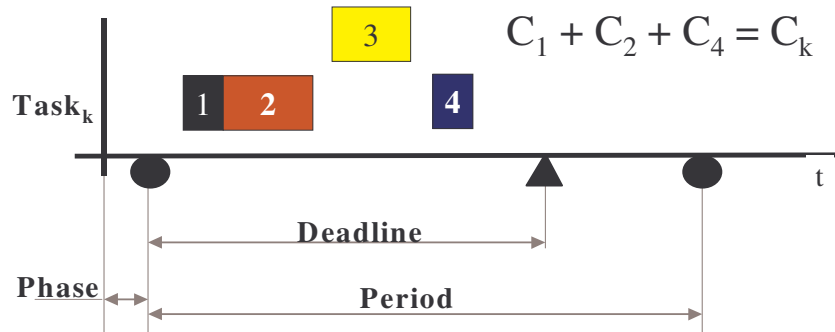
Time delays



Basic Control Loop



Control Task Model



$$T_K = (C_K, D_k, P_k, \Phi_k)$$

Control task

...

...

loop

convert _sensor _analog_ digital (y);

compute _control _action (u);

compute _error (e)

compute _control _action (u) ←

send _converted _control_ action (u);

update _internal_variables(y,u, ...);

Next _Iteration:= Next _Iteration + Period;

delay until Next _Iteration;

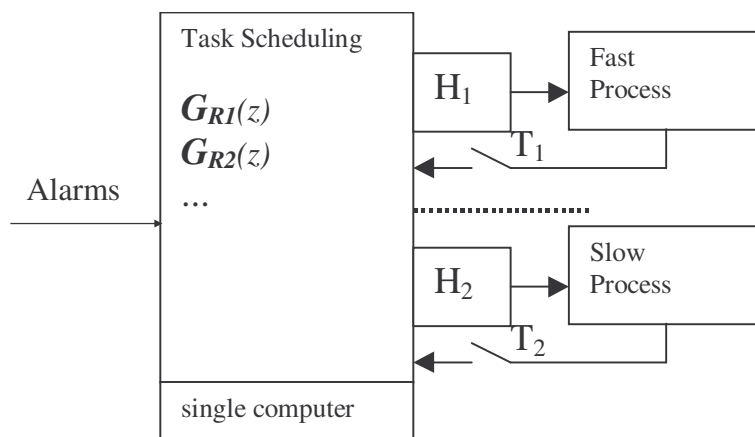
end loop;

...

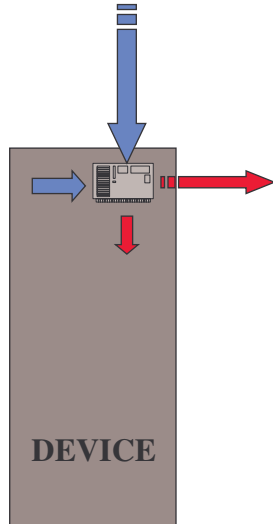
Basic assumptions in Computer control

- The Data Acq. system provides the required data
- The actuators' drivers deliver the control actions
- The CPU computes on-time the control action
- The required data are stored in the memory
- The sampling pattern is regular (constant, synchronous and uniform for any control task)
- The control algorithm is well defined
- Alternative controllers are independent
- Power supply is guaranteed

Implementation



Embedded Control Systems



- Inputs:
 - Device
 - Environment
 - Cable
 - Wireless
 - User
- Outputs:
 - Control actions
 - Self organizing
 - Data

Embedded Control Systems

- Embedded systems with:
 - hard RT constraints
 - guarantee of safe operation
 - best possible performances
- Additional issues from viewpoint of:
 - implementation
 - computation
 - algorithmic

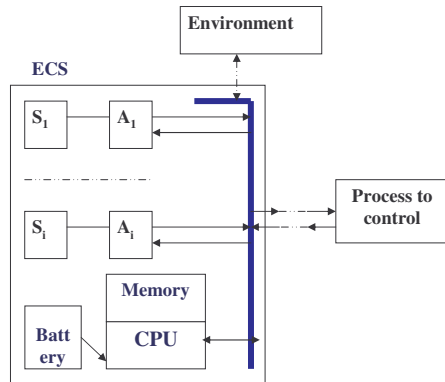
RT Control Issues

- RT Constraints:
 - Maximize the time determinism
 - For many controllers a worst-case approach works well e.g., PI, PID, State Feedback, ...
 - however, many exceptions:
 - hybrid controllers that switch between different modes with different characteristics
 - model-predictive controllers (MPC)
 - convex optimization problem solved every sample
 - execution time can vary an order of magnitude
 - Compensate the variations:
 - Measure and react
 - Feedback robustness

Control Performances

- Relevance of the control actions
 - The Control Effort concept*
- Sensitive to time delays
- Changes in the sampling period:
 - Controller parameters
 - Past data
- Commutation bumping

Control requirements



- Multiloop control
- Non-uniform sampling
- Missing data
- Variable delays
- Sampling period changes
- Mode changes
- Fault tolerant
- Safe operation
- CPU optimization
- Battery control

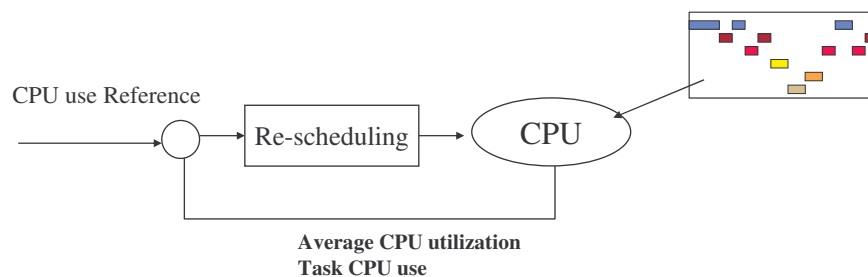
ECS: Implementation

- The same resources must be **shared** between different tasks
- **Alternative** control **algorithms** should be ready to get the control of the process
- **Working conditions**, such as priority, allocated time and memory or signals availability may change
- **Variable delays** should be considered
- **Validation** and certification

ECS: Computational viewpoint

- Economic algorithms
- Information updating
- Optional tasks
- Hybrid systems
- CPU use measurement and optimisation
- On-line scheduling
- Memory saving
- Economic hardware redundancy
- Fault detection and isolation

Feedback re-scheduling



Re-scheduling actions:

- Increase/Decrease task periods
- Increase/decrease processor frequency
- Change the set of tasks (mode change)
- ...

ECS: Control algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
- Missing data control
- Event-triggered control
- Hybrid control systems
- Decision and supervisory control
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

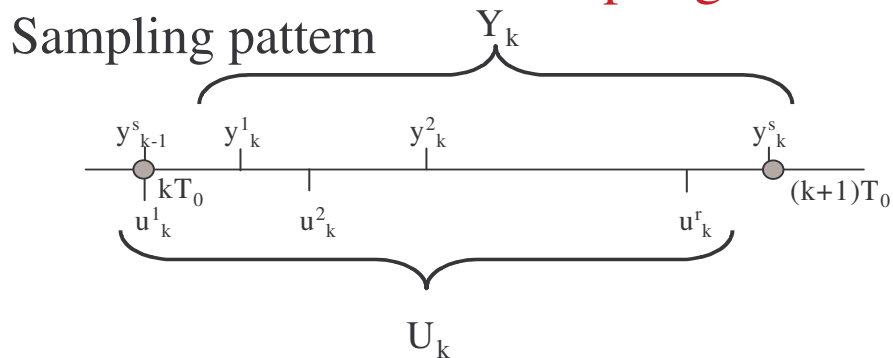
ECS: Control algorithm viewpoint

- **Reduced order models**
- Non-conventional sampling and updating patterns
- Missing data control
- Event-triggered control
- Hybrid control systems
- Decision and supervisory control
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

ECS: Control algorithm viewpoint

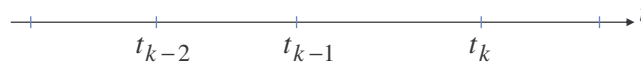
- Reduced order models
- **Non-conventional sampling and updating patterns**
- Missing data control
- Event-triggered control
- Hybrid control systems
- Decision and supervisory control
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Non-uniform sampling



- Irregular sampling
- Time delays
- Relevance of variables

Variable Sampling Time



PID Controller:

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i \int_0^t e(\tau) d\tau;$$

$$u_k - u_{k-1} = q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}$$

$$q_0 = K_p + \frac{K_d}{t_k - t_{k-1}} = K_p + \frac{K_d}{T_1}$$

$$T_1 = t_k - t_{k-1}$$

$$q_2 = \frac{K_d}{t_{k-1} - t_{k-2}} = \frac{K_d}{T_2}$$

$$T_2 = t_{k-1} - t_{k-2}$$

$$q_1 = -K_p - K_d \frac{t_k - t_{k-2}}{(t_k - t_{k-1})(t_{k-1} - t_{k-2})} + K_i(t_k - t_{k-1}) = -K_p - K_d \frac{T_1 + T_2}{T_1 T_2} + K_i T_1$$

Control task

...

...

loop

convert _sensor _analog_ digital (y), get t_k ;

compute _control _action (u);

compute T_1, T_2

compute coefficients q_i

compute _error (e)

compute _control _action (u) $\leftarrow \Delta$

send _converted _control _action (u);

update _internal _variables(y,u, ...);

Next _Iteration := Next _Iteration + Period;

delay until Next _Iteration;

end loop;

...

PID: Time delay effect

- Open loop control
- Discretized controller, independently of the plant
- Degrading as the time delay increases

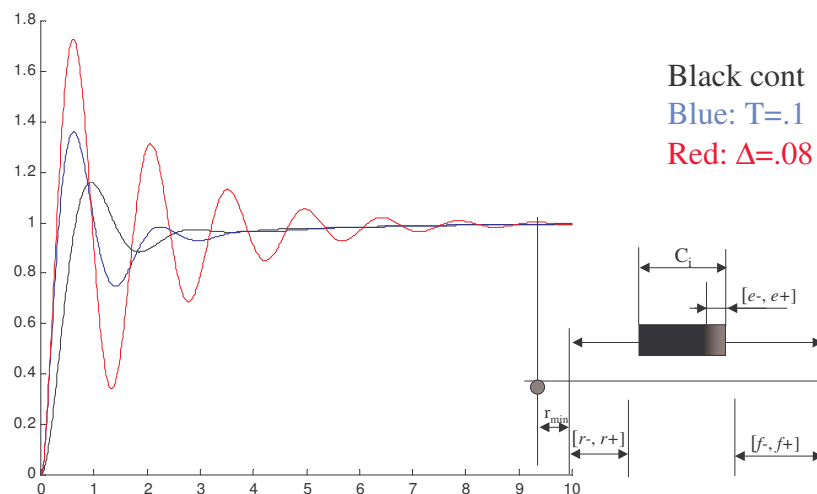
• EXAMPLE: Plant:
$$G(s) = \frac{1.5}{(s+0.5)(s+1.5)}$$

Parameters: $K_P = 8 \quad T_D = 0.2 \quad T_I = 3.2$

Sampling period: $T = 0.1 \text{ sec}$

$$q_0 = K_p + \frac{K_d}{T}; \quad q_1 = -K_p - \frac{2K_d}{T} + K_i T; \quad q_2 = \frac{K_d}{T}$$

PID: computational delays



Delay Counteraction

- Input/output delay

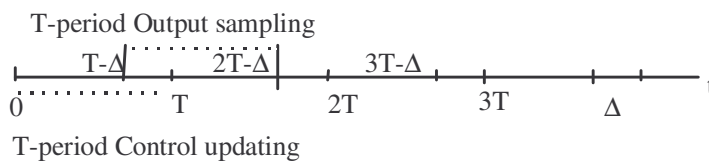
$$\dot{x}(t) = Ax(t) + Bu(t - \tau); y(t) = Cx(t)$$

$$\dot{x}(t) = Ax(t) + Bu(t); y(t) = Cx(t - \tau)$$

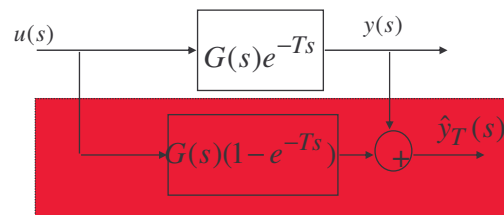
$$y(s) = G(s).e^{-s\tau}u(s)$$

- Smith Predictor
- Error prediction
- Output prediction

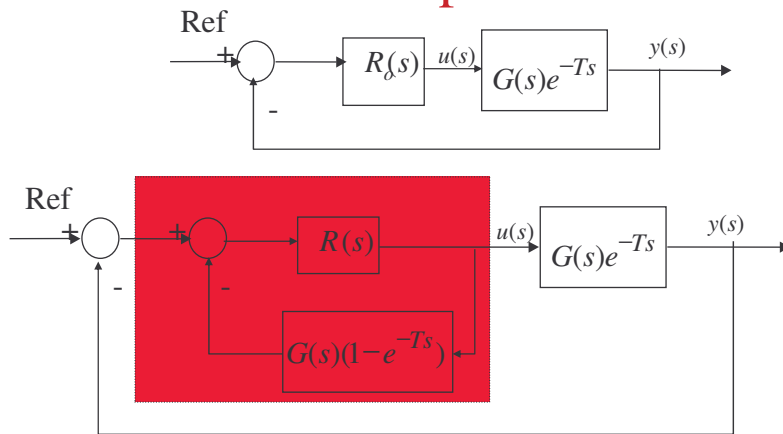
Delayed Sampling



- Classical Smith Predictor Option



Smith predictor



PID: error prediction

loop

$$e_{k,\Delta} = e_k + \frac{\Delta}{T}(e_{k,\Delta} - e_{k-1,\Delta})$$

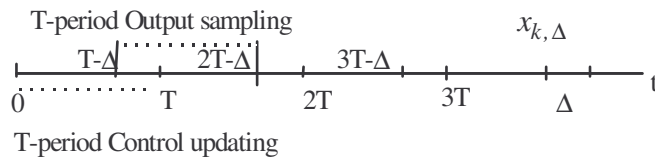
$$e_{k,\Delta} = \frac{T}{T-\Delta}e_k - \frac{\Delta}{T-\Delta}e_{k-1,\Delta}$$

```

convert_sensor_analog_digital(y);
compute_control_action(u);
compute_error(e)
compute_actual_error(eΔ)
compute_control_accion(u)
send_converted_control_action(u);
update_internal_variables(eΔ,y,u, ...);
Next_iteration:= Next_iteration + Period;
delay until Next_iteration;
end loop;
    
```

$$u_{k,\Delta} = q_0 e_{k,\Delta} + q_1 e_{k-1,\Delta} + q_2 e_{k-2,\Delta} + u_{k-1,\Delta}$$

Output/State prediction



$$\bar{x}_k = A(\Delta)x_{k,\Delta} + B(\Delta)u_{k-1} \quad u_k = K_o.[A(\Delta)x_{k,\Delta} + B(\Delta)u_{k-1}] + r_k$$

$$y_{k,\Delta} = C x_{k,\Delta}, \quad x_{k,\Delta} = A(-\Delta)x_k - A(-\Delta)B(\Delta)u_{k-1}$$

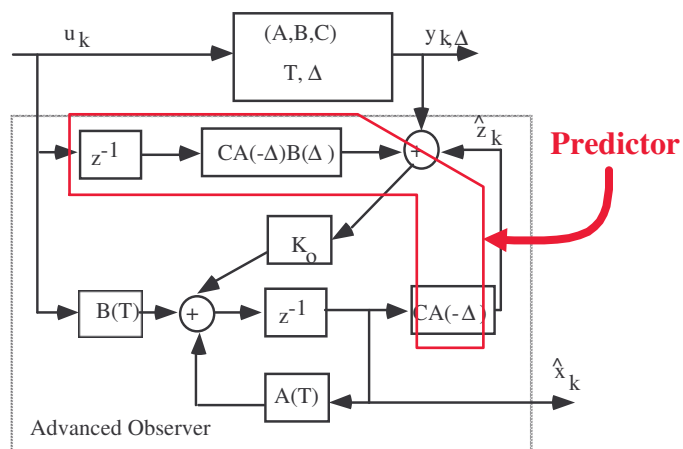
$$z_k = C \cdot A(-\Delta)x_k = y_{k,\Delta} + C \cdot A(-\Delta) \cdot B(\Delta)u_{k-1}$$

$$\hat{x}_{k+1} = A(T)\hat{x}_k + B(T)u_k + K_o(z_k - \hat{z}_k)$$

$$\hat{z}_k = CA(-\Delta)\hat{x}_k$$

Delay Predictor

- Advanced (Predictor) Observer



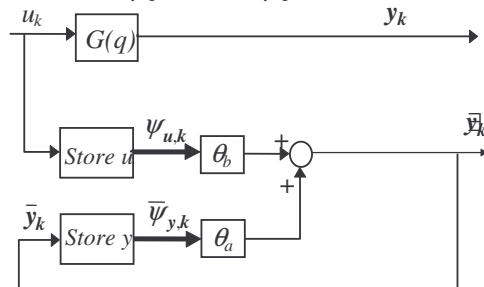
ECS: Control algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
- **Missing data control**
- **Event-triggered control**
- Hybrid control systems
- Decision and supervisory control
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Output prediction

Model-based open-loop prediction

$$\hat{y}_k = -\sum_{i=1}^n a_i \hat{y}_{k-i} + \sum_{i=1}^n b_i u_{k-i} = \hat{\psi}_{k-1}^T \cdot \theta$$



Drawbacks:

- Error dynamics is that of the process.
- Lack of robustness against disturbances.

Output prediction (2)

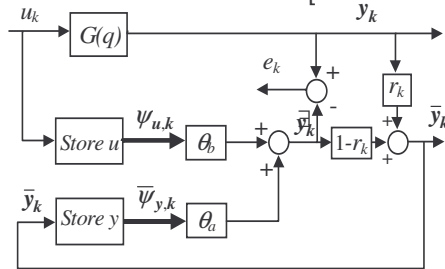
Output estimation with mixed vector of past outputs

$$\bar{y}_k = \bar{\psi}_y(k-1)^T \cdot \theta_a + \psi_u(k-1)^T \cdot \theta_b$$

$$\bar{y}_k = (1-r_k) \cdot \bar{y}_k + r_k \cdot y_k$$

$$\bar{\psi}_y(k) = [-\bar{y}_k \quad \Lambda \quad -\bar{y}_{k-(n-1)}]^T$$

$$\psi_u(k) = [u_k \quad \Lambda \quad u_{k-(n-1)}]^T$$



Convergence?

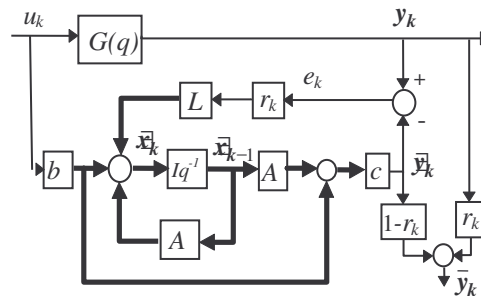
Other options

Enlarged polynomials
$$\frac{d_1 q^{-1} + \Lambda + d_L q^{-L}}{1 + c_1 q^{-1} + \Lambda + c_L q^{-L}} = \frac{(b_1 q^{-1} + \Lambda + b_n q^{-n}) \cdot E(q^{-1})}{(1 + a_1 q^{-1} + \Lambda + a_n q^{-n}) \cdot E(q^{-1})} = \frac{D(q^{-1})}{C(q^{-1})}$$

Linear state observer:

$$\hat{x}_k = A^N \hat{x}_{k-N} + A^{N-1} b u_{k-N} + \Lambda + b u_{k-1} + L(y_k - c(A^N \hat{x}_{k-N} + A^{N-1} b u_{k-N} + \Lambda + b u_{k-1}))$$

$$\hat{y}_{k+j} = c(A^j \hat{x}_k + A^{j-1} b u_k + \Lambda + b u_{k+j-1}), j=1, K, N-1$$

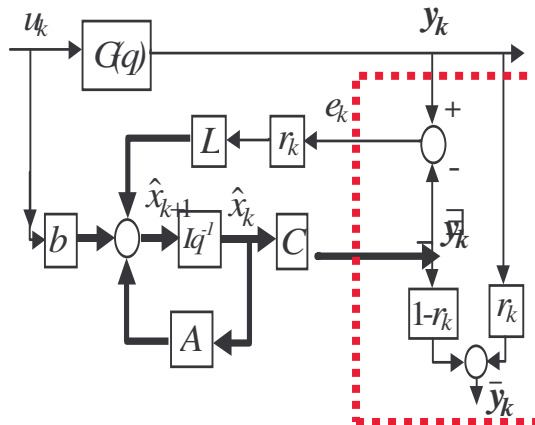


$$(A^N)^T, (cA^N)^T \Rightarrow L$$

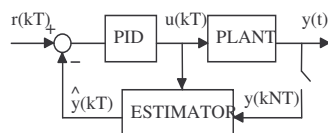
Missing Data

The output is only available at some time instants: $r_k = 1; \quad r_k = \{0,1\}$

KALMAN Filter



Missing data Control



Example.

$$G(s) = 1/(s^2 + 4s + 3)$$

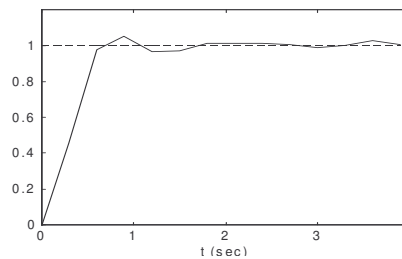
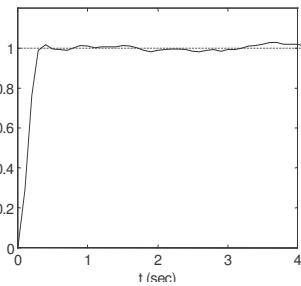
$$N \cdot T = 300 \text{ ms}$$

$$K_p = 8, T_d = 0.25, T_i = 1.2$$

Extended order predictor

$$T = 100 \text{ ms} (N=3)$$

$$(K_p = 20, T_d = 0.25, T_i = 1.2)$$



$$E = 1 + 0.67q^{-1} + 0.33q^{-2}$$

Parameter estimation

Regular pattern: dual rate model

$$G(z) = \frac{B_N(z^{-1})}{A_N(z^{-1})} = \frac{c_{N,n-1}z^{-1} + c_{N,n-2}z^{-2} + \Lambda + c_1z^{-(N-n)} + c_0z^{-N}}{1 + d_{n-1}z^{-N} + \Lambda + d_1z^{-N(n-1)} + d_0z^{-Nn}}$$

Regression vector and parameters:

$$\psi_k = [-y_{k-N} - y_{k-2N} \Lambda - y_{k-n} u_{k-1} u_{k-2} \Lambda u_{k-n}]^T$$

$$\theta_N = [d_{n-1} d_{n-2} \Lambda d_0 c_{n,N-1} c_{n,N-2} \Lambda c_0]^T$$

Irregular pattern or fast model

$$\hat{\psi}_k = [-\hat{y}_{k-1} - \hat{y}_{k-2} \Lambda - \hat{y}_{k-n} u_{k-1} \Lambda u_{k-n}]^T$$

Output predictor:

$$\hat{y}_k = f(\hat{\theta}_k, y_j r_j, u_j, \hat{y}_j)$$

$$\hat{y}_k = (1 - r_k) (\hat{\psi}_y(k-1)^T \cdot \theta_a + \psi_u(k-1)^T \cdot \theta_b) + r_k y_k$$

Fast parameter estimation

$$\gamma_k = \frac{P_k}{\lambda + \bar{\psi}_k^T P_k \bar{\psi}_k}$$

$$\bar{\theta}_k = \bar{\theta}_{k-1} + \gamma_k \cdot \bar{\psi}_k (y_k - \bar{\psi}_k^T \cdot \bar{\theta}_{k-1}) \cdot r_k$$

$$P_{k+1} = \frac{1}{\lambda} (I - \gamma_k \bar{\psi}_k \bar{\psi}_k^T) P_k \cdot r_k + P_k (1 - r_k)$$

$$\bar{y}_k = f(\bar{\theta}_k, y_j r_j, u_j, \bar{y}_j)$$

($r_k=1$ if measurement and 0 if not).

- Convergence depend on sampling period and data availability rate
- The stability of the output predictor is a necessary but not sufficient condition for convergence.
- For small T wrong attractors appear close to the dual-rate poles.

(Poles in those positions have an oscillating impulse response with the periodicity of the lower-rate sampling).

Convergence

Example: $G(s) = \frac{2}{s^2 + 3s + 1}$, with $N=3$.

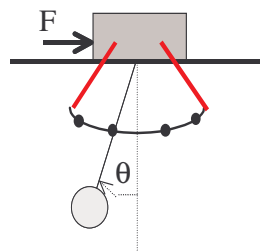
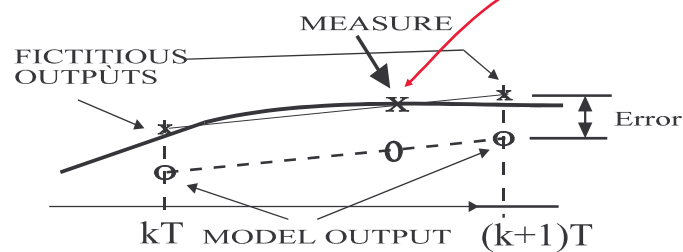
If $T = 0.02$, the estimates are

$$G^*(z) = 10^{-4} \frac{0.224z + 5}{z^2 + 0.986z + 0.984},$$

instead of the correct one $10^{-3} \frac{0.392z - 0.384}{z^2 - 1.941z + 0.942}$.

If $T = 0.7$, the estimates are the correct parameters.

Event-driven control



Event-driven sensor

$$M\ddot{x} = F + T\sin\theta - kx$$

$$m\ddot{y}_m = -T\sin\theta - m\ddot{y}_m = T\cos\theta - mg$$

$$x_m = x_l + L\sin\theta, y_m = -l\cos\theta$$

$$\ddot{\theta} = \frac{-(F - kx)\cos\theta - ml\sin\theta\cos\theta\ddot{\theta} - gM\sin\theta}{ML - mL\cos^2\theta}$$

$$\bar{M}\ddot{x} = F - kx - ml\cos\theta\ddot{\theta} + ml\sin\theta\dot{\theta}^2$$

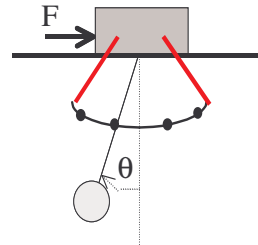
$$\bar{M} = M + m$$

$$T=0.9; M=40:$$

$$M=15:$$

$$G(z) = \frac{-7.39 \cdot 10^{-4}(z-1)(z-0.9952)}{(z-0.9945)((z-0.5242)^2 + 0.8464^2)}$$

$$G(z) = \frac{-7.541 \cdot 10^{-4}(z-1)(z-0.9953)}{(z-0.9928)((z-0.6261)^2 + 0.7751^2)}$$



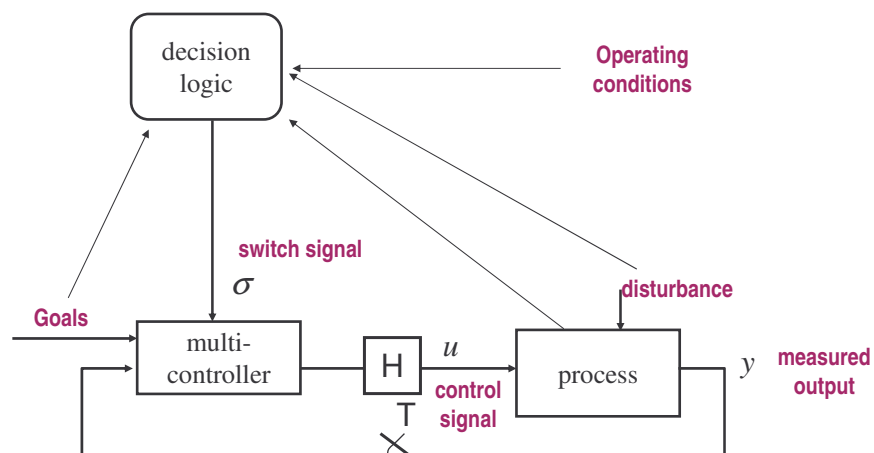
Partial parameter set

$$\theta \approx \begin{pmatrix} -2.043 \\ 2.038 \\ -0.986 \\ -7.39 \cdot 10^{-4} \\ 3.5 \cdot 10^{-6} \\ 7.36 \cdot 10^{-4} \end{pmatrix} + \begin{pmatrix} -0.0135 \\ 0.0135 \\ 0 \\ -10^{-6} \\ 2.8 \cdot 10^{-9} \\ 10^{-6} \end{pmatrix} \Delta m$$

ECS: Control algorithm viewpoint

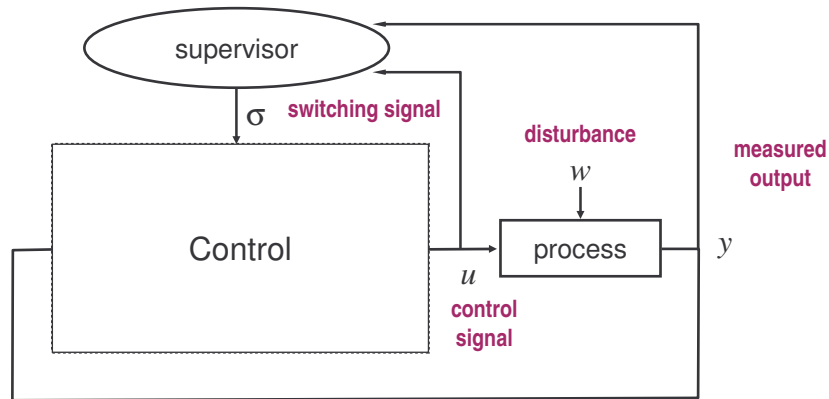
- Reduced order models
- Non-conventional sampling and updating patterns
- Missing data control
- Event-triggered control
- **Hybrid control systems**
- **Decision and supervisory control**
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Hybrid control system



Supervisory control

{ Alternative controllers
Switch among them online



Supervisor:

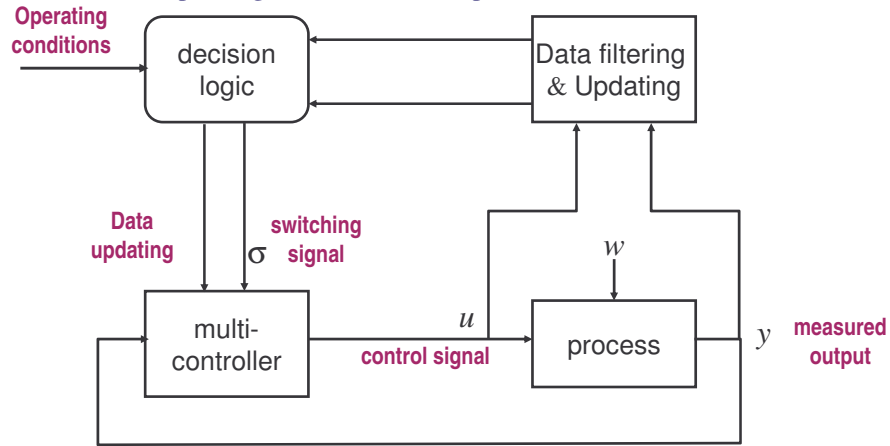
- places in the feedback loop the 'best' controller
- switching **effects**

Types of supervision

- **Pre-routed supervision**
 - try one controller after another in a pre-defined sequence
- **Performance-based supervision**
 - stop when the performance seems acceptable
- **Estimator-based supervision**
- **Goal and Operating-conditions based supervision**
 - keep controller while observed performance is acceptable
 - when performance of current controller becomes unacceptable, **switch** to controller that leads to best expected performance based on available data
 - estimate process model from observed data
 - select controller based on current estimate – *Certainty Equivalence*
- **On-line re-scheduling of the control tasks**

External commuting

Goal and operating-condition based supervision



Controller commuting

- Given a change on:
 - Goal
 - Working point
 - Operating conditions



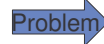
Change in controller

$$q_0 = K_p + \frac{K_d}{T}; \quad q_1 = -K_p - \frac{2K_d}{T} + K_i T; \quad q_2 = \frac{K_d}{T}$$

- Controller selection (or computation)

$$u_k = (K + K^* T)x_k$$

- Transfer:



- **Stability**
- **Performances**

- Parameter updating
- Controller initialization

Controller commuting: Stability issues

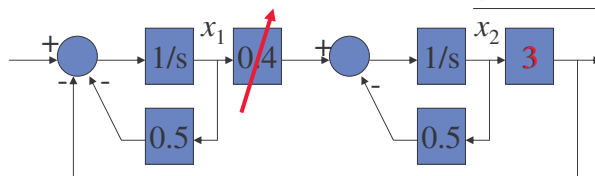
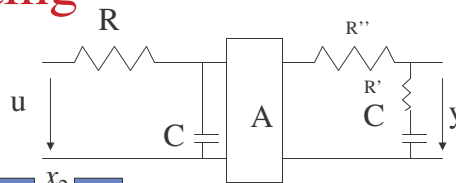
**Each controller stabilizes the plant under control,
 But ... what under commuting?**

- Common Lyapunov function
- Controller initialization
- Controller resetting

Controller Commuting

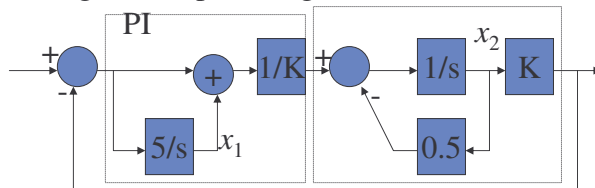
Change in the process:

Change in the amplifier gain:



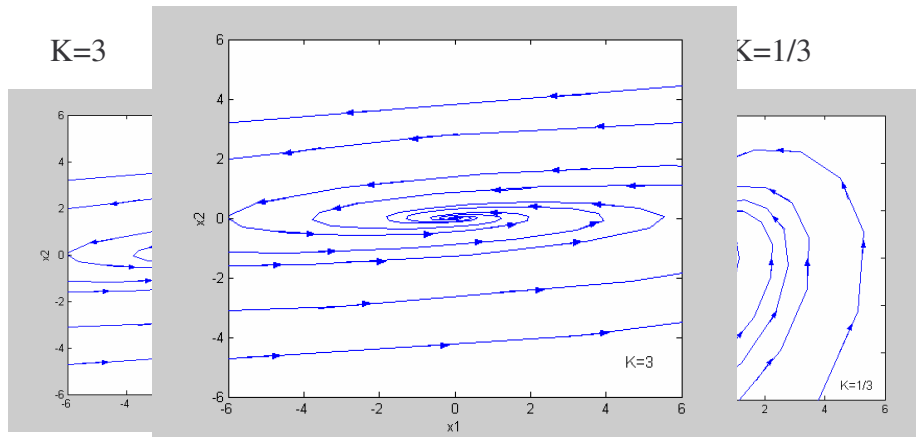
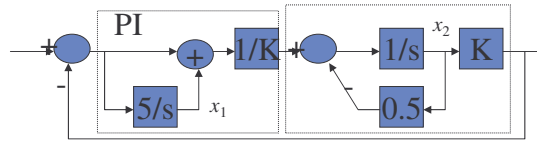
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5 & -3 \\ 0.4 & -0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Change in the process gain:

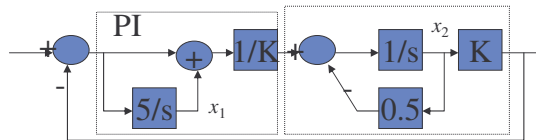


$$K=3 \rightarrow 1/3$$

Stability

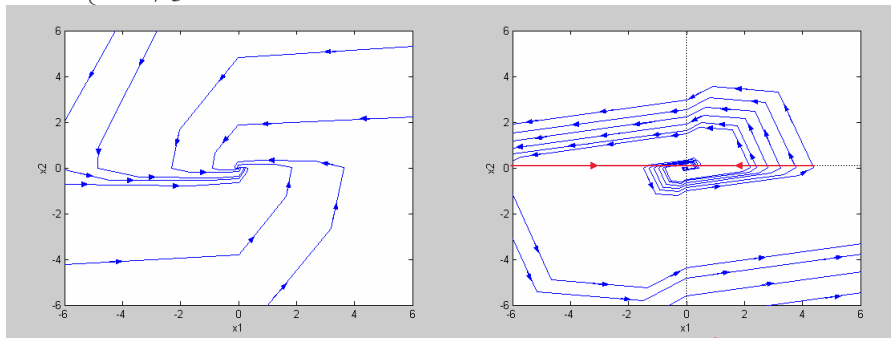


(Hybrid) Stability



$$\sigma(t) = \begin{cases} K = 3 & x_1 x_2 \leq 0 \\ K = 1/3 & x_1 x_2 > 0 \end{cases}$$

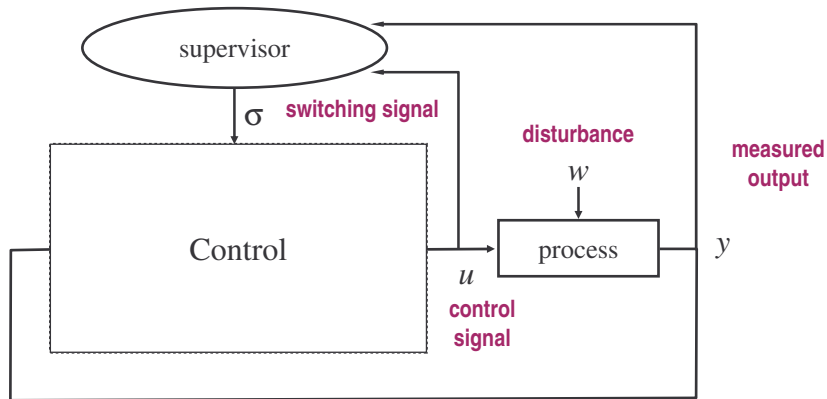
$$\sigma(t) = \begin{cases} K = 3 & x_1 x_2 \geq 0 \\ K = 1/3 & x_1 x_2 < 0 \end{cases}$$



resetting

Supervisory control

Online re-scheduling of control tasks



Supervisor:

- Evaluates the effects of delays
- Assign priorities

Control Effort

- The maximum allowable time delay is given by the phase margin, derived from the frequency analysis of the open loop output feedback controlled system

$$\Delta \leq \frac{\psi_m}{\omega_c} \quad |G(j\omega_c)| = 1 \quad \psi_m = \pi + \angle G(j\omega_c)$$

State feedback:

- Given $\dot{x}(t) = Ax(t) + bu(t)$ being $a(s) = |sI - A| = \prod_{i=1}^n (s - a_i)$

$$u(t) = -kx(t) + r(t)$$

$$A_n = -\sum_{i=1}^n a_i$$

- The state feedback control k places the closed loop poles: $p(s)$

$$p(s) = |sI - A + bk| = \prod_{i=1}^n (s - p_i)$$

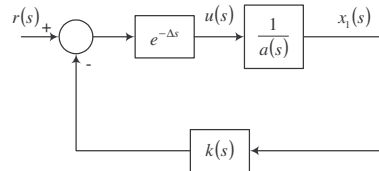
$$P_n = -\sum_{i=1}^n p_i$$

- Definition:** Define *Control Effort* (CE) as the shift of the poles, from the open to the closed loop position in the s-plane

$$CE = \sum_{i=1}^n (a_i - p_i) = -\sum_{i=1}^n (p_i - a_i) = k_n$$

Control Effort vs. Time Delay

- Now, assume a control action time delay Δ , and $P_n \gg A_n$, that is, the loop poles are shifted well on the left



- From the loop frequency transfer function of the system (without delay)

$$G(j\omega) = \frac{k_n (j\omega)^{n-1} + K + k_1}{(j\omega)^n + A_n (j\omega)^{n-1} + K + A_1}$$

$$|G(j\omega_c)| = 1, \rightarrow 1 \cong \left| \frac{k_n}{j\omega_c + A_n} \right|$$

$$\rightarrow \omega_c \cong k_n = P_n - A_n$$

- That is, an approximate expression for the maximum allowable time delay is given by

$$\Delta < \frac{\psi_m}{P_n - A_n} = \frac{\psi_m}{k_n}$$

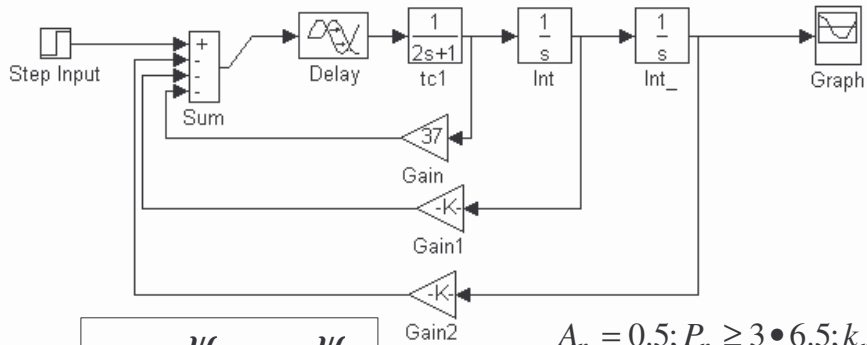
Example

Given the process: $G(s) = \frac{1}{s^2(1+2s)}$

Design a controller to achieve:

- over-damped step response
- 0.5 sec settling time

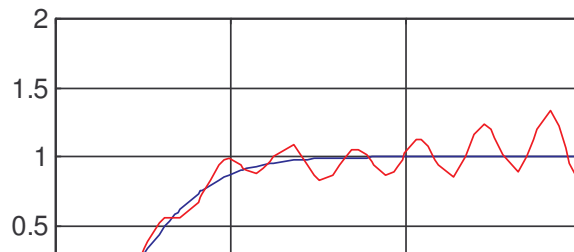
- Estimate the maximum allowable delay in the loop to keep stability



$$\Delta < \frac{\psi_m}{P_n - A_n} = \frac{\psi_m}{k_n}$$

$$A_n = 0.5; P_n \geq 3 \bullet 6.5; k_n \cong 20$$

$$\psi_m \approx 80^\circ \rightarrow \Delta \approx \frac{4\pi}{180} \approx 0,07$$

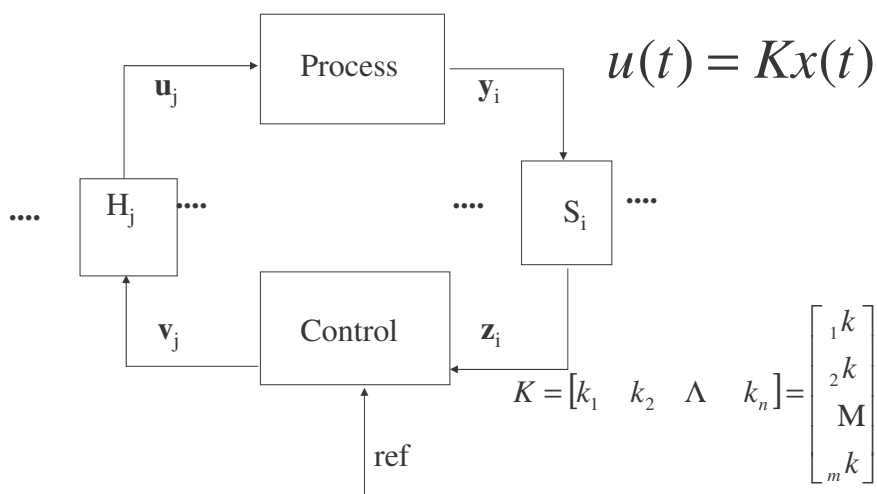


$\Delta=0.073\text{sec}$

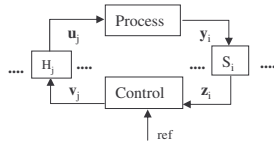
Performance degrading

- The *Control Effort*, defined as the shift in damping from the open loop poles to the closed loop poles, provides a useful way to obtain the maximum allowable time delay, for both, continuous and discrete systems.
- The longer the sampling period T is, the more sensitive to the time delay the design is.

MIMO controlled plant



MIMO controlled plant



$C = I; y(t) = x(t); u(t) = Kx(t)$
 Measurement relevance:

$$K_j = [k_1 \quad k_2 \quad \dots k_{j-1} \quad 0 \quad k_{j+1} \dots k_n]$$

$$\frac{x_i(s)}{z_i(s)} = {}_i c (sI - A + BK_i)^{-1} k_i$$

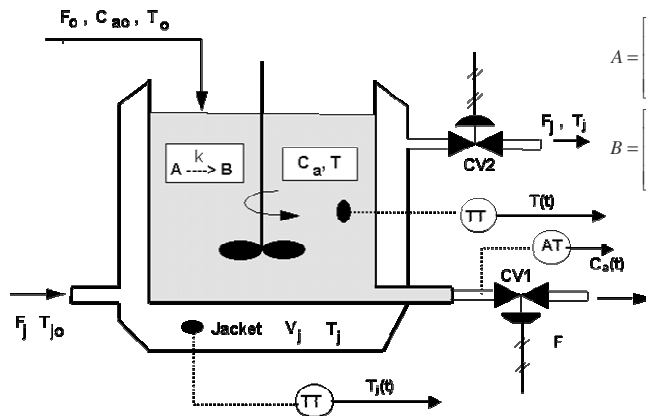
Control relevance:

$${}_i K = \begin{bmatrix} {}_1 k \\ {}_2 k \\ \vdots \\ M \\ 0 \\ \vdots \\ {}_{j+1} k \\ \vdots \\ M \\ m k \end{bmatrix}$$

$$\frac{v_j(s)}{u_j(s)} = {}_j k (sI - A + B_j K)^{-1} b_j$$

Maximum Time Delay

Reactor: model

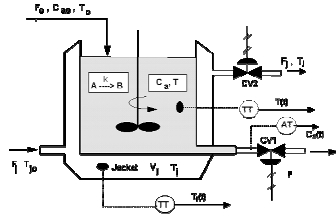


$$A = \begin{bmatrix} -1.705 & -0.2519 & 0 \\ 23.088 & -28.71 & 20.9 \\ 0 & 200.3 & -216.89 \end{bmatrix};$$

$$B = \begin{bmatrix} 2.918 & 0 \\ -28.6 & 0 \\ 0 & -415.29 \end{bmatrix}$$

$$x = [C_a \quad T \quad T_j]^T; u = [F \quad F_j]^T$$

Reactor: control



$$\{a_i\} = \{eig(A)\} = \{-2.5878, -7.73, -236.987\} \quad A_n = -247.3$$

Control Goal: $p = \{-320, -340, -360\}$; $P_n = -1020$

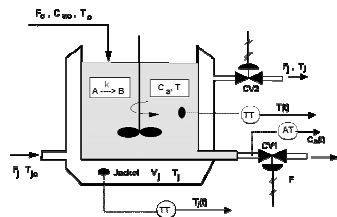
$$K = \begin{bmatrix} 858.5 & 68.5676 & 4.6683 \\ -40.463 & -4.2238 & -0.5505 \end{bmatrix}$$

Assume F active and F_j open

$$\{eig(A - b_{1,1}k)\} = \{-405.7, -336.1, -49.5\}; \rightarrow S_2 = -791.4$$

$$S_2 - P_n = 228.7$$

Reactor: variables' relevance



The signal relevance depends on the control solution!!

Variable	u_2	u_1	x_1	x_2	x_3
o-l poles	-405.7 -336.1 -49.5	0.2 -135.6 -340.6	0.71492.9 -0.4585	-2513.3 -340.5 -127.2	-340.9 -225.2 $\pm 179.5i$
S=Sum of poles	-719.4	-475.9	1485.1	-2981	-719.4
Relevance = $S - P_n$	228.7	544	2963.6	-1961	228.7

Balance the performances

As a result of the selected controllers and scheduling:

compute Control Effort

Schedule all the tasks and compute

Delays

Redesign the control taking into account delays

Re-compute the **CE** and update the *delays*

Check the schedulability

Performance's balance

Computation delays depend on:

- Operating mode: Control Algorithm Complexity
- Priority
- CPU's load

Delay's effect depends on:

- Sampling period
- Control effort

Performance degrading depends on:

- Sampling period
- Delays
- Loops interaction

$$J_D = \sum_{i=1}^N K_i \Delta_i$$

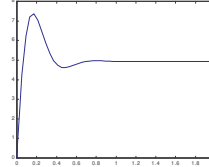
Example

Initial Plants $G_1(s) = \frac{100}{s+6}$ $G_2(s) = \frac{100}{s+1}$

Control Goal: $M(s) = \frac{100(s+7)}{s^2+13s+142}$

Required controllers

$G_{R,1}(s) = \frac{1}{s+7}$ $G_{R,2}(s) = \frac{5s+135}{100(s+7)}$

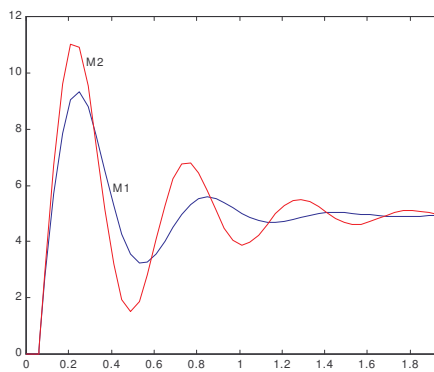


Plant delay tolerance

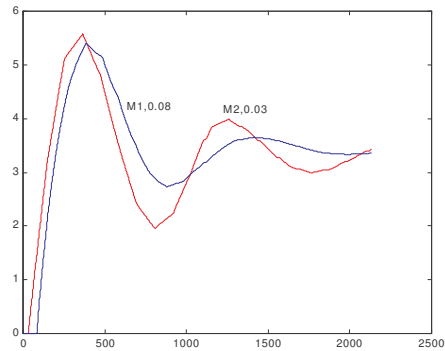
Plant	ω_c (rad/sec)	ψ_m (rad)	Δ_{max}
S ₁	7,6	1,41	0,185
S ₂	11	1,22	0,111

Delays' Balance

Same delay: 60 msec

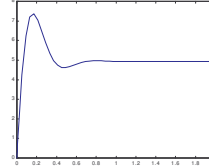


80 & 30 msec delays



Initial Plants $G_1(s) = \frac{100}{s+6}$ $G_2(s) = \frac{100}{s+1}$

Control Goal: $M(s) = \frac{100(s+7)}{s^2+13s+142}$



$G_{R,1}(s) = \frac{1}{s+7}$ $G_{R,2}(s) = \frac{5s+135}{100(s+7)}$

Multitask system:
 2 G_1
 1 G_2
 1 System task

Table 4. DM scheduling of the 4 tasks

	WCET	Period	Min. Delay (msec)	Max. Delay (msec)	Average Delay, Δ (msec)	CAI %	Control Effort K	Degrading (rad) $K*\Delta$	TOTAL degrad.
T1	22	70	22	22	22	0,0	0	0	1,309
T2	15	100	15	37	26,0	22,0	7,6	0.198	
T3	17	110	17	54	35,5	33,6	11	0.391	
T4	19	110	36	95	65,5	53,6	11	0.720	

DM scheduling with CAI reduction

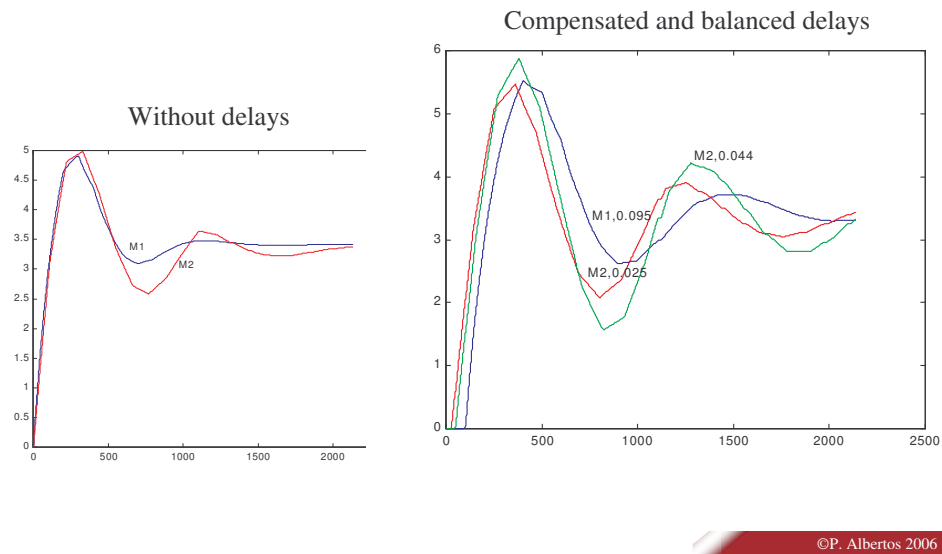
	WCET	Period	Min. Delay (msec)	Max. Delay (msec)	Average Delay, Δ (msec)	CAI %	Control Effort K	Degrading (rad) $K*\Delta$	TOTAL degrad.
T1	22	70	27	28	27,5	1,4	0	0	1,927
T2	15	100	39	41	40,0	2,0	7,6	0,304	
T3	17	110	53	56	54,5	2,7	11	0,600	
T4	19	110	91	95	93,0	3,6	11	1,023	

Re-scheduling minimizing the control performance degrading

	WCET	Period	Min. Delay (msec)	Max. Delay (msec)	Average Delay, Δ (msec)	CAI %	Priority	Degrading (rad) $K*\Delta$	TOTAL degrad.
T1	22	70	56	60	58	5,7	3	0	1,392
T2	15	100	94	97	95,5	3,0	4	0,726	
T3	17	110	22	23	22,5	0,9	1	0,247	
T4	19	110	38	40	39	1,8	2	0,429	

Results

Step responses



ECS: Control algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
- Missing data control
- Event-triggered control
- Hybrid control systems
- Decision and supervisory control
- **Multimode control**
- **Sampling rate changes**
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

PID: Sampling period effect

- Open loop control
- Discretized controller, independently of the plant
- Degrading as the sampling period increases

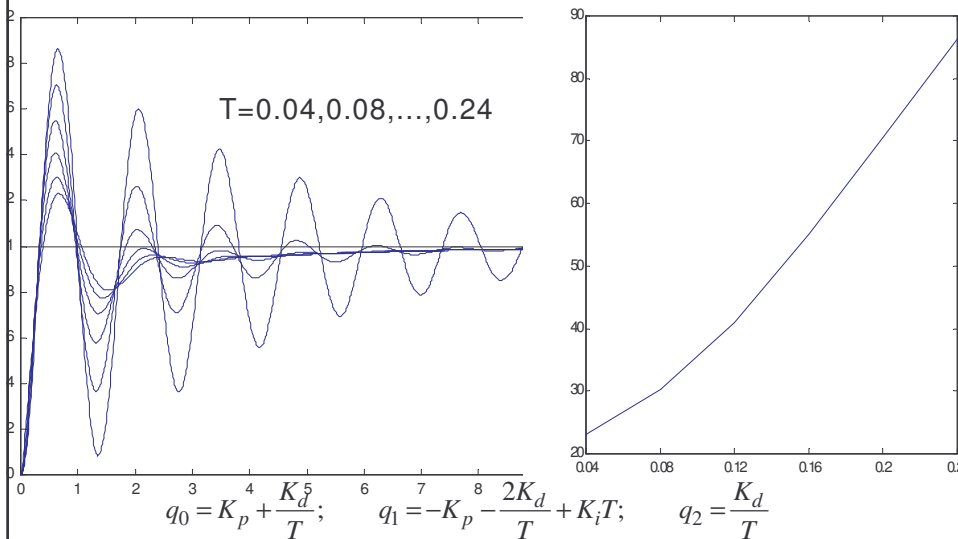
• EXAMPLE: Plant:
$$G(s) = \frac{1.5}{(s+0.5)(s+1.5)}$$

Parameters:
$$K_P = 8 \quad T_D = 0.2 \quad T_I = 3.2$$

Sampling periods: $T = 0.04, 0.08, \dots, 0.24$ sec

$$q_0 = K_p + \frac{K_d}{T}; \quad q_1 = -K_p - \frac{2K_d}{T} + K_i T; \quad q_2 = \frac{K_d}{T}$$

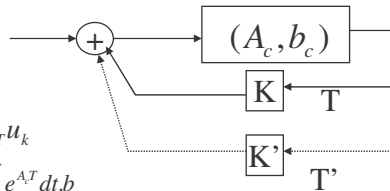
T effect



Variable sampling period

- State Feedback:**

$$\dot{x}(t) = A_c x(t) + b_c u(t)$$



$$u(t) = Kx(t) + r(t)$$

$$\text{eig}(A_c + b_c K) = \text{eig}(\bar{A}) = \{p\}$$

$$x_{k+1} = A_T x_k + b_T u_k$$

$$A_T = e^{A_c T} \quad b_T = \int_0^T e^{A_c t} dt b_c$$

$$u_k = K_T x_k + r_k$$

$$p \cong \text{eig}(A_T + b_T K_T) \cong \text{eig}(A_T + b_T K_T)$$

$$e^{\bar{A}T} = A_T + b_T K_T$$

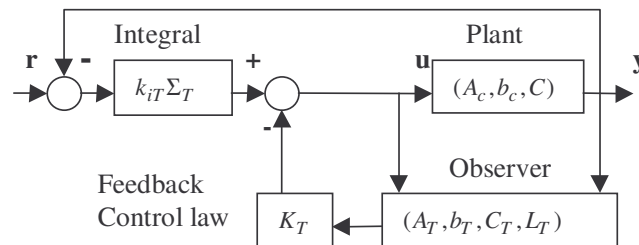
$$T \Rightarrow T + \Delta;$$

$$K_T \cong K(I + \bar{A}T/2)$$

$$K_{T+\Delta} \cong K_T + K\bar{A}\Delta/2$$

Output Controller

- Integral error**
- Output feedback**



Observer

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + b_c u(t) \\ y(t) &= Cx(t) \end{aligned}$$

$$\dot{\hat{x}}(t) = A_c \hat{x}(t) + b_c u(t) + L[y(t) - C\hat{x}(t)]$$

$$\dot{\tilde{x}} = (A_c - LC)\tilde{x} = A_o \tilde{x}$$

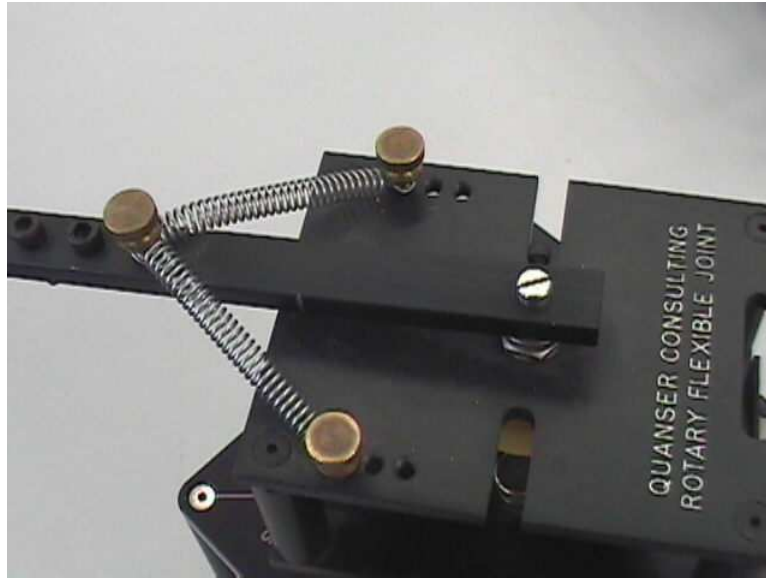
$$T \quad e^{A_o T} = A_T - L_T C \Rightarrow L_T = L_0 T + L_1 T^2 / 2 \quad \begin{cases} L_0 = L \\ L_1 = (LCA_c + A_o LC)C^\# \end{cases}$$

We must update the model (A_T, b_T) as well as the gain L_T

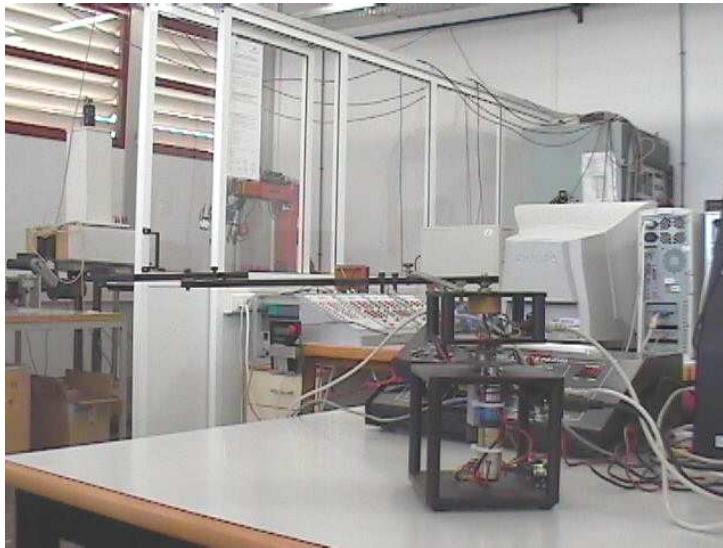
$$\begin{aligned} T \Rightarrow T + \Delta; \quad & A_{T+\Delta} = A_T + A_c(I + A_c T)\Delta \\ & b_{T+\Delta} = b_T + b_c(I + A_c T)\Delta / 2 \\ & L_{T+\Delta} = L_T + (L + A^2 TC^\#)\Delta \end{aligned}$$

Flexible arm prototype

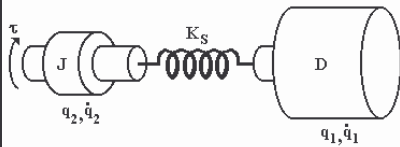




Flexible arm prototype



Mathematical model



The dynamical equations of this system are:

$$J\ddot{q}_2 - K_S q_1 = \tau$$

$$D\ddot{q}_1 + D\dot{q}_2 + K_S q_1 = 0$$

The torque is generated by the voltage applied to the d.c. motor:

$$V = IR_m + K_e \dot{q}_2 \Rightarrow I = \frac{1}{R} V - \frac{K_e}{R} \dot{q}_2$$

in this way

$$\tau = K_e I = \frac{K_e}{R} V - \frac{K_e^2}{R} \dot{q}_2$$

State+integral feedback control

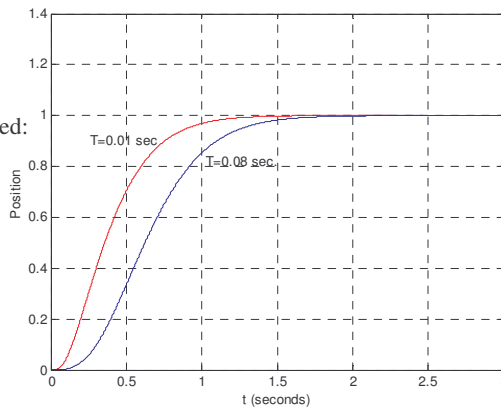
Controllers' gain,

for $T=0.08$ sec $[K_2 \ K_1] = [0.824 \ 7.032 \ -0.271 \ -0.113 \ 1.1623]$

for $T=0.01$ sec $[K_2 \ K_1] = [64.91 \ -16.25 \ 8.173 \ 7.2 \ 156.61]$

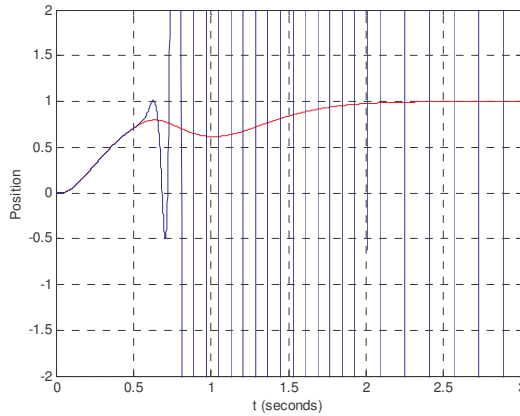
Sampling period effect

The performance of the controlled system when each controller is applied:



Change in the sampling period

$T_1 = 0.01$ to $T_2 = 0.08$ at $t = 0.5$ sec



Updating the parameters (red)

Keeping the same controller (blue)

Information updating Dynamic Controller

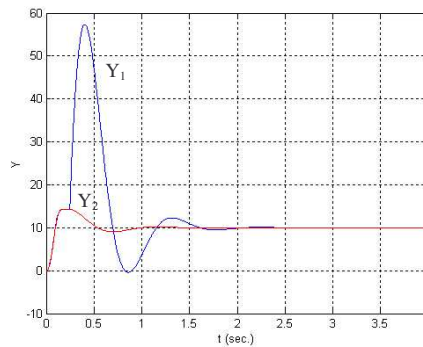
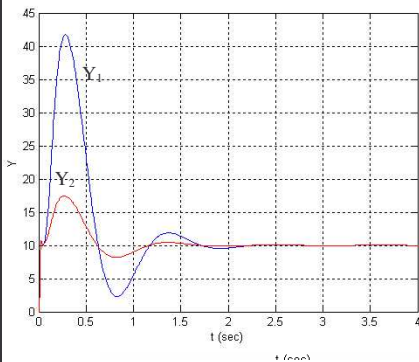
for $T = 0.08$ sec. $G_k(z) = \frac{(3.12z^2 - 3.903z + 1.349)}{(z + 0.5119)(z - 1)}$

for $T = 0.01$ sec. $G_k(z) = \frac{(105.6424z^2 - 202.7365z + 97.7)}{(z + 0.498)(z - 1)}$

Plant: $T: 0.01 \rightarrow 0.08$

for $T = 0.08$ sec.

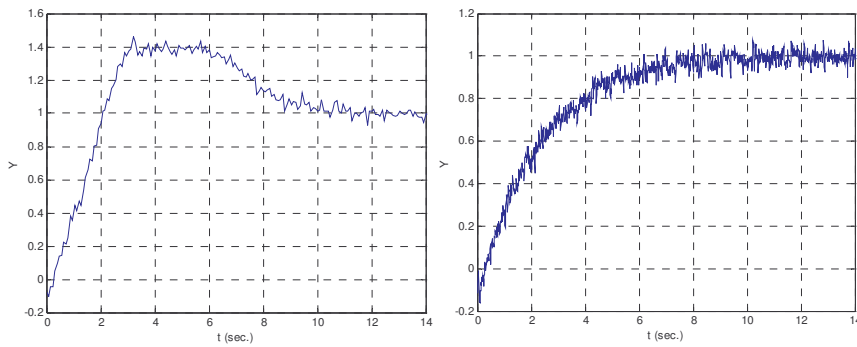
$G_n(z) = \frac{(3.12z^2 - 3.903z + 1.349)}{(z + 0.5119)(z - 1)}$ $T: 0.08 \rightarrow 0.01$



Experimental rig: Change in sampling period

for $T_2=0.08$ sec. $K = [0.206 \quad 1.758 \quad -0.06775 \quad -0.02825 \quad 0.290575]$

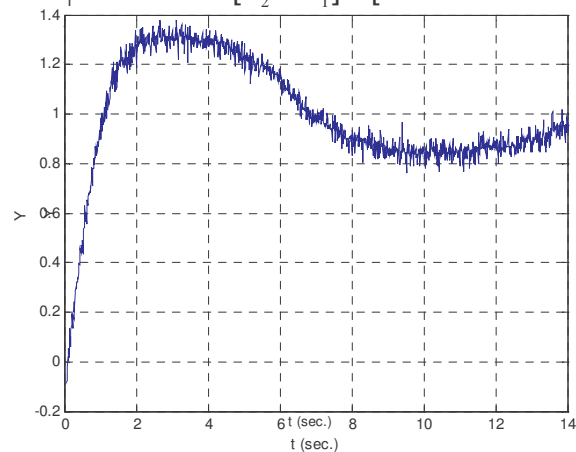
for $T_1=0.01$ sec. $[K_2 \quad K_1] = [16.22 \quad -4.06 \quad 0.1734 \quad 0.154 \quad 7.17]$



Experimental rig: Change in sampling period

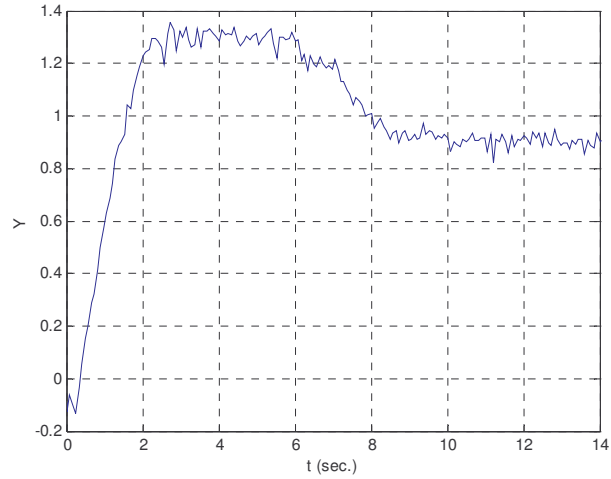
for $T_2=0.08$ sec. $K = [0.206 \quad 1.758 \quad -0.06775 \quad -0.02825 \quad 0.290575]$

for $T_1=0.01$ sec. $[K_2 \quad K_1] = [16.22 \quad -4.06 \quad 0.1734 \quad 0.154 \quad 7.17]$



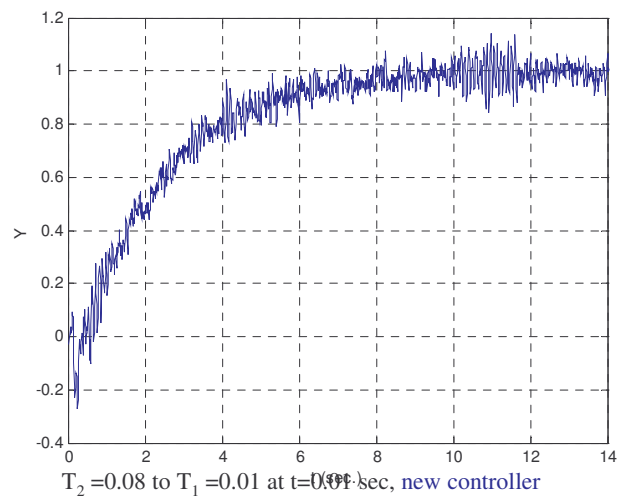
$T_2=0.08$ to $T_1=0.01$ at $t=0.08$ sec, same controller

Experimental rig: controller updating



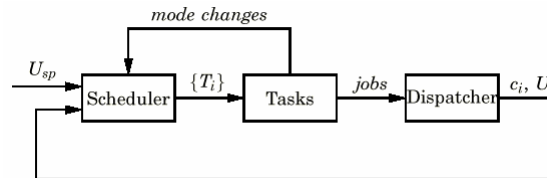
$T_1 = 0.01$ to $T_2 = 0.08$ at $t=0.01$ sec, new controller

Experimental rig: Controller updating



$T_2 = 0.08$ to $T_1 = 0.01$ at $t=0.01$ sec, new controller

Feedback re-scheduling



- According to the mode of operation change periods, re-schedule and check if feasible
- Apply transfer controllers

ECS: Control algorithm viewpoint

- Reduced order models
- Non-conventional sampling and updating patterns
- Missing data control
- Event-triggered control
- Hybrid control systems
- Decision and supervisory control
- Multimode control
- Sampling rate changes
- Fault-tolerant control
- Degraded and back-up (safe) control strategies
- Battery monitoring and control

Conclusions

- ES is a growing area of interest
 - Software issue
 - Control presence
- ECS
 - Implementation
 - Computational
 - Algorithmic
- ECS Design

ARTIST2

Real-Time Issues



Alfons Crespo

Universidad Politécnica de Valencia
Instituto de Automática e Informática Industrial
<http://www.gii.upv.es/personal/alfons>
acrespo@disca.upv.es

03/20/06

© Alfons Crespo 2006

Goals

The goal of this lecture is to provide an overview of the basic concepts of the real-time embedded systems.

03/20/06

© Alfons Crespo 2006

Outline

- Introduction
- Real-time task model
- Schedulability analysis
- Real-time operating system support
- Real-Time Languages

Introduction

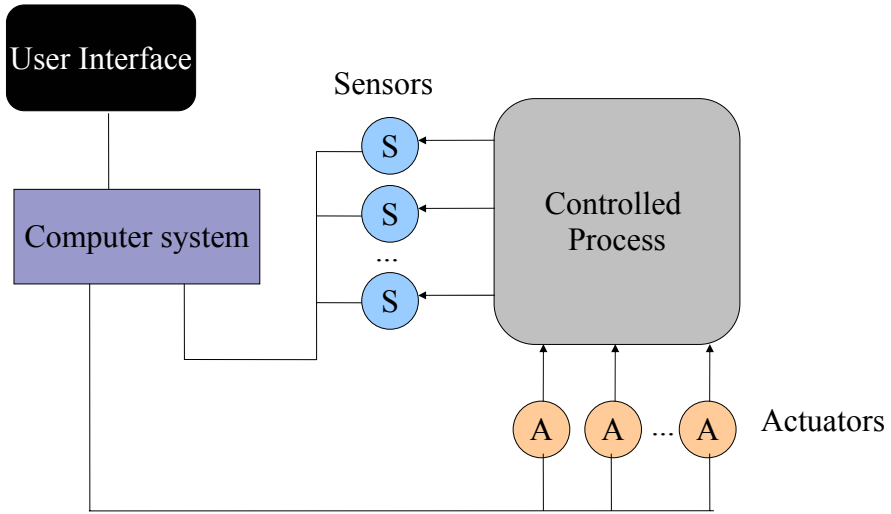
Embedded Control System

A embedded control system is composed by a **computer system** embedded in a **larger engineering system** and **performing control functions** in all or part of this environment

Examples

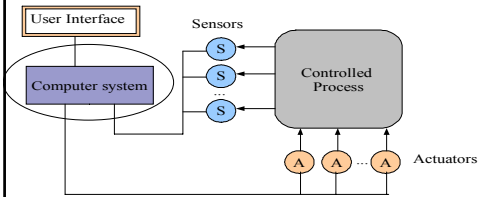
Aerospace systems, trains, cars, robotic systems, communication systems,

Introduction



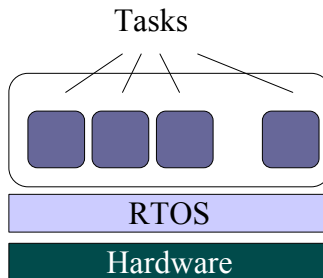
03/20/06

Introduction



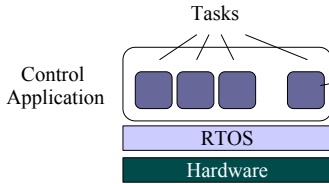
Computer System

Control Application



03/20/06

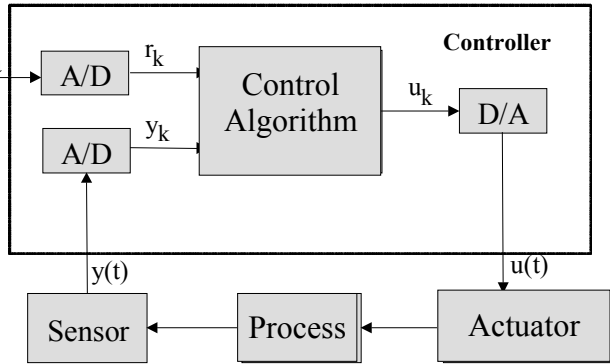
Introduction



Each task in the application has the following structure

```

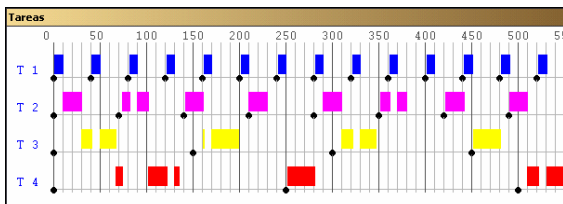
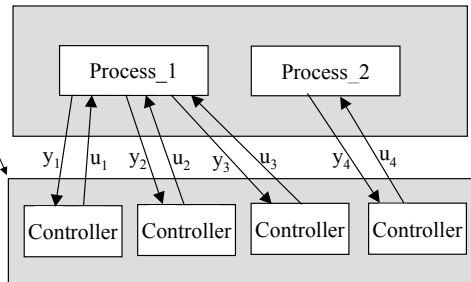
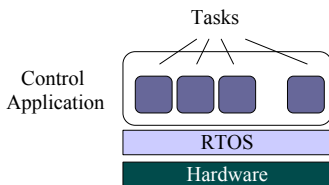
task Controller
each Sample do
  get_sensor (y);
  determine_action(u);
  send_action(u);
  aupdate(e,y,u,...);
end do;
end task;
    
```



03/20/06

© Alfons Crespo 2006

Introduction



03/20/06

© Alfons Crespo 2006

Characteristics

Functionality

- Continuous control
- Discrete event control
- Data display
- Data logging
- Operator commands
- Communications

Implementation requirements

- Concurrency
- Timeliness/ dependability
- Reliability
- Special hardware platforms
- Limited resources
- Efficiency

There is a growing need for a larger size and complexity of embedded control systems

Simplification factors

- Do not use display (specialized displays)
- No disk (no file systems)
- Monouser
- Limited security constraints
- Limited number of tasks
- User access limited
- Closed system

Concurrency

Activities in the real world are simultaneous

- physical variables change at the same time
- events occur asynchronously, and even at the same time

Control systems have to cope with this simultaneity

- e.g. multivariable control, asynchronous events

... but computers are sequential machines

- different activities must run on the same processor
- simultaneous execution is simulated by multiplexing the usage of the processor among different execution sequences

Concurrency: multiplexed execution of several activities on a computer

- concurrent activities are called **processes, threads, or tasks**

Real-Time

Concurrent activities with temporal constraints:

- The actions taken by the computer system have to be produced within a specified interval
- The algorithm (task) result has to be **logically** and **temporally** correct.

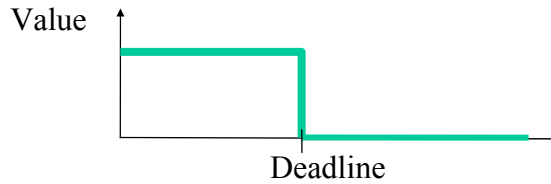
Temporal constraints:

- Sampling of analog variables (sensing) must be periodic
- Control actions must be issued in time (actuation)
- Reactions to events have to be executed within some deadline

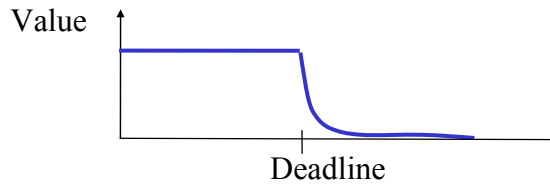
Temporal Constraints

Based on the deadline, task can be:

Hard deadline (Hard Real-Time Systems)



Soft deadline (Soft Real-Time Systems)



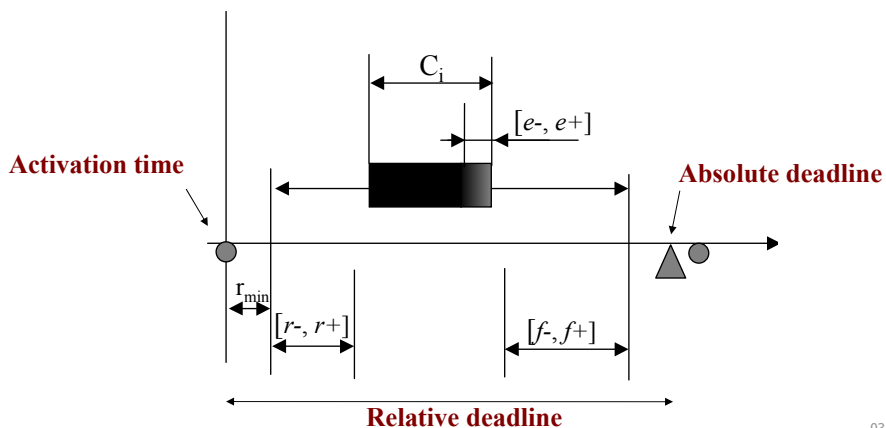
13

03/20/06

© Alfons Crespo 2006

Temporal constraints

A **real-time task** is required to execute within a given time interval usually characterized by an **activation pattern** and a **relative deadline**



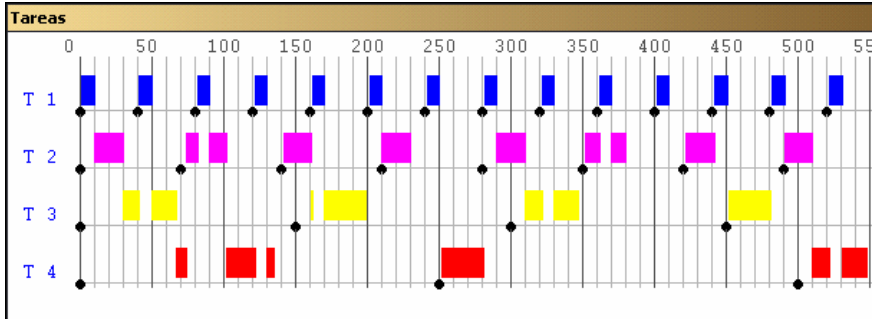
14

03/20/06

© Alfons Crespo 2006

Temporal constraints

But ... task are executed in a **processor multiplexing** following a **scheduling policy**



So, a task (depending on the scheduling policy) can have several delays.

03/20/06

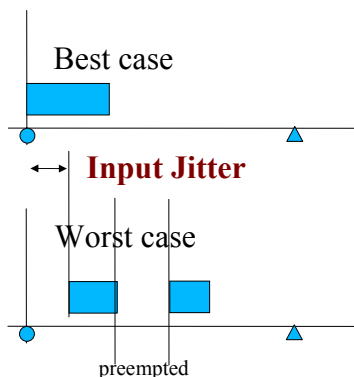
15

© Alfons Crespo 2006

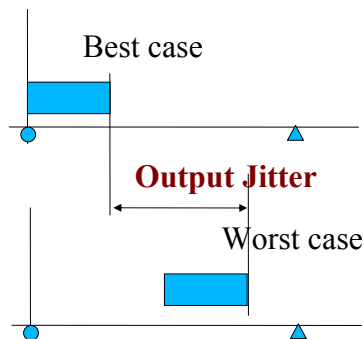
Temporal constraints

A task can suffer delays

At the beginning



At the end



The effect of **jitter** is difficult to analyse, both from the computing and control. Deadline and activation requirements can be used to limit jitter.

03/20/06

16

© Alfons Crespo 2006

Task Model

Based on the activation patterns for control tasks

- **Periodic**: the task is activated at regular intervals with period T

$$T_i = (C_i, D_i, P_i, \phi_i)$$

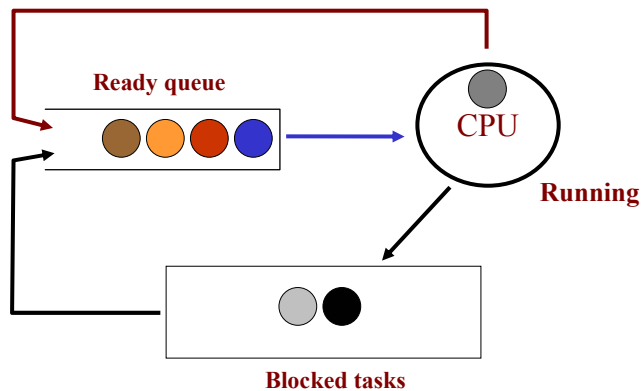
Activation: $a_k = \phi + k \cdot T$

- **Aperiodic**: the task is activated when some event occurs. Event arrival can be modelled in different ways (e.g. Poisson distribution)
- **Sporadic**: aperiodic, with a minimum inter-arrival time T between activation events

$$a_k \geq a_{k-1} + T$$

Scheduling

The scheduling algorithm determines which is the next task to be executed.



Scheduling

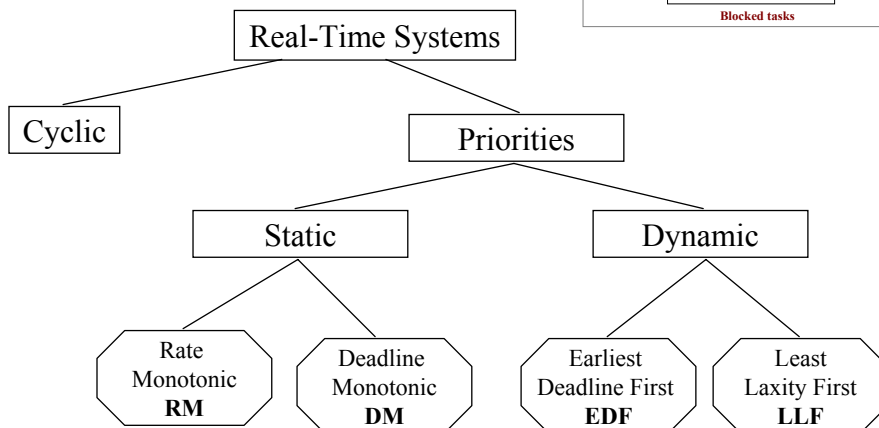
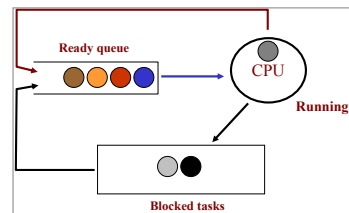
❖ It has a crucial role in ensuring temporal requirements enforcing of activation patterns

- releasing periodic tasks for execution at the proper times
- releasing aperiodic tasks when the activation event is detected
- ensuring minimum inter-arrival time for sporadic tasks

❖ Implementing appropriate processor sharing algorithms in order to guarantee deadlines

- the aim is not to maximize throughput or to improve average performance, but to guarantee deadlines
- hard deadlines have to be guaranteed even in worst-case load conditions, i.e. $R_i \leq D_i$

Schedulers



Cyclic Scheduling

Each task is characterised by a tupla (C_i, T_i, D_i)

- C_i is the worst case execution time
- T_i is the period
- D_i is the deadline

If all **tasks are periodic**, it is possible to design a **fixed execution plan** that is repeated each main cycle T_m

- Main cycle corresponds to the hyperperiod $H = \text{mcm}(T_i), i=1..n$
- Main cycle can be split in **secondary cycles** $T_S \Rightarrow T_M = kT_S$

Cyclic Scheduling

```
procedure Level_Control is
begin
  Level := Get_Level;
  Compute(R_Level, Level, ValOut);
  Put_Valve(ValOut);
end Level_Control;

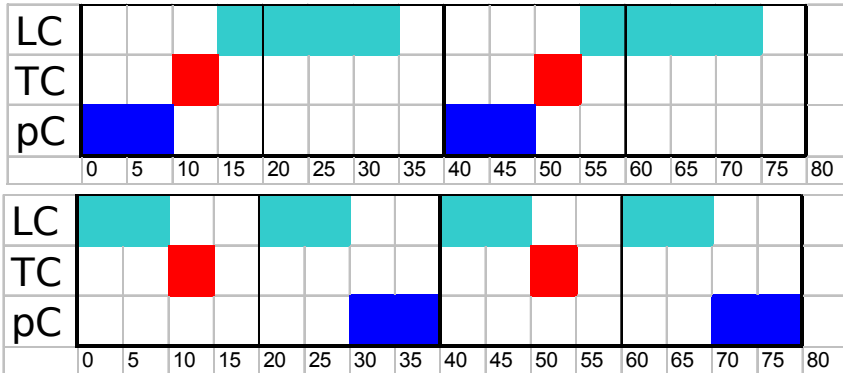
procedure pH_Control is
begin
  pH := Get_pH;
  Compute (R_pH, pH, Val_pH);
  Put_Valve(Val_pH);
end pH_Control;

procedure Temp_Control is
begin
  Temp := Get_Temp;
  Compute(R_Temp, Temp, Val_Temp);
  Put_Valve(Val_Temp);
end Temp_Control;
```

Cyclic Scheduling

Level_Control: C1 = 10 ms; P1 = 20ms
 Temp_Control: C2 = 5 ms; P2 = 40ms
 pH_Control: C3 = 10 ms; P3 = 40ms

HyperPeriod = 40ms
 SecondaryCycle = 20ms;



03/20/06

© Alfons Crespo 2006

Cyclic Scheduling

Level_Control: C1 = 10 ms; P1 = 20ms
 Temp_Control: C2 = 5 ms; P2 = 40ms
 pH_Control: C3 = 15 ms; P3 = 40ms

HyperPeriod = 40ms
 SecondaryCycle = 20ms;

cycle := 0;

Next_Activation := Clock; -- get the current time

loop

delay until Next_Activation;

Next_Activation := Next_Activation + SecondaryCycle ;

case (cycle mod 2) is

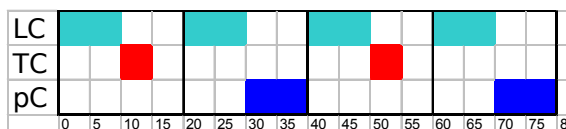
when 0 => Level_Control; Temp_Control;

when 1 => Level_Control; pH_Control;

end case;

cycle := cycle + 1;

end loop;



03/20/06

© Alfons Crespo 2006

Designing a cyclic plan

- Constraints on Secondary Cycle T_s
 - 1.- T_s any task has to be included
 - $\forall i : T_s \geq C_i$
 - 2.- Should be a submultiple of the Main Cycle (T_p):
 - $T_p = k \cdot T_s$
 - 3.- A whole T_s has to be included between an activation and its deadline for any task.
 - $\forall i : 2 T_s - \text{mcd}(T_s, T_i) \leq D_i$

03/20/06

© Alfons Crespo 2006

Cyclic Scheduling

• Example

Tarea	C	T	D
t_1	10	40	40
t_2	18	50	50
t_3	10	200	200
t_4	20	200	200

T_s Selection

- 1.- $\forall i : T_s \geq C_i \rightarrow T_s \geq 20$
- 2.- $\exists i : \lfloor T_i / T_s \rfloor = T_i / T_s \rightarrow T_s \in \{20, 25, 40, 50, 100, 200\}$
- 3.- $\forall i : 2 T_s - \text{mcd}(T_s, T_i) \leq D_i$

Sups $T_s = 20$:

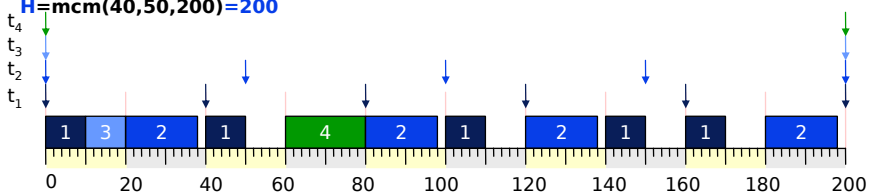
t_1 : $2 \cdot 20 - \text{mcd}(20, 10) = 30 \leq 40$ OK

t_2 : $2 \cdot 20 - \text{mcd}(20, 50) = 30 \leq 50$ OK

t_3 y t_4 : $2 \cdot 20 - \text{mcd}(20, 200) = 20 \leq 200$ OK

$$U = 10/40 + 18/50 + 10/200 + 20/200 = 0.76 < 1 \text{ OK}$$

$$H = \text{mcm}(40, 50, 200) = 200$$



03/20/06

© Alfons Crespo 2006

Cyclic Scheduling: Summary

- Robust method, **appropriate for simple systems**
 - temporal behaviour guaranteed by construction
 - easy to implement and analyse for correctness
- **Too rigid for complex systems**
 - static schedule difficult to build (NP-hard in the general case)
 - changes in code require rebuilding the schedule
 - difficult to accommodate sporadic tasks

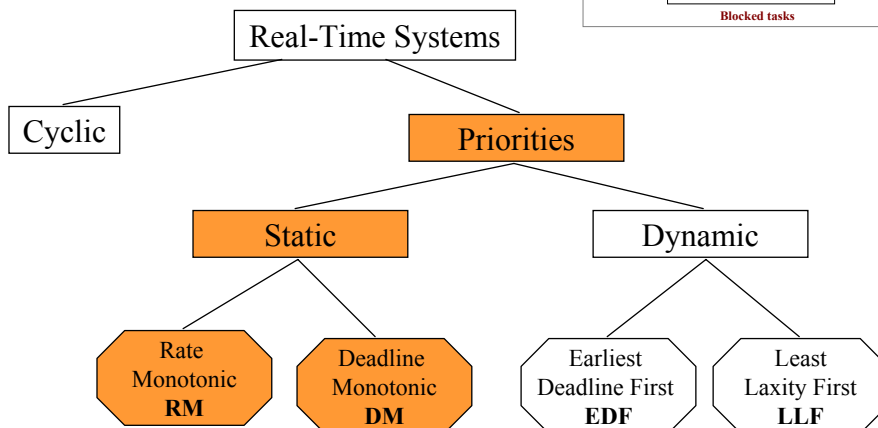
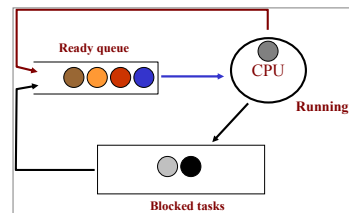
Summary: **a low-level method**

- more flexible scheduling schemes are easier to implement and maintain

03/20/06

© Alfons Crespo 2006

Schedulers



03/20/06

Priority Scheduling

Priority based is a **scheduling method for real-time** activities using threads or task.

Each **task has a priority** related to some assignement criteria (importance, urgency,)

- Priorities can be assigned off-line (**static**) or during the execution based on some parameter (**dynamic**)
- The scheduling policy is implemented in the kernel and selects the **highest priority task** (thread) among the ready tasks.

Schedulers can permit the **preemption or not preemption** of running tasks.

Fixed Priority Scheduling

Fixed Priority preemptive scheduling: **Priorities are assigned at design time**

Criteria:

- Designer decides the priority based on the importance of the task (semantic criteria)

- Based on the urgency (deadline):

❖ **Rate Monotonic** (periods = deadline). Higher priority to more frequent tasks

❖ **Deadline Monotonic** : Higher priority to more urgent task (Shorter deadline)

OPTIMAL

Rate Monotonic Analysis

Assumptions:

- Periods = Deadlines
- Tasks are independent (no shared resources)

Analysis: Utilisation based feasibility test (Liu & Layland, 73):

A task $T_i = (C_i, P_i)$, uses the CPU $U(1) = \frac{C_i}{P_i}$

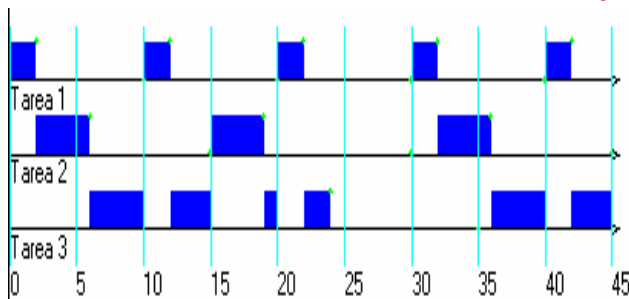
For n tasks: $\frac{C_1}{P_1} + \frac{C_2}{P_2} + \dots + \frac{C_n}{P_n} \leq U(n) = n \cdot (2^{\frac{1}{n}} - 1)$

Utilisation bound for n tasks

**It is sufficient but
not necessary condition**

n	U(n)
1	1.00
2	0.82
3	0.77
4	0.75
5	0.74
...	...
∞	0,69

Rate Monotonic Analysis



Example 1

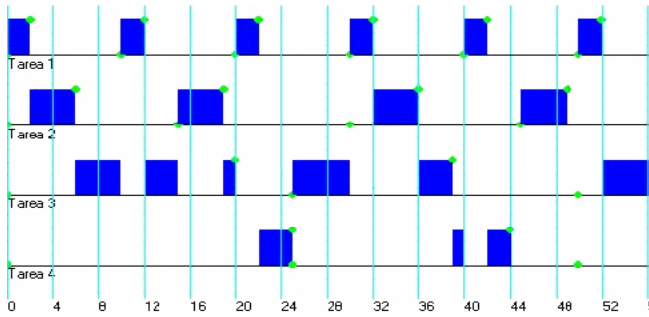
$C1 = 2$ $P1 = 10$ $U1 = 0.2$
 $C2 = 4$ $P2 = 15$ $U2 = 0.267$
 $C3 = 10$ $P3 = 35$ $U3 = 0.286$
 $U1 + U2 + U3 = 0.753$

75.3 % < U(3) = 77.9 % OK

24.7 % of the CPU can be used for other non real-time activities

Example 2

Rate Monotonic Analysis



C1 = 2	P1 = 10	U1 = 0.2
C2 = 4	P2 = 15	U2 = 0.267
C3 = 5	P3 = 25	U3 = 0.2
C4 = 6	P4 = 35	U4 = 0.17
U1 + U2 + U3 + U4 = 0.837		
83,7 % > U(4) = 75.6 % No		

But the system is schedulable

03/20/06

© Alfons Crespo 2006

Response time analysis

A set of n periodic tasks is schedulable under any priority assignment

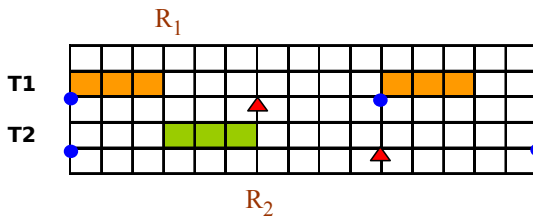
iff: All tasks finish its execution in the first period before the deadline

(Critical instant) $R_i \leq D_i$

The worst case response time (R_i) of a task T_i occurs when all tasks with higher priority start at the same time than T_i .

$$R_i = C_i + I_i$$

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{W_j}{T_j} \right\rceil C_j$$



$$I_i^j = \left\lceil \frac{W_j}{T_j} \right\rceil C_j$$

$$I_2^1 = \left\lceil \frac{6}{10} \right\rceil \cdot 3 = 3$$

03/20/06

© Alfons Crespo 2006

Response time analysis

Critical Instant

The value of interference (I_i) depends on the relative phases of the task activation times

I_i is maximum when a task is activated at the same time as all the tasks with a higher priority

It suffices to compute interference for the first period after a critical instant – no need for a full hyperperiod

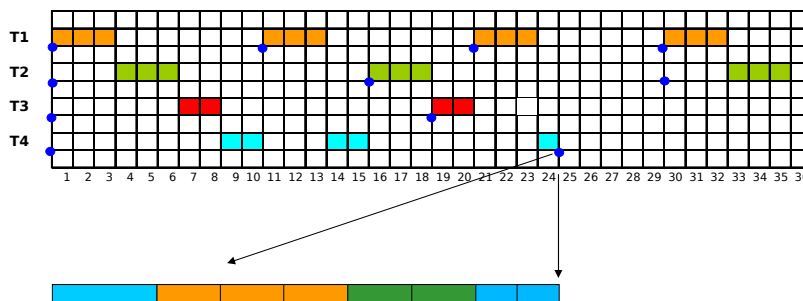
This reduces complexity of analysis to polynomial case

03/20/06

© Alfons Crespo 2006

Response time analysis

Interference calculation



03/20/06

© Alfons Crespo 2006

Response time analysis

R_i can be determined by:
$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{T_j} \right\rceil * C_j$$

It can be solved by a linear iteration:
$$R_i^0 = C_i$$

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j^n}{T_j} \right\rceil C_j$$

Termination condition $R_i^{n+1} = R_i^n$

The system is schedulable if for all task $R_i^{n+1} \leq D_i$

03/20/06

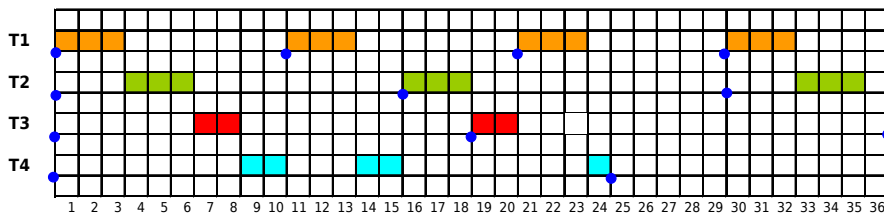
© Alfons Crespo 2006

Response time analysis

	T1	D _i	C _i
T1	10	10	3
T2	15	15	3
T3	18	18	2
T4	24	24	5

$$R_1^0 = C_1$$

$$R_1^1 = C_1 + 0 = 3 \leq D_1 = 10$$



03/20/06

© Alfons Crespo 2006

Response time analysis

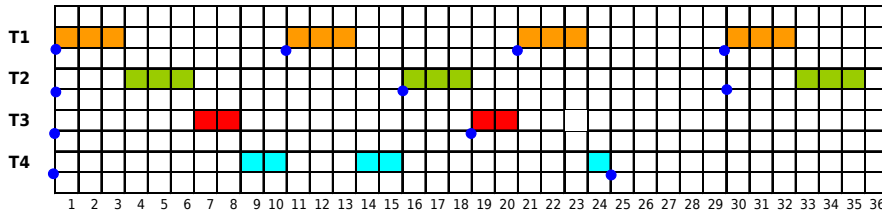
	T1	Di	Ci
T1	10	10	3
T2	15	15	3
T3	18	18	2
T4	24	24	5

$$R_2^0 = C_2 = 3$$

$$R_2^1 = C_2 + \sum_{\forall j \in \{1\}} \left\lceil \frac{R_2^0}{T_j} \right\rceil = 3 + \left\lceil \frac{3}{10} \right\rceil 3 = 6$$

$$R_2^2 = C_2 + \sum_{\forall j \in \{1\}} \left\lceil \frac{R_2^1}{T_j} \right\rceil = 3 + \left\lceil \frac{6}{10} \right\rceil 3 = 6$$

$$R_2^1 = R_2^2 = 6 \leq D_2 = 15$$



03/20/06

© Alfons Crespo 2006

Response time analysis

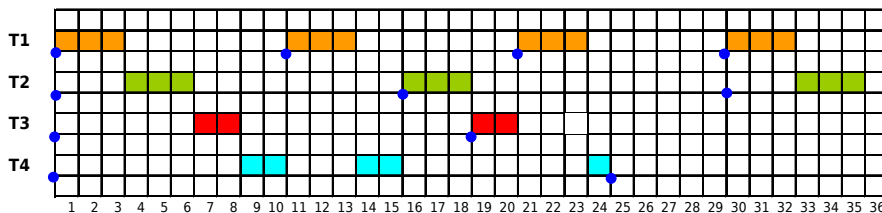
	T1	Di	Ci
T1	10	10	3
T2	15	15	3
T3	18	18	2
T4	24	24	5

$$R_3^0 = C_3 = 2$$

$$R_3^1 = C_3 + \sum_{\forall j \in \{1,2\}} \left\lceil \frac{R_3^0}{T_j} \right\rceil = 2 + \left\lceil \frac{2}{10} \right\rceil 3 + \left\lceil \frac{2}{15} \right\rceil 3 = 8$$

$$R_3^2 = C_3 + \sum_{\forall j \in \{1,2\}} \left\lceil \frac{R_3^1}{T_j} \right\rceil = 2 + \left\lceil \frac{8}{10} \right\rceil 3 + \left\lceil \frac{8}{15} \right\rceil 3 = 8$$

$$R_3^1 = R_3^2 = 8 \leq D_3 = 18$$



03/20/06

© Alfons Crespo 2006

Response time analysis

	T1	Di	Ci
T1	10	10	3
T2	15	15	3
T3	18	18	2
T4	24	24	5

$$R_i^0 = C_i = 5$$

$$R_i^1 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^0}{T_j} \right\rceil = 5 + \left\lceil \frac{5}{10} \right\rceil 3 + \left\lceil \frac{5}{15} \right\rceil 3 + \left\lceil \frac{5}{18} \right\rceil 2 = 13$$

$$R_i^2 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^1}{T_j} \right\rceil = 5 + \left\lceil \frac{13}{10} \right\rceil 3 + \left\lceil \frac{13}{15} \right\rceil 3 + \left\lceil \frac{13}{18} \right\rceil 2 = 16$$

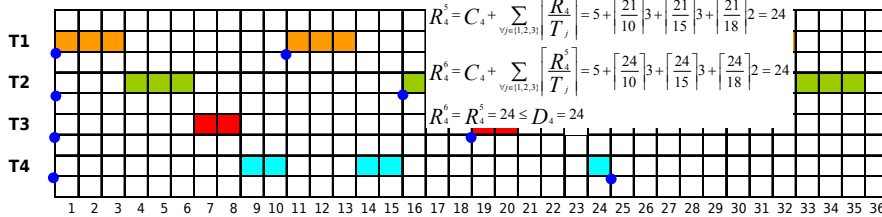
$$R_i^3 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^2}{T_j} \right\rceil = 5 + \left\lceil \frac{16}{10} \right\rceil 3 + \left\lceil \frac{16}{15} \right\rceil 3 + \left\lceil \frac{16}{18} \right\rceil 2 = 19$$

$$R_i^4 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^3}{T_j} \right\rceil = 5 + \left\lceil \frac{19}{10} \right\rceil 3 + \left\lceil \frac{19}{15} \right\rceil 3 + \left\lceil \frac{19}{18} \right\rceil 2 = 21$$

$$R_i^5 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^4}{T_j} \right\rceil = 5 + \left\lceil \frac{21}{10} \right\rceil 3 + \left\lceil \frac{21}{15} \right\rceil 3 + \left\lceil \frac{21}{18} \right\rceil 2 = 24$$

$$R_i^6 = C_i + \sum_{j \in \{1,2,3\}} \left\lceil \frac{R_i^5}{T_j} \right\rceil = 5 + \left\lceil \frac{24}{10} \right\rceil 3 + \left\lceil \frac{24}{15} \right\rceil 3 + \left\lceil \frac{24}{18} \right\rceil 2 = 24$$

$$R_i^6 = R_i^5 = 24 \leq D_i = 24$$



03/20/06

© Alfons Crespo 2006

Extending Response Time Analysis

RTA can be extended to more complex task models

- tasks with any priority assignment
- deadlines shorter than periods
- sporadic tasks
- communication with shared variables
- activation jitter

... and also to distributed systems

- provided the communication link has a bounded transmission time

The computation models covered by RTA can be implemented on a number of operating systems and programming languages

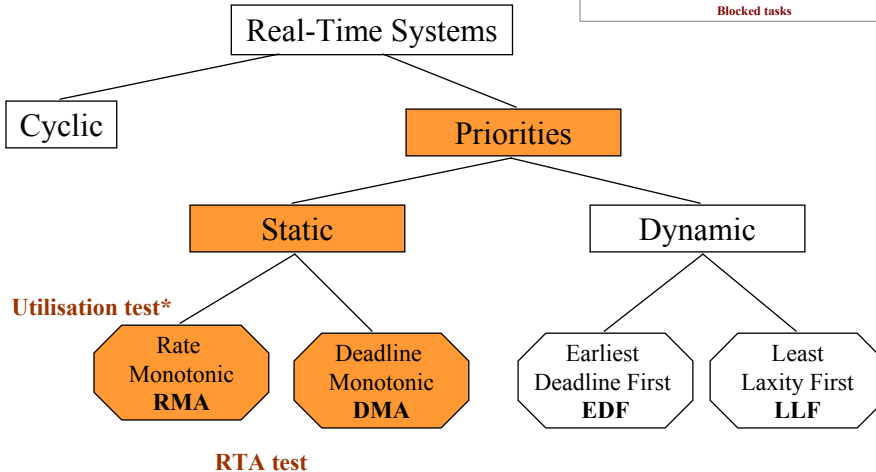
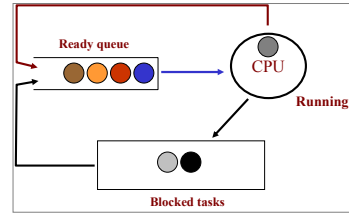
This enables many kinds of real-time systems to be built and analysed for deadline guarantees

- mandatory in high-integrity systems

03/20/06

© Alfons Crespo 2006

Schedulers



Sporadic tasks

Sporadic events are characterized by a minimum inter-arrival time T

The worst case is when the task is activated as often as possible

- pseudo-periodic activation with period T

Sporadic tasks are converted (from the analysis point of view) in periodic task with period equal T and, usually, deadlines shorter than T

the pseudo-periodicity assumption: sporadic tasks behave as periodic tasks in the worst case

Aperiodic tasks

Aperiodic events are characterized by different kinds of models, most of them stochastic

- e.g. Poisson process

Aperiodic tasks do not usually have any hard deadlines but are required to respond as fast as possible

Aperiodic tasks can be scheduled in a DMS framework using a variety of aperiodic servers

An aperiodic server is a periodic task serving aperiodic events with a limit of computation by period (budget)

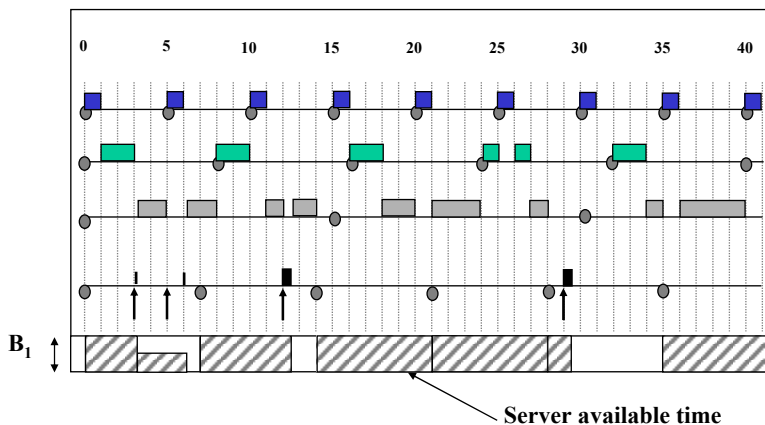
$$S_i = (B_i, P_i)$$

- In general, the budget is replenished each period
- As a periodic task it can be integrated into RTA

03/20/06

© Alfons Crespo 2006

Aperiodic tasks



03/20/06

© Alfons Crespo 2006

Jitter evaluation

- Output jitter can play an important role in the control performance
- Tasks are pseudo-periodic
- The distance between two consecutive input or output is variable
- Important aspect: determine the Input/Output Jitter.

03/20/06

© Alfons Crespo 2006

Jitter evaluation

Worst case response time \Rightarrow Maximum finishing time of the task:

$$WC_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{WC_i^n}{T_j} \right\rceil C_j$$

Best case response time \Rightarrow Minimum initial time of the task:

$$BC_i^{n+1} = C_i^{\min} + \sum_{j \in hp(i)} \left\lfloor \frac{BC_i^n - T_j}{T_j} \right\rfloor C_j^{\min}$$

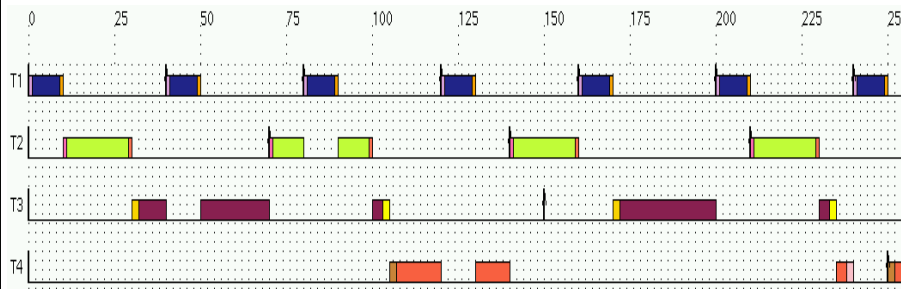
Control Action Interval

$$CAI_i = \frac{WC_i - BC_i}{T_i}$$

03/20/06

© Alfons Crespo 2006

Jitter evaluation



Task	WCET	Period	Deadline	Offset
T1	10	40	40	0
T2	20	70	70	0
T3	35	150	150	0
T4	30	250	250	0

WCET	Begin		End		CAI
	Min	Max	Min	Max	
T1	0	0	10	10	0%
T2	0	10	20	30	14%
T3	0	20	45	105	40%
T4	0	105	30	240	84%

03/20/06

© Alfons Crespo 2006

Jitter reduction

A simple way to reduce the Jitter in a fixed priority scheme is to increase the priority of the task.

- ❖ Reducing the deadline of a task (DM assignment)
- ❖ Increasing the priority

Reduction of the jitter in a task produce an increase in others. But not all tasks are equal sensible to higher jitter.

Other alternatives are also possible:

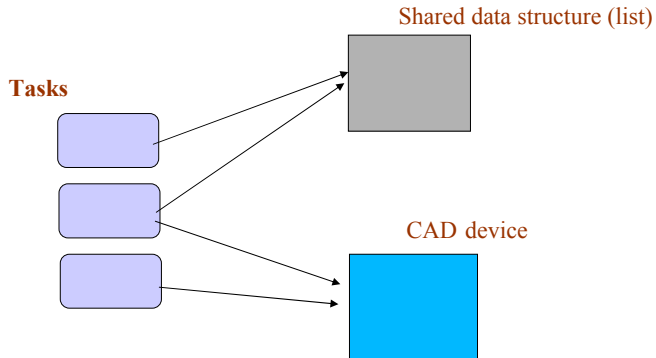
- ❖ Task partitioning
- ❖ Bands of priorities for Input/Output/computation phases
- ❖

03/20/06

© Alfons Crespo 2006

Using shared resources

Tasks usually shared variables or devices



03/20/06

© Alfons Crespo 2006

Using shared resources

❖ Code segments in which shared data are accessed are called **critical sections**

❖ Shared data must be **protected** so that critical sections are **executed in mutual exclusion**

operating systems provides many mechanisms: semaphores, mutexes, Ada protected objects

❖ In real-time systems, mutual exclusion may give rise to priority inversion this effect is also called blocking

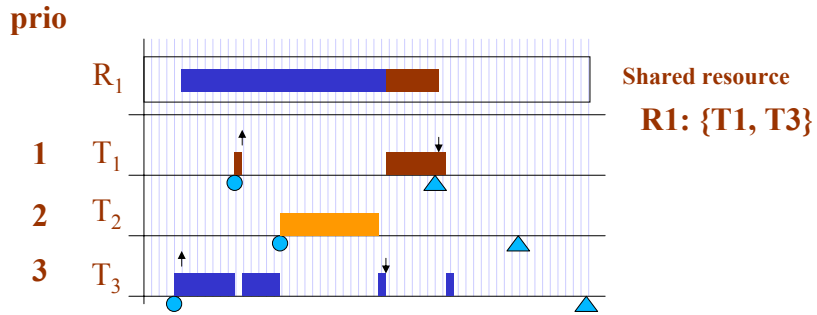
- an undesirable side effect of mutual exclusion
- it can even affect tasks not sharing any variables
- it can produce unbounded blocking

This may result in unbounded response times

03/20/06

© Alfons Crespo 2006

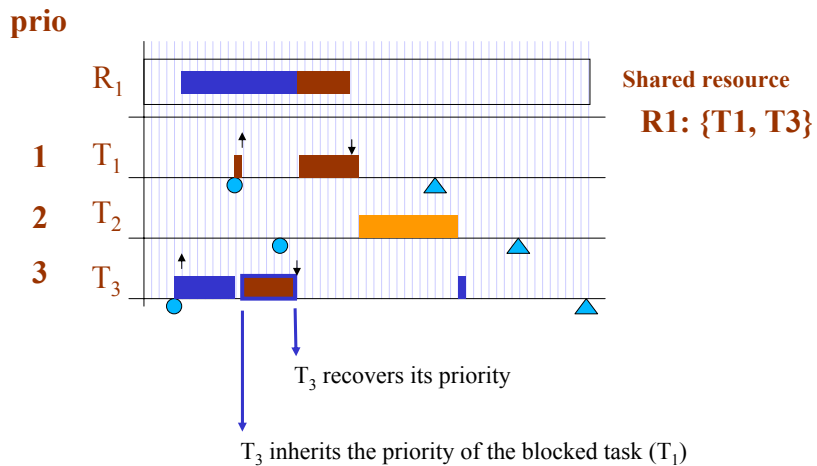
Using shared resources: Priority inversion



03/20/06

© Alfons Crespo 2006

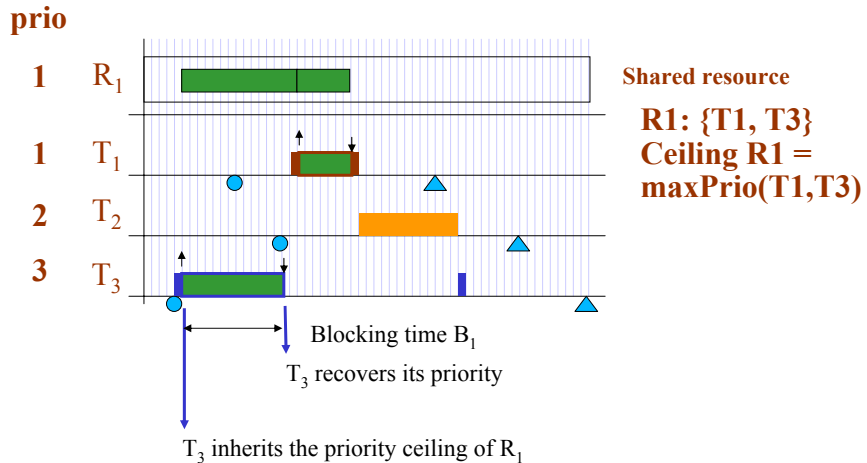
Using shared resources: Priority inheritance protocol



03/20/06

© Alfons Crespo 2006

Using shared resources: Immediate ceiling protocol



03/20/06

© Alfons Crespo 2006

Immediate ceiling protocol

- ❖ A high-priority task **blocks at most once** in each execution cycle
 - ❖ It is independent of how many shared data items it uses
- ❖ The **maximum duration of blocking** for a task T_i equals the duration of the longest critical section executed by a lower-priority task using a data item with a ceiling priority greater or equal to the priority of T_i
- ❖ There are **no deadlocks**, in spite of possible circular wait situations
- ❖ The effect of (bounded) blocking can be added to the response time equation:

$$R_i = C_i + \boxed{B_i} + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{T_j} \right\rceil * C_j$$

03/20/06

© Alfons Crespo 2006

Real-time Operating Systems

- **Provides infrastructure for running software on an embedded hardware platform**
 - Provides a hardware abstraction
 - Provides access to the devices
 - Provides support for multiple processes and threads
 - Provides development tools
 - Compiler / Linker
 - Downloader
 - Debugger
- **Provides deterministic performance**
 - Guaranteed interrupt management Latency
 - Guaranteed Context Switch
 - Small and bounded scheduling overhead
 - Timer and clock access

RTOS requirements

- **Real-time task scheduling**
- **Preemptive scheduling**
- **Interrupt response guarantee**
- **Static or dynamic priorities**
- **Synchronous and Asynchronous I/O**
- **Fast data acquisition**
- **Deterministic network communications**
- **Portability**
- **Efficient memory management**
- **Real-Time languages support**
- **Standard API (POSIX®)**

Commercial RTOS

Interrupt handling

Scheduling

- FIFO or Round-robin for equal-priority threads (EDF scheduling is not provided)
- Priority inversion control (PIP)

Memory management

- Virtual memory (not pagination)
- Memory protection (some of them)

- LynxOS
- pSOS
- QNX
- VRTX
- VxWorks
-

Most of them provide real-time extensions

Open Source RTOS

Real-Time Linux:

RT-Linux

KURT

RED Linux

RTAI

Linux/RK (from Mach/RK)

RTEMS

ORK

MARTE OS (Ada)

Embedded Linux

MiniRTL

Extension to RTLinux (OCERA)

ELKS project

HA-Linux (High Availability)

Lineo Embeddic

VME Linux Project

Open Source RTOS

	Linux 2.4	RTLinux/GPL	RTAI	OCERA
Processors	I386, PPC, ARM, SH, m68k, PARI SK, Sparc, MIPS	I386, PPC*, ARM*	I386, PPC, ARM, m68k, MIPS	I386, PPC*, ARM*
Multi-processors	Yes	Yes	Yes	Yes
Process	Yes	No	No	No
Threads	Yes	Yes	Yes	Yes
Scheduling policies	FIFO, RR	FIFO, EDF, SPORADIC	FIFO	FIFO, EDF, SPORADIC, CBS, IRI S, ADS
Priority inversion	None	Ceiling	Inheritance	Ceiling
Priority range	0-100	0-100000	0x3ffff-0	0-100000
Protected memory	Yes	No	No	Yes
Dynamic memory	Yes	No	Yes	Yes
Semaphores	Yes	Yes	Yes	Yes
Mutex	Yes	Yes	Yes	Yes
Message queues	No	No	Yes	Yes
Barriers	No	No	No	Yes
rd/wr locks	No	No	No	No
Signals	Yes	No	No	Yes
Timers	No	No	No	Yes
Execution Timers	No	No	No	Yes
Time resolution	Configurable (HRT)	Configurable	Configurable	Configurable
User timers	Yes	No	No	Yes
Network	IP, UDP, TCP, ...	No	IP, UDP	IP, UDP, TCP, ...
Filesystems	Ext2/3, ReiserFS, DOS, RAM, Flash, XFS, QNX4, ...	No	No	Yes
API's	POSIX, pSOS, VxWorks	POSIX 1003.1c, PSE	Custom, POSIX 1003.1c (compat)	POSIX 1003.1c, PSE

Real-Time Programming Languages

- Sequential languages as **C/C++ languages** do not support directly real-time facilities
 - Facilities are **supported by OS API** (POSIX or other API)
- **Ada** was specifically designed for **embedded real-time systems**
 - **Provides support for real-time and concurrency and protected objects**
- **Java** provides support for threads and protected shared data. But it does not provide support for real-time
 - RTSJ: A specification for Real-Time Java

Ada

- ❖ Ada supports many real-time concepts at the programming language level
 - – concurrency
 - – protected shared data
 - – fixed priority scheduling & ICPP
 - – device & interrupt drivers
- ❖ Ada 95 is the current standard, Ada 2005 to come soon
 - – Ravenscar profile for high-integrity systems
 - – additional scheduling methods
 - – Java-like interfaces
 - – execution-time clocks
 - – etc.

03/20/06

© Alfons Crespo 2006

RTJava

- ❖ Java extension for real-time systems
 - – developed by the Real-Time for Java Expert Group (version 1.0, 2001)
- ❖ Extensions of Java computation model in several areas
 - – scheduling
 - – memory management
 - – synchronization
 - – event handling
 - – physical memory access
- ❖ Implementations available
 - – TimeSys, AICAS (Jamaica)

...but not so mature as Ada

03/20/06

© Alfons Crespo 2006

Summary

- ❖ Real-time systems have temporal requirements
 - ❖ not just functionally correct, things must be done in time
- ❖ Scheduling is crucial in guaranteeing and analysing temporal behaviour
 - ❖ timing properties depend on the way processor & other resources are shared
- ❖ Analysable task model based on fixed-priority scheduling
 - – deadline or rate-monotonic priorities
 - – controlled access to shared data
 - – extensible to offsets, jitter, distributed systems, etc.
- ❖ Other scheduling methods
 - EDF (earliest-deadline first) » efficient but not so robust
 - Static, time-driven scheduling » robust, but complex to implement

Tuesday 4th of April



ARTIST2

Embedded Control Systems: Control Kernel

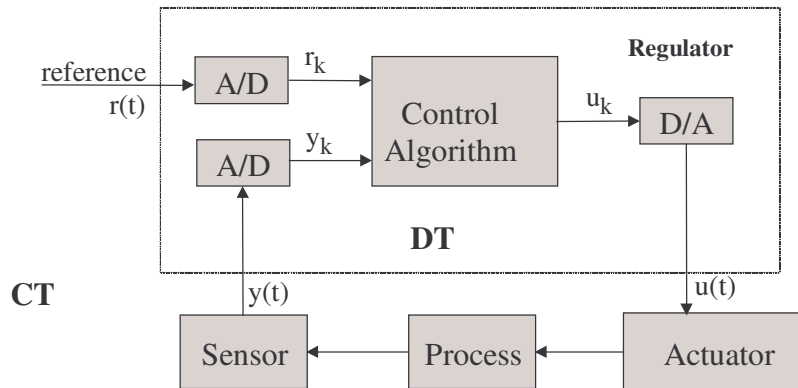
P. Albertos

Universidad Politécnica de Valencia
* Dept. of Systems Engineering and Control,
E-46071 Valencia, Spain. Fax: +34 96 3879579
e-mail: pedro@aii.upv.es

The kernel concept

- OS kernel:**
- Basic services:
 - Task and time management
 - Interrupt handling
 - Interface to the applications (API)
 - Mode changes
 - Fault tolerance
 - Additional services
 - File management
 - Quality of service
 - Tracing and debugging

Basic Control Loop



Control task

...

...

loop

```
convert _sensor_analog_digital (y);
```

```
compute _control_action (u);
```

```
compute _error (e)
```

```
compute _control_action (u) ←
```

```
send _converted_control_action (u);
```

```
update_internal_variables(y,u, ...);
```

```
Next_Iteration:= Next_Iteration + Period;
```

```
delay until Next_Iteration;
```

```
end loop;
```

...

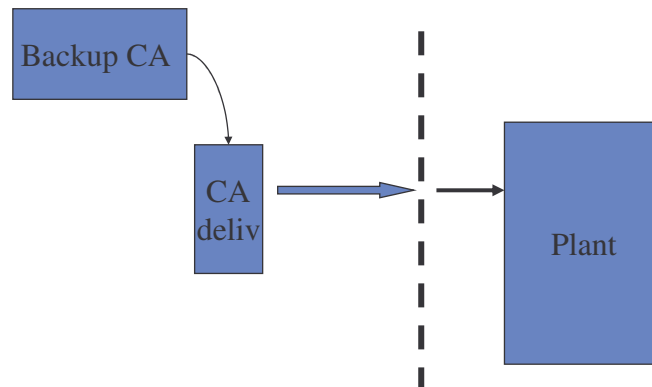
The Control Kernel concept

- Ensures control action (CA) delivering
- Data acquisition of major signals
- Transfer to new control structure

- Additional CA computing facilities
- Communication facilities
- Coordination facilities

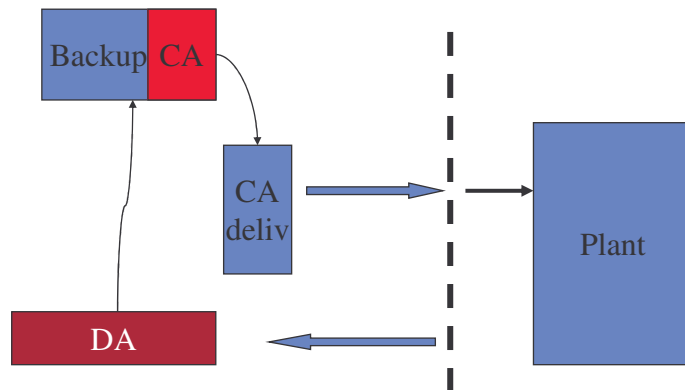
Control Kernel

- Ensures control action (CA) delivering
 - Safe (back-up) CA computation
 - Safe CA computation based on previous data



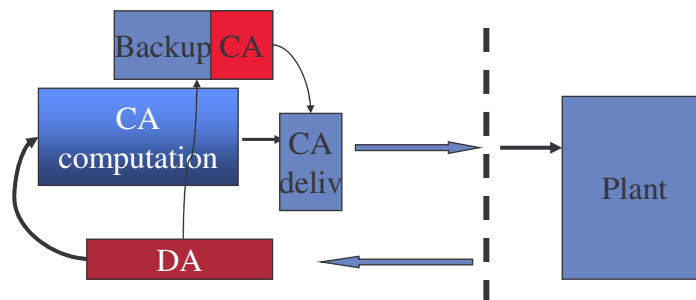
Control Kernel (2)

- Data acquisition of major signals
 - Safe CA computation based on current data



Control Kernel (3)

- Transfer to new control structure
 - Basic control structure parameters computation
 - CA computation
- Full DA
 - Control structures evaluation and selection
 - CA computation (different levels)



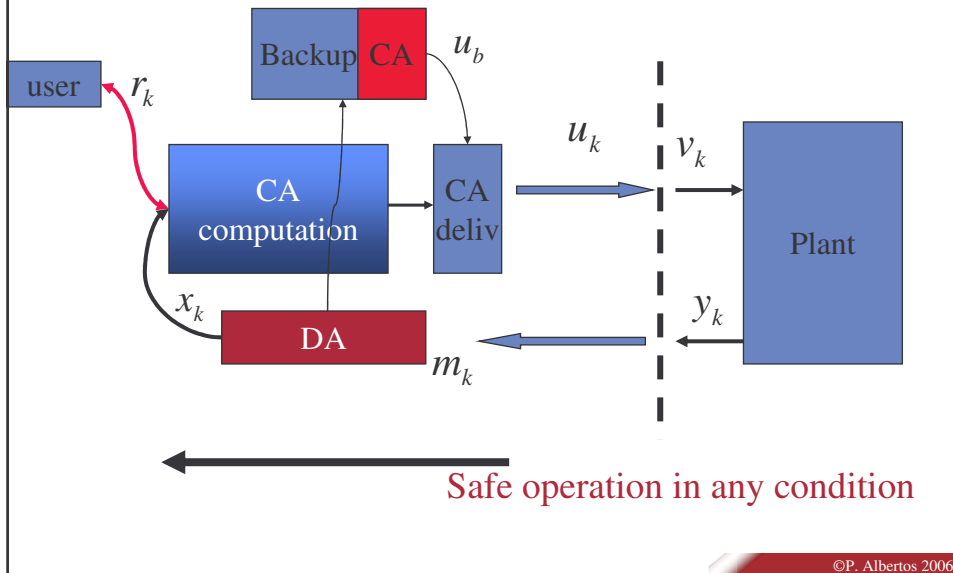
Control Kernel (4)

- Communication facilities
 - with the environment
 - the operator
 - other ECS
- Coordination facilities

Control Kernel

- Ensures control action (CA) delivering
 - Safe (back-up) CA computation
 - Safe CA computation based on previous data
- Data acquisition of major signals
 - Safe CA computation based on current data
- Transfer to new control structure
 - Basic control structure parameters computation
 - CA computation
- Full DA
 - Control structures evaluation and selection
 - CA computation (different levels)
- Communication facilities
- Coordination facilities

The control kernel concept

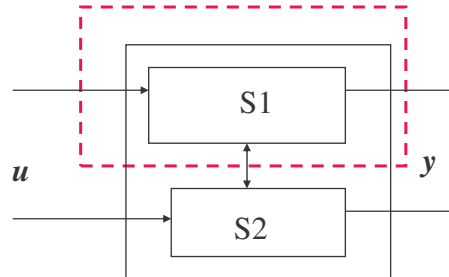


Control Kernel Algorithm

- CA delivering $v_k = u_k$
 - Backup CA $u_k = u_b$
 - Backup CA Computation $u_k = f(u_b, x_{k-1})$
-
- Current safe b-up CA comp. $u_k = f(u_b, x_k)$
 - Basic CA computation $u_k = f_1(r_k, x_k)$
 - CA comp $u_k = f_i(r_k, x_k)$
 - CA comp. (Process model) $u_k = F(r_k, x_k)$
 - Essential
 - Partial
 - Complete

Control Kernel Algorithm

- Model reduction:
 - Partial control (parts of the plant)



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_k; \quad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_k = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k$$

$$x_{1k+1} = A_{11}x_{1k} + B_{11}u_{1k}; \quad y_{1k} = C_{11}x_{1k}$$

Control Kernel Algorithm

- Model reduction: time scale x_1 : slow modes; x_2 : fast modes

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u_k; \quad y_k = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k$$

Deleting the “slow” mode: $x_{1,k}$: constant

$$x_{2,k+1} = A_{22}x_{2,k} + A_{21}x_{1,k} + B_2u_k$$

$$y_k = C_2x_{2,k} + C_1x_{1,k}$$

Control Kernel Algorithm

- Model reduction: deleting the “fast” mode $x_{2,k+1} = x_{2,k}$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u_k; \quad y_k = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k$$

$$x_{2,k} = (I - A_{22})^{-1} (A_{21} x_{1,k} + B_2 u_k) \quad \longrightarrow$$

$$\begin{aligned} x_{1,k+1} &= \bar{A}_1 x_{1,k} + \bar{B}_1 u_k & \bar{A}_1 &= A_{11} + A_{12} (I - A_{22})^{-1} A_{21} \\ y_k &= \bar{C}_1 x_{1,k} + \bar{D} u_k & \bar{B}_1 &= A_{12} (I - A_{22})^{-1} B_2 + B_1 \\ & & \bar{C}_1 &= C_1 + C_2 (I - A_{22})^{-1} A_{21} \\ & & \bar{D} &= C_2 (I - A_{22})^{-1} B_2 \end{aligned}$$

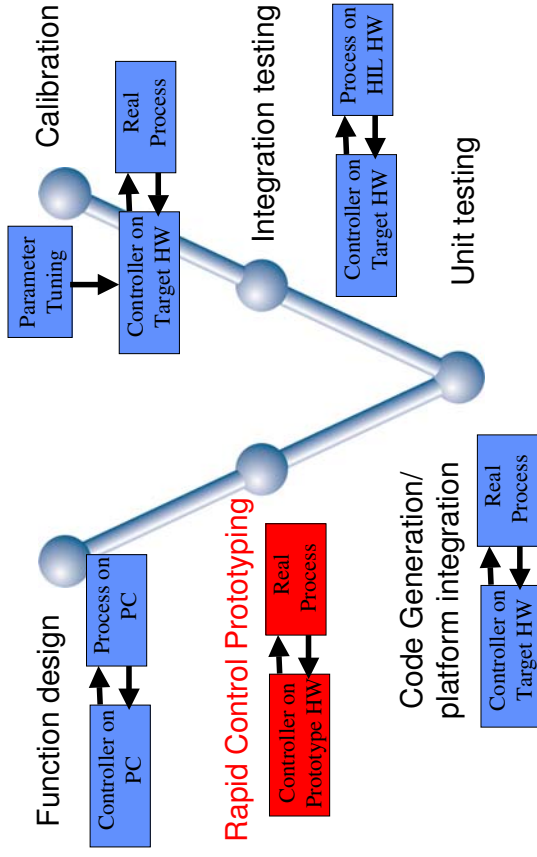
To recover the “fast” state:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k^+ = \begin{bmatrix} I \\ (I - A_{22})^{-1} A_{21} \end{bmatrix} x_{1,k} + \begin{bmatrix} 0 \\ (I - A_{22})^{-1} B_2 \end{bmatrix} u_k$$

Conclusions

- Kernel concept
- Code rewriting
- Interaction with OS kernel
- Fast, reliable and safe operation
- Include ... CPU, power control

Tools for model based control engineering



ARTIST2

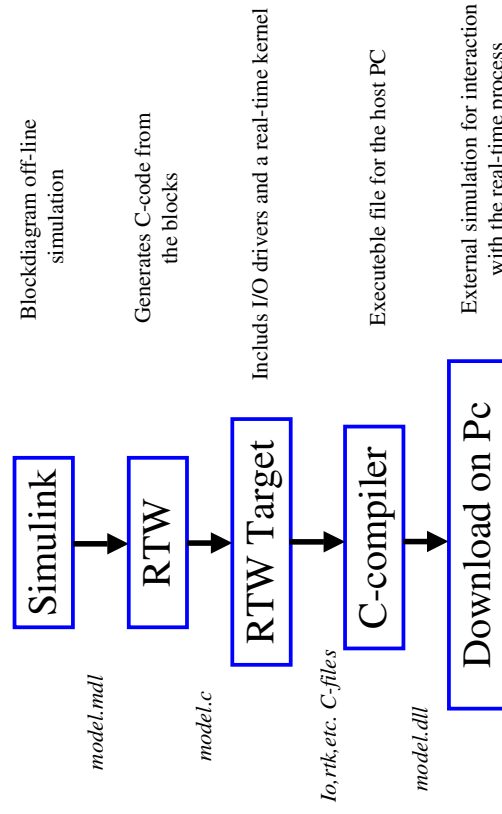
Control design practical issues

M. Törnren and B. Eriksson

Division of Mechatronics, Dept. of Machine Design
KTH - Royal Institute of Technology, Stockholm
www.md.kth.se e-mail: martin@md.kth.se



Rapid Control Prototyping

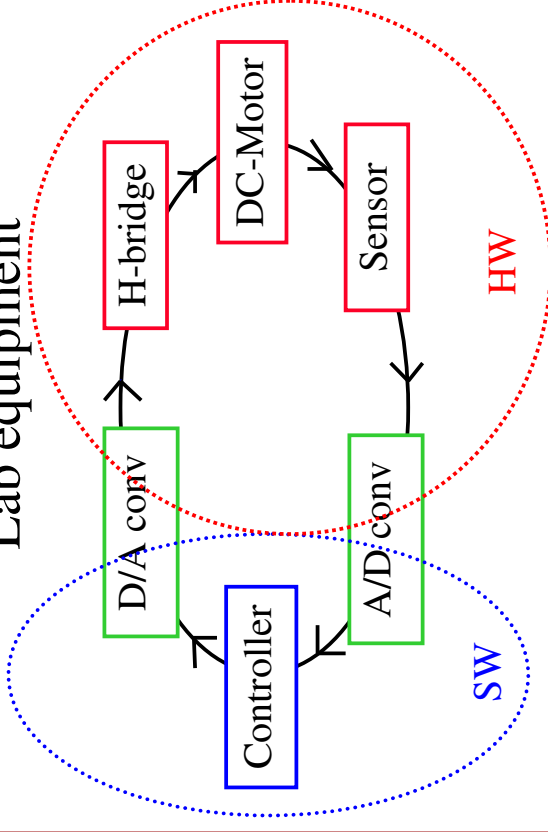


Controlling a Dc-motor

Main goals of the workshop

- Control structure: P, PD, PID, and filtered PID
- Design method: Tuning v.s. model based (poleplacement) model parameter identification
- Reference: Setpoint v.s. Trajectories as a function of time
- Sensor noise and nonlinear friction

Lab equipment



- Open the Simulink file Poscontrol and try to select the controlparameters P & D
C:\Matlab6p5\work\Labdesign
- Try this with different sampling times e.g. $T_s = 80..10$ ms
- Observe reference and measured positions and velocities, and the control signal
- What is the fastest i.e., the highest achievable gain (P)
- For model based design we need a model: Open the Simulink file ID
- Find the two model parameters, Time constant and Gain.
- Go back to Poscontrol and use the PD block to design the controlparameters
- A good guess of sample time is 20-30 times faster than the fastest closed loop pole
 $T_s = 2 * \pi / ([10..20] * \omega)$
- What is now the fastest possible closed loop, P?
- Change to sine input in the signal generator, amplitude 1, frequency 0.1 Hz.
- Observe what happens when the velocity changes sign.
- Open the file Poscontrol2 look through the subsystems start with the same PD design as from above. Use fast and slow trajectories, what are the main problems for fast and slow trajectories.
- Compare the results using the other controllers. What is the highest gain for a slow trajectory????

$$G(s) = \frac{K}{\tau s^2 + s}$$

ARTIST2

Embedded Control Systems: Integrated Control Design & Implementation

Karl-Erik Årzén and Anton Cervin

Department of Automatic Control

Lund University

Sweden

{karlerik,anton}@control.lth.se



1

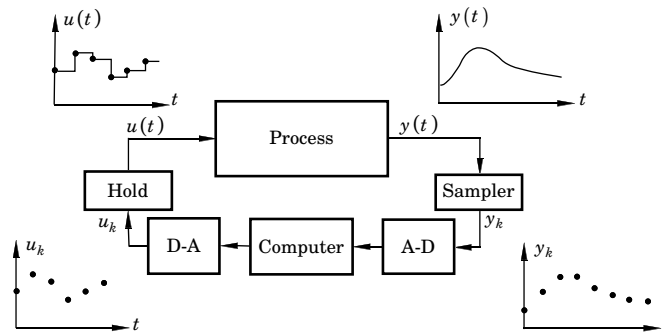
Session Outline

- **Control Loop Timing Parameters**
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- The Jitter Margin
- The Control Server Model
- Arithmetics

2

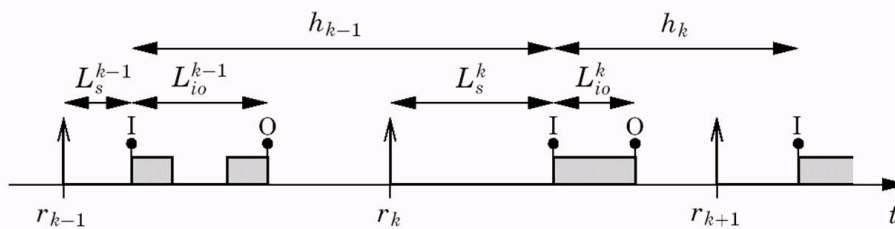
Assumptions

In this session we will focus on periodically sampled control loops.



3

Continuous Controllers: Timing Parameters



- Task released at $r_k = hk$
- Sampling latency L_s
- Sampling jitter $J_s \stackrel{\text{def}}{=} L_s^{\max} - L_s^{\min}$
- Sampling interval jitter $J_h \stackrel{\text{def}}{=} h^{\max} - h^{\min}$
- Input-output latency jitter $J_{io} \stackrel{\text{def}}{=} L_{io}^{\max} - L_{io}^{\min}$

4

Control Loop Timing

Classical control assumes deterministic sampling

- in most cases periodic
- sampling interval determined by desired closed loop performance and the nature of the disturbances acting on the system
- too long sampling interval or too much jitter cause poor performance or instability

Control Loop Timing

Classical control assumes negligible or constant input-output latency

- if the latency is small compared to the sampling interval it can be ignored
- if the latency is constant it can be included in the control design
- too long latency or too much jitter cause poor performance or instability

Embedded Control Loop Timing

Embedded control systems with limited computing resources implies temporal non-determinism

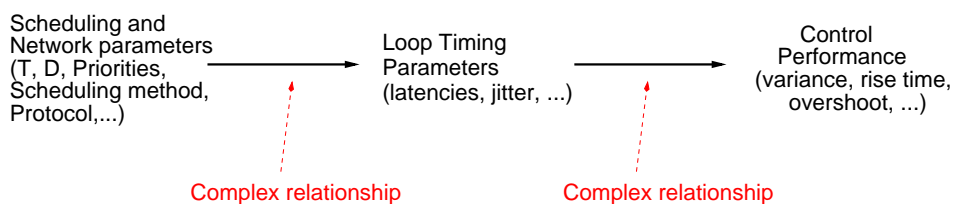
- multiple tasks compete for computing resources
- preemption by higher-priority tasks, blocking when accessing shared resources, varying computation times, non-deterministic kernel primitives

Networked control systems with limited communication resources implies temporal non-determinism

- network interface delay, queuing delay, transmission delay, propagation delay, link layer resending delay, transport layer ACK delay, ...
- lost packets

7

Timing Relationships



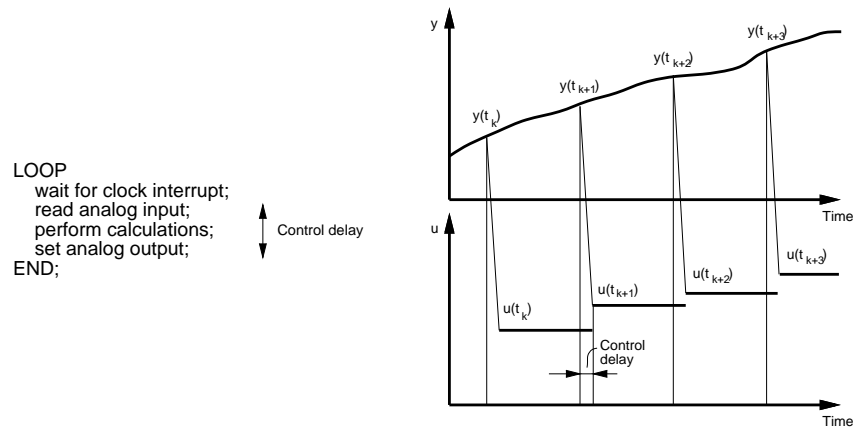
Possibilities:

- Simulation – the TrueTime tool
- “Numerical Analysis” – the Jitterbug tool
- Theoretical results – e.g., the Jitter Margin

8

Input-Output Latency

Always present in computer-based control systems.



9

Rules of Thumb

A short latency is better than a long latency

A short, but jittery, latency is better than a long constant latency

But, anomalies exist!

10

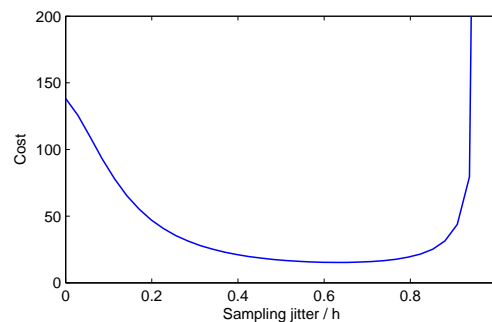
Example from Kushner and Tobias (1969)

Plant: $P(s) = \frac{6}{(s+1)(s+2)}$

Controller: $C(z) = 1$ (unit negative feedback)

Sampling period: $h = 1.42 + \text{uniform sampling jitter}$

Unit white input noise. Cost function: $J = \mathbb{E}y^2(t)$



11

Basic Questions

1. How robust is a control loop to temporal nondeterminism?
2. How do we implement the control loop in order to maximize the temporal determinism?
3. Can we use control techniques in order to improve the temporal robustness?

12

Implementing Periodic Controller Tasks

Three Main Issues:

1. How do we achieve periodic execution?
2. When is the sampling performed?
3. When is the control signal sent out?

1. How Do We Achieve Periodic Execution?

1. Using a static schedule (cyclic executive)?
 - High temporal determinism but inflexible
 - Does not require any sophisticated RTOS support
2. In interrupt handlers (interrupt service routines) associated with timers
3. As self-scheduling threads in a RTOS/kernel using time primitives such as sleep/delay/WaitTime (relative wait) or sleepUntil/delayUntil/WaitUntil (absolute wait)
4. Using an RTOS/kernel with built-in support for periodic tasks
 - implement the tasks as simple procedures/methods that are registered with the kernel
 - not yet common in commercial RTOS

Implementing Self-Scheduling Periodic Tasks

Attempt 1:

```
LOOP
  PeriodicActivity;
  WaitTime(h);
END;
```

Does not work.

Period $> h$ and time-varying.

The execution time of PeriodicActivity is not accounted for.

Implementing Self-Scheduling Periodic Tasks

Attempt 2:

```
LOOP
  Start = CurrentTime();
  PeriodicActivity;
  Stop = CurrentTime();
  C := Stop - Start;
  WaitTime(h - C);
END;
```

Does not work. An interrupt causing suspension may occur between the assignment and WaitTime.

In general, a WaitTime (Delay) primitive is not enough to implement periodic processes correctly.

A WaitUntil (DelayUntil) primitive is needed.

Implementing Self-Scheduling Periodic Tasks

Attempt 3:

```
t = CurrentTime();  
LOOP  
  PeriodicActivity;  
  t = t + h;  
  WaitUntil(t);  
END;
```

Will try to catch up if the actual execution time of PeriodicActivity occasionally becomes larger than the period (a too long period is followed by a shorter one to make the average correct)

Reasonable for alarm clocks, but perhaps not for controllers.

17

2. When is the Sampling Performed?

Two options:

- At the beginning of the controller task
 - gives rise to sampling jitter and, hence, sampling interval jitter
 - still quite common
- At the nominal task release instants
 - using a dedicated high-priority sampling task or in the clock interrupt handler
 - somewhat more involved scheme
 - minimizes the sampling jitter

18

3. When Is the Control Signal Sent Out?

Three Options:

- At the end of the controller task
 - creates a longer than necessary input-output latency
- As soon as it can be sent out
 - minimizes the input-output latency
 - controller task split up in two parts: CalculateOutput and UpdateState
- At the next sampling instant
 - minimizes the latency jitter
 - gives a longer latency than necessary
 - often gives worse performance, also if the constant delay is compensated for
 - delay compensation easy

19

Minimize Input-Output Latency

General Controller representation:

$$\begin{aligned}x(k+1) &= Fx(k) + Gy(k) + G_r y_{ref}(k) \\ u(k) &= Cx(k) + Dy(k) + D_r y_{ref}(k)\end{aligned}$$

Do as little as possible between AdIn and DaOut

```
PROCEDURE Regulate;
BEGIN
  AdIn(y);
  (* CalculateOutput *)
  u := u1 + D*y + Dr*yref;
  DaOut(u);
  (* UpdateStates *)
  x := F*x + G*y + Gr*yref;
  u1 := C*x;
END Regulate;
```

20

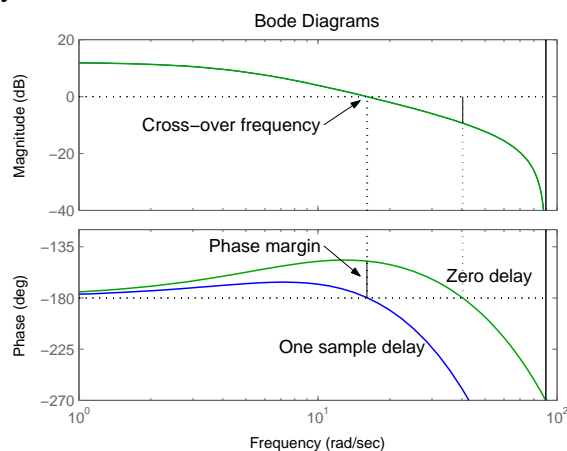
Session Outline

- Control Loop Timing Parameters
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- The Jitter Margin
- The Control Server Model
- Arithmetics

Why is Input-Output Latency Bad?

A constant input-output latency decreases the phase margin.

Example: Loop gain (controller · process) with zero delay or one sample delay:



Computing the Delay Margin*

We have

- Phase margin $\phi_m = 32.4^\circ$
- Crossover frequency $\omega_c = 16.0$ rad/s

How large delay L can be tolerated before we lose stability?

The delay is modeled by $G(s) = e^{-sL}$

At crossover frequency: $\arg G(i\omega_c) = \arg e^{-i\omega_c L} = -\omega_c L$

To retain a positive phase margin, we must have

$$\omega_c L < \phi_m$$

$$16.0 L < 32.4^\circ \frac{\pi}{180^\circ}$$

$$L < 0.035$$

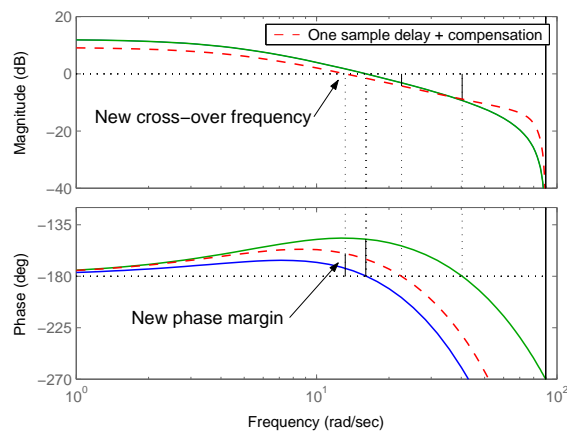
* Since we have a sampled system, the analysis is only approximate

23

Delay Compensation

If the delay is constant and known, it is straightforward to compensate for it in the design.

Delay compensation:



24

Compensation for Fixed Delays

Continuous-Time Designs

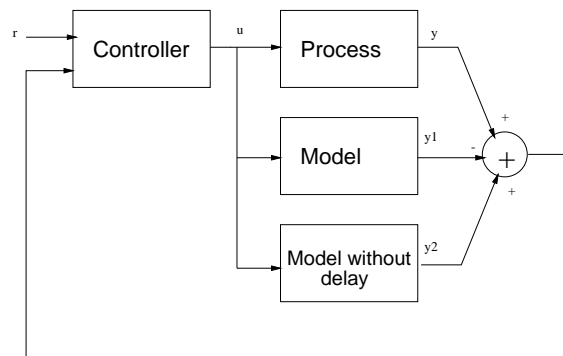
- Otto-Smith controller

Discrete-Time Designs

- Augment the plant model

The Smith Predictor

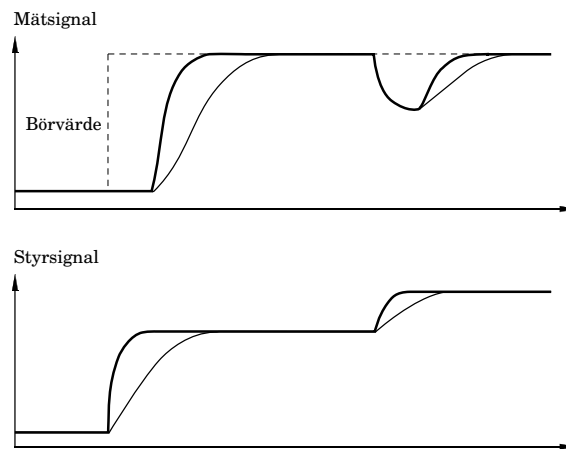
Prediction using the control signal rather than the output derivative



With perfect model the controller does not see any delay

The control performance the same as without any delay (with the exception that the output will be delayed)

PI versus Smith



However, a delay compensating controller can never undo the delay

27

The Smith Predictor

Assume that the process is given by $P(s) = P_0(s)e^{-sL}$ and that we have a perfect model $\hat{P}(s) = P(s)$.

This gives the transfer function

$$Y(s) = \frac{P_0 C}{1 + P_0 C} e^{-sL} R(s)$$

The same as if without any delay + a pure delay

Ideally the controller can be designed for without delay

In practice due to model errors and disturbances the delay must be taken into account in the control design (a more conservative design)

28

Delays in Discrete Time

Include the delay in the discrete time model

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t - \tau), \quad \tau < h$$

$$x(kh + h) - \Phi x(kh)$$

$$\begin{aligned} &= \int_{kh}^{kh+h} e^{A(kh+h-s)} B u(s - \tau) ds \\ &= \int_{kh}^{kh+\tau} e^{A(kh+h-s)} B ds u(kh - h) + \int_{kh+\tau}^{kh+h} e^{A(kh+h-s)} B ds u(kh) \end{aligned}$$

$$= \Gamma_1 u(kh - h) + \Gamma_0 u(kh)$$

29

LTI system!

A state-space model (with extra state $z(kh) = u(kh - h)$)

$$\begin{pmatrix} x(kh + h) \\ z(kh + h) \end{pmatrix} = \begin{pmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x(kh) \\ z(kh) \end{pmatrix} + \begin{pmatrix} \Gamma_0 \\ I \end{pmatrix} u(kh)$$

Can easily be extended to $\tau > h$

Design:

- apply arbitrary discrete time design using the augmented model
- e.g., LQG-design

30

LQG with Deadtime Compensation

Designs a discrete-time LQG controller with direct term for a continuous-time system assuming a constant sampling interval h and a constant time delay τ .

Controller:

$$\begin{aligned}u(k) &= -L\hat{x}_e(k|k) \\ \hat{x}_e(k|k) &= \hat{x}_e(k|k-1) + K_f(y(k) - C_e\hat{x}_e(k|k-1)) \\ \hat{x}_e(k+1|k) &= \Phi_e\hat{x}_e(k|k-1) + \Gamma_e u(k) + K(y(k) - C_e\hat{x}_e(k|k-1))\end{aligned}$$

Used in most of our examples.

Jitterbug command: `lqgdesign`

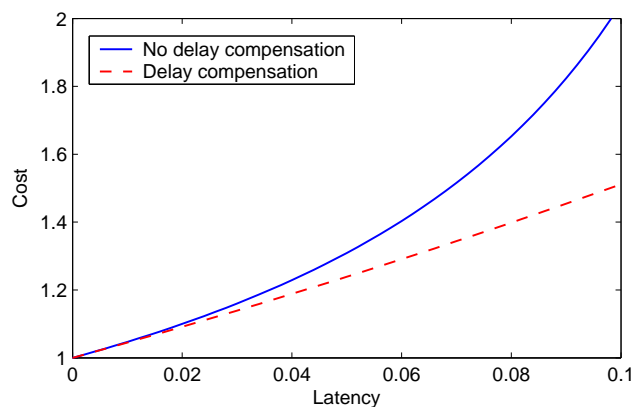
Why is Jitter Bad?

- The controllers were designed assuming a constant h
- The jitter can be interpreted as a process disturbance
- Very hard to analyze in the general case
 - counter-intuitive anomalies can be found
- The Jitterbug toolbox can be used to evaluate the effect of jitter for a given case
- Many jitter compensation schemes have been developed

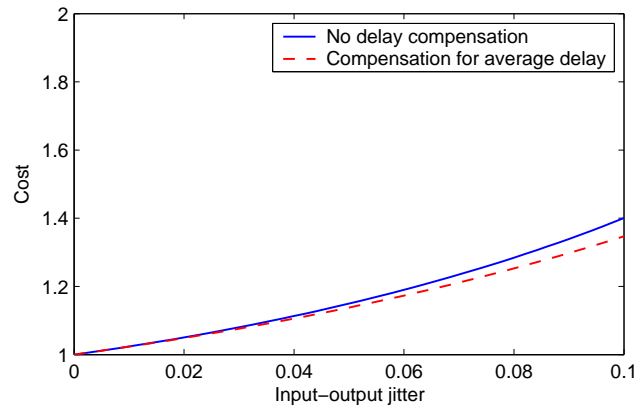
Example: DC Servo with IO Latency and Jitter

- Process: $P(s) = \frac{1}{s(s+1)}$
- LQG controller with or without delay compensation
- Process noise $R_{1c} = 1$, measurement noise $R_2 = 0.01$
- Cost function: $J = \mathbb{E}\{y^2(t) + 0.001u^2(t)\}$
- Periodic sampling with $h = 0.1$
- Constant or random (uniform distribution) IO latency

Constant Input-Output Latency



Input-Output Jitter



Note: having uniform jitter J is only slightly worse than having a constant latency of $L = J/2$.

35

Compensation for Sampling Jitter

Rule of thumb: Jitter that is less than 10% of the nominal sampling period need not to be compensated for

Two approaches:

- Gain scheduling
- Robust design methods

36

Gain Scheduling

Assume that the sampling period can be measured

Store several sets of pre-calculated controller parameters in a table with the sampling period as input parameter.

Switch controller parameters when the sampling period changes

Assumes that the sampling period varies slowly, i.e., not so realistic for jitter

May cause switching transients

Gain Scheduling

What if the sampling period varies fast?

Parameterize the controller parameters in terms of the sampling period

For example:

$$\frac{dx(t)}{dt} \approx \frac{x(t_{k+1}) - x(t_k)}{h_k}$$

Works often well for low order controllers, e.g., PID.

Ad hoc method with no formal guarantees

Robust Design Methods

Design the controller to be robust against timing variations

Several robust design methods are available

- H_∞
- Quantitative Feedback Theory (QFT)
- μ -design
- ...

Session Outline

- Control Loop Timing Parameters
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- The Jitter Margin
- The Control Server Model
- Arithmetics

Switching Controller Task Parameters

Jitter in sampling and latency

- stochastic changes in controller task parameters (period and execution time)
- caused by the implementation platform

Sometimes it can be useful to change the controller task parameters intentionally

- deterministic changes in order to adapt to changing work loads
- generated by a controller when it changes modes (e.g. changes its execution time demands)
- generated by a scheduler when the resources change

May cause both scheduling and control problems

41

Mode Changes and Scheduling

A task set that is schedulable under fixed priority scheduling before the mode change occurs and after the mode change has occurred, may not necessarily be schedulable during the mode change (in transition phase)

Special mode change protocols are needed

Easier under EDF (Earliest Deadline First) scheduling than under fixed priority scheduling

42

Switching-Induced Instabilities

Deterministic changes of task parameters may lead to instability

Example:

Process:

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx\end{aligned}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ -10000 & -0.1 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C = [1 \quad 0]$$

The system is stable with poles in $p_{1,2} = -0.05 \pm 100i$.

Sampled with $h_1 = 0.002\text{s}$ and $h_2 = 0.094\text{s}$

$$\begin{aligned}x_{k+1} &= \Phi_i x_k + \Gamma_i u_k \\ y_k &= C_i x_k \\ i &\in \{1, 2\}\end{aligned}$$

where $\Phi_i = e^{Ah_i}$, $\Gamma_i = \int_0^{h_i} e^{As} B ds$

Both discrete-time systems are stable

Control Design:

State feedback controllers: $u = -K_i x$

LQ-design:

$$J = \int_0^{\infty} (x(t)^T Q_c x(t) + u(t)^T R u(t)) dt$$

with

$$Q_c = \begin{bmatrix} 20000 & 0 \\ 0 & 20000 \end{bmatrix} \quad R = 50$$

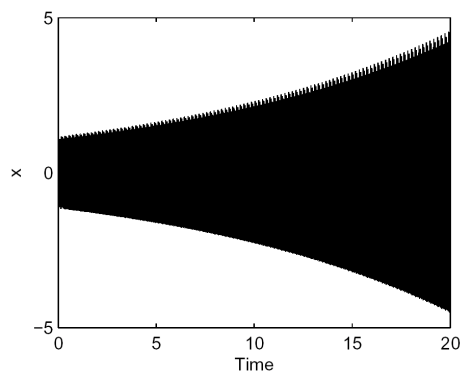
45

Both closed-loop systems, $\Phi_i - \Gamma_i K_i$, are stable

- eigenvalues inside unit circle

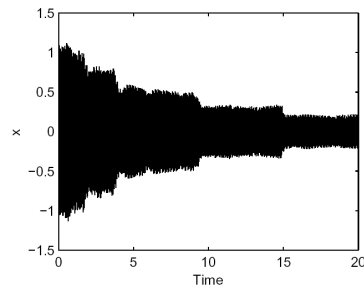
However, the switching sequence $h_1, h_2, h_2, h_1, h_2, h_2, \dots$ gives an unstable system

- eigenvalues of $(\Phi_2 - \Gamma_2 K_2)^2 (\Phi_1 - \Gamma_1 K_1)$ outside the unit circle



46

If we instead switch between the two controller stochastically using the relative frequency 67% for h_2 and 33% for h_1 the resulting system is stable (in the mean-square sense).



The phenomenon can in principle occur also in other cases:

- change of sampling interval for the same controller
- change of input output latency

However, it is rare and so far we have not seen any “realistic” examples where it has occurred.

47

Switching & Controller State

Switching sampling intervals may also cause problems for controllers on input-output form

$$u(k) = a_1y(k) + a_2y(k-1) + a_3y(k-2) + b_1u(k-1) + b_2u(k-2)$$

Remedy:

- only allow switches in stationarity
- use an observer (Kalman filter) to estimate the signal values at the new points in time

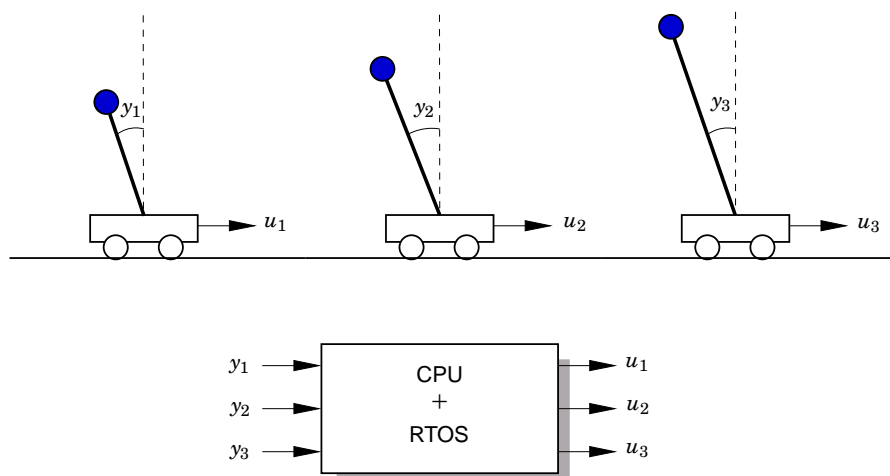
48

Session Outline

- Control Loop Timing Parameters
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- **The Jitter Margin**
- The Control Server Model
- Arithmetics

Inverted Pendulum Example

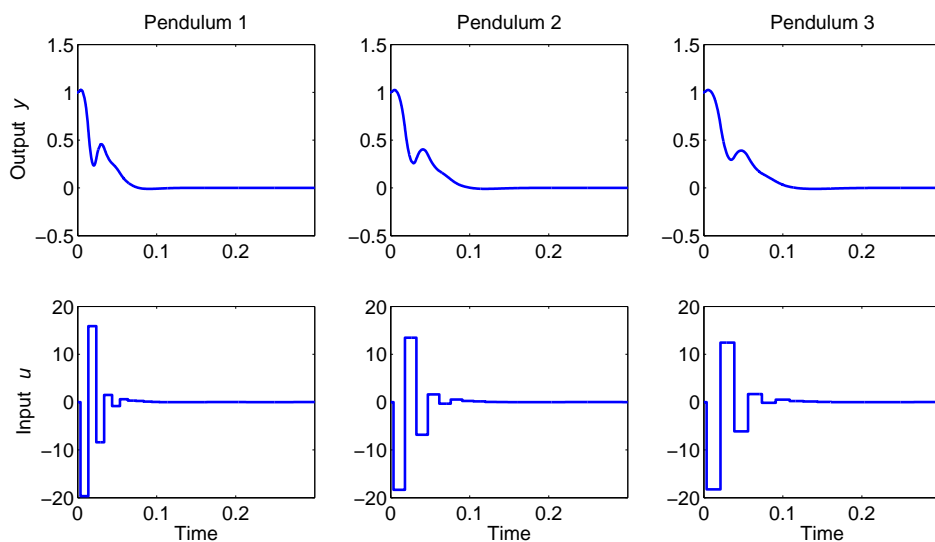
Suppose you want to control three inverted pendulums using one CPU:



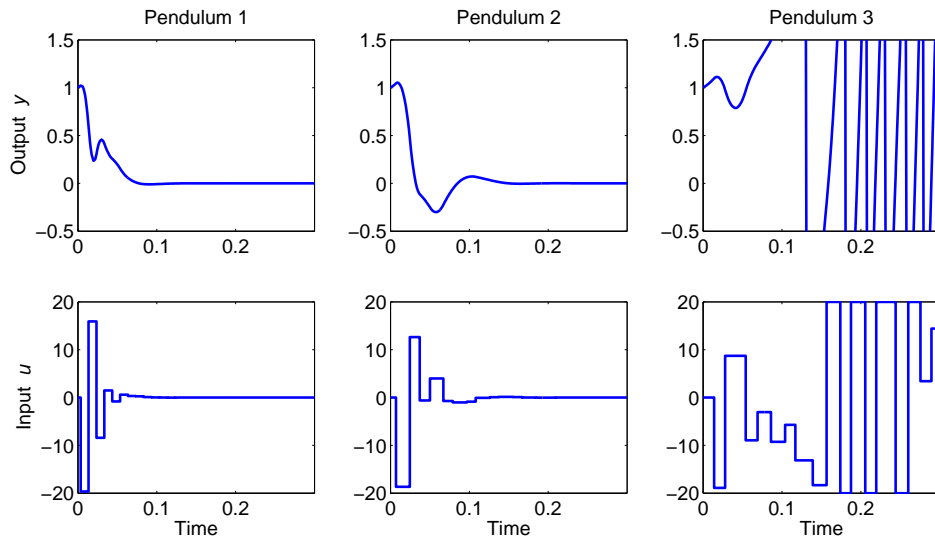
Design

- Discrete-time LQG controllers
- Sampling intervals: $(T_1, T_2, T_3) = (10, 14.5, 17.5)$ ms
- Assumed execution time: $C_i = 3.5$ ms
- Controllers designed assuming delay of 3.5 ms
 - Jitterbug command: `lqgdesign`
- Schedulable under both RM and EDF (with $D_i = T_i$)

Simulation 1 – No Interference

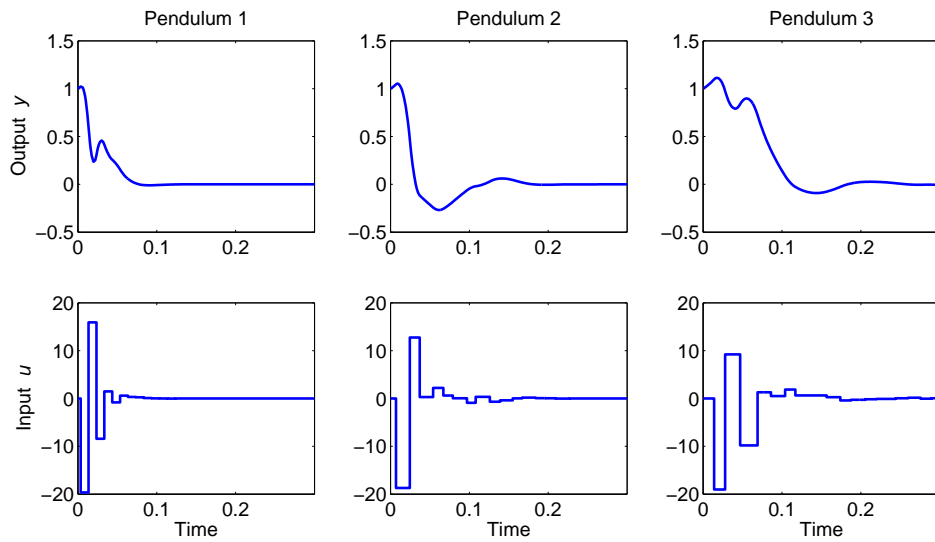


Simulation 2 – Rate-Monotonic Scheduling



53

Simulation 3 – Earliest-Deadline-First Scheduling



54

Questions

- How much jitter is there under various scheduling policies?
 - Simulation
 - Jitter analysis
- How much jitter do the control loops tolerate?
 - Simulation
 - The jitter margin

Jitter analysis + the jitter margin give *hard stability results*

Jitter Analysis – Rate-Monotonic Scheduling

- R_i – worst-case response time of task i

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- R_i^b – best-case response time of task i

$$R_i^b = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^b}{T_j} - 1 \right\rceil C_j$$

- J_i – worst-case input-output jitter of task i :

$$J_i = R_i - R_i^b$$

(Analysis for earliest-deadline-first scheduling also exists)

The Pendulum Example – RM Scheduling

Task	T	C	R	R^b	J
1	10	3.5	3.5	3.5	0
2	14.5	3.5	7.0	3.5	3.5
3	17.5	3.5	14.0	3.5	10.5

The Delay Margin

- L_m – delay margin, the longest delay a loop can tolerate without becoming unstable
- Simple to compute
 - Continuous-time system: $L_m = \varphi_m / \omega_c$
 - * φ_m – phase margin [rad]
 - * ω_c – cross-over frequency [rad/s]
 - Sampled-data system: need to compute a root locus with respect to the delay

Delay Margins in the Pendulum Example

The maximum delay is equal to the response time R

Compute the delay margin L_m for each controller:

Task	T	C	R	L_m
1	10	3.5	3.5	9.8
2	14.5	3.5	7.0	12.5
3	17.5	3.5	14.0	14.6

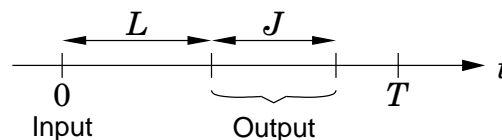
$\forall i : R_i < L_{mi}$. Still, system 3 was seen to be unstable!

The delay margin is only valid for constant delays!

The Jitter Margin

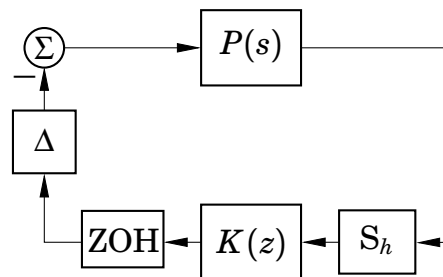
Assumptions:

- Periodic sampling (high-prio/interrupt-driven)
- Arbitrarily time-varying input-output delay $\Delta \in [L, L + J]$
 - L – constant part
 - J – jitter



Jitter margin $J_m(L)$ – the largest J for which stability can be guaranteed given a value of L

Checking Stability

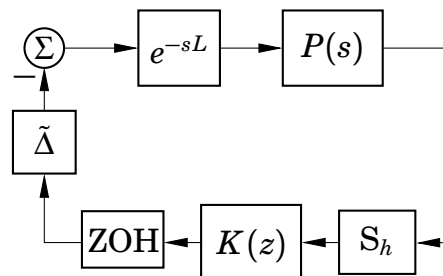


- Continuous-time plant $P(s)$
- Discrete-time controller $K(z)$
- Arbitrarily time-varying delay $\Delta \in [L, L + J]$
- Closed-loop system assumed stable for $\Delta = L$

61

Checking Stability

Include the constant delay L in the plant:

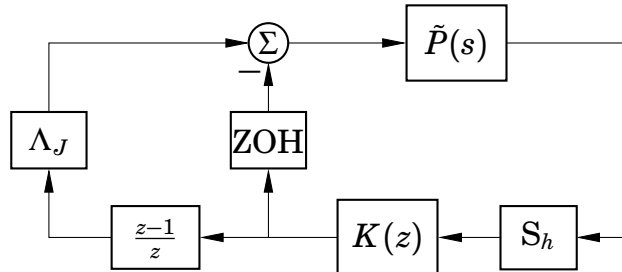


- New time-varying delay $\tilde{\Delta} \in [0, J]$
- New plant $\tilde{P}(s) = P(s)e^{-sL}$

62

Checking stability

Rewrite the control output as one direct path and one error path:

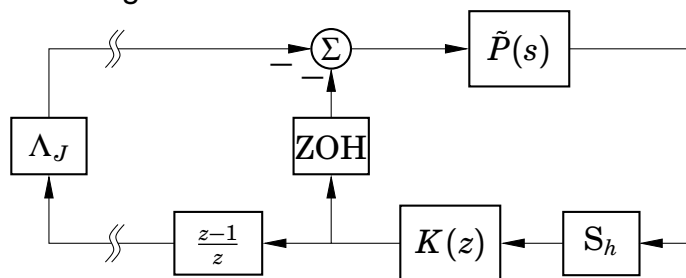


- Difference operator $\frac{z-1}{z}$
- Time-varying gate function Λ_J (open at most J seconds every sample)

63

Checking Stability

Apply the small gain theorem:



- L_2 -gain of gate function: $\|\Lambda_J\| = \sqrt{J}$
- L_2 -gain of the rest:

$$\|H\| = \max_{\omega} \left\{ \left| \frac{P_{\text{alias}}(\omega)K(e^{i\omega})}{1 + P_{\text{ZOH}}(e^{i\omega})K(e^{i\omega})} \right| |e^{i\omega} - 1| \right\}$$

64

Checking Stability

The closed-loop system is stable if $\|\Lambda_J\| \|H\| < 1 \Leftrightarrow$

$$\left| \frac{P_{\text{alias}}(\omega)K(e^{i\omega})}{1 + P_{\text{ZOH}}(e^{i\omega})K(e^{i\omega})} \right| < \frac{1}{\sqrt{J}|e^{i\omega} - 1|}, \quad \forall \omega \in [0, \pi]$$

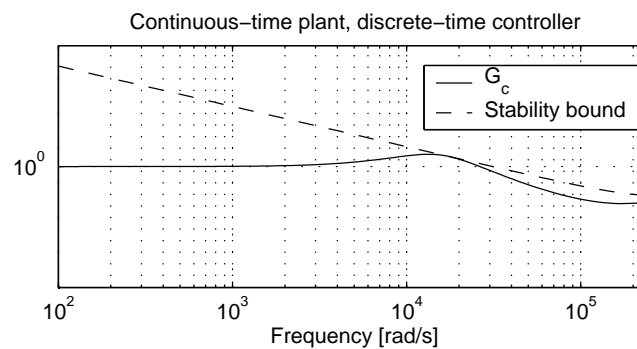
Here,

- $P_{\text{alias}}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} |\tilde{P}(i(\omega + 2\pi k)\frac{1}{h})|^2}$
- $P_{\text{ZOH}}(z)$ is the ZOH-discretization of $\tilde{P}(s)$

(For small h , $P_{\text{alias}}(\omega) \approx P_{\text{ZOH}}(e^{i\omega})$)

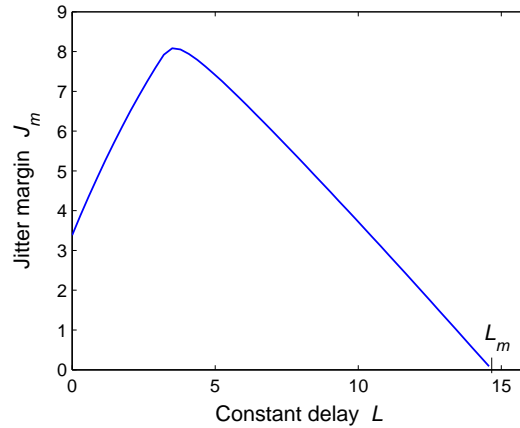
Checking Stability

Graphical test:



Jitter Margin – Example

$J_m(L)$ for pendulum controller 3:



- $L_m = 14.6$
- $J_m(3.5) = 8.1$

67

Deadline Assignment

Stability of the closed-loop systems can be guaranteed by assigning relative deadlines

$$D_i = J_m(L_i) + L_i$$

and verifying that the resulting task set is schedulable.

(In our example, assigning such deadlines gives an unschedulable system under fixed-priority scheduling)

68

The Pendulum Example – RM

- Compute the jitter margin $J_m(L)$ for each task
- $J < J_m(L) \Rightarrow$ Stable

Task	R	$L = R^b$	J	$J_m(L)$	Stable
1	3.5	3.5	0	4.4	Yes
2	7.0	3.5	3.5	6.4	Yes
3	14.0	3.5	10.5	8.1	No?

The Pendulum Example – EDF

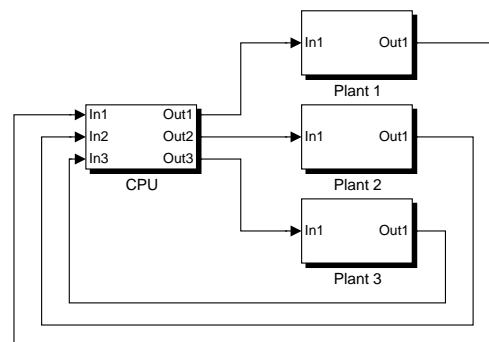
Task	R	$L = R^b$	J	$J_m(L)$	Stable
1	3.5	3.5	0	4.4	Yes
2	7.5	3.5	4.0	6.4	Yes
3	10.5	3.5	7.0	8.1	Yes

Session Outline

- Control Loop Timing Parameters
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- The Jitter Margin
- **The Control Server Model**
- Arithmetics

71

The Control–Scheduling Co-Design Problem

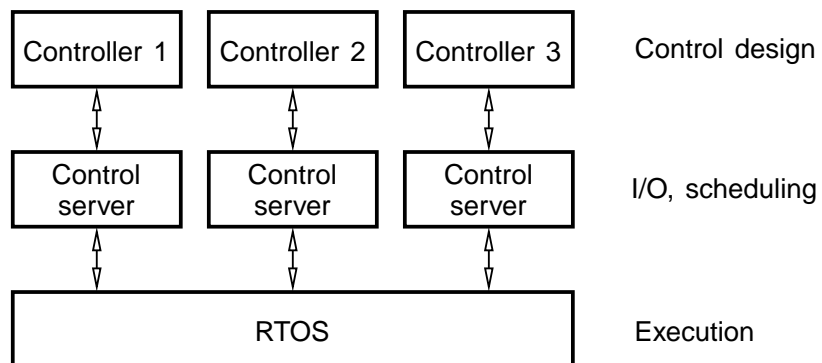


- Multiple plants controlled by a CPU with limited resources
- Controller *and* scheduling parameters should be chosen to optimize the overall control performance
- Very complex problem, due to latencies and jitter

72

The Control Server

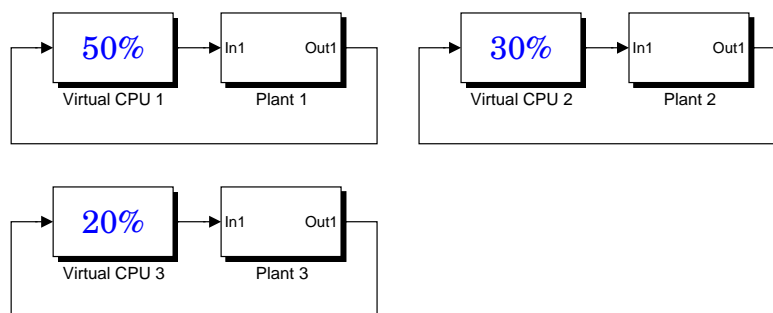
- Extra layer between the controller and the RTOS
- Provides *temporal isolation* between the controllers



73

Co-Design Using Control Servers

The CPU is divided into many virtual CPUs:



- A share of the processor is assigned to each control task
- Each control loop can be analyzed independently
- Simple to do trade-offs between the loops

74

How to Achieve the Virtual CPUs?

Step 1: *Constant bandwidth servers* (Abeni & Buttazzo, 1998)

A constant bandwidth server (CBS) is described by

- a server period, T_s
- a server bandwidth (maximum utilization), U_s

Scheduling mechanism (assuming one task per server):

- Based on earliest-deadline-first scheduling
- Each period T_s , the task has the budget $C_s = U_s T_s$
- Normally, the task has the relative deadline $D = T_s$
- When an overrun occurs (when the budget is exhausted)
 - the task deadline is moved one period forward
 - the budget is recharged to C_s

75

How to Achieve the Virtual CPUs?

- Ideal processor sharing with CBSes requires $T_s \rightarrow 0$
- Overhead $\rightarrow \infty$

However, the *observable behavior* of a task only consists of

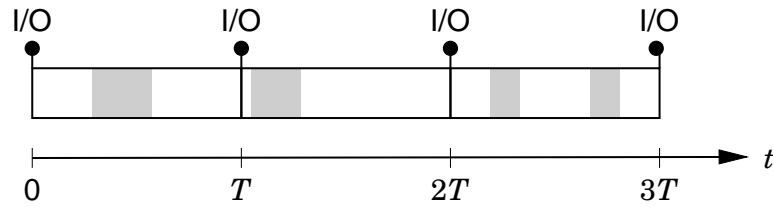
- the input actions
- the output actions

Step 2: *Add time-triggered I/O points*

- I/O at highest (interrupt) priority
- Unaffected by the task scheduling

76

A Simple Control Server Task



- Period T
- CPU share U
- I/O at beginning and end of each period

Behaves as a task executing alone on a CPU with speed U of the original speed

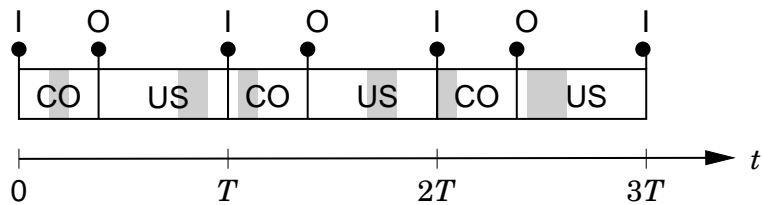
77

Control Server Tasks with Segments

Multiple segments can be used to reduce the I-O latency

Example:

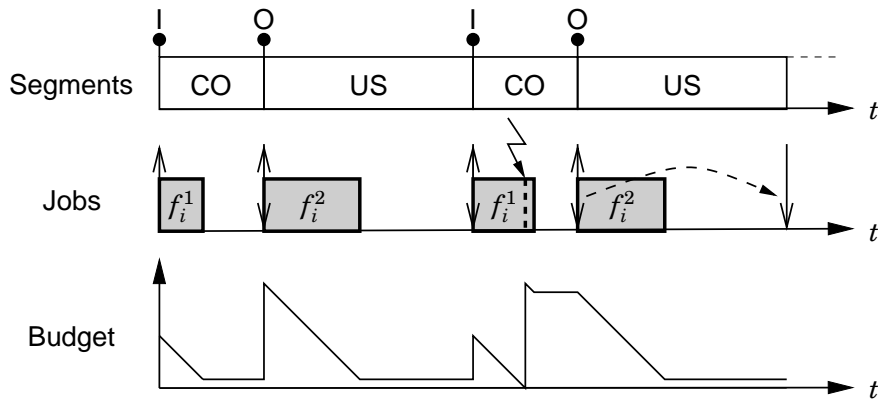
- Calculate output
- Update state



78

Execution

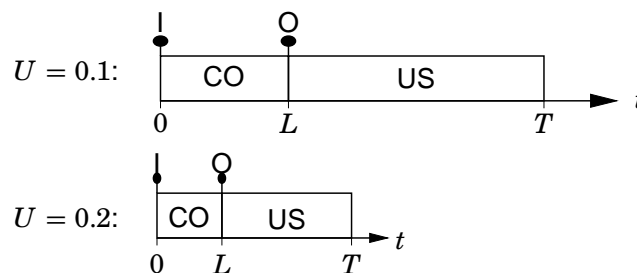
- One job is released in each segment
- Deadline at end of segment
- Scheduled by CBS with variable period = segment length



79

Scalable Control Components

View the CPU share U as a design parameter. Example:

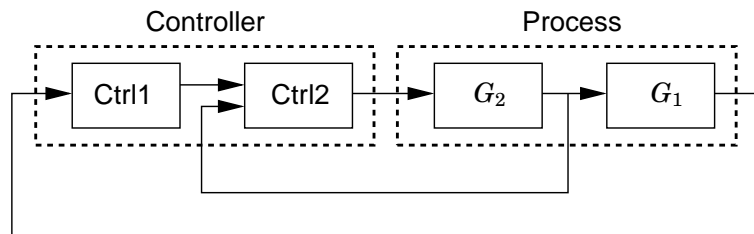


- Sampling period $T \propto 1/U$
- Input-output latency $L \propto 1/U$
- Performance $J = J(U)$

80

Composite Control Tasks

Cascade controller:



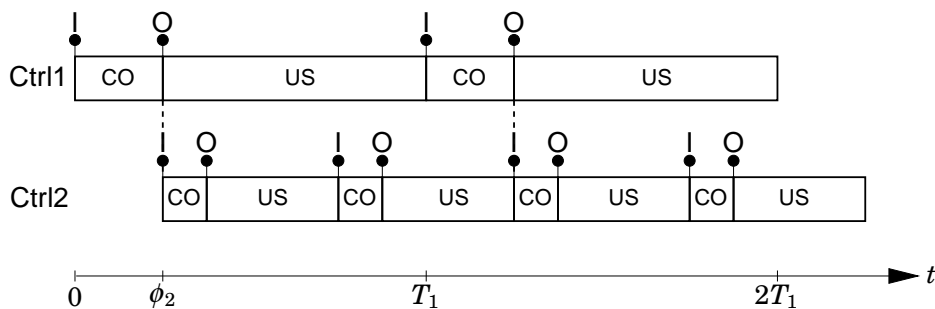
The cascade controller could be built from two control components with different rates

- $T_1 = 2T_2 \Leftrightarrow U_2 = 2U_1$
- Synchronization required

81

Offsets

Communicating tasks can be synchronized using offsets:



82

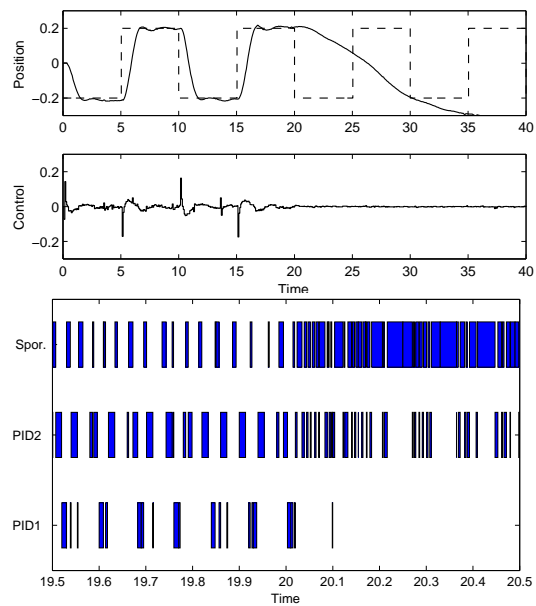
Control Experiments

- Ball and beam process
- Multirate cascade PID controller (PID1, PID2)
- Sporadic disturbance task (Spor.)
- Comparison of
 - RM
 - EDF
 - Control server



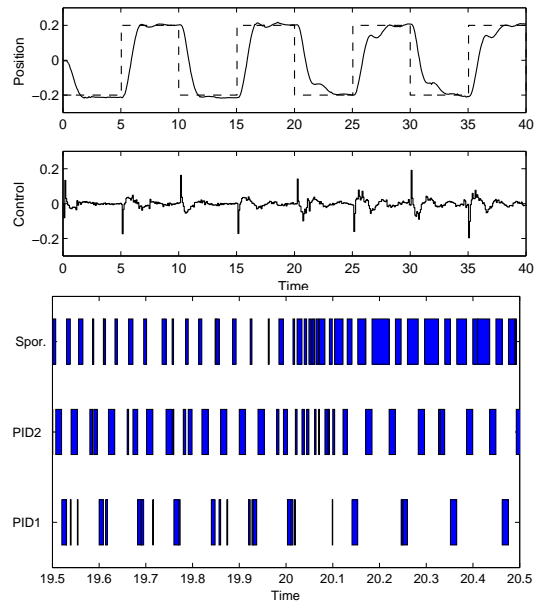
83

Results – RM



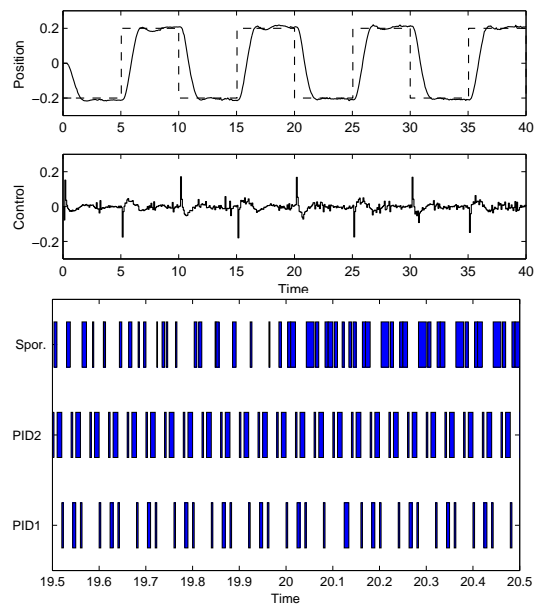
84

Results – EDF



85

Results – Control Server



86

Session Outline

- Control Loop Timing Parameters
- Temporal Non-Determinism
 - Input-Output Latency
 - Sampling
- Switching
- The Jitter Margin
- The Control Server Model
- **Arithmetics**

Computer Arithmetics

Control analysis and design assumes floating point arithmetics (i.e. high range and resolution)

Hardware-supported on modern high-end processors (e.g., floating point ALUs (Arithmetic-Logic Units))

Representation:

$$\pm f \times 2^{\pm e}$$

- f : mantissa, significant, fraction
- 2: radix or base
- e : exponent

IEEE 754 Standard

Used by almost all floating-point processors (except certain DSPs)

Single precision (Java/C float):

- 32-bit word divided into 1 sign bit, 8-bit exponent, and 23-bit mantissa
- Range: $2^{-126} - 2^{128}$

Double precision format (Java/C double):

- 64-bit word divided into 1 sign bit, 11-bit exponent, and 52-bit mantissa.
- Range: $2^{-1022} - 2^{1024}$

Supports infinity and NaN

Floating-Point Emulation

Emulate floating-point arithmetics in software

Approaches:

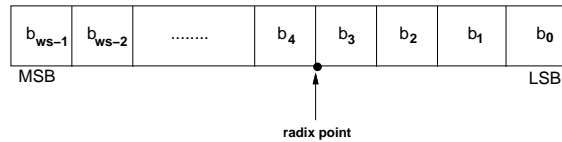
- compiler supported
- manually
 - e.g., floating point variables represented as C structs
 - floating point operations in the form of a library

Problems:

- Code size becomes too large
- Slows down execution speed
- Non-trivial

Fixed-Point Arithmetics

Use the binary word directly for representing numbers



- MSB - Most significant bit
- LSB - Least significant bit
- ws - word-size

Unsigned versus signed

Fixed-Point Arithmetics

Integer arithmetics:

- radix point to right of LSB
- 16 bits signed integer gives range $-32768 \leq \hat{x} \leq 32767$
($(-2^{15}) - (2^{15} - 1)$)

Fractional arithmetics:

- radix point to right of MSB (signed)
- 0.10011001

Generalized fixed point arithmetics:

- application-defined radix point
- 1101.0110
- Scaling: $x = \hat{x}/2^4$ – shifting the radix point

Fixed-Point Arithmetics

Fixed point arithmetics: N bits (signed) integer

- Example: $N=16$ gives range $-32768 \leq \hat{x} \leq 32767$
- We can use fixed scale to get decimals:

$$x = \hat{x}/2^8$$

E.g., $\hat{x} = 315 \Rightarrow x = 1.2305$

- Multiplication then requires rescaling:

$$z = x \cdot y = \hat{x}/2^8 \cdot \hat{y}/2^8 \Rightarrow \hat{z} = (\hat{x} \cdot \hat{y})/2^8$$

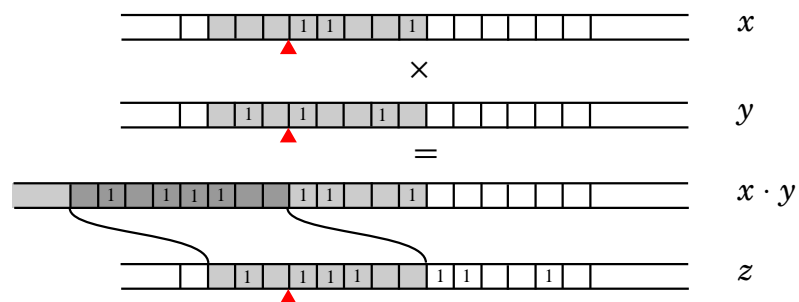
$$z = 1.2305 \cdot 2.4609 = 315/2^8 \cdot 630/2^8 \Rightarrow \hat{z} = (315 \cdot 630)/2^8$$

$$z = 3.0281 \Rightarrow \hat{z} = 775$$

93

Fixed-Point Calculations

Fixed point multiplication involves quantization



Fixed-point addition is error-free

Quantization (truncation or rounding)

- modeled as “noise”

Overflow (wrap-around or saturation)

94

Example: Scalar Products

Many controllers and filters involve calculations of scalar products, e.g.,

$$u = -Lx = -[l_1 \ l_2 \ l_3][x_1 \ x_2 \ x_3]^T = -l_1x_1 - l_2x_2 - l_3x_3$$

Consider the vectors

$$\begin{aligned} a &= (100 \ 1 \ 100) \\ b &= (100 \ 1 \ -100) \end{aligned}$$

The true scalar product is 1

When computed in fixed point representation using a precision corresponding to three decimal places, the result will be 0 (100 × 100 + 1 × 1 is rounded to 10000)

The result depends on the order of the operations.

To avoid this it is common to use higher resolution in the accumulator and round to a smaller resolution afterwards.

95

Fixed-Point Arithmetics Problems

- Quantization
 - Fixed-point values are rounded or truncated.
 - Coefficient Quantization: Poles and zeros end up somewhere else
 - Signal (state) Quantization:
 - * Noise is added in each operation
 - * Quantization may cause signal bias
 - * Quantization may cause limit cycles. Either in the output only (LSB) or in the entire system through feedback.
- Overflow
 - Adding/Multiplying two sufficiently large numbers can produce a result that does not fit into the representation.
 - Scaling important both of variables and of coefficients.
 - Overflow characteristics. Saturation or wrap-around? Hardware supported overflow detection or not.

96

Example: Coefficient Quantization

An example controller

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.5776}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184}$$

8-bit fixed point coefficients with $x = \hat{x}/2^4$, so

$$x \in [-8.0 \dots 7.9375]$$

4 integer bits



▲ 4 fractional bits
 2^4

Example: Coefficient Quantization

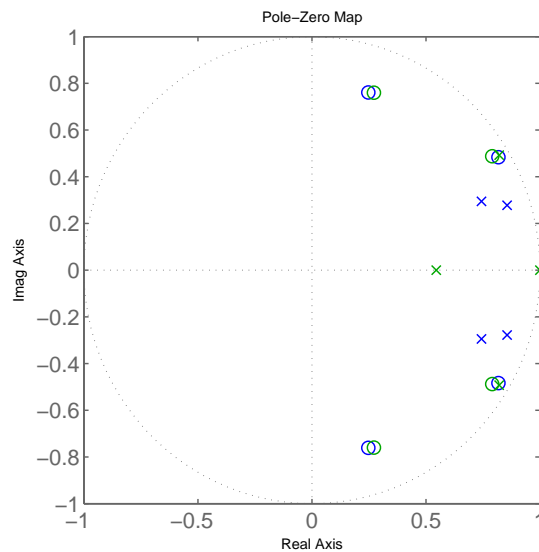
- Original:

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.576}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184}$$

- Quantized:

$$C(z) = \frac{z^4 - 2.125z^3 + 2.375z^2 - 1.5z + 0.5625}{z^4 - 3.188z^3 + 4z^2 - 2.312z + 0.5}$$

Example



99

© Lund University 2006

Issues: Realization of Digital Controllers

A digital controller

$$u(k) = H(q^{-1})y(k) = \frac{b_0 + b_1q^{-1} + \dots + b_mq^{-m}}{1 + a_1q^{-1} + a_2q^{-2} + \dots + a_nq^{-n}}y(k)$$

can be realized in a number of different ways with equivalent input-output behavior (different choice of state variables)

Issues:

- number of storage elements (memory)
- number of non-zero non-one coefficients
- coefficient range
- sensitivity towards coefficient quantization
- sensitivity towards state quantization
 - order of computations matters

100

© Lund University 2006

Direct and Companion Forms

$$u(k) = \sum_{i=0}^m b_i u(k-i) - \sum_{i=1}^n a_i y(k-i)$$

Not minimal ($n + m$ states)

Companion forms (e.g., observable canonical form or controllable canonical form):

$$x(k+1) = \begin{pmatrix} -a_1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n-1} & 0 & \cdots & 1 \\ -a_n & 0 & \cdots & 0 \end{pmatrix} x(k) + \begin{pmatrix} b_1 \\ \vdots \\ b_m \\ 0 \end{pmatrix} y(k)$$

$$u(k) = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} x(k)$$

Minimal

Coefficients in the characteristic polynomial are the coefficients in the realization. Sensitive to computational errors if the systems are of high order and if the poles or zeros are close to each other.

101

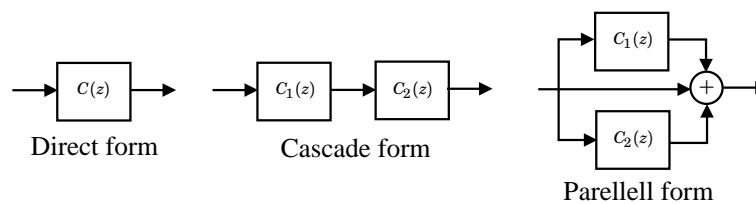
Example

A linear system can be rewritten in many ways:

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.5776}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184}$$

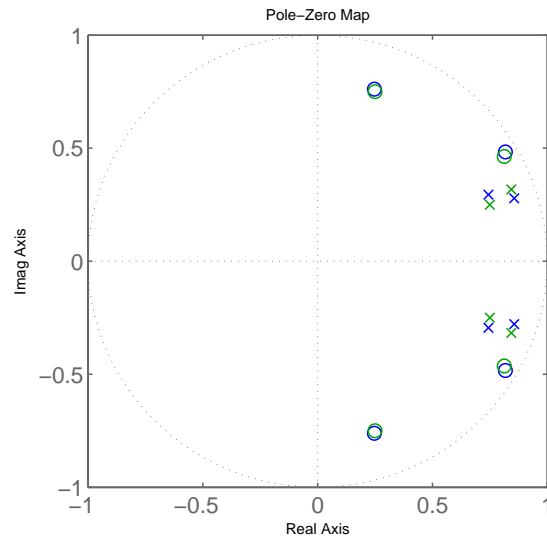
$$= \left(\frac{z^2 - 1.635z + 0.9025}{z^2 - 1.712z + 0.81} \right) \left(\frac{z^2 - 0.4944z + 0.64}{z^2 - 1.488z + 0.64} \right)$$

$$= 1 + \frac{-5.396z + 6.302}{z^2 - 1.712z + 0.81} + \frac{6.466z - 4.907}{z^2 - 1.488z + 0.64}$$



102

Cascade Form

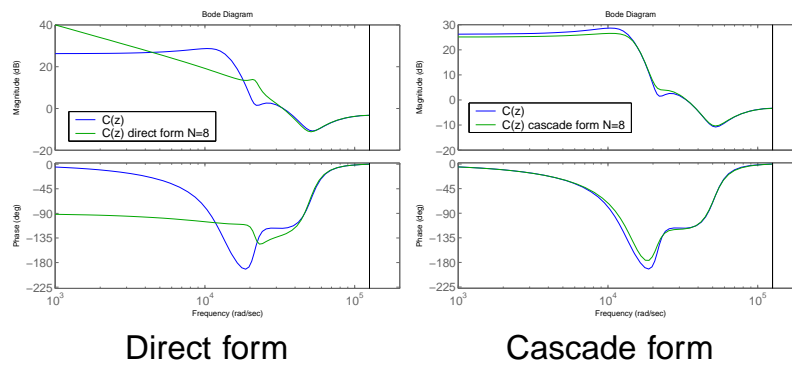


103

Well-Conditioned Realizations

Parallel (diagonal/Jordan) and cascade (series) forms have normally the best numerical properties.

If poles (zeroes) are far apart, direct form is usable.

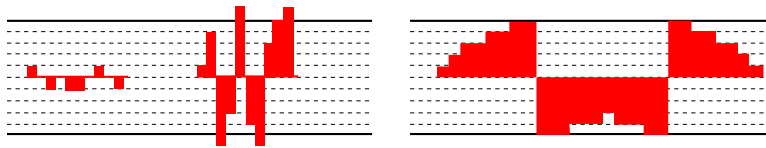


104

State Saturation

For fixed point arithmetics, there is a balance:

- Too high gain in some part of system will cause state to overflow.
- Too low gain in some part of system will cause a lot of quantization errors.



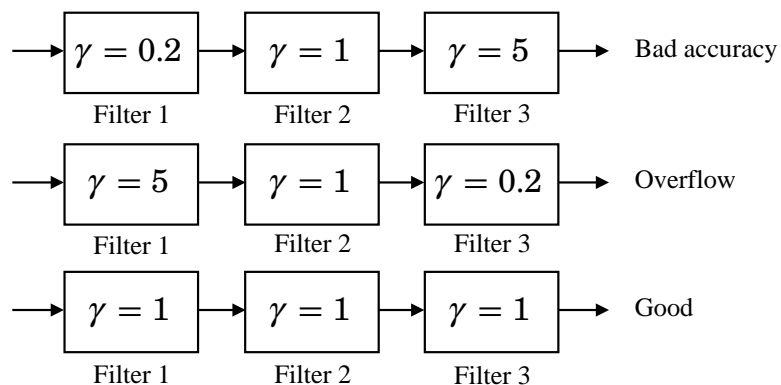
Your digital system should have gain $\gamma \approx 1$.

What is γ ? *The gain of the system for the kind of input signal we expect*

105

State Saturation

Spread the gain:



106

State Saturation

How to pair and order poles and zeros?

Jackson's rules (1970):

- Pair the pole closest to the unit circle with its closest zero. Repeat until all poles and zeros are taken.
- Order the filters in increasing or decreasing order based on the poles closeness to the unit circle.

This will push down high internal resonance peaks.

Summing Up

Problems and solutions:

- Coefficient quantization:
 - Avoid direct forms and companion forms
 - Always split systems into first- and second-order systems (cascade, parallel form)
- State quantization:
 - Can be modeled as noise sources after multipliers
 - Use double-size accumulator
- State saturation:
 - Have equal gains ($\gamma \approx 1$) for all systems
 - Use Jackson's rules for pole-zero sorting

Wednesday 5th of April



ARTIST2

Embedded Control Systems: Control of Computing Systems

Karl-Erik Årzén and Anton Cervin

Department of Automatic Control

Lund University

Sweden

{karlerik,anton}@control.lth.se



Session Outline

1. Overview
2. Some General Observations
3. Control of Web servers
4. Feedback Scheduling of Controllers

Control of Computer Systems

Apply control as a techniques to manage uncertainty and achieve performance and robustness in computer and communication systems.

One of the strongest increasing areas in real-time computing (adaptive/flexible scheduling) and networking.

Applications in

- Internet protocols, e.g., TCP and extensions
- Internet servers (HTTP, Email)
- Cellular phone systems (power control, ...)
- CPU scheduling

Control used to manage finite resources (Resource allocation as a control problem = feedback scheduling)

3

Control of Computer Systems

New area

- However, feedback has been applied in ad hoc ways for long without always understanding that it is control

Textbooks are emerging:

- “Feedback Control of Computer Systems”, Hellerstein, Diao, Parekh, Tilbury
- Book by Stankovic, Abdelzaher, ...

4

Control of Computer Systems

Control of computing systems can benefit from a lot of the classical control results

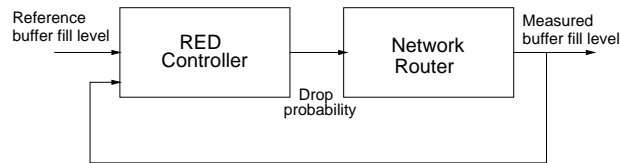
- However, several new challenges
- First principle modeling not so natural
- Complex dynamics no longer the problem

Example: Internet Protocol

The congestion control in TCP is one of the major reasons why Internet has been able to expand at the current high rate and still work properly.

- Congestion window (cw) decides how many un-ack'ed packets a host can have
- When cw below threshold it grows exponentially
- When cw above threshold it grows linearly
- Whenever there is a timeout the threshold is set to half the cw and cw is set to 1.
- Nonlinear behavior

Example: Internet Protocols



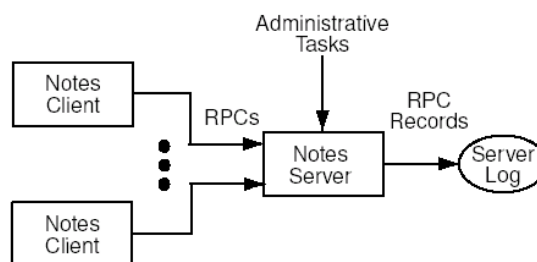
Random Early Detection (RED) of Router Overloads

- Prevent router buffers from overflowing
- Random drops of packets before the buffer is full

A lot of ongoing work on improvements of IP based on models and theory rather than on ad hoc fixes

7

Example: Lotus Notes E-Mail Server



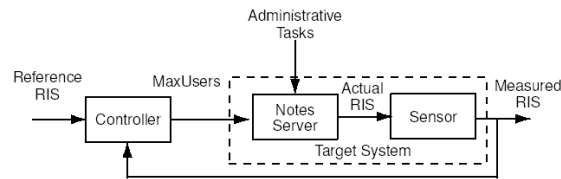
Client-server application

Interaction using Remote Procedure Calls (RPC)

Server log of RPC statistics

8

Example: Lotus Notes E-Mail Server



- Control the number of RPCs in the server (*RIS*) by adjusting the maximum allowed users (*MaxUsers*)
- First-order model derived from data:

$$y(k+1) = 0.43y(k) + 0.47u(k)$$

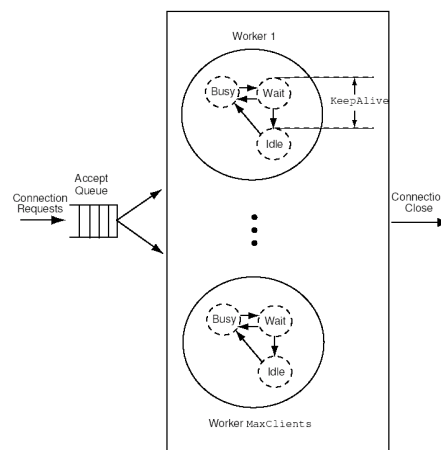
where $y(k) = RIS(k) - RIS_0$ and $u(k) = MaxUsers(k) - MaxUsers_0$

- First-order LP-filter added to remove outliers
- Resulting second-order system controlled by PI-controller

9

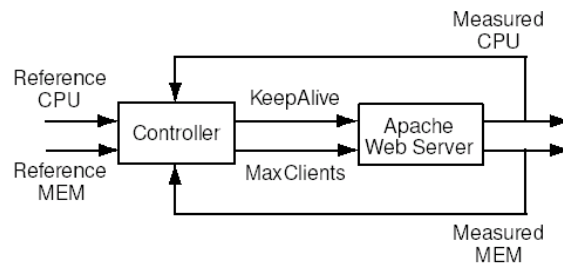
Example: Apache HTTP Server

- HTTP requests from clients to server
- Pool of workers being either Idle, Busy or Waiting (Persistent Connections)
- *MaxClients* limits the size of the worker pool
- *KeepAlive* determines how long a worker is waiting before it becomes idle



10

Example: Apache HTTP Server



Control of CPU utilization and memory utilization

Too large *MaxClients* → large consumption of CPU and memory

Too large *KeepAlive* → under-utilization

Too small *KeepAlive* increases CPU consumption since connections must be re-established for requests from the same user

11

Example: Apache HTTP Server

Two first-order transfer functions derived from input-output data

$$CPU(z) = G_{MC}(z)MC(z) + G_{KA}(z)KA(z)$$

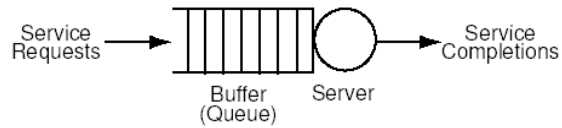
around a certain operating point

PI-controller using *KeepAlive* control signal

Design using pole placement

12

Example: Queuing Systems



Work requests (customers) arrive and are buffered

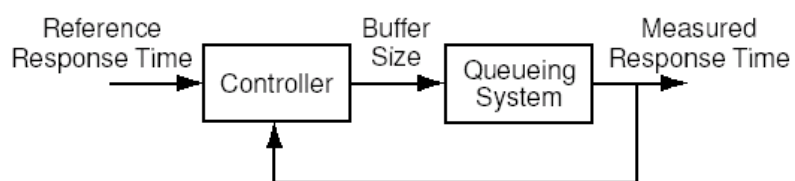
Service level objectives (response time for request belonging to class X should be less than Y time units)

Reduce the delay caused by other requests, i.e., adjust the buffer size and redirect or block other requests

Admission control

13

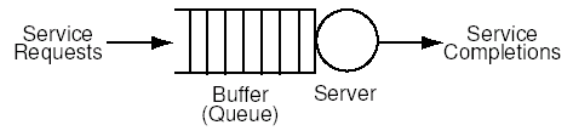
Example: Queuing Systems



14

Example: Queue Length Control

Assume an M/M/1 - queuing system:

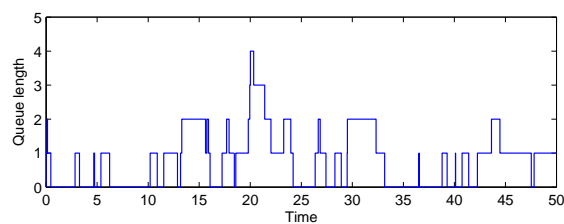


- Random arrivals (requests), average λ per second
- Random service times, average $1/\mu$ and exponentially distributed
- queue containing x requests

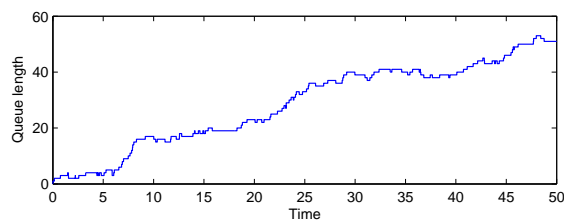
Intuition: $x \rightarrow \infty$ if $\lambda > \mu$

Queue Length Control: Simulation

$\lambda = 0.5, \mu = 1$:



$\lambda = 2.0, \mu = 1$:



Queue Length Control: Model

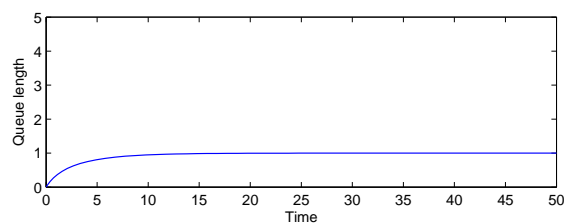
Approximate the system with a nonlinear flow model (Tipper's model from queuing theory)

The expectation of the queue length x is

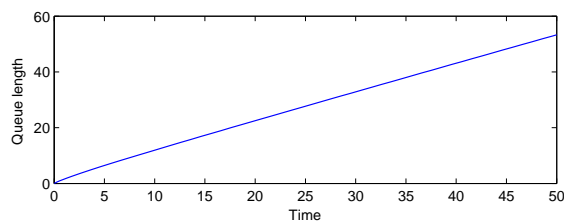
$$\dot{x} = \lambda - \mu \frac{x}{x+1}$$

Queue Length Control: Model

$\lambda = 0.5, \mu = 1$:



$\lambda = 2.0, \mu = 1$:



Queue Length Control: Model

Control the queue length by only admitting a fraction u (between 0 and 1) of the requests

$$\dot{x} = \lambda u - \mu \frac{x}{x+1}$$

Admission control

Queue Length Control: Linearization

Linearize around $x = x^\circ$

Let $y = x - x^\circ$

$$\dot{y} = \lambda y - \mu \frac{1}{(x^\circ + 1)^2} y = \lambda y - \mu a y$$

Queue Length Control: P-Control

$$u = K(r - y)$$

$$\dot{y} = \lambda K(r - y) - \mu a y$$

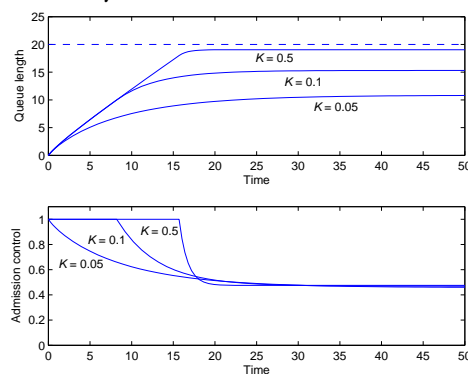
$$(s + \lambda K + \mu a)Y(s) = \lambda K R(s)$$

$$G_{cl}(s) = \frac{\lambda K}{s + \lambda K + \mu a}$$

With K the closed loop poles can be placed arbitrarily

Queue Length Control: P-Control

Simulations for $\lambda = 2$, $\mu = 1$, $x^o = 20$ and different values of K



- Stationary error
- Nonlinear system (control signal limitations)

Queue Length Control: PI-Control

$$G_P(s) = \frac{\lambda}{s + \mu a}$$

$$G_R(s) = K \left(1 + \frac{1}{sT_i} \right)$$

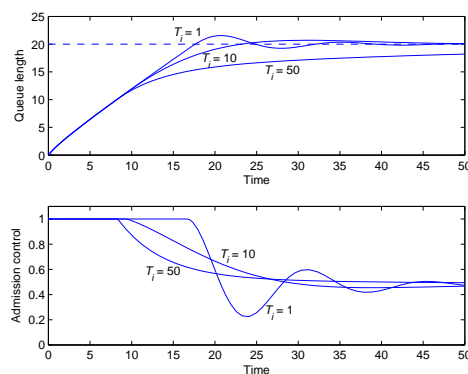
$$G_{cl}(s) = \frac{G_P G_R}{1 + G_P G_R} = \frac{\lambda K \left(s + \frac{1}{T_i} \right)}{s(s + \mu a) + \lambda K \left(s + \frac{1}{T_i} \right)}$$

With K and T_i the closed loop poles can be placed arbitrarily

23

Queue Length Control: PI-Control

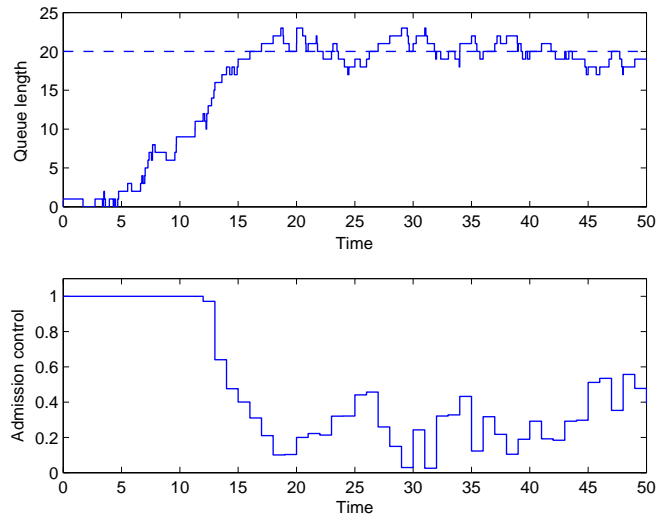
Simulations for $\lambda = 2$, $\mu = 1$, $x^\circ = 20$, $K = 0.1$ and different values of T_i



- Stationary error removed
- Tracking (anti-windup) important

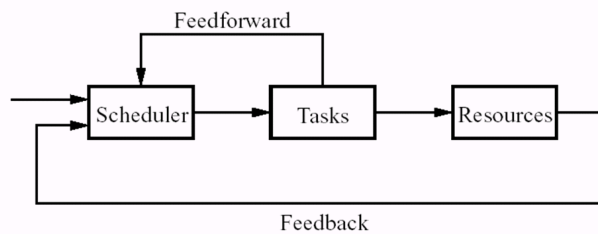
24

Queue Length Control: PI-Control on Process



25

Example: Task Scheduling



Control CPU utilization by adjusting

- task periods
- task execution demands
- priorities

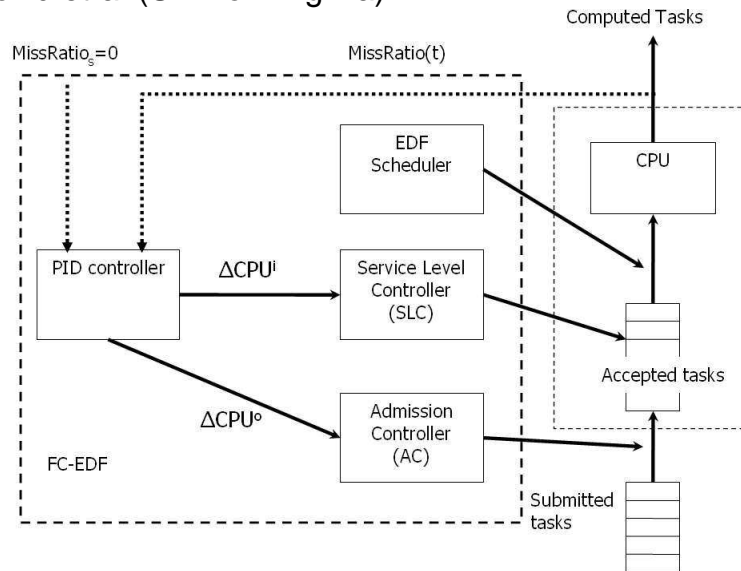
Setpoint = schedulability bound

Feedforward to handle mode changes

26

Feedback Control Real-Time Scheduling

Stankovic et al (Univ of Virginia)



27

© Lund University 2006

Feedback Control Real-Time Scheduling

EDF scheduler in combination with PID controller

PID that controls the task deadline miss ratio

- setpoint values = 0

The control signal (u) is the total amount of CPU load that should be added to or removed from the system

Two actuators:

- Service Level Controller: adjusts the service levels (execution time demands) of the accepted tasks
- Admission Controller (AC): decides if a new task should be admitted to the system or not

28

© Lund University 2006

Session Outline

1. Overview
2. **Some General Observations**
3. Control of Web servers
4. Feedback Scheduling of Controllers

General Observations

The plant under control rarely have any real dynamics or only very simple dynamics

- static nonlinearities + time delays (possibly time-varying)
- first or (maybe) second-order dynamics

Dynamics introduced through the sensors

- Time averages

Event-based control seems a more natural approach than time-based (though very few try to apply it)

General Observations

Seldom any measurement noise

- high gain feedback a possibility

Decentralized control in communication with local and global constraints

- control the resource allocation of tasks or jobs
- local minimum constraints
- global maximum constraints
 - schedulability conditions
 - total available amount of resource limited (e.g. power)

General Observations

Much to learn from control engineering

- Control principles
- Model-based design
- Optimization-based design

Simple controllers often enough

- P, I, PI + feedforward, PD
- anti-windup to achieve good performance

Lack of first principles knowledge that can be used to derive models

- queuing systems an exception
 - however, the models here are averages over long horizons
 - how use these for control?
- models often derived from input-output data

General Observations

So far, primarily applications of classical linear and non-linear time-driven control

It could be expected that there is room for special control theory developed to better fit these types of application

The interest for this area is currently higher in the computer community than in the control community (unfortunately)

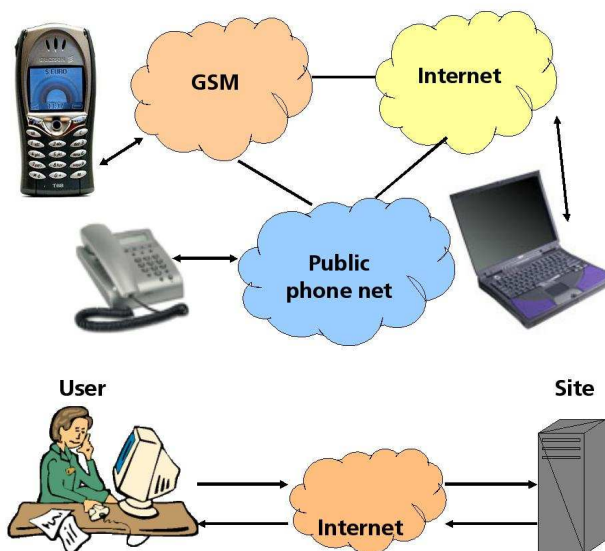
Session Outline

1. Overview
2. Some General Observations
3. [Control of Web servers](#)
4. Feedback Scheduling of Controllers

Outline

- Introduction
- Problem Formulation
- Queuing Model Based Absolute Delay Control
 - L. Sha, X. Liu, UIUC and Y. Lu, T. Abdelzaher, UVa
- Improved Feed-forward Prediction
 - Y. Lu, T. Abdelzaher, UVa and D. Henriksson, LTH
- Analysis and Design of Admission Controllers
 - A. Robertsson, B. Wittenmark, M. Kihl, LTH
 - Not covered in the lectures. See papers

Control at Different Levels



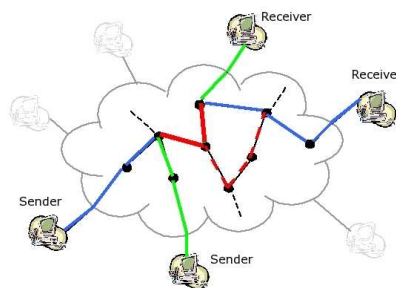
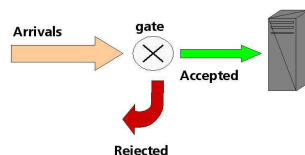
Controlling Computer Systems

- Feedback control is embedded in the TCP protocol in the form of a sliding window mechanism.
- Introduced in the 70's to solve the congestive failure problems that had brought down the network.
- We have not experienced system-wide congestive failures again even though the network has grown orders of magnitude.
- This is a testament of the effectiveness of feedback control in a highly dynamic, decentralized, and fast changing environment.
- Can feedback control be applied to accurately control the performance of web server systems?

37

What to Control?

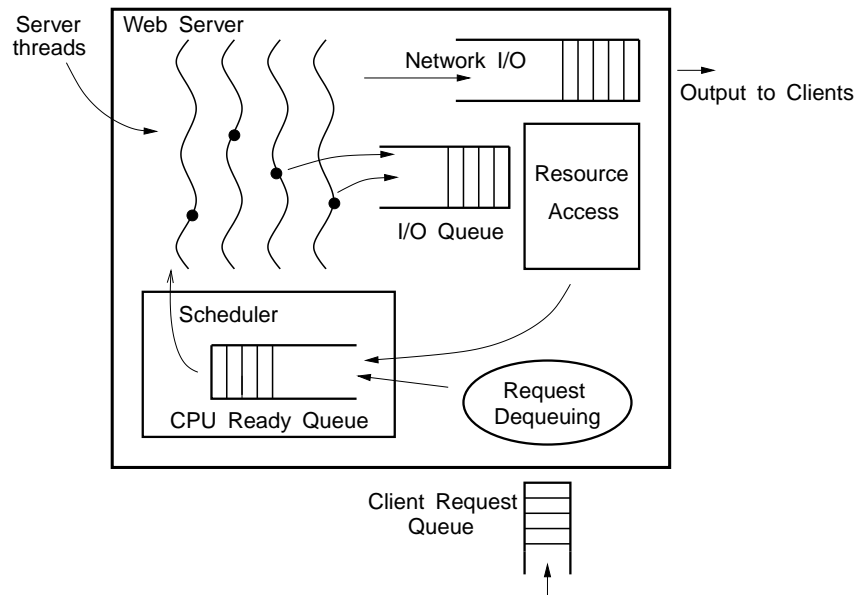
- Temporal
 - local (at server)
 - global (End-to-End/TCP)
- Spatial (routing)



We will focus on temporal control issues at the server.

38

Web Service Performance Control



39

Difficulties

- Web server systems are stochastic with highly non-linear behavior.
 - Response times increase exponentially with utilization at heavy load.
 - Input and output saturations.
- The parameters of the stochastic process, e.g. arrival rate, can change abruptly without warning.
- How should the server system be modeled?
- What is the control objective?
- How can we influence the system, i.e., which actuators are available?

40

Web Server Modeling

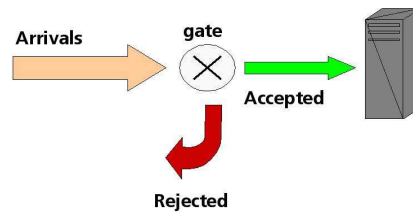
- Queuing theory models:
 - Discrete-event models
 - Markov chains
- Control theory models:
 - Non-linear flow models (continuous time)
 - Discrete-time models
- Differential (or difference) equation models traditionally used in control theory have their limitations.
- Works well in the case of heavy workload when the web server can be modeled using fluid approximations.

Control Objective

- The main objective is to control the service delay of individual requests.
- Can be controlled directly or indirectly by manipulating the server queue lengths.
- The stochastic nature of the system requires averaging (inherent in the non-linear flow model).
- Want to be able to control both long-term averages and transient responses.

Actuator Mechanisms

- The difference between the service rate, μ , and the arrival rate, λ , determines the delay experienced by the requests.
- Changing the arrival rate, admission control:



- Changing the service rate:
 - Number of server threads
 - Quality adaptation
 - Dynamic voltage scaling

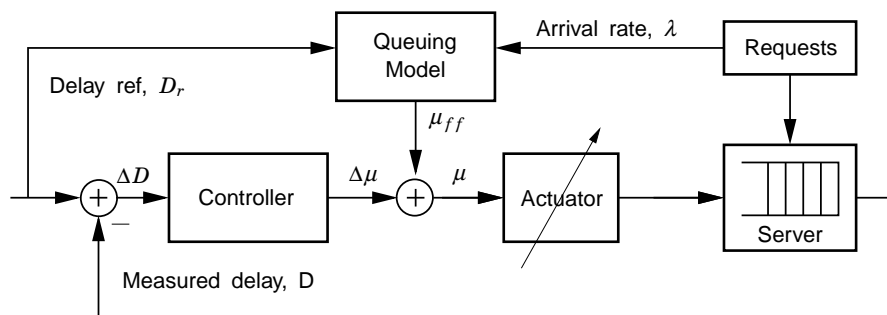
Outline

- Introduction
- Problem Formulation
- **Queuing Model Based Absolute Delay Control**
 - L. Sha, X. Liu, UIUC and Y. Lu, T. Abdelzaher, UVA
- **Improved Feed-forward Prediction**
 - Y. Lu, T. Abdelzaher, UVA and D. Henriksson, LTH
- **Analysis and Design of Admission Controllers**
 - A. Robertsson, B. Wittenmark, M. Kihl, LTH

Control Objective

- Want to keep the average timing delay experienced by users close to a desired value, D_r .
- The delay specification, D_r , relates to the QoS agreement with the end user.
- Delays consistently longer than the specification are unacceptable to the users,
- and delays consistently shorter than the specification indicate over-provisioning of resources.

Absolute Delay Control



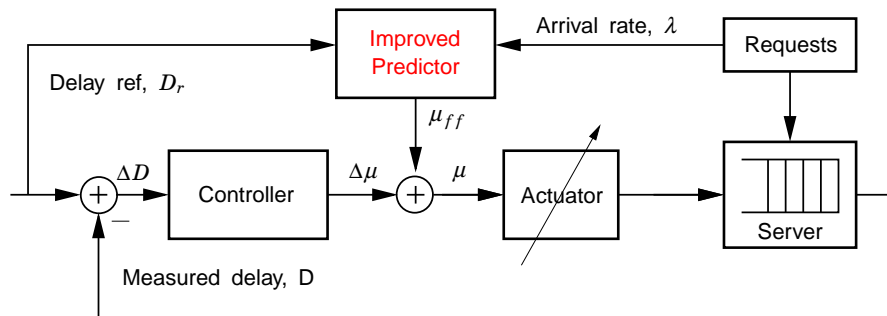
Key Ideas

- Use queuing theory to model the non-linear behavior of the web server.
- Use the steady-state solution of the queuing model as feed-forward control to bring the system to an equilibrium point near the desired delay set-point.
- Example: M/M/1 queuing model where $\hat{D} = \frac{1}{\mu - \lambda}$. Use feed-forward control, $\mu_{ff} = \frac{1}{D_r} + \lambda$.
- Use linear feedback control to suppress approximation errors and transient errors around the operating point.

Problems

- Queuing theory predicts delay as a function of arrival and service rates.
- The prediction applies only to long-term averages.
- Insensitive to sudden load changes and does not handle transient responses very well.
- Internet load is very bursty and may change abruptly in a frequent manner.
- Inaccurate assumptions in the queuing model, e.g., Poisson distributed arrival and service processes.

Improved Feed-forward Predictor



- Based on instantaneous measurements instead of long-term averages.

Notation

\hat{C} = average number of processor cycles required by a request

μ = server speed

N = number of waiting requests

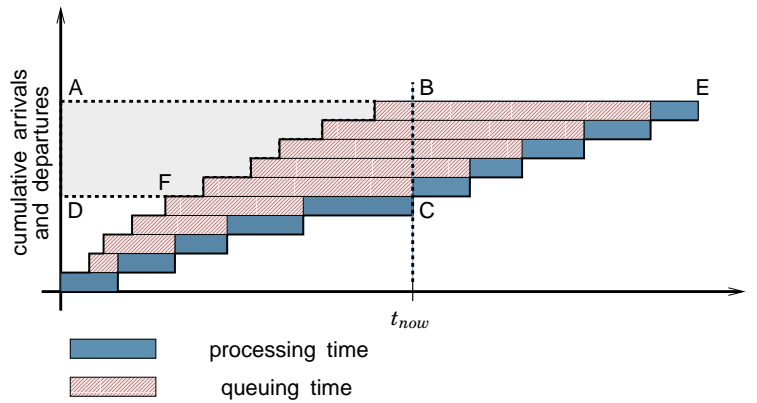
\hat{D} = average delay experience by the N requests

$N\hat{D}$ = total delay experienced by the N requests

$\hat{A}_i = \frac{1}{N} \sum_{k=i}^{i+N-1} A_k$ = the average arrival time

$Q_i = t_{now} - \hat{A}_i$ = average queuing time for the requests being dequeued in the i 'th sample

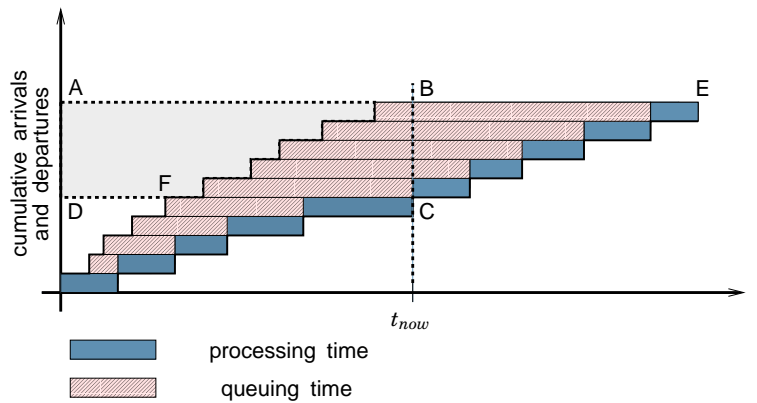
The Predictor



$$BECF = ABCD + BEC - ABFD$$

$$N\hat{D} = N \cdot t_{now} + \frac{N \cdot (N\hat{C}/\mu)}{2} - \sum_{k=i}^{i+N-1} A_k$$

The Predictor



$$\hat{D} = t_{now} - \hat{A} + \frac{N\hat{C}}{2\mu} = \hat{Q} + \frac{N\hat{C}}{2\mu}$$

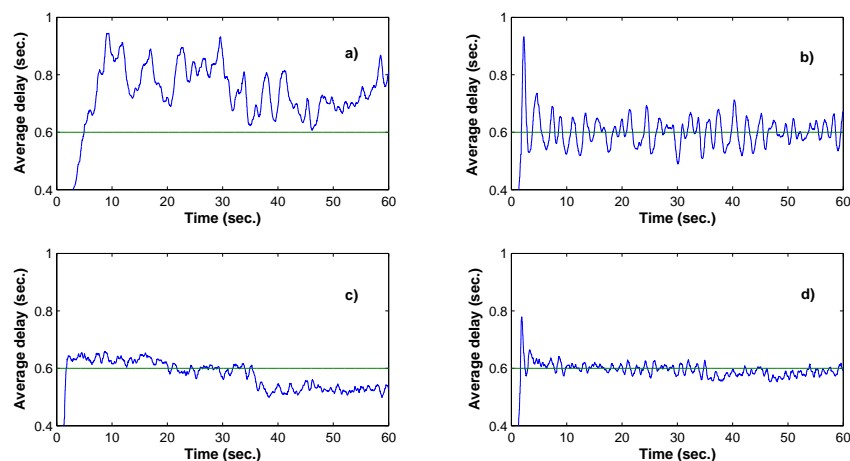
$$\mu_{ff} = \frac{N\hat{C}}{2(D_r - \hat{Q})}$$

The Feedback Controller

- Event-triggered PI-controller with sliding window action.
- Need a long observation window, N_{obs} , to accurately estimate the average values of arrival rates and processing times of requests.
- Long observation window does not imply slow control action. Control updated every $N < N_{obs}$ event (request departure).
- Quick update steps reduce the variance and control efforts in each sample.
- The PI-controller is implemented using gain-scheduling
 - tuned for different operating points (arrival rate and delay set-point, D_r).
- Anti-windup crucial.

53

Simulations



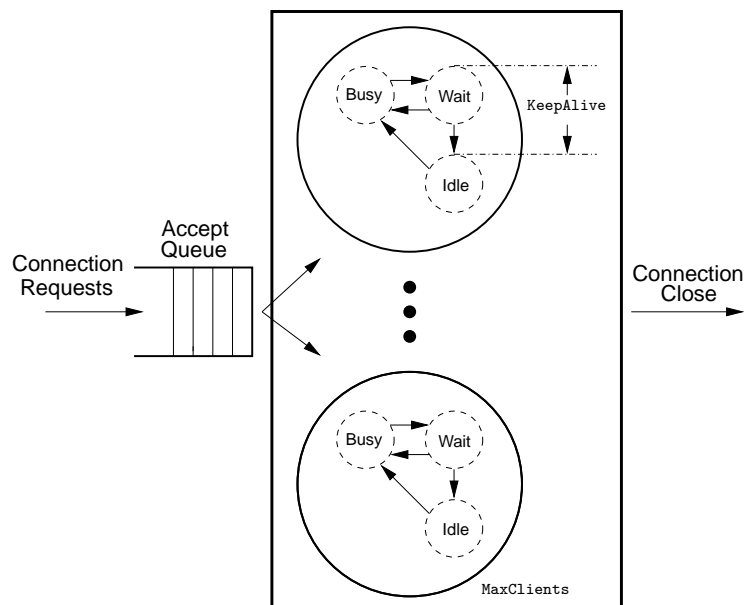
- Performed using the TrueTime simulator.
- **a:** M/M/1, **b:** M/M/1 + PI, **c:** predictor, **d:** predictor + PI

54

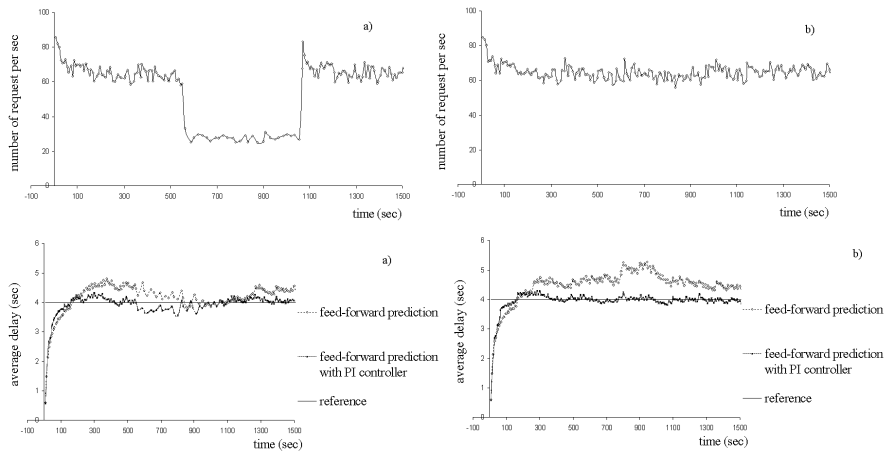
Experiments

- Performed at an Apache web server test-bed at the University of Virginia.
 - Sensor: average response time of incoming requests
 - Actuator: number of server processes
- Load generation: Scalable URL Reference Generator (SURGE). <http://www.cs.bu.edu/faculty/crovella/links.html>
- Platform: Linux-based PC cluster on 100 Mbit Ethernet.
- 4 machines of which one ran the server with HTTP 1.1, and the rest ran clients to stress the server.

Apache Web Server Session Flow



Results



57

Outline

- Introduction
- Problem Formulation
- Queuing Model Based Absolute Delay Control
 - L. Sha, X. Liu, UIUC and Y. Lu, T. Abdelzaher, UVA
- Improved Feed-forward Prediction
 - Y. Lu, T. Abdelzaher, UVA and D. Henriksson, LTH
- **Analysis and Design of Admission Controllers**
 - A. Robertsson, B. Wittenmark, M. Kihl, LTH

58

Overview

Work being done in collaboration between the control and telecommunication departments in Lund

Admission control-based approach

Based on the Tipper's non-linear flow model, shown previously

Non-linear analysis taking actuator saturation and queue length constraint into account

Resulting controller based on PI-control with anti-reset windup

Session Outline

1. Overview
2. Some General Observations
3. Control of Web servers
4. **Feedback Scheduling of Control Task Utilization**

Resource Allocation as a Control problem

In applications with multiple tasks or jobs the dynamic allocation of resources to the tasks can be viewed as a control problem in itself

Use feedback as a technique for mastering uncertainty and guaranteeing performance

Feedback Scheduling

Dynamic on-line allocation of computing resources

Feedback from actual resource utilization

In principle, any computing resource

Here,

- Scheduling of the execution of real-time tasks
- In particular, real-time controller tasks

Optimal Feedback Control

Mostly ad-hoc approaches

Adjust sampling periods and/or execution time demands so that the task set is schedulable

In control it is important to take the application performance into account

- E.g. adjust scheduling parameters in such a way that the global performance is optimized

Requires performance metrics that are properly parameterized

- E.g. quadratic cost functions (Jitterbug)

Optimal Period Assignment

Assume that the performance of each controller i can be described by a cost function $J_i(x_i, h_i, T_{FBS})$

- x_i – the current state of plant i
- h_i – the sampling period of controller i
- T_{FBS} – the optimization horizon of the feedback scheduler

The objective is to minimize the combined performance with respect to the utilization bound:

$$\min_{h_1 \dots h_n} \sum_{i=1}^n J_i(x_i, h_i, T_{fbs})$$

$$\text{subj. to } \sum_{i=1}^n \frac{C_i}{h_i} \leq U_{sp}$$

Optimal Period Assignment, cont'd

- Convex problem if functions $J_i(x_i, 1/f_i, T_{FBS})$ convex in f_i .
- Explicit solution if all cost functions have the same shape,

$$J_i = \alpha_i + \beta_i h_i^\nu$$

- $\nu = 1$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/2} \frac{\sum_{j=1}^n (C_j \beta_j)^{1/2}}{U_{sp}}$$

- $\nu = 2$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/3} \frac{\sum_{j=1}^n C_j^{2/3} \beta_j^{1/3}}{U_{sp}}$$

- Linear cost functions ($\nu = 1$) are often good approximations

Linear-Quadratic Controllers

The cost function for an LQ controller is given by

$$J(x, h, T_{fbs}) = x^T S(h) x + \frac{T_{fbs}}{h} \left(\text{tr} S(h) R_1(h) + J_v(h) \right)$$

- $S(h)$ – solution to the LQ Riccati equation
- $R_1(h)$ – sampled process noise variance
- $J_v(h)$ – inter-sample cost term

Example: Integrator Process

- Process: $dx = u dt + dv_c$
 - v_c – Wiener process with unit incremental variance
- Design cost function: $J = \int_0^{T_{fbs}} x^2(t) dt$
- Resulting cost:

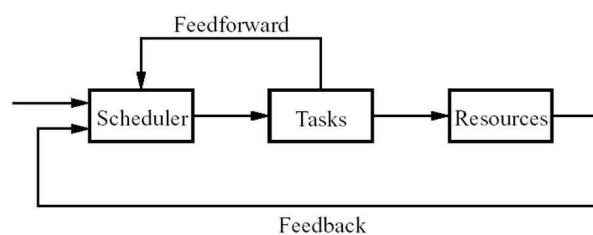
$$J(x, h, T_{fbs}) = \left(x^2 \frac{\sqrt{3}}{6} + T_{fbs} \frac{\sqrt{3} + 3}{6} \right) h$$

- Linear in h
- Explicit solution for multiple controllers:

$$h_i \propto \sqrt{\frac{C_i}{x_i^2 + T_{fbs}(1 + \sqrt{3})}}$$

67

Feedback Scheduling Structures



Feedback

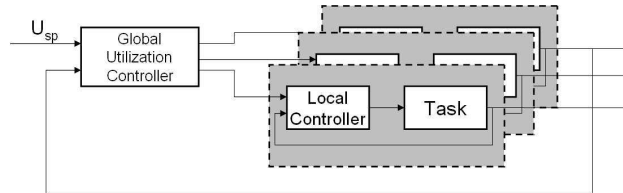
- Reactive

Feedforward

- Proactive
- Mode changes and admission control

68

Feedback Scheduling Structures



Cascaded/Layered Structure:

- Global utilization controller that outputs the desired utilization share for each task
- Local controllers that adjust the task parameters accordingly
- Combine with reservation-based scheduling to provide temporal protection, cp. the Control Server

69

Indirect Feedback Scheduling

Most proposed approaches are indirect

By adjusting the scheduling parameters (T,D,C) one makes sure that the task set is schedulable

The scheduling parameters determine the timing attributes (latency, jitter) which in turn determine the control performance

Problem:

- Complex, nonlinear relationship between scheduling parameters and timing attributes
- Non-trivial relationship between timing attributes and control performance

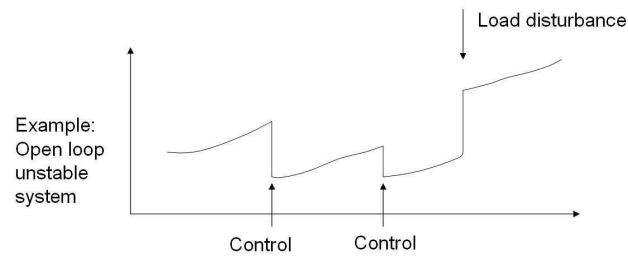
70

Direct Feedback Control

Use the instantaneous cost as a dynamic task priority

Fast sampling and cost evaluation (hardware/interrupt handler)

Always execute the task with the highest instantaneous cost



71

Problems

Event-based sampling / aperiodic system

Very little theory available

The sampling operation is no longer linear

72

Feedback Scheduling Actuators

Two main actuators:

- Changing the task periods
- Changing the allowed execution times

Task periods:

- Easy for simple controllers, e.g. PID & state feedback
- More difficult for complex controllers
- Update the internal state of the controller appropriately

Execution times:

- Not applicable to most controllers

73

Anytime Controllers

Controllers where the quality of the control signal is gradually refined the more time that is available

Model-based Predictive Control (MPC)

- On-line convex optimization problem solved each sample
- Highly varying execution times
- For fast processes the latency may effect the control performance considerably
- The control algorithm is based on a quality-of-service type cost measure, cp instantaneous cost
- As long as a “feasible control signal” has been found the iterative search can be aborted before it has reached completion
- Maps well to the imprecise task model
 - Mandatory part
 - Optional part

74

Quality-of-Control

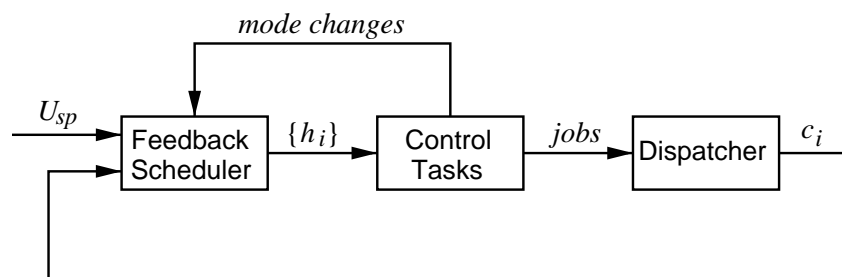
The control performance can in many cases be viewed as a quality-of-service parameter

Several open issues:

- Specification of acceptable performance ranges
- Run-time negotiation methods

Research issue within the FLEXCON project

A Feedback Scheduling Structure



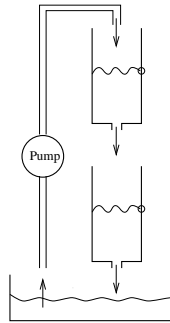
Control system analogy:

- Setpoint: Desired CPU utilization, U_{sp}
- Measurement: Execution times, c_i
- Control: Sampling periods, $\{h_i\}$
- Feedforward: Mode changes

Case Study: Set of Hybrid Controllers

The double-tank process:

Use pump, $u(t)$, to control
level of lower tank, $y(t)$



Hybrid control strategy:

- PID control in steady state
- Optimal control for setpoint changes

77

PID Controller

$$P(t) = K(y_{sp}(t) - y(t))$$

$$I(t) = I(t-h) + a_i(y_{sp}(t) - y(t))$$

$$D(t) = a_d D(t-h) + b_d(y(t-h) - y(t))$$

$$u(t) = P(t) + I(t) + D(t)$$

Average execution time: $C = 2.0$ ms

78

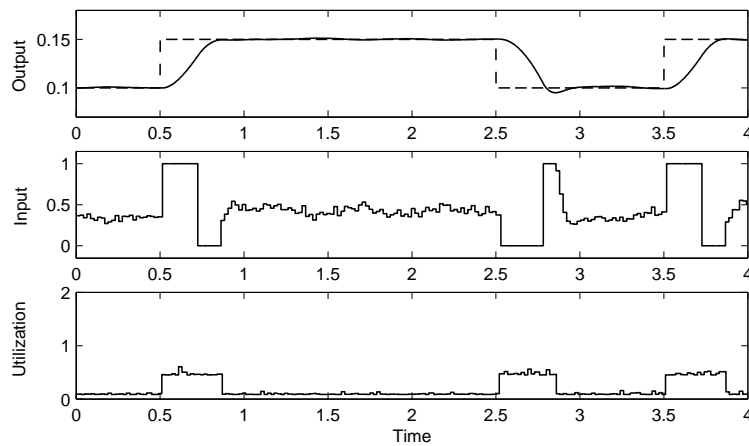
Optimal Controller

$$x_2(x_1) = \frac{1}{a} \left((ax_1 - b\bar{u}) \left(1 + \ln \left(\frac{ax_1^R - b\bar{u}}{ax_1 - b\bar{u}} \right) \right) + b\bar{u} \right)$$

$$V_{close} = \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix}^T P(\theta, \gamma) \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix} + \text{more } \dots$$

Average execution time: $C = 10.0$ ms

Nominal Behavior, $h = 21$ ms



Scheduling Experiments

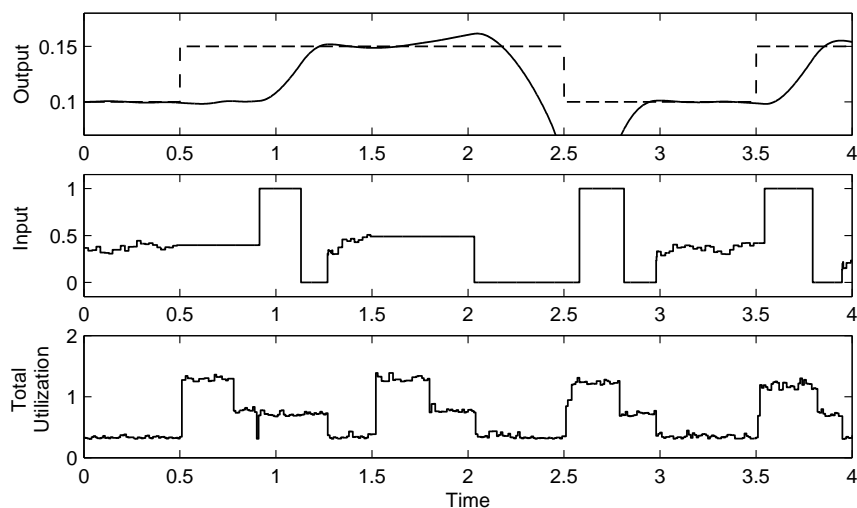
- Three hybrid controllers execute on one CPU
- Nominal sampling periods: $(h_1, h_2, h_3) = (21, 18, 15)$ ms
- Potential problem: All controllers in Optimal mode \Rightarrow
$$U = \sum \frac{C}{h} = 170\%$$

Compare strategies:

1. Open-loop scheduling
 2. Feedback scheduling
 3. Feedback + feedforward scheduling
- Co-simulation of scheduler, controllers, and double tanks
 - Focus on the lowest-priority controller

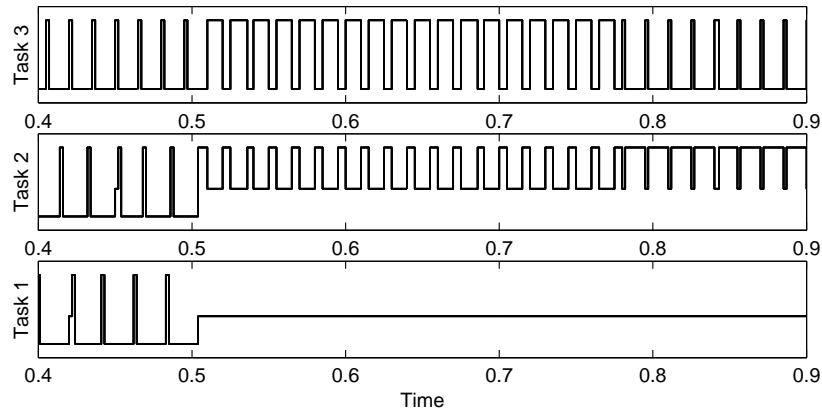
81

Open-Loop Scheduling



82

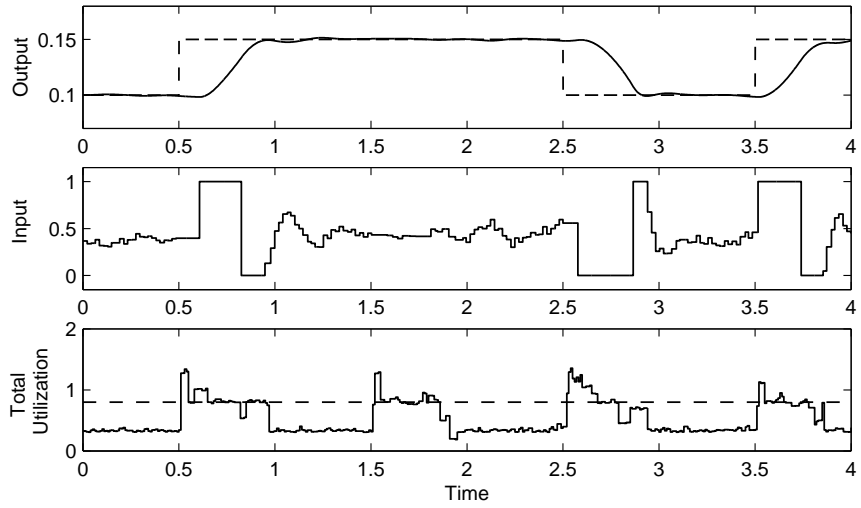
Open-Loop Scheduling



Feedback Scheduler

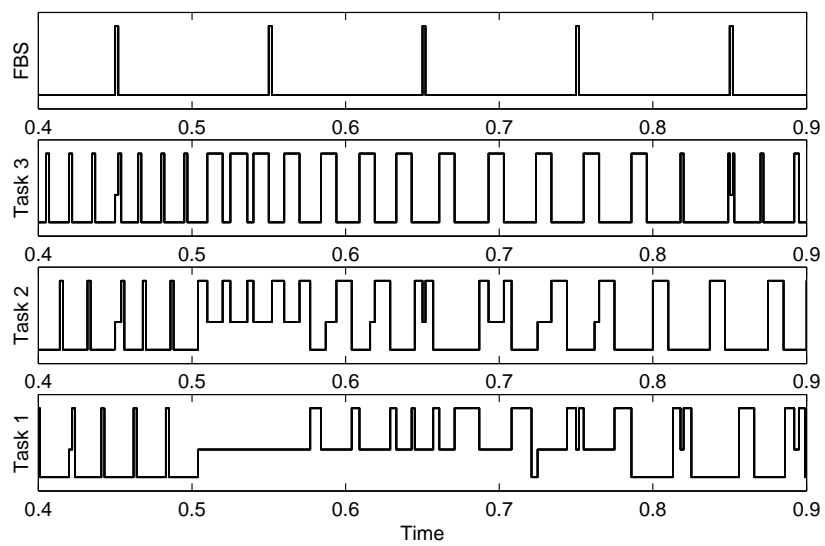
- A high-priority task, $T_{FBS} = 100$ ms, $C_{FBS} = 2$ ms
- Setpoint: $U_{sp} = 80\%$
- Estimate execution times using first-order filters
- Control U by adjusting the sampling periods
 - Simple linear rescaling of $\{h_i\}$ such that $U = U_{sp}$
 - Non-optimal
 - No regard for current plant states

Feedback Scheduling



85

Feedback Scheduling

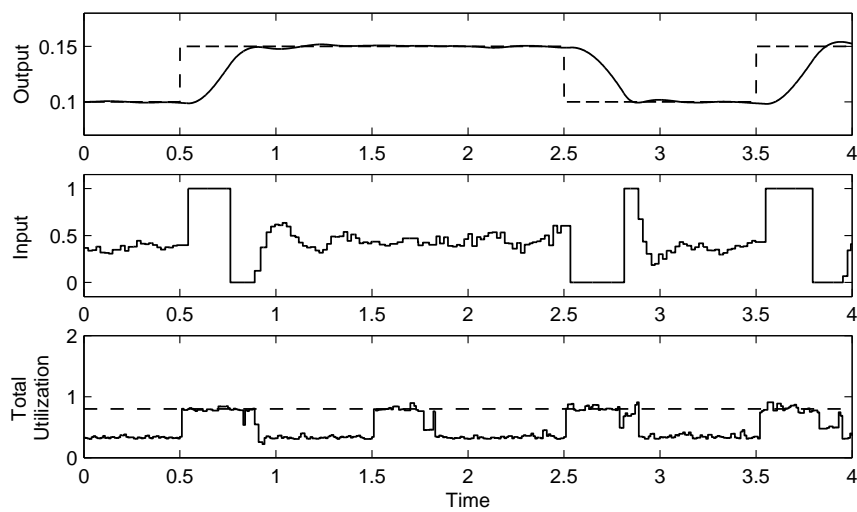


86

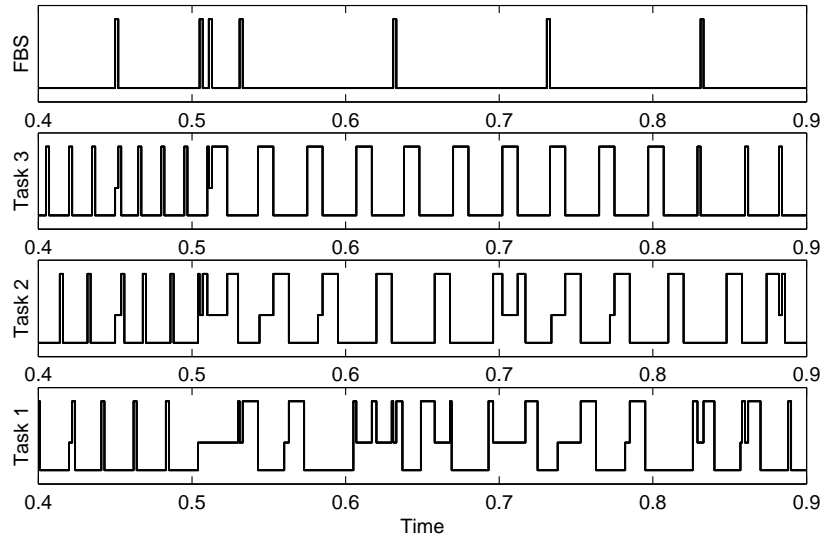
Feedforward

- Controller notifies feedback scheduler when switching from PID to Optimal mode
- Scheduler is released immediately

Feedback + Feedforward Scheduling

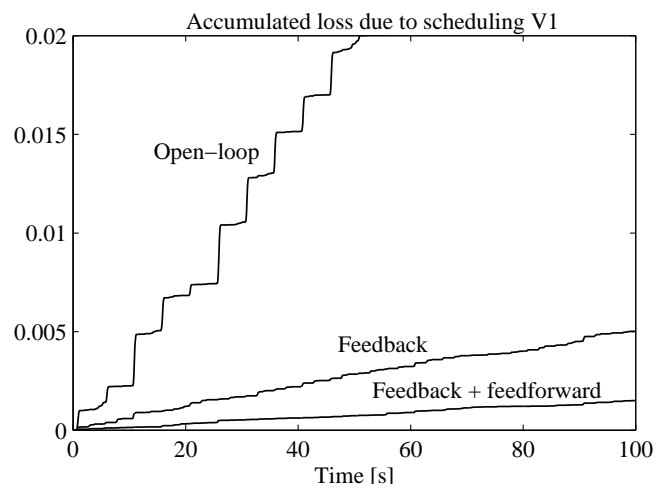


Feedback + Feedforward Scheduling



89

Control Performance Evaluation



90

ARTIST2

Embedded Control Systems: TrueTime & Jitterbug

Anton Cervin and Karl-Erik Årzén

Department of Automatic Control

Lund University

Sweden

{anton,karlerik}@control.lth.se



1

Session Outline

- TrueTime introduction
- TrueTime laboratory session
- Jitterbug overview & demonstration

2

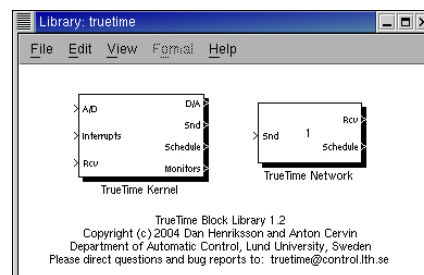
TrueTime – Main Idea

Co-simulation of controller task execution, network transmissions, and continuous plant dynamics.

The approach enables us to:

- investigate the true, timely behavior of control loops
- develop dynamic compensation schemes
- experiment with flexible scheduling techniques
- simulate event-based and networked control loops

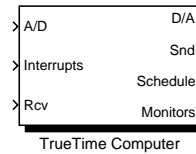
Simulink Blocks



- Offers a *Kernel* and a *Network* block
 - Simulink S-functions written in C++
 - Event-based implementation using the Simulink built-in zero-crossing detection

The Kernel Block

- Simulates an event-based real-time kernel
- Executes user-defined tasks and interrupt handlers
- Arbitrary user-defined scheduling policy
- Supports external interrupts and timers
- Supports common real-time primitives (sleepUntil, wait/notify, setPriority, etc.)
- Generates a task activation graph
- More features: context switches, overrun handlers, task synchronization, data logging



5

Tasks

- Used to model the execution of user code (mainly control algorithms)
- Tasks may be periodic or aperiodic and are triggered by creation of jobs
- For periodic tasks the kernel sets up an internal timer to periodically create task jobs
- Each task is described by a number of (static and dynamic) task attributes and a code function

```
ttCreateTask(name, deadline, priority, codeFcn, data)
ttCreateJob(taskname)
ttKillJob(taskname)
ttCreatePeriodicTask(name, offset, period, prio, codeFcn, data)
```

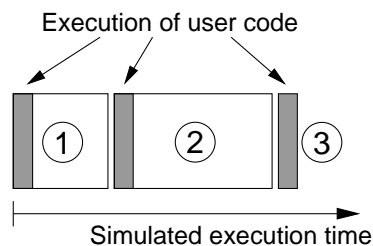
6

TrueTime Code

Three choices:

- C++ code (fast)
- Matlab code (medium)
- Simulink block diagram (slow)

Code Execution Model



- Execution is modeled by a code function (C++ or MATLAB m-file) consisting of a sequence of segments
- The execution time of each segment is returned by the code function (may be data-dependent, random, etc.)
- User code executed at the beginning of each segment
- The task can only interact with other tasks and the environment at the beginning of each code segment

Example of a Code Function

```
function [exectime, data] = P_Ctrl(segment, data)
switch (segment),
    case 1,
        r = ttAnalogIn(1); % Read reference
        y = ttAnalogIn(2); % Read process output
        data.u = data.K * (r-y); % Compute and store control
                                % signal in task data
        exectime = 0.002; % Return execution time
    case 2,
        ttAnalogOut(1, data.u); % Output control signal
        exectime = 0.001;
    case 3,
        exectime = -1; % finished
end
```

9

Initialization

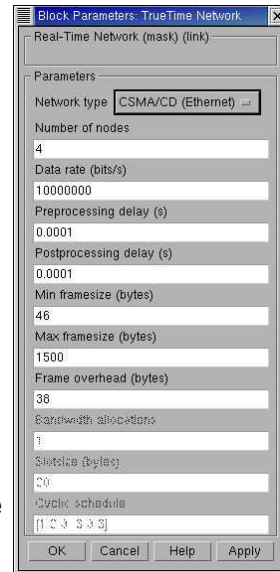
Each kernel block is initialized in a script (block parameter):

```
nbrInputs = 3;
nbrOutputs = 3;
ttInitKernel(nbrInputs, nbrOutputs, 'prioFP');
periods = [0.01 0.02 0.04];
code = 'myCtrl';
for k = 1:3
    data.u = 0;
    taskname = ['Task ' num2str(k)];
    offset = 0; % Release task at time 0
    period = periods(k);
    prio = k;
    ttCreatePeriodicTask(taskname, offset, period, prio, code, data);
end
```

10

The Network Block

- Supports six common MAC layer policies:
 - CSMA/CD (Ethernet)
 - CSMA/AMP (CAN)
 - Token-based
 - FDMA
 - TDMA
 - Switched Ethernet
- Variable network parameters
- Generates a transmission schedule



11

Implementation Details

- TrueTime implements a complete real-time kernel with
 - A ready queue for tasks ready to execute
 - A time queue for tasks waiting to be released
 - Waiting queues for monitors and events
- Queues manipulated by the kernel or by calls to kernel primitives
- The simulated kernel is ideal (no interrupt latency and no execution time associated with real-time primitives)
 - However, possible to specify a constant context switch overhead

12

Scheduling Policy

- The scheduling policy of the kernel is defined by a priority function, which is a function of task attributes
- Pre-defined priority functions exist for fixed-priority, rate-monotonic, deadline-monotonic, and earliest-deadline-first scheduling
- EDF priority function

```
double prioEDF(UserTask* t)
    return t->absDeadline;
}
```

```
void ttAttachPrioFcn(double (*prioFcn)(UserTask*))
```

13

Scheduling Hooks

- Code that is executed at different stages during the execution of a task
 - Arrival hook – executed when a job is created
 - Release hook – executed when the job is first inserted in the ready queue
 - Start hook – executed when the task executes its first segment
 - Suspend hook – executed when the task is preempted, blocked or voluntarily goes to sleep
 - Resume hook – executed when the task resumes execution
 - Finish hook – executed after the last code segment
- Facilitates implementation of arbitrary scheduling, such as, e.g, server-based scheduling

```
ttAttachHook(char* taskname, int ID, void (*hook)(UserTask*))
```

14

Overrun Handlers

- Two special interrupt handlers may be associated with each task (cf. Real-Time Java)
 - A deadline overrun handler
 - An execution time overrun handler
- Can be used to dynamically handle prolonged computations and missed deadlines
- Implemented by internal timers and default scheduling hooks

```
ttAttachDLHandler(taskname, hdlname)  
ttAttachWCETHandler(taskname, hdlname)
```

15

A Real-World Application

- Multiple processors and networks
- Based on VxWorks and IBM Rational Rose RT
- Using TrueTime to describe timing behavior
- Has ported TrueTime to a mechatronics simulation environment

Embedded Systems
INSTITUTE



"We found TrueTime to be a great tool for describing the timing behavior in a straightforward way."

16

TrueTime Limitations

- The network block only supports data-link layer protocols
 - TCP transport protocol available as an example
- A global clock (= simulation time)
 - No support for local clocks with offsets and drift
- How decide the execution times?
 - Integration with compiler/WCET tools?
- How integrate legacy code?
- How support automatic code generation from TrueTime models?
 - Generate POSIX-thread compatible code?
 - Generate monolithic code (“TrueTime Virtual Machine”)?
- Based on Matlab/Simulink

17

More Material

- The toolbox (TrueTime 1.2) together with a complete reference manual can be downloaded at:

<http://www.control.lth.se/~dan/truetime/>

18

Session Outline

- TrueTime Introduction
- **TrueTime Laboratory Session**
- Jitterbug Overview & Demonstration

Example: PID-control of a DC-servo

- Intended to give a basic introduction to the TrueTime simulation environment
- Consists of a single controller task implementing a standard PID-controller
- Continuous-time process dynamics

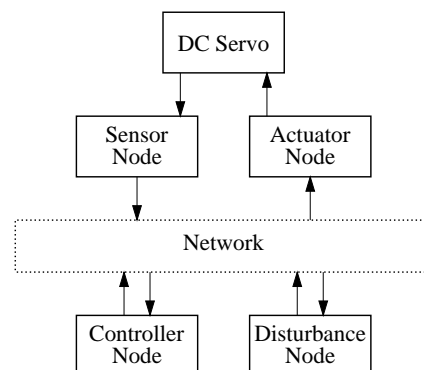
$$G(s) = \frac{1000}{s(s+1)}$$

- Can evaluate the effect of sampling period and input-output latency on control performance

Example: Three Controllers on one CPU

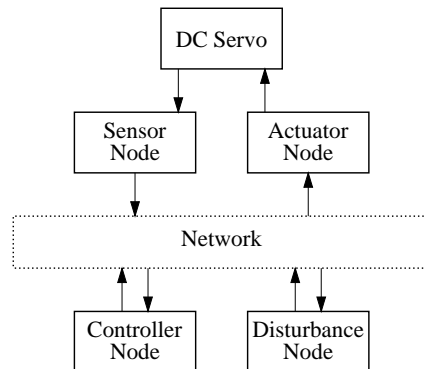
- Three controller tasks controlling three different DC-servo processes
- Sampling periods $h_i = [0.006 \ 0.005 \ 0.004]$ sec.
- Execution time of 0.002 sec. for all three tasks for a total utilization of $U = 1.23$
- Possible to evaluate the effect of the scheduling policy on the control performance
- Can use the logging functionality to monitor the response times and sampling latency under the different scheduling schemes (connection to Jitterbug)

Example: Networked Control System



- Time-driven sensor node
- Event-driven controller node
- Event-driven actuator node
- Disturbance node generating high-priority traffic

Example: Networked Control System



- Will try changing the bandwidth occupied by the disturbance node
- Possible to experiment with different network protocols and network parameters
- Can also add a high-priority task to the controller node

23

Session Outline

- TrueTime Introduction
- TrueTime Laboratory Session
- **Jitterbug Overview & Demonstration**

24

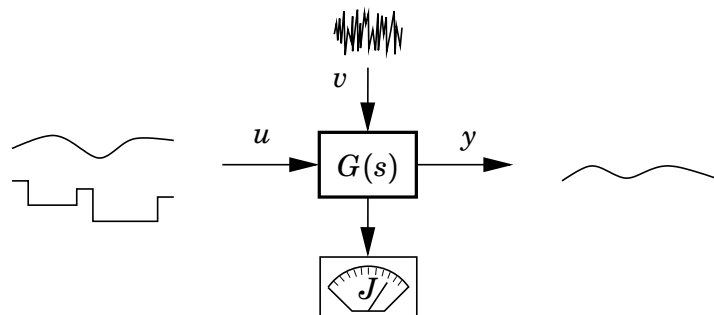
Jitterbug

- MATLAB toolbox
- Stochastic analysis of (mean) control performance
- Control loop described by connected continuous-time and discrete-time linear systems
- Execution of discrete-time systems described by stochastic timing model (random delays)
- Systems driven by white noise
- Performance measured by quadratic cost function

$$J = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T x^T(t) Q x(t) dt$$

25

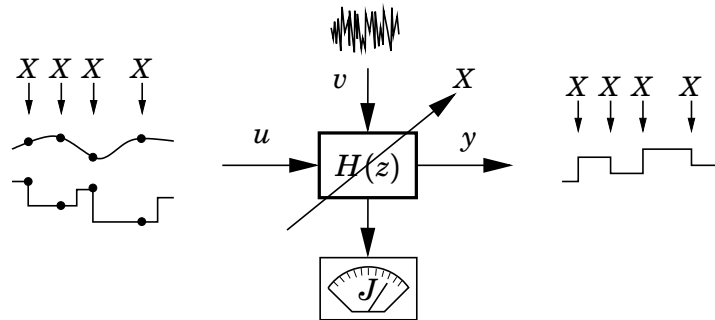
Continuous-time systems



- Continuous or discrete input u
- Continuous output y
- Continuous white noise v
- Continuous cost J

26

Discrete-time systems

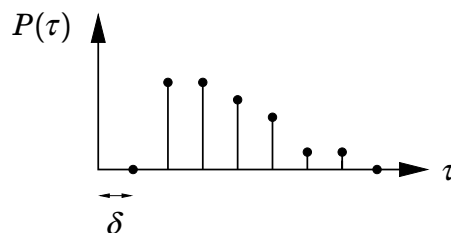
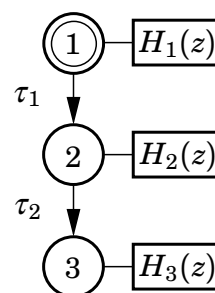


- Continuous or discrete input u , sampled at instants X
- Discrete output y , updated at instants X
- Discrete white noise v , updated at instants X
- Continuous cost J

27

Timing Model

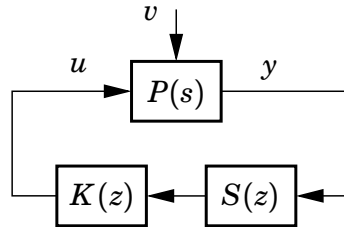
- Describes the timing of the triggering of the discrete systems
- Each node can trigger one or more systems
- The first node may be periodic or aperiodic
- Delays between nodes described by discrete probability distributions



28

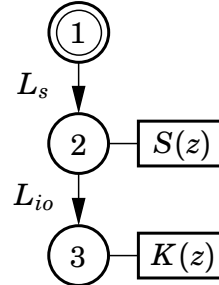
Jitterbug Model – Example

Signal model:



- $P(s)$ – process
- $S(z)$ – sampler
- $K(z)$ – controller/actuator

Timing model:



- L_s – sampling latency distribution
- L_{io} – input-output latency distribution

29

Jitterbug example script

```
Ptau1 = 1;
Ptau2 = [zeros(1,round(L/dt)) 1];
N = initjitterbug(dt,h);
% timegrain dt, periodic system with period h
N = addtimingnode(N,1,Ptau1,2);
% add timing node with delay distribution to next node
N = addtimingnode(N,2,Ptau2,3);
N = addtimingnode(N,3);
% add timing node with no next node
```

30

```

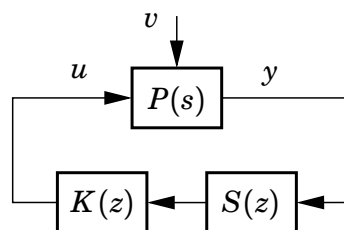
N = addcontsys(N,1,plant,3,Q,R1,R2);
% add cont-time LTI system taking its input from syst 3
N = adddiscsys(N,2,1,1,2);
% add disc-time LTI system (sampler) taking its input
% from system 1 and executing in timing node 2
N = adddiscsys(N,3,ctrl,2,3);
% add disc-time LTI system (controller) taking its
% input from system 2 and executing in timing node 3
N = calcdynamics(N);
% Calculate internal dynamics
J = calccost(N)
% Calculate (and display) cost

```

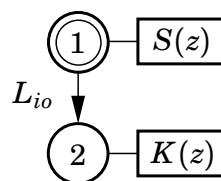
31

Example – “simple”

Signal model:



Timing model:



- $P(s) = \frac{1}{s^2 - 1}$ (inverted pendulum)
- $S(z) = 1$
- $K(z) = \text{lqgdesign}(P, \dots)$

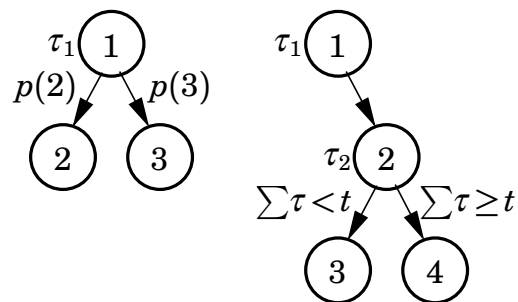
32

Example – “simple”

Investigate performance with

- different sampling periods
- constant I-O latency with/without delay comp.
- I-O jitter with/without delay comp.

More Complicated Cases



- random choice of path
- choice of path depending on delay
- different update equations in different nodes
- aperiodic systems

Pros and cons of Jitterbug

Pros:

- Analytical performance computations
- Fast to evaluate cost for wide range of parameters
- Guarantees stability (in mean-square sense) if $J < \infty$

Cons:

- Simplistic timing models
- Only linear systems, quadratic cost
- Hard to know the latency distributions
- Cannot handle dependencies between periods
- Based on Matlab

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

ARTIST2

Embedded Control Systems: Deployment

M. Törngren and B. Eriksson
 Division of Mechatronics, Dept. of Machine Design
 KTH - Royal Institute of Technology, Stockholm
 www.md.kth.se e-mail: martin@md.kth.se



KTH Industrial Engineering and Management

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Deployment

- Pronunciation: di-'ploi,
- Function: *verb*
- Etymology: French *déployer*, literally, to unfold, from Old French *desploier*, from *des-* *dis-* + *ploiier*, *plier* to fold
- Uses:
 - to extend (a military unit) especially in width
 - to place in battle formation or appropriate positions
 - to spread out, utilize, or arrange especially strategically

Here: Used to in the context of ECS “implementation”

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- **ECS implementation issues and trade-offs**
 - Technical issues in controller implementation, trade-off examples
 - Platform selection: Choice of RTOS and processor
- **Development approaches:**
 - Traditional, CBD, MBD, platform based
 - Tool support for deployment
- **Application example**
 - Smart-1 spacecraft
- **Concluding Remarks**

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

A sample of technical issues in ECS design and implementation

- Discretization
- Quantization
- Delays
- Jitter in delays and periods
- Aliasing
- Triggering and tasking partitioning, scheduling
- Code implementation
- Sensor and actuator limitations
- Calibration/diagnostics
- Error detection and error handling
- Check lists for well known pitfalls, e.g. RT and parallel programming faults

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Controller implementation and co-design

Requirements $\tau(k), h(k)$ memory, accuracy, reliability, etc. Constraints

Platform SW HW

Optimization issues:

- Distribution, task partitioning
- Code structuring
- Other functionality
- Trade-offs

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Vehicle stability Suspension Diagnostics Driver in the loop

Local control function Local control function

Sensor Actuator Sensor Sensor Actuator

Gateway

Computer hardware & software Drive unit Actuator(s) Signal cond. Sensor(s)

Computer hardware & software Drive unit Actuator(s) Signal cond. Sensor(s)

Computer hardware and software Drive unit Circuitry Actuator(s) Signal cond. Sensor(s)

Vehicle Mechanics

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Mapping related terms

- Assignment in space and in time
 - Computations and communication → resources
 - Scheduling per resource (nodes & communication)
- Partitioning node software into tasks

Affecting (among others)

- Resource loads
- E2e delays
- Single points of failure
- Maintenance

Constrained by

- Organization
- Policies
- Legacy, technology

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

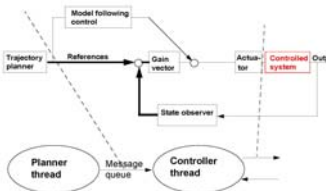
Partitioning example

3/21/2006 ©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Partitioning guidelines and example

<u>Actual partitioning</u>	<u>Example reason</u>
<ul style="list-style-type: none"> • Planner and controller in separate threads • Controller divided into two threads. 	<ul style="list-style-type: none"> (i) Runs at <i>different rates</i>. (ii) Set-points, over network (iii) To ensure constant delay and fixed sampling period <i>~different timing requirements in one activity.</i>
<ul style="list-style-type: none"> • Needs and possibilities also depend on <ul style="list-style-type: none"> - Criticality of functions - Dependencies (coupling and cohesion) - Development/production modularization - Type of operating system & tool support 	



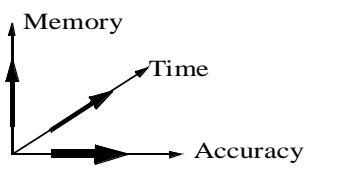
3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

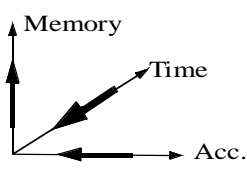
Controller implementation trade-offs

Goal: minimize control delay, provide high accuracy in computations, at a given cost; where is the catch?

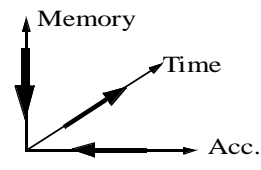
The trade-offs: among them the achievable accuracy, required memory size and execution speed are typically in conflict.



(a) Increase in computational accuracy desired



(b) Increase in speed desired



(c) Scarce memory

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Controller implementation optimization, trade-off examples

- Improved accuracy:
 - better algorithm; typically requires more execution time and memory
- Improved timing:
 - e.g. through simpler algorithms (may result in reduced performance), function in-lining requires more memory
- Less memory usage:
 - e.g. int16 rather than int32; reduced resolution, less accurate results

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Controller code optimization, cont.

- Minimize computational delay by code structuring
 1. Sample
 2. Compute Control Output
 3. Actuate
 4. State update
- Use of hardware specific instructions --> Drawback: maintenance
- Fixed point operations instead of hardware floating point support
 - reduced cost, but reduction in accuracy
- Logging/debugging problem: affects execution time!

Code generation: can facilitate controller implementation optimization

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- ECS implementation issues and trade-offs
 - Technical issues in controller implementation, trade-off examples
 - **Platform selection: Choice of RTOS and processor**
- Development approaches:
 - Traditional, CBD, MBD, platform based
 - Tool support for deployment
- Application example
 - Smart-1 spacecraft
- Concluding Remarks

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Choosing electronics technology

- Off-the shelf control systems, for example,
 - Programmable logic controllers (PLC)
- Own design
 - Analog (low flexibility, only simple functionality)
 - Microprocessor based
 - Customized solution for large series
 - ASIC (Application Specific Integrated Circuit)
 - Flexible programmable hardware
 - FPGA (Field Programmable Gate Array)
- Choosing a microprocessor
 - Micro-controller, DSP, general purpose
 - Fixed point vs. floating point
 - I/O, memory and communication capabilities

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Example trade-offs in choosing processor

Choosing a low cost micro-controller:

- + Cheap microcontroller, low cost production
- + Use on-chip memory, no external circuits, robust solution
- Scarce resources: Little room for later functional extensions
- Can performance requirements be met?
- Increased development (and possibly maintenance) costs

Choosing a highly performing processor:

- + Easier to solve performance and flexibility requirements
- + Reduced development (and possibly maintenance) costs
- Increased production cost
- Power consumption, fan

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Floating point support in hardware?

	Floating point hw	Fixed point hw
Development cost	Lower - facilitated design	Higher - more difficult design; scaling effort, quantization errors
Production cost	Higher	Lower
Maintenance cost	Improved	Higher - more effort
Speed	'High'	Trade-off between accuracy, speed and required memory
Real-time	Care with concurrency handling	Usual considerations

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Selecting the software platform: RTOS

Has to be determined based on the actual requirements at hand!

Technical requirements:

- Static vs. dynamic configuration?
- Single processor, communication centric, I/O-intensive?
- RTOS footprint ~ resources required
- Critical vs. non critical? Error detection & handling by RTOS?

Other criteria:

- Costs - Development, Production, Maintenance
 - e.g. Free-source Linux vs. license cost/system
 - Tools, documentation, competence
- Standards and compatibility?
 - Specific domains may require compliance, e.g. to OSEK
 - Certification requirements

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

RTOS; when, what, how?

- Many categories of RTOS'es – reflecting the broad scope of applications.
- Major benefits of using an RTOS: *reduced cost* and *increased reliability*
- The need for an RTOS increases with the product complexity (amount of activities, types of timing requirements, etc.)
- Many companies developing embedded systems still use proprietary kernels

Is the answer open source or standards?
Research trends instead point towards increased high level configuration and synthesizing the kernel as required

3/21/2006
©M. Törngren 2006





ARTIST2 *Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006*

Outline

- ECS implementation issues and trade-offs
 - Technical issues in controller implementation, trade-off examples
 - Platform selection: Choice of RTOS and processor
- **Development approaches:**
 - **Traditional, CBD, MBD, platform based**
 - Tool support for deployment
- Application example
 - Smart-1 spacecraft
- Concluding Remarks

3/21/2006
©M. Törngren 2006

Different deployment approaches










Broad variety of applications & different requirements:

- from critical to non-critical
- long to short life time, etc.

Room for many methodologies:

- software vs. control engineering
- company traditions
- top-down/bottom-up/platform based

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Static vs. dynamic configuration

- Pre run-time configuration of software and hardware
- Example: A system controlling two motors, no more, no less, configured into e.g. three tasks
- Preferred approach for safety critical systems
- Dependability at the expense of flexibility (modes of operation can increase flexibility)

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Static vs. dynamic configuration

- Pre run-time established infrastructure configuration, dynamically varying mobile users and load at run-time
- Example: A telecom system with 0-1M subscribers, spawning handler tasks during run-time as required.
- Dynamically varying configuration, e.g. hand
- Configuration driven by flexibility requirements

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Top-down, vs. bottom up vs. platform based design

Top-down: Synthesizing an appropriate/optimal system
(but product development seldom starts from scratch)

Bottom-up: Purely reusing components
(reuse can be dangerous; what about system requirements, and required changes?)

Platform based: Finding common architectures that can support a variety of applications – as well as – future evolutions of applications.

One definition: A platform comprises the complete technological base (SW and HW) required to execute an application.
Note: A platform can be seen as a relative concept

In practice the approaches are often combined.

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Traditional implementation

Control design

Software design

Assembly
C/C++

C Compiler

Assembler

Linker

Object Files

Program loader

Application software

Supporting software

Input/output drivers

Hardware

Hierarchy

Microprocessor and/or programmable logic
Memories, input/output, and communication

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Component-based development

A component is a unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently and is subject to third-party composition (Szyperski, 1998).

Component models and infrastructures in use (examples):

- CORBA (telecommunication)
- COM/DCOM, .NET – process industry
- OPC (OLE process control Foundation)

CBD is closely related to the software platform definition.

CBD today supports mainly functionality and run-time flexibility.

Consideration of non-functional requirements (timing, resource consumption, reliability, etc.) is an intense research area

Trends: → Is evolving towards model-based development
→ Domain specific efforts: e.g. AUTOSAR (automotive)

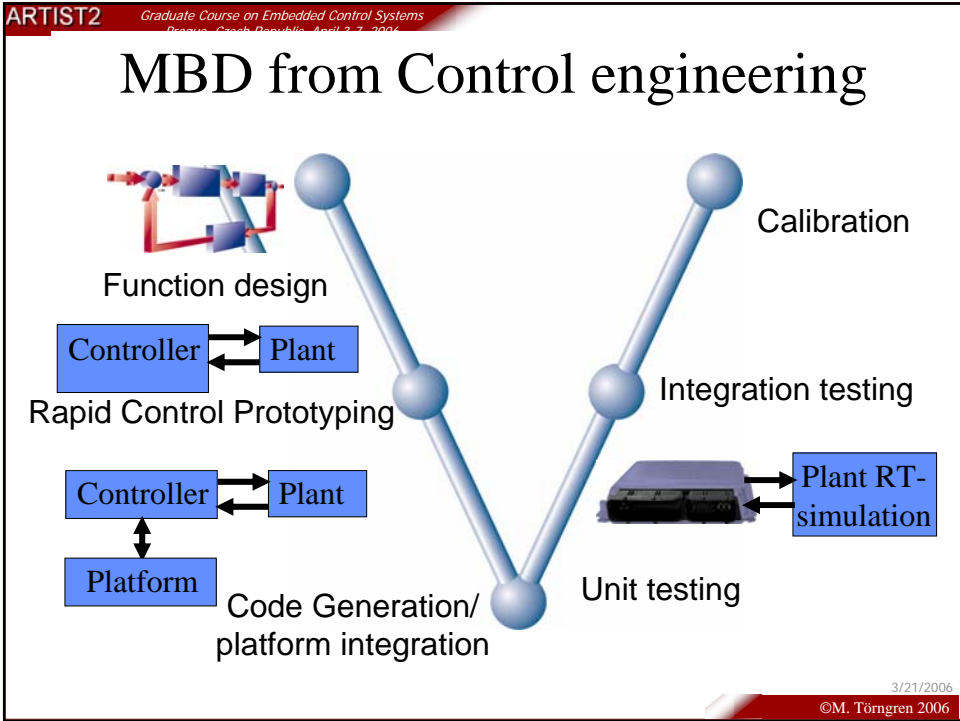
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Model-based development

- Closely related to engineering!
Work supported by abstract representations, with well defined syntax and semantics, and with visual representations
- Tool support for communication, analysis and synthesis
- Required to better manage complexity
- Many interpretations and domains where MBD is used:
 - e.g. UML in software engineering
 - e.g. CAD, STEP and PDM in mechanical engineering
- Large differences in maturity
- Key issue: How to handle the multitude of aspects of interest
 - Model and tool integration remains a challenge

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Question

Is software engineering model based?
I.e. – does coding practices and the use of any ordinary programming language represent model based development?

Compare work by OMG in defining what model driven software development really is about.

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

MBD- Software and systems

- Unified modeling language: UML → UML2
- for software, still evolving,
- Systems modeling language - SysML
- for systems engineering, (in progress)
- Architecture and analysis description language - AADL
- evolving from avionics (SAE standard)
- emphasizing real-time and reliability
- An Automotive Description Language – EAST-ADL
-(in progress)

This is an active research area

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

OMGs viewpoint on meta-models

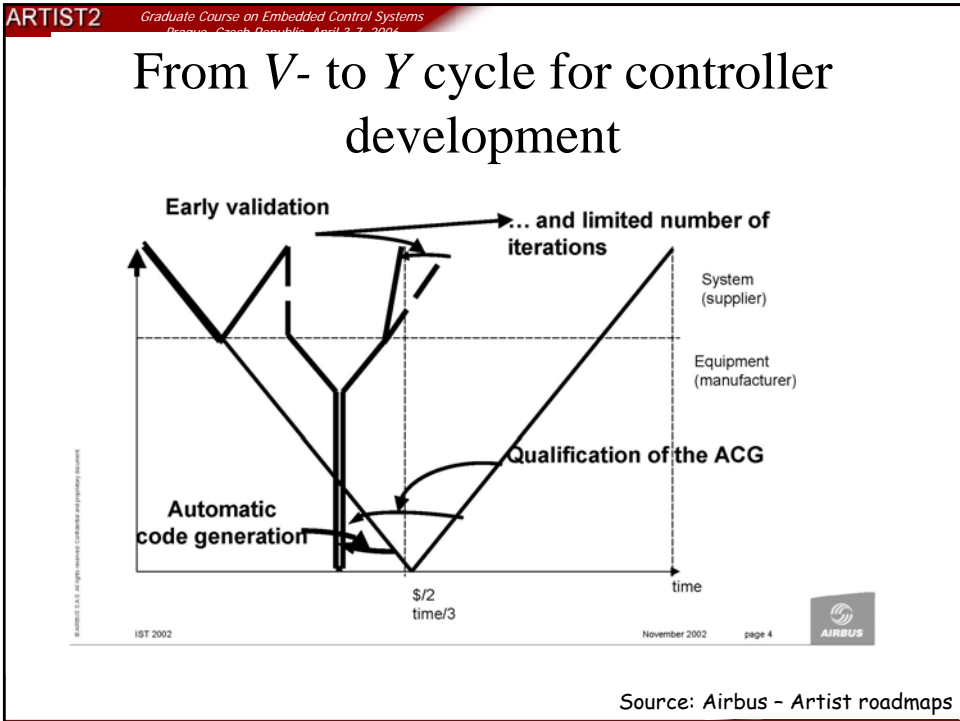
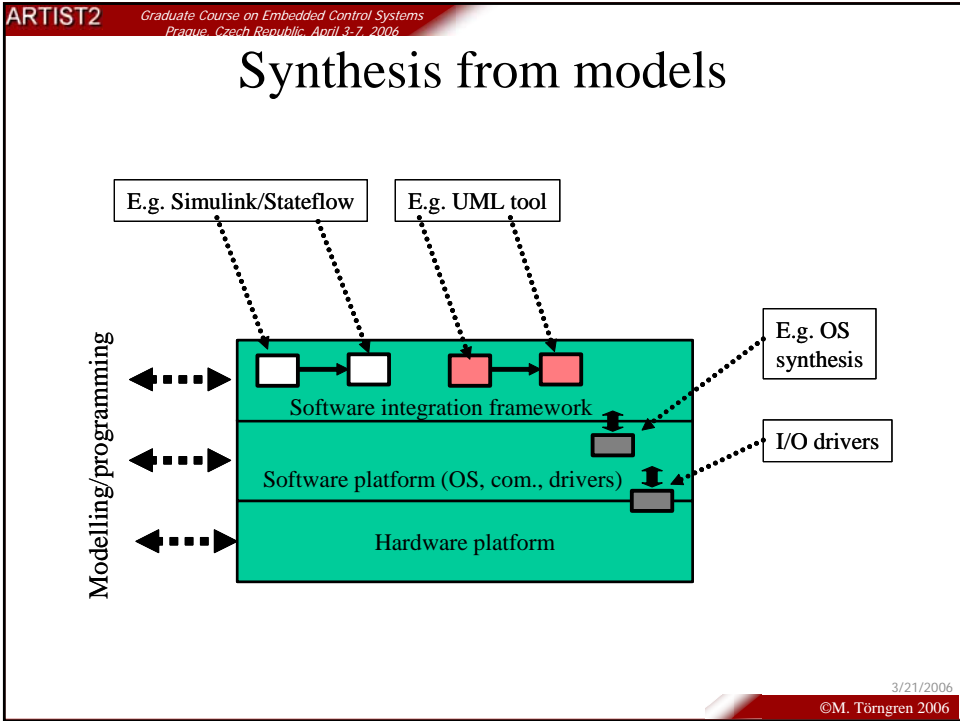
M_3 metamodel The MOF (some kind of "representation ontology")

M_2 metamodel The UML metamodel and other MMs

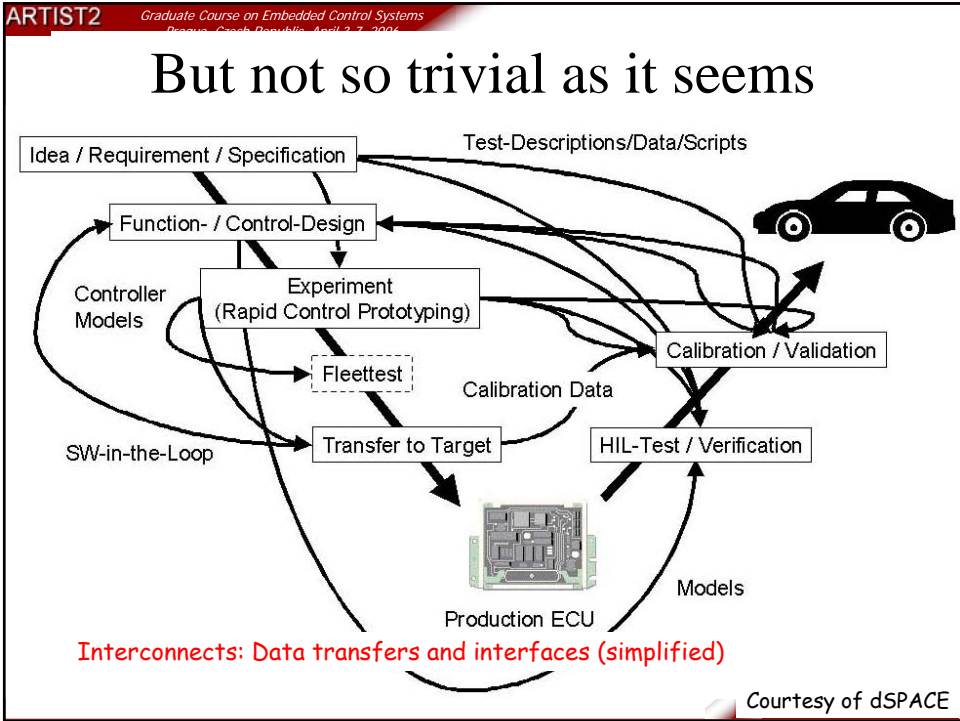
M_1 model Some UML Models and other Ms

M_0 "the real world" Various usages of these models

3/21/2006
©M. Törngren 2006



Source: Airbus - Artist roadmaps



- ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006
- ## General needs and trends
- Early validation and throughout development
 - Early integration and throughout development
 - Reuse of all relevant assets; function designs, test definitions, scripts, tool support
 - Increasing tool support for complex systems
→ model based development
- 3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- ECS implementation issues and trade-offs
 - Technical issues in controller implementation, trade-off examples
 - Platform selection: Choice of RTOS and processor
- Development approaches:
 - Traditional, CBD, MBD, platform based
 - **Tool support for deployment –a taster**
- Application example
 - Smart-1 spacecraft
- Concluding Remarks

3/21/2006
©M. Törngren 2006

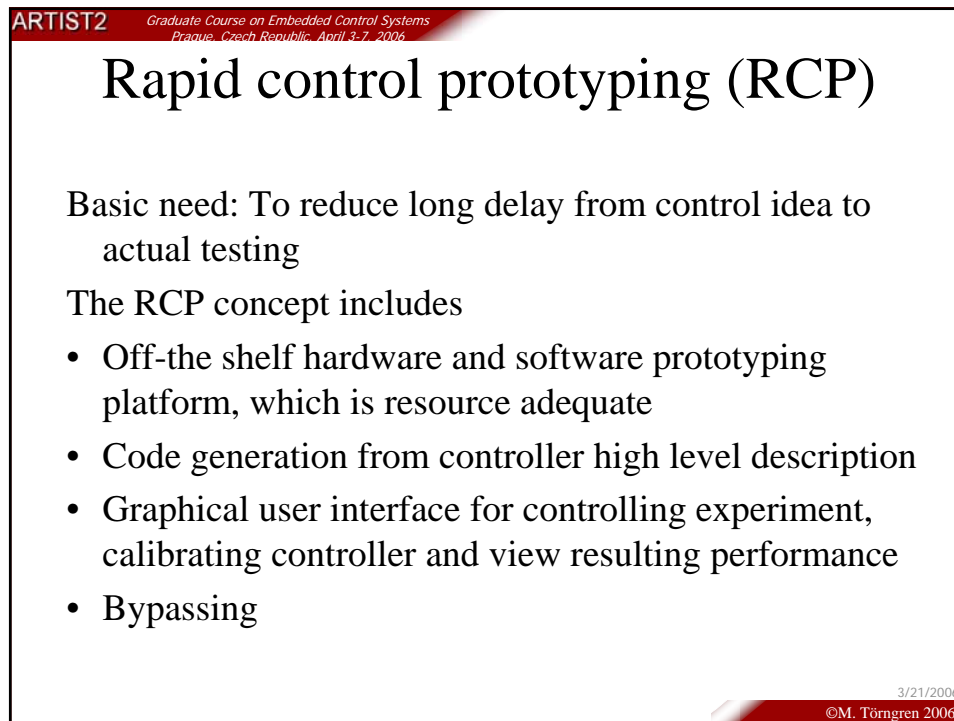
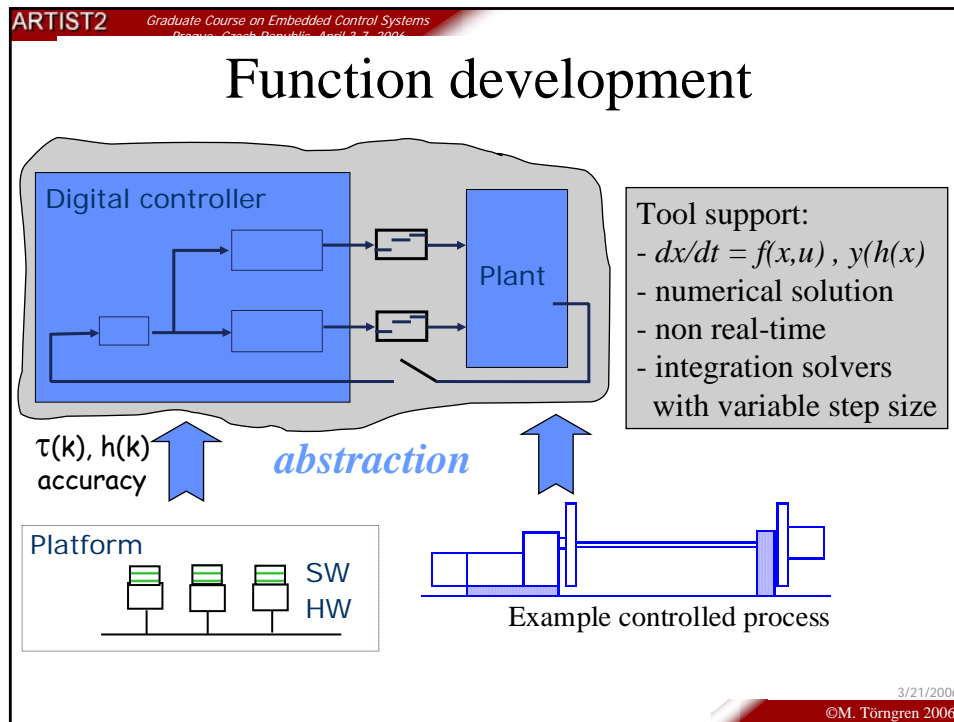
ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Tools supporting controller implementation

- Function development – “model in the loop”
- Rapid controller prototyping
- Software in the loop
- Processor in the loop
- Hardware in the loop

Referring to different analysis modes where the different parts of the controller and plant are either simulated or “real”

3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Rapid control prototyping (RCP)

Digital controller

Output, e.g. DA

Input, e.g. AD

Tool support:

- code generation
- real-time execution
- fixed step size
- I/O blocks
- experiment control
- also for systems identification

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Software in the loop

- Running implementation code within the simulation environment
- Otherwise similar to ‘model in the loop’

Production software

Digital controller

Plant

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Processor-in-the-Loop

simulation on PC

C-Code

EVM

RS 232

```

***** Summary for model: controller
*****
Function                               Code [bytes]
-----
Sa2_start_mode                           62
controller                               82
Sa2_running_mode                         150
Sa1_RP_Control_enabled                   496
    
```

12.11.1999

Courtesy of dSPACE

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Hardware in the loop - example

ECU
Real electronic control unit

Vehicle Model

Engine Model

Tool support:

- code generation
- real-time simulation of plant model
- fixed step size
- 'Inverted' I/O blocks & I/O functionality
- test control

1/2006
©M. Töngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Hardware in the loop simulation

Special requirements

- Hardware outputs: sensor emulation
- Hardware inputs: read controller outputs
- Electrical loads, fault injection at inputs and outputs
- Models with sufficient accuracy, parametrizable and real-time

+ Tests under any desired conditions are possible
 → safety critical failures and stress tests

+ Reproducible tests and automated testing

- “Black“ box testing: additional debugging tools required
 → Requires test methodology and initial investment

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- ECS implementation issues and trade-offs
 - Technical issues in controller implementation, trade-off examples
 - Platform selection: Choice of RTOS and processor
- Development approaches:
 - Traditional, CBD, MBD, platform based
 - Tool support for deployment
- **Application example**
 - **Smart-1 spacecraft**
- Concluding Remarks

3/21/2006
©M. Törngren 2006

The SMART-1 spacecraft: mission to the moon

SMART: Small Missions for Advanced Research & Technology

- *ESA*: European Space Agency
- Prime contractor: Swedish Space Corporation. Subsystems were made all over the world
- *SMART-1*:
launched through Ariane 5 vehicle from French Guiana, on 27 September 2003

<http://www.esa.int/SPECIALS/SMART-1/index.html>

3/21/2006

©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

The SMART-1 spacecraft cont.

Mission

- Scientific experiments,
- Demonstration of electric primary propulsion
- Commercial off the shelf components including CAN
- Energy sources: solar cells, xenon gas and hydrazine
- One <75 mN stationary plasma thruster ("7 grams pulling force")
 - Very efficient engine (only 70kg xenon)
 - Poor acceleration
 - Potential usage for long space journeys in the future
- 2 years lifetime, 350Kg weight

3/21/2006

©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Smart-1

The diagram shows a satellite in an elliptical orbit around Earth. A dashed circle around the satellite indicates the area of 'Disturbances'. Labels with arrows point to various components: 'Energy from thrusting' points to the satellite's body; 'Sun sensing' and 'Sun energy' point to a sensor on the satellite; 'Star tracking' points to a sensor on the satellite; 'Telemetry and telecommand' points to a communication link between the satellite and Earth; 'earth' points to the planet. A dashed circle also encloses the satellite and its immediate surroundings.

Disturbances: Gravity, particles and aero drag

Courtesy of Swedish Space Corporation

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Smart-1 and examples of actuators and sensors

The cutaway diagram shows the internal structure of the satellite. Labels with arrows point to specific components: 'Sun sensors (3 in total)' are located on the top surface; 'Reaction wheels (4 in total)' are arranged in a central cluster; 'Star trackers (2 in total)' are mounted on the side; 'Hydrazine thrusters (8 in total)' are distributed around the satellite's perimeter; and the 'EP thruster and orientation mechanism' is located at the bottom.

Courtesy of Swedish Space Corporation

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

Smart-1 sensors and actuator

- **Sensors:**
 - Sun sensors,
 - Star trackers
 - Angular rate sensors
 - Thermal sensors
- **Actuators:**
 - Reaction wheels,
 - 1N attitude control hydrazine thrusters
 - Actuators for deploying and rotating the solar arrays
 - Electric propulsion
 - Thruster orientation mechanism (gimbal drives)
 - Heaters

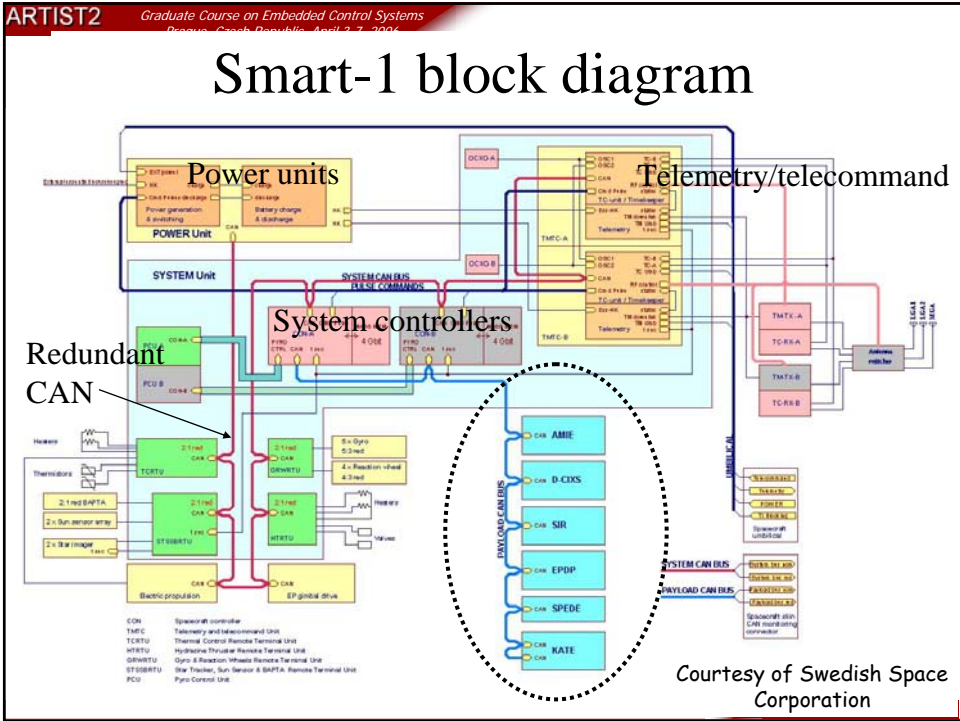
3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006

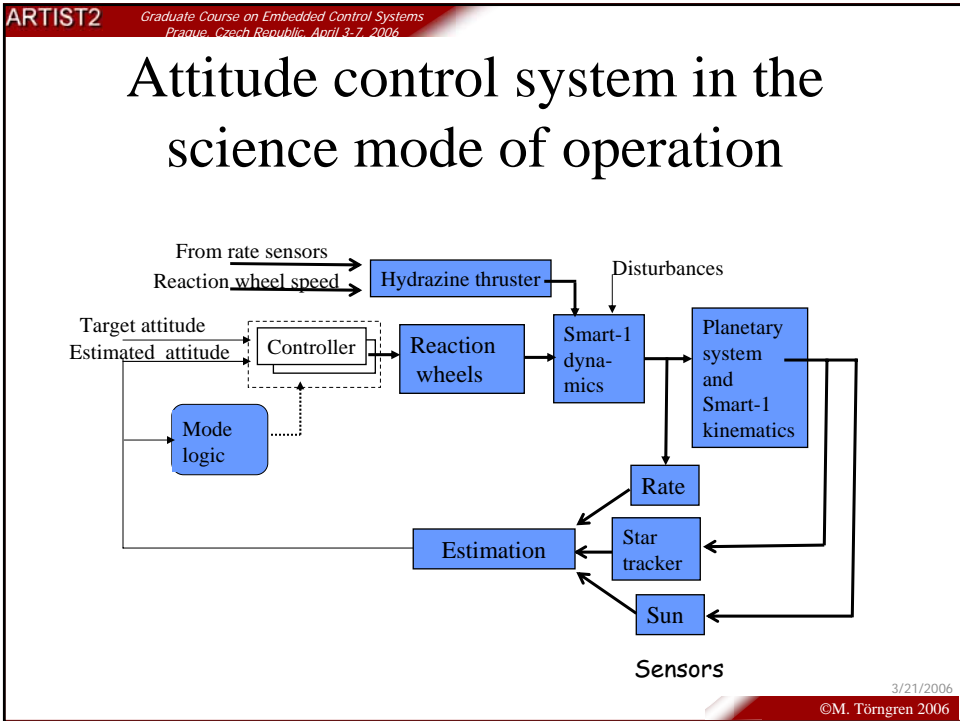
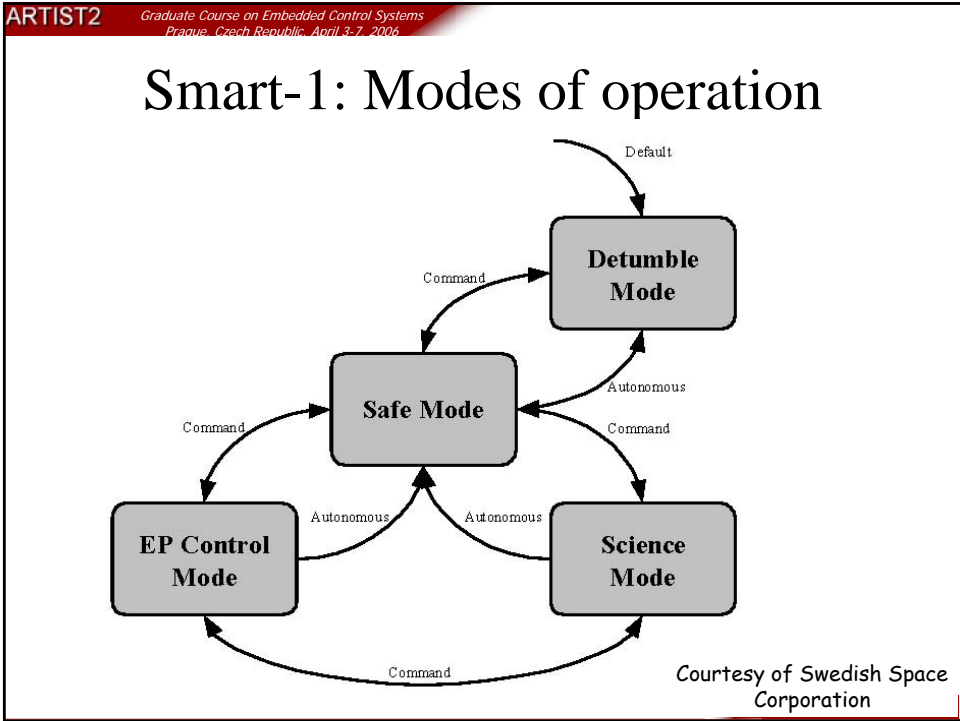
Smart-1 block diagram briefly explained

- One science subsystem (indicated by dotted oval) with dedicated CAN network
- One control subsystem including
 - Telemetry and telecommand
 - Power unit, batteries and solar cells (1850W)
 - System controllers (one redundant)
 - Sensor and actuator subsystems

3/21/2006
©M. Törngren 2006



- ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 27, 2006
- ## Characteristics: Smart-1 computer
- Distributed I/O system: system controller and smart I/O units – loops closed over network
 - Dual CAN network
 - Main portion: Master/slave polling
 - Also employing asynchronous potential of CAN
 - Master slave clock synchronisation
 - TMTV maintains and distributes "space elapsed time"
- 3/21/2006
©M. Törngren 2006



ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Attitude and orbit control system

- Main computation is done in the system unit
- Distributed I/O, closed loop operation over the network
- Sampling period: 1s
- Timing analysis carried out to ensure small enough end to end delay
- Passive backup controller: state backed up in the power unit (PCDU)

3/21/2006
©M. Törngren 2006

Specific concerns for spacecraft controllers

- Energy supply
- Autonomous operation
 - Must survive without ground contact
 - Must handle tough environment & subsystem failures
- Cosmic radiation:
 - Bit-flips in digital electronics by ionizing particles (alpha, heavy ions, neutrons, myons etc.)
 - Single event upset intensity for SRAM in 0,6 μm :
 - Space: in the order of 10^{-3} faults/hour/MB
 - Ground level: 10^{-7} - 2×10^{-6} faults/hour/MB
- A potential problem for future ground electronics as sizes shrink?

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Reliability considerations in Smart-1

Designed for fault-tolerance and autonomy:

- Few single point failures: spare for most functions, examples
 - Passive redundancy for system controller
 - Active redundancy for power unit
- Error detection:
E.g. watchdog timers, periodic messages, checksums, ...
- Control hierarchy (supervision chain)
- Automatic reconfiguration and graceful degradation:
safe mode & ground interaction

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Main on-board computer

TCS695E single chip ERC32 processor, 20 MHz clock

EEPROM 2 Mbyte EDAC (error detection& correction) protected

SRAM 3 Mbytes EDAC protected,

Additional memory protection through 'scrubbing' (e.g. Background software, reading, detecting and correcting faults)

MassMemory 512 Mbyte, EDAC protected

CAN controllers, 2 kBytes FIFO in reception path
→ Purchased VHDL code; radiation tolerant FPGA

Watchdog

JTAG I/F for EEPROM software upload and board HW checkout

Bottom line: "Space compliant technology + VHDL CAN"

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Smart-1 system development

- Matlab/Simulink, rapid prototyping and code generation
- VxWorks RTOS
- Several refined control system releases, subject to increasing testing, including HIL
- Software upgrade/upload has been carried for the star camera

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Outline

- ECS implementation issues and trade-offs
 - Technical issues in controller implementation, trade-off examples
 - Platform selection: Choice of RTOS and processor
- Development approaches:
 - Traditional, CBD, MBD, platform based
 - Tool support for deployment
- Application example
 - Smart-1 spacecraft
- **Concluding Remarks**

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

Concluding remarks

- A very broad spectra of ECS applications – e.g. seen in multitude of RTOS'es
- Trade-offs required for resource constrained implementations: Memory, accuracy and execution time usually in conflict
- Different approaches to development and deployment: Traditional compile/link/load over CBD to MBD
- Evolving modeling languages for ECS

3/21/2006
©M. Törngren 2006

ARTIST2 Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

A few selected research topics

- Co-design and architectural design
 - Functions, software, hardware
- Model integration and management
- Distributed systems support

3/21/2006
©M. Törngren 2006

Distributed control systems development

- Tools and development today:
 - Mainly single node
 - In some domains, there are tools that provide some support
- Active tools area:
 - Plenty of research and prototype tools
 - Emerging tools (TTP, Flexray etc.)
- Distributed control systems require “system-wide” configuration:
 - Execution and communication strategy
 - Error detection and handling strategy

Thursday 6th of April



ARTIST2

Off-line scheduling

Zdenek Hanzalek

Department of Control Engineering

FEE, CTU in Prague

<http://dce.felk.cvut.cz/hanzalek>, hanzalek@fel.cvut.cz

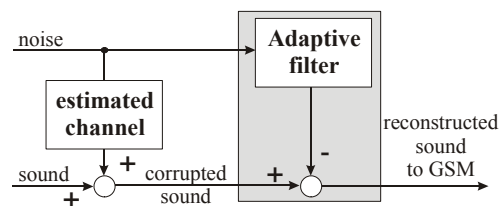
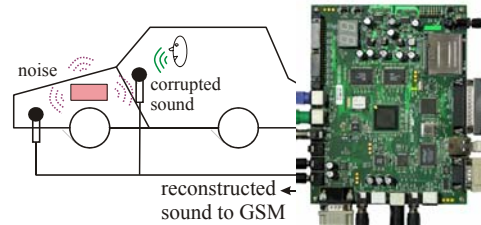


Outline

- Motivation and Standard notation $\alpha/\beta/\gamma$
- Monoprocessor scheduling – state of the art
 - C_{\max} minimization
- Scheduling on FPGA with arithmetic unit
 - Start Time Related Deadlines
 - Cyclic scheduling

Motivation Example

- Digital Signal Processing
- Active noise cancellation
- sampling frequency 44kHz
- FPGA hardware Virtex II

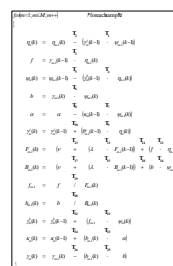


3/17/2006

© Zdenek Hanzalek 2006

Parallel implementation of algorithms

- High-level **synthesis**
 - off-line scheduling
- **Specific HW** architecture (FPGA, DSP)
 - high degree of parallelism
 - dedicated unit
 - Pipelining
- **Optimality**
 - Branch&Bound algorithms
 - ILP formulations



3/17/2006

© Zdenek Hanzalek 2006

Scheduling

- set **T** of n tasks $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$
- set **P** of m processors $\mathbf{P} = \{P_1, P_2, \dots, P_m\}$
- set of additional resources

Scheduling = **assignment of task to processors** in order to complete the tasks under imposed constraints

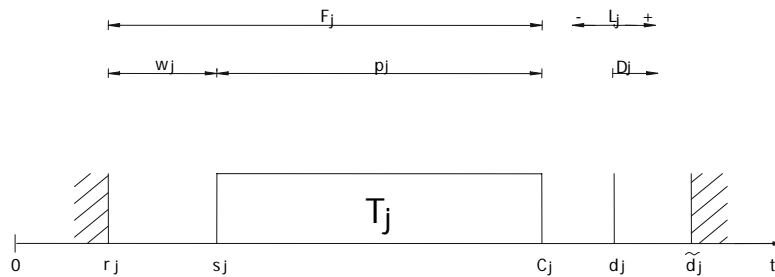
- **off line** – all parameters of the tasks and resources are known in advance
- deterministic parameters – combinatorial **optimization** algorithms
- schedules are represented by **Gantt charts**
- J. Blazewicz, K. Ecker, G. Schmidt, J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer, 2001

Basic constraints

- each task is to be processed by at most **one processor** at a time
- each processor is capable of processing at most **one task** at a time
- task T_j is processed in time interval $[r_j, \infty)$
- all tasks are **completed**
- if tasks T_i, T_j are in the relation $T_i < T_j$, the processing of T_j is not started before completion of T_i
- in the case of **non preemptive** scheduling no task is preempted, otherwise the number of preemptions is **finite**
- additional resource constraints, if any, are satisfied

Task parameters

- release (arrival) time r_j
- start time s_j
- due date d_j
- deadline \tilde{d}_j is hard real time limit by which T_j must be completed
- completion time C_j
- waiting time w_j
- processing time p_j
- flow (response) time $F_j = C_j - r_j$
- lateness $L_j = C_j - d_j$
- tardiness $D_j = \max\{C_j - d_j, 0\}$



3/17/2006

© Zdenek Hanzalek 2006

6

Standard notation $\alpha/\beta/\gamma$ by Graham

$$\alpha = \alpha_1 \alpha_2$$

- $\alpha_1 = \dots$
- 1 processor
 - P parallel identical processors
 - Q parallel uniform proc. $p_{ij} = p_j/b_i$
(b_i is proc. speed)
 - R parallel unrelated proc. p_{ij} is arbitrary
 - O dedicated machines "open-shop"
 - F dedicated machines "flow-shop"
 - J dedicated machines "job-shop"
- $\alpha_2 = \dots$
- variable number of processors
 - k given number of processors

3/17/2006

© Zdenek Hanzalek 2006

7

Tasks: $\beta = \beta_1 \beta_2 \beta_3 \beta_4 \beta_5 \beta_6 \beta_7 \beta_8$

$\beta_1 \in \{\dots, \text{pmtn}\}$	preemption
$\beta_2 \in \{\dots, \text{res}\}$	additional resources
$\beta_3 \in \{\dots, \text{prec, tree, chain}\}$	precedence constraints
$\beta_4 \in \{\dots, r_j\}$	release time
$\beta_5 \in \{\dots, p_j=k, p_L \leq p_j \leq p_U\}$	var/const/limited proc.time
$\beta_6 \in \{\dots, d^-\}$	deadline
$\beta_7 \in \{\dots, n_j \leq k\}$	limited number of jobs in Job-shop
$\beta_8 \in \{\dots, \text{no-wait}\}$	buffers with infinite/zero capacity

Optimality criterion γ

$$\gamma \in \{C_{\max}, \Sigma C_j, \Sigma W_j C_j, L_{\max}, \dots\}$$

$$C_{\max} = \max_{\forall j} \{C_j\}$$

$$L_{\max} = \max_{\forall j} \{C_j - d_j\}$$

For example standard notation $P || C_{\max}$ means:
variable number of identical parallel processors, non-preemptive, no precedence constraints, all tasks starting at time 0, variable processing time, make-spawn minimisation

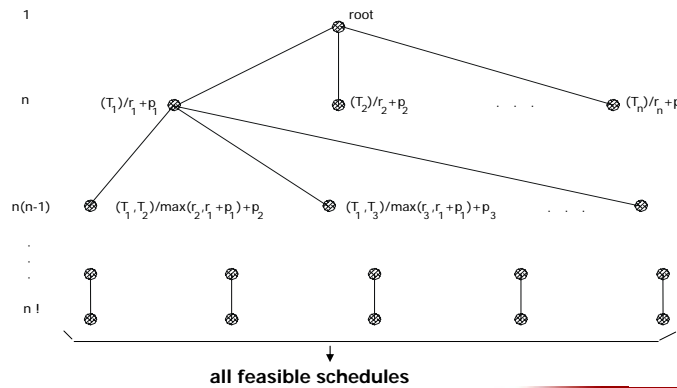
Outline

- Motivation and Standard notation $\alpha/\beta/\gamma$
- Monoprocessor scheduling – state of the art
 - C_{\max} minimization
- Scheduling on FPGA with arithmetic unit
 - Start Time Related Deadlines
 - Cyclic scheduling

C_{\max}

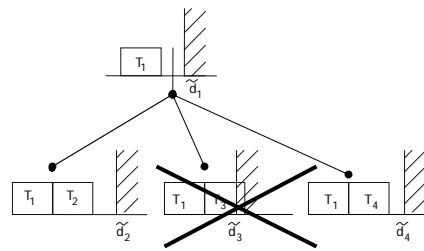
- $1|\text{prec}|C_{\max}$ – simple
 - if tasks are assigned in whatever order in accordance with precedence relation, then $C_{\max} = \sum p_j$
- $1||C_{\max}$ – simple
- $1|r_j|C_{\max}$ – simple
 - tasks are scheduled in order of nondecreasing release times
- $1|d_j^-|C_{\max}$ – simple
 - tasks are scheduled in order of nondecreasing deadlines (**EDF** – earliest deadline first)
 - EDF provides optimal solution iff there exists a schedule that meets all the deadlines

- $1 | r_j, d_j \sim | C_{\max}$ – NP hard problem
 - transformation from the 3-PARTITION problem
 - polynomial algorithm can be found if $p_j=1$
 - general problem can be solved by applying **Branch&Bound** algorithm by **Bratley**



3/17/2006

- (i) **exceeding deadlines**
- if completion time associated with at least one of the nodes under node v in level $k-1$ then all nodes under v can be eliminated

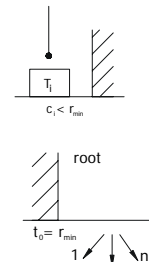


due to this vertex it is needed to eliminate both "brother" vertices

- (ii) **probl. decomposition**
- if the completion time C_i of all scheduled tasks is less than or equal to smallest release time of all unscheduled tasks

situation at level k

it remains to schedule (n-k) tasks



Optimality test of Bratley's algorithm

block is a group of tasks such that the first task starts at its release time and all the following tasks to the end of the schedule are processed without idle time

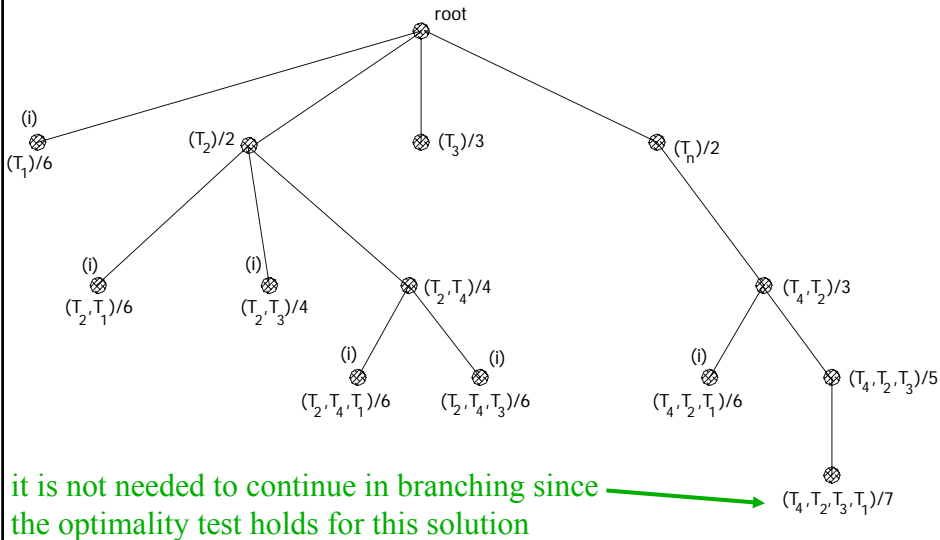
block satisfies **release time property** if release time of all tasks in the block are greater or equal to the release time of the first task in the block

Lemma: A schedule is optimal iff it contains a block that satisfies the release time property

Proof:

- if part (each schedule with block satisfying RTP is optimal) - follows from the definition of RTP
- only if part (each optimal schedule has block satisfying RTP) – by contradiction – suppose schedules that do not have block with RTP, none of them is optimal

Example: $r = [4, 1, 1, 0]$, $p = [2, 1, 2, 2]$, $d^- = [7, 5, 6, 4]$



3/17/2006

16

© Zdenek Hanzalek 2006

Outline

- Motivation and Standard notation $\alpha/\beta/\gamma$
- Monoprocessor scheduling – state of the art
 - C_{\max} minimization
- Scheduling on FPGA with arithmetic unit
 - Start Time Related Deadlines
 - Cyclic scheduling

3/17/2006

17

© Zdenek Hanzalek 2006

Specific HW architecture

FPGA based High-Speed Logarithmic Arithmetic (HSLA) -
one **dedicated** ADD/SUB unit

Operation on HSLA (19-bit)	+ (-)	\times , $/$, $\sqrt{-}$
Processing time ρ [clock cycles]	1	1
In-Out Latency [clock cycles]	9	2
Number of units	1	∞



HSLA is pipelined ... leads to the precedence delays in G

3/17/2006

18

© Zdenek Hanzalek 2006

Problem Statement

Off-line scheduling problem

- Monoprocessor – dedicated unit HSLA in FPGA
- Task T_i with processing time p_i
- Precedence relations and precedence delays $w_{ij} \geq p_i$
- Start time related deadlines – real-time requirements

Objective - to find a feasible schedule with a minimum C_{max}

Related work:

- [M. Jacomino, D. Gutfreund & J. Pulou 99] - **Scheduling Real-time processes with timing constraints and its applications to cyclic systems** – problem formulation, mobile phones, heuristics.
- [P. Brucker, T. Hilbig & J. Hurink 99] - **A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags** – B&B alg.

3/17/2006

19

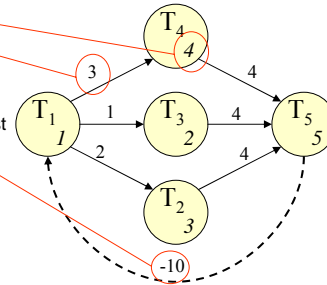
© Zdenek Hanzalek 2006

Problem formalization

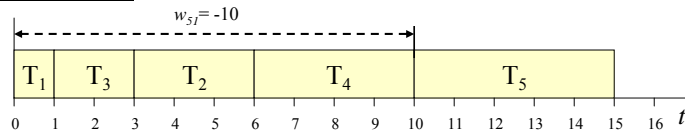
Algorithm representation by **oriented graph G**

- **node** ~ instruction ~ task T_i with processing time p_i
- **forward arc** /positive sign/ ~ express precedence delay, including **pipelining** or processing time on **nondedicated processors**
- **backward arc** /negative sign/ ~ express deadline, the latest starting time s_j of T_j relative to the starting time s_i of T_i
- both forward and backward edges are weighted by w_{ij} satisfying:

$$s_j - s_i \geq w_{ij}$$



Optimal feasible schedule:



Problem complexity

Our problem is NP-hard, since it is P-reducible from Bratley's problem (P-reducible from 3-PARTITION prob.)

Instance of Bratley's problem \Rightarrow instance of our problem

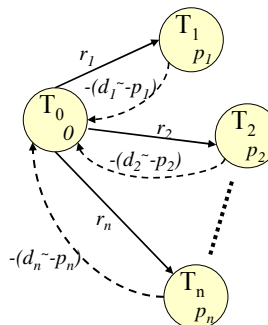
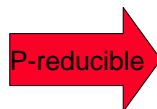
Independent tasks

$$1 \mid r_j, d_j^- \mid C_{max}$$

$$r = [r_1, r_2, \dots, r_n]$$

$$p = [p_1, p_2, \dots, p_n]$$

$$d^- = [d_1^-, d_2^-, \dots, d_n^-]$$



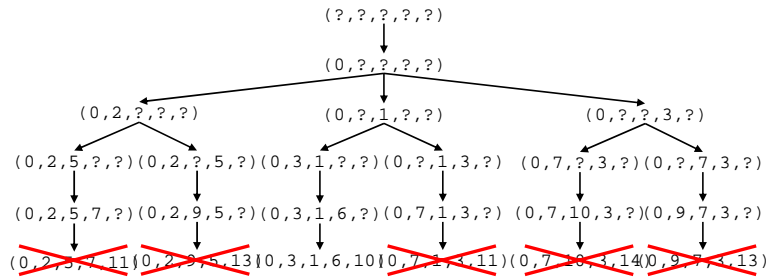
Branching procedure with basic bounding

At each step, the set of tasks is partitioned into **three disjoint subsets**:

\mathcal{T}_s already scheduled tasks

\mathcal{T}_c candidate tasks (ready to be scheduled)

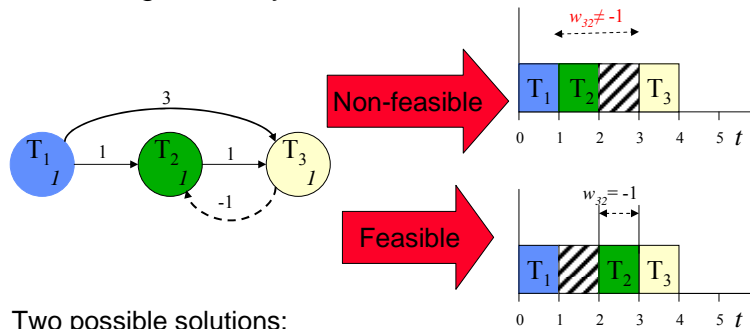
\mathcal{T}_r remaining tasks (not ready due to precedence relations)



3/17/2006

Scheduling Anomaly

Prior to eliminate a solution we have to check scheduling anomaly, since the **deadlines are relative**.



Two possible solutions:

Shifting procedure – same order of tasks

Decision by LP

3/17/2006

Critical Path Bounding

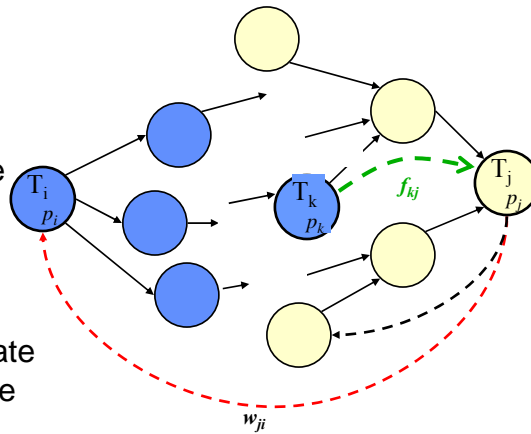
- 1) Longest path calculation by Floyd's algorithm
- 2) For each **activated backward edge** we calculate **lower bound** of s_j as:

$$\tilde{s}_j(S) = f_{kj} + s_k$$

- 3) Does it lead to the **feasible solution**?

$$s_i - \tilde{s}_j \geq w_{ji}$$

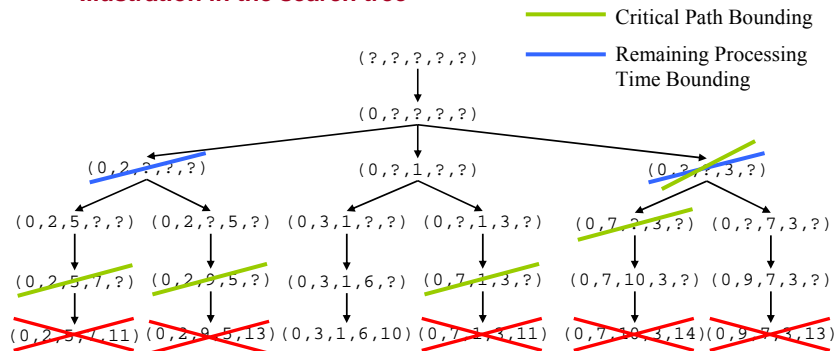
- 4) If not, then eliminate **father's vertex** in the search tree



Remaining Processing Time Bounding

Use a sum of processing times of **unscheduled tasks** to calculate **lower bound** on s_j

Illustration in the search tree



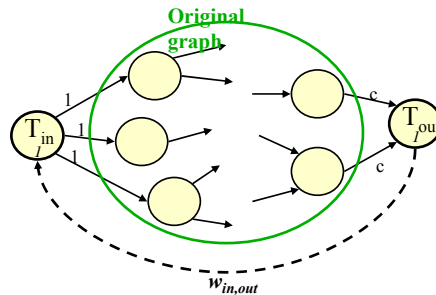
C_{max} Bounding

The **best known solution** is used to **eliminate** a partial solution leading to worse C_{max}

C_{max} is **estimated** using lower bound on remaining work

Dynamic **graph transformation** by adding:

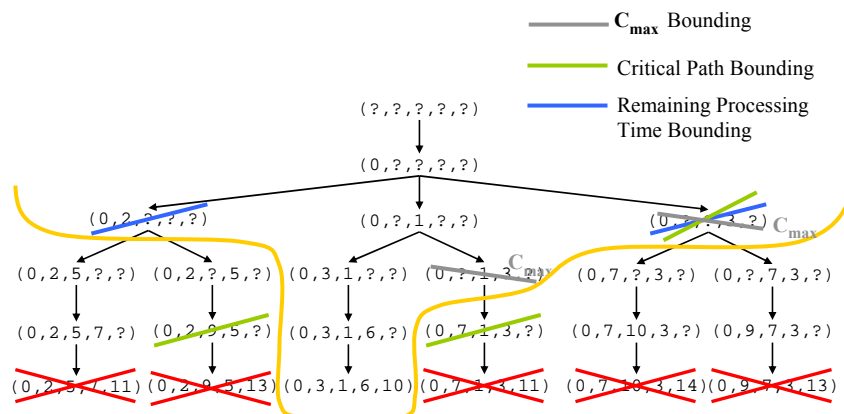
- input and output node
- edges to source / sink nodes
- dynamic backward edge



3/17/2006

Illustration in the search tree

C_{max} bounding is **inactive** at the **beginning** of the algorithm, but it is **very efficient** when C_{max}^* is found



3/17/2006

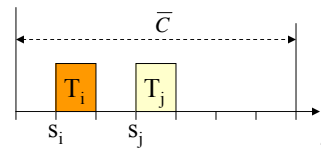
Problem Formulation by ILP

processor constraints ($(n^2-n)/2$ decision variables and constraints)

$$\forall i, j \in \langle 1, n \rangle, i < j, \quad p_j \leq s_i - s_j + x_{ij} \cdot \bar{C} \leq \bar{C} - p_i$$

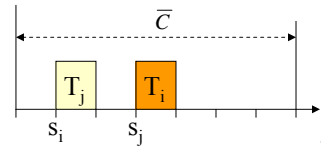
a) when T_i precedes T_j ($x_{ij} = 1$)

$$s_j \geq s_i + p_i$$



b) when T_i succeeds T_j ($x_{ij} = 0$)

$$s_i \geq s_j + p_j$$



3/17/2006

28

© Zdenek Hanzalek 2006

ILP program

$$\begin{aligned} & \min C_{\max} \\ & \text{subject to:} \\ & \quad s_j - s_i \geq w_{ij} \\ & \quad p_j \leq s_i - s_j + \bar{C} \cdot x_{ij} \leq \bar{C} - p_i \\ & \quad s_i + p_i \leq C_{\max} \\ & \text{where:} \\ & \quad s_i \in \langle 0, \bar{C} - 1 \rangle, \quad x_{ij} \in \langle 0, 1 \rangle, \quad C_{\max} \in \langle 0, \bar{C} \rangle \\ & \quad s_i, x_{ij} \text{ are integers.} \end{aligned}$$

objective function -
minimizes makespan

precedence constraint -
restriction given by graph G

processor constraints -
at maximum one task is
executed at a given time

3/17/2006

29

© Zdenek Hanzalek 2006

Outline

- Motivation and Standard notation $\alpha/\beta/\gamma$
- Monoprocessor scheduling – state of the art
 - C_{\max} minimization
- Scheduling on FPGA with arithmetic unit
 - Start Time Related Deadlines
 - Cyclic scheduling

Scheduling of Iterative Algorithms

Cyclic scheduling

- N , the number of iterations, is **large enough**
- can lead to **overlapped schedule** - operations belonging to different iterations can execute **simultaneously**

Results in periodic schedule - **one iteration** repeated each **period w**

Objective - to find a periodic schedule with a **minimum period**

Related work:

[C. Hanen & A. Munier 95] - **Basic Cyclic Scheduling** – infinite number of processors – $O(n^3 \log n)$

[A. Munier 96] – problem is NP-hard for m processors, polynomial for 2 processors and unique processing time

[Phillipe Chrétienne 2000] – approximation algorithm for m processors

[Sindorf & Gerez 2000] – problems with communication delays

BCS - Basic Cyclic Scheduling

Algorithm representation by **oriented graph G**

- vertex ~ instruction ~ task
- arc ~ precedence relation
- arc **height h_{ij}** ~ **shift of the iteration index**
- arc length l_{ij} ~ processing time

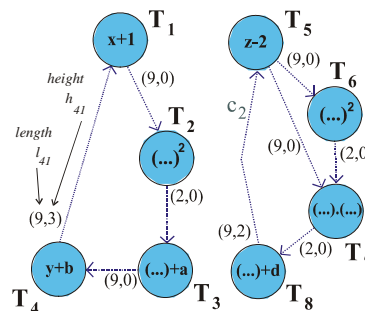
for $k=1$ **to** N **do**

$$y(k) = (x(k-3) + 1)^2 + a$$

$$x(k) = y(k) + b$$

$$z(k) = (z(k-2) - 2)^3 + d$$

end



3/17/2006

32

© Zdenek Hanzalek 2006

Formulation of BCS Problem

- Periodic schedule:
(identical for each iteration)

$$\forall i \in T, \quad \forall k \geq 1, \quad s_i(k) = s_i + w \cdot (k-1)$$

Start time of T_i in 1st iteration

- Precedence relations:

$$\forall k \geq 1, \quad s_i(k) + l_{ij} \leq s_j(k + h_{ij})$$

Start time of T_i in k^{th} iteration

- Optimization criterion - min. average cycle time:

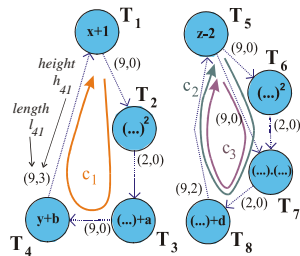
$$w = \lim_{k \rightarrow \infty} \frac{\max_{i \in T} \{s_i(k) + p_i\}}{k}$$

3/17/2006

33

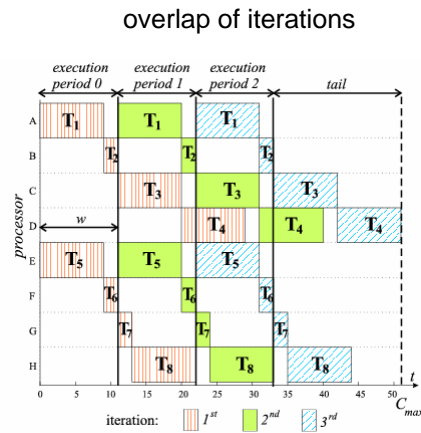
© Zdenek Hanzalek 2006

Standard solution of BCS Problem



critical circuit

$$w(G) = \max_{c \in C(G)} \frac{L(c)}{H(c)}$$

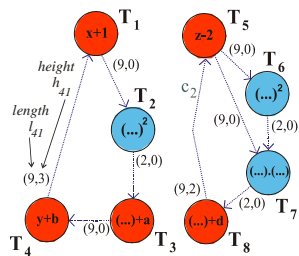


3/17/2006

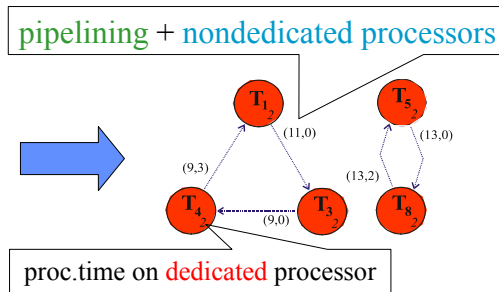
© Zdenek Hanzalek 2006

Problem statement

- Cyclic scheduling - **overlap** of iterations
- Some tasks run on the **dedicated** processor
- The dedicated processor is **pipelined**



graph G



reduced graph G'

3/17/2006

© Zdenek Hanzalek 2006

Problem complexity

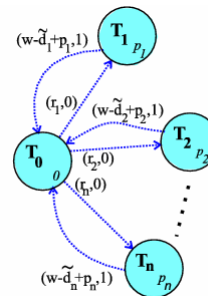
Our problem is NP-hard, since it is P-reducible from Bratley's problem (P-reducible from 3-PARTITION prob.)

Instance of Bratley's problem \Rightarrow instance of our problem

Independent tasks

$$\begin{aligned} &1 / r_j, d_j^- / C_{max} \\ r &= [r_1, r_2, \dots, r_n] \\ p &= [p_1, p_2, \dots, p_n] \\ d &= [d_1^-, d_2^-, \dots, d_n^-] \end{aligned}$$

P-reducible \rightarrow



3/17/2006

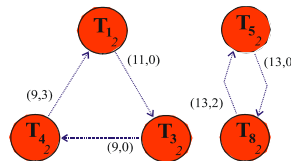
36

© Zdenek Hanzalek 2006

Dedicated processor – Problem Formulation by ILP

precedence constraints (n_e constraints)

$$\forall e_{ij} \in G, \quad s_j - s_i \geq l_{ij} - w \cdot h_{ij}$$



Start time of T_i in 1st iteration

$$s_i = \mathbf{s}_i + \mathbf{q}_i \cdot w$$

“offset”

“segment”

$$\mathbf{s}_i \in \langle 0, w \rangle$$

$$\mathbf{q}_i \geq 0$$

$$\forall e_{ij} \in G, \quad (\mathbf{s}_j + \mathbf{q}_j \cdot w) - (\mathbf{s}_i + \mathbf{q}_i \cdot w) \geq l_{ij} - w \cdot h_{ij} \quad (1)$$

3/17/2006

37

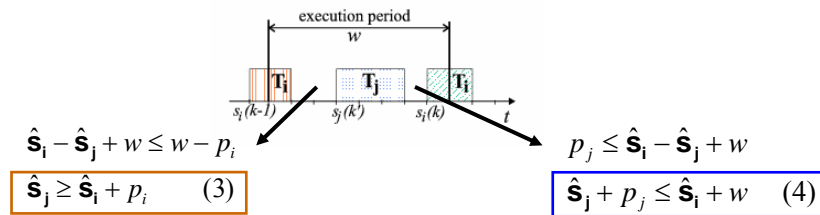
© Zdenek Hanzalek 2006

Problem Formulation by ILP

processor constraints ($(n^2-n)/2$ constraints)

$$\forall i, j \in \langle 1, n \rangle, i < j, \quad p_j \leq \hat{s}_i - \hat{s}_j + \mathbf{x}_{ij} \cdot w \leq w - p_i \quad (2)$$

a) when T_i precedes T_j ($\mathbf{x}_{ij}=1$)



3/17/2006

38

© Zdenek Hanzalek 2006

Problem Formulation by ILP (cont.)

processor constraints ($(n^2-n)/2$ constraints)

$$\forall i, j \in \langle 1, n \rangle, i < j, \quad p_j \leq \hat{s}_i - \hat{s}_j + \mathbf{x}_{ij} \cdot w \leq w - p_i \quad (2)$$

b) when T_i precedes T_j ($\mathbf{x}_{ij}=0$)

... both inequalities in double inequality (2) holds as well

- proven by exchanging index i with index j in inequalities (3) and (4)

$$\hat{s}_i \geq \hat{s}_j + p_j$$

$$\hat{s}_i + p_i \leq \hat{s}_j + w$$

3/17/2006

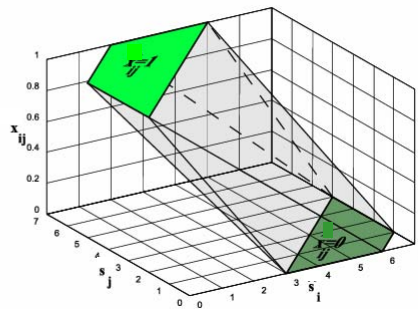
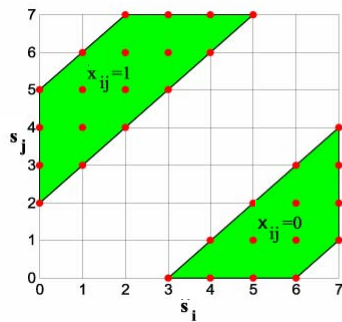
39

© Zdenek Hanzalek 2006

Processor constraints (cont.) – OR relation

Example:

T_i and T_j without precedence constraints, $p_i = 2$, $p_j = 3$, $w = 8$



3/17/2006

40

© Zdenek Hanzalek 2006

ILP program for fixed w

$$\min \sum_{i=1}^n \hat{q}_i$$

subject to:

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l_{ij} - h_{ij} \cdot w$$

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i$$

where:

$$\hat{s}_i \in \langle 0, w - 1 \rangle, \hat{q}_i \geq 0, \hat{x}_{ij} \leq w - p_i$$

\hat{q}_i, \hat{x}_{ij} are integers.

objective function -
minimizes the
iteration overlap

precedence constraint -
restriction corresponding to
algorithm of filter

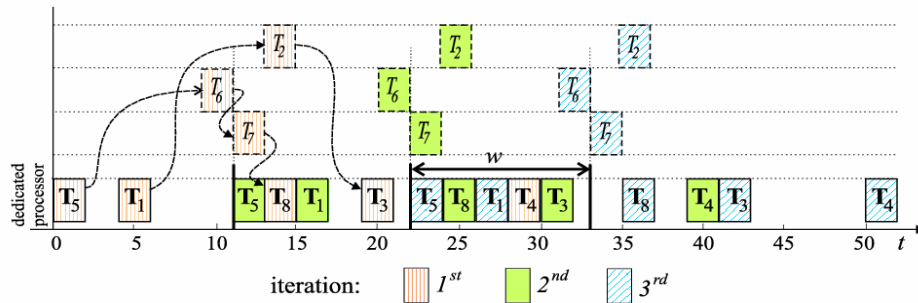
processor constraints -
at maximum one task is
executed at a given time

3/17/2006

41

© Zdenek Hanzalek 2006

Feasible Schedule ($w^* = w = 11$)



w^* - the shortest period resulting in feasible schedule

w^* is found by formulating one **ILP** program for each integer $w \in [\text{lowerbound}, \text{upperbound}]$

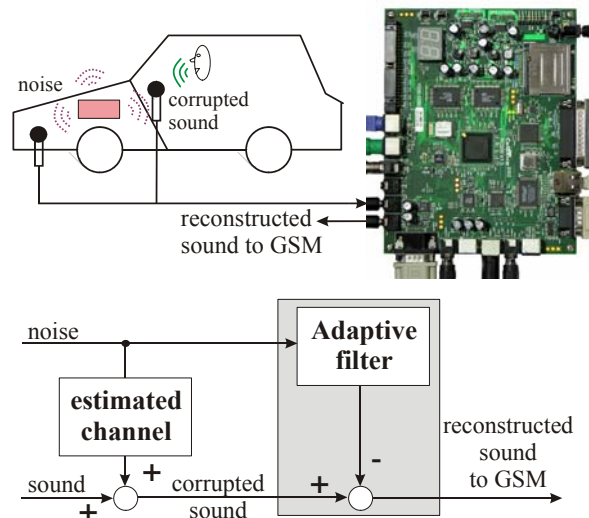
... interval bisection method

3/17/2006

42

© Zdenek Hanzalek 2006

Example



3/17/2006

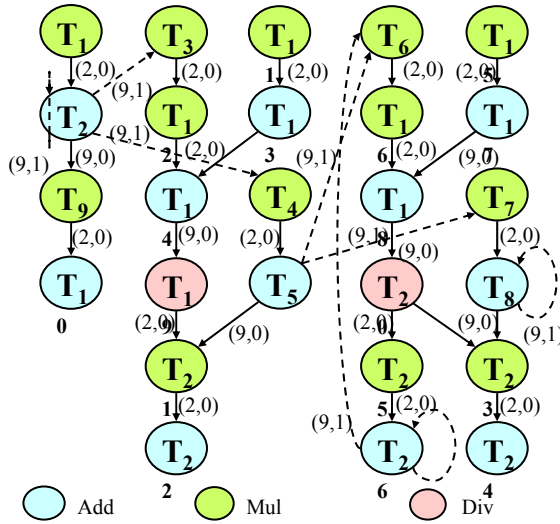
43

© Zdenek Hanzalek 2006

RLS (Recursive Least Square) Algorithm

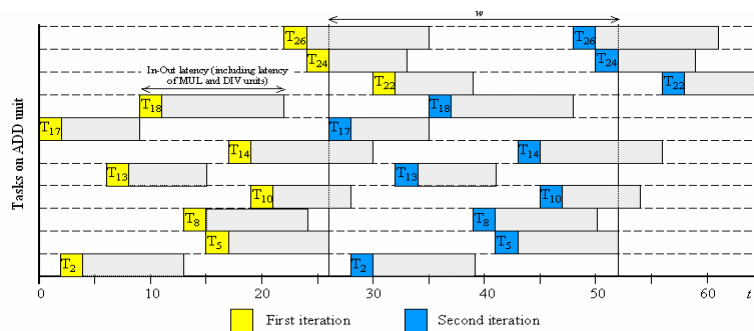
```

for(m=1; m<=M; m++) #for each sample
{
    for(k=1; k<=N; k++) #for iterations
    {
        T1 = zc(k-1) * vna(k-1)
        na(k) = na(k) - (zc(k-1) * vna(k-1))
        f = zc(k-1) * na(k)
        vna(k) = vna(k-1) + (zc(k-1) * na(k))
        b = zc(k) * vna(k)
        alpha = alpha - (zc(k-1) * vna(k))
        zc(k) = zc(k-1) + (zc(k-1) * na(k))
        Ea(k) = (v + (lambda * Ea(k-1)) + (f * na(k)))
        Ba(k) = (v + (lambda * Ba(k-1)) + (b * vna(k)))
        fa = f / Ea(k)
        ba = b / Ba(k)
        zc(k) = zc(k-1) + (fa * vna(k))
        xc(k) = xc(k-1) + (ba * vna(k))
        zc(k) = zc(k) - (ba * vna(k))
    }
}
    
```



3/17/2006

Schedule for RLS Algorithm

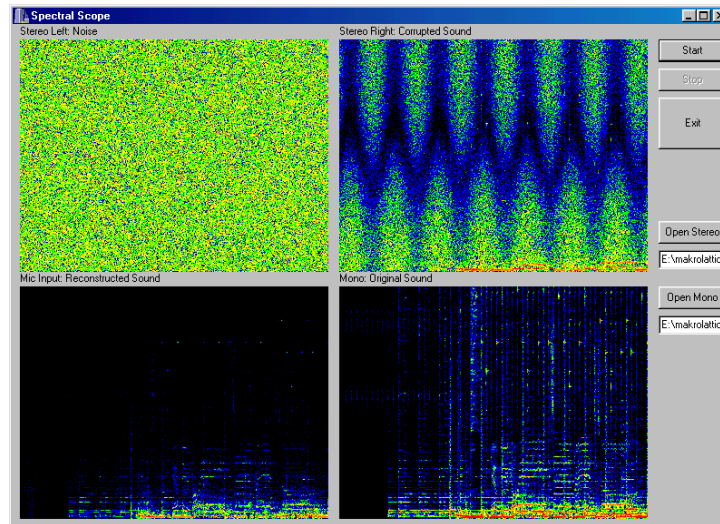


Celoxica rc200e board
50MHz in Virtex II
sampling frequency 44kHz

	w	MFLOPS	Filter order
Manual	45	80	75
Automatic	26	137.5	129

3/17/2006

Qualitative Parameters of Adaptive Filter



3/17/2006

Achievements

- ILP gives rather good results even for realistic examples in reasonable time (2 seconds)
 - model is dependent on number of tasks but it is **independent of w**
- Filter performance increased by 70%
- Better utilization of arithmetic unit
- Automatic scheduling
 - systematic design
 - **algorithm | graph | schedule | code**
 - rapid prototyping ...simulation of the schedule prior to time consuming implementation

3/17/2006

ARTIST2

Platform for Advanced Process Control and Real Time Optimization

Vladimír Havlena, Jiří Findejs
ACS Advanced Technology Laboratory Prague
Honeywell Intl.
havlena@htc.honeywell.cz, findejs@htc.honeywell.cz

Honeywell

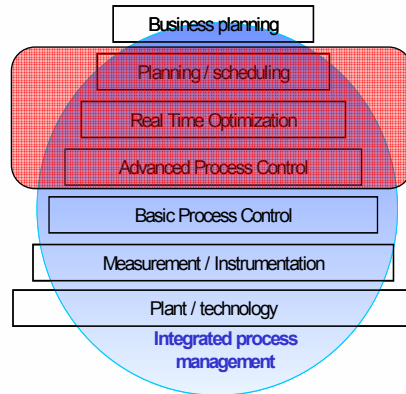


Agenda

- Introduction
 - Advanced Process Control scope, status, drivers
- Unified Energy Solutions (UES) portfolio
 - Architecture, components, technology
- Unified Real Time (URT) platform for advanced control applications
 - Objectives, architecture, benefits
 - Demo of key features
- Conclusions

Where we are ...

- Analog age
- 1980's computer technology / DCS enabled large scale implementations of regulatory control (>10 000 I/O points)
- 1990's Advanced Process Control (MPC) & Real Time Optimization layers followed
- Today's scope
 - Control strategies not restricted to cascaded blocs - controllers, state observers, etc.
 - APC/RTO + Performance monitoring, fault detection & recovery, operator decision support / what-if analysis ...
 - Complex solutions – integration cost becoming prohibitive to further extension and innovation
 - Based on open standards (OPC) and solution components



Where we go ...

- Software enabled control
 - Interaction between control & optimization algorithms and sw/hw platforms – dynamic environment
 - Embedded systems – “hard” real time
 - Specific APC/RTO features
 - Supervisory level – “soft” real-time
 - Extensive platform management features - separated from control / optimization functionality
- Optimization-based solutions
 - Mathematical optimization = basic enabling technology in advanced control
 - Off-line vs. on-line applications: numerical issues of algorithm design
 - Specific APC/RTO features
 - Large-scale solutions can absorb economics-related information as a part of internal criteria
 - Efficient methods to solve huge problems ...
 - BUT**
 - Need for structured, decentralized, hierarchical solutions – human effort to set up model / maintain / operate
 - Need for consistency between models on individual hierarchical levels (APC, RTO, planning/scheduling)



Agenda

- Introduction
 - Advanced Process Control scope, status, drivers
- Unified Energy Solutions (UES) portfolio
 - Architecture, components, technology
- Unified Real Time (URT) platform for advanced control applications
 - Objectives, architecture, benefits
 - Demo of key features
- Conclusions

Unified Energy Solutions



Portfolio of solution components covering

- APC for power generation and industrial energy (process steam production)
 - Predictive pressure/temperature ... controller
 - Combustion coordinator
- Real Time Optimization of
 - Combustion process
 - Load allocation
 - Power delivery/ancillary services
- Planning/scheduling
 - Contract planning
 - Unit commitment
- Performance monitoring
 - Efficiency, thermal stress
 - Soot blowing optimization
 - What-if analysis



UES portfolio overview

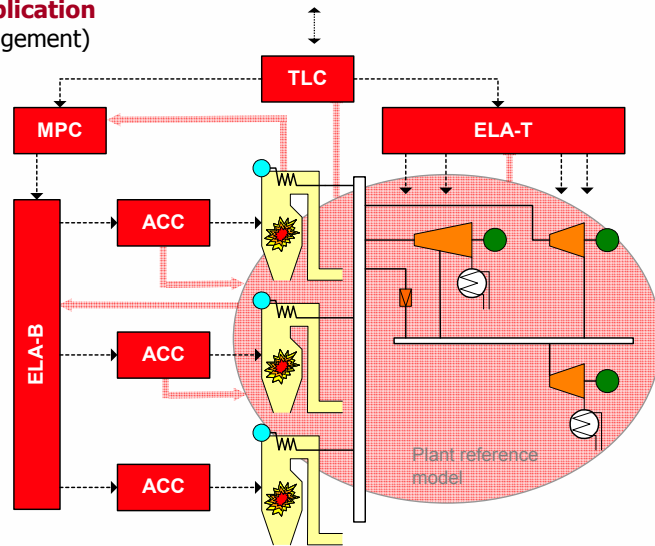
Industrial energy application (common headers arrangement)

Objectives

- Process steam
- Heating steam
- Power contract
- Costs/profit
- Responsiveness

Distributed arch.

- Control
- Local opt.
- Global opt.
- Interactions
- Variety of operation modes



6

© V. Havlena, J. Findejs, 2006

UES portfolio overview

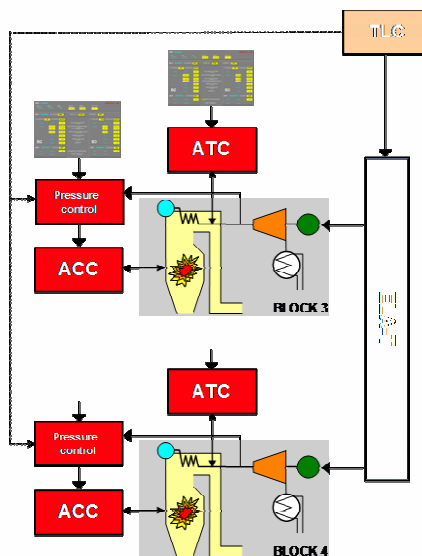
Utility application (block arrangement)

Objectives

- Power generation
- Availability/contract execution
- Costs/profit
- Responsiveness
(eligibility for ancillary services)

Distributed arch.

- Control
- Local/Global opt.
- Shared solution components



7

© V. Havlena, J. Findejs, 2006

UES components

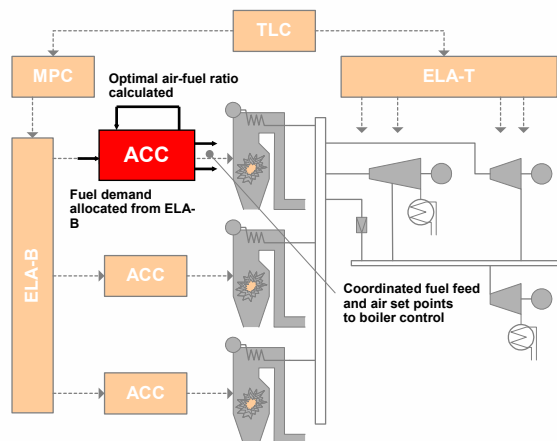


- TLC = Tie Line Control
 - Power delivery contracting tool
 - Power contract real-time monitoring/execution
- ELA-T = Economic Load Allocation for Turbines
 - Optimally allocates generated power on multiple generators
 - Multiheader setups with condensing, back-pressure turbines
- MPC = Master Pressure Controller
 - Steam balance in headers, using demand prediction
- ELA-B = Economic Load Allocation for Boilers
 - Optimally allocates steam production on multiple boilers
 - Minimizes total steam production cost
- ACC = Advanced Combustion Control
 - Advanced control of combustion process
 - Optimizes boiler thermal efficiency, keeps emissions within given limits
- ATC = Advanced Temperature control
- PRM = Plant Reference Model
 - Consistent models, real-time responsiveness

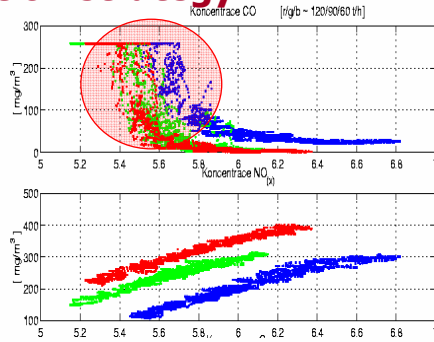
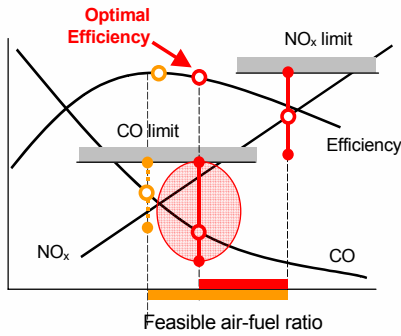
Advanced Combustion Control (ACC)

Functionality

- optimal A/F Ratio
- dynamic A/F coordination
- advanced features
 - low-NO_x burning
 - staged air design
 - flue-gas recirculation



ACC cautious optimization strategy

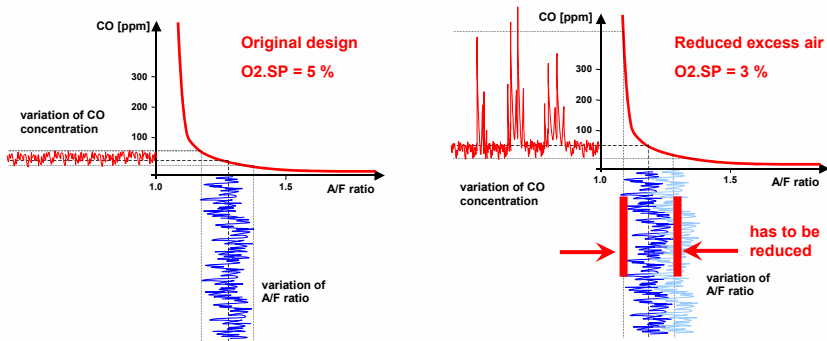


- Maximum achievable efficiency under emission constraints
- Turbulence-driven process – not very deterministic
- Cautious strategy = optimization under uncertainty
- Constraints defined in terms of posterior probability content
- Compatible with statistical emission evaluation (e.g. 15 min. average)
- Operator acceptance – “single knob” solution

$$P\{CO > CO_{max}\} < \epsilon$$

... Need for dynamic coordination

Boiler operation with reduced excess air only *not feasible*



Problem

- strongly non-linear A/F → CO mapping:

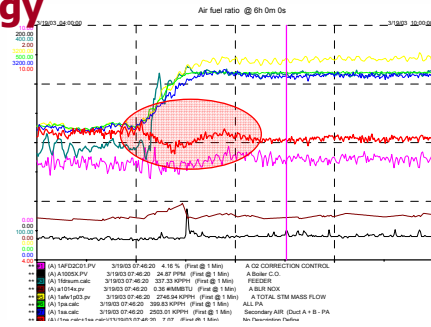
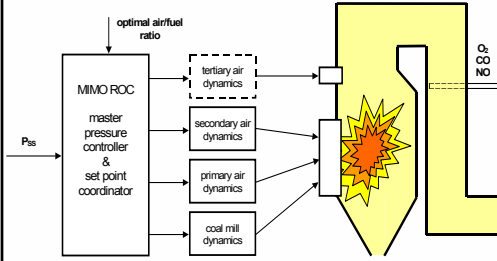
$$E\{f(x)\} = f(E\{x\}) + d^2f(x)/dx^2 \cdot \text{Var}\{x\}$$

Solution

- minimize A/F ratio variation

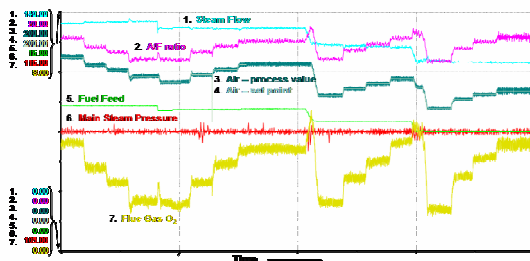
Optimal A/F ratio depends on achievable A/F coordination performance

ACC Coordination strategy

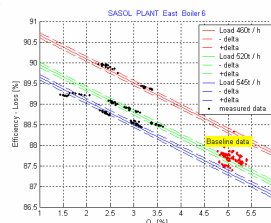
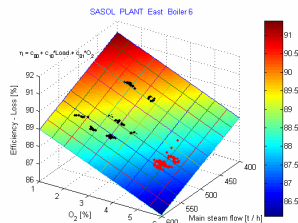
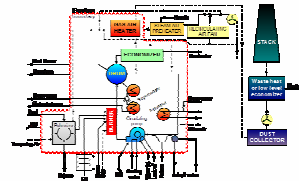


- Low NOx/optimal efficiency burning
 - Reduced excess air
 - Reduced A/F variation
- Model-predictive control formulation + extensions
 - Different dynamics
 - Ratio control - air/fuel "burner nozzle flow" coordination
 - Set-range concept → calm control – improved coordination performance

ACC Performance



ASME power test code



Economic Load Allocation for Boilers (ELA-B)

Functionality

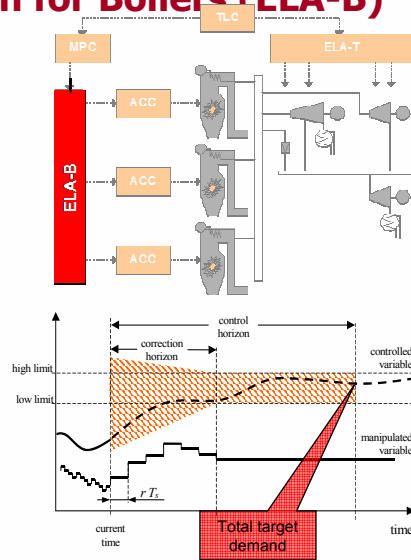
- Allocate total steam production
- Efficiency + fuel cost
- Independent steady-state/dynamic solution

Algorithmic solution

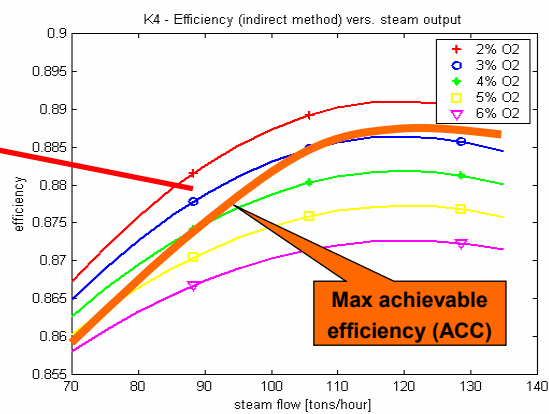
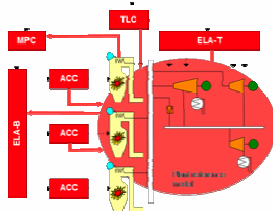
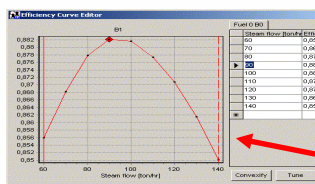
- SQP with IST - Iteration spread in time
- Feasibility constrained SQP

APC/RTO interaction

- Classical approach
 - Wait for steady state, run RTO
- MPC-enabled approach
 - Predict target steady state
 - Optimize at target set point
- RTO execution rate
 - Optimum tracking capabilities
(hours → minutes/seconds)



ELA-B – Optimized Load Allocation



Adaptation based on ACC performance (unit-level optimization)

Tie Line Control (TLC)

Functionality – on-line / off-line

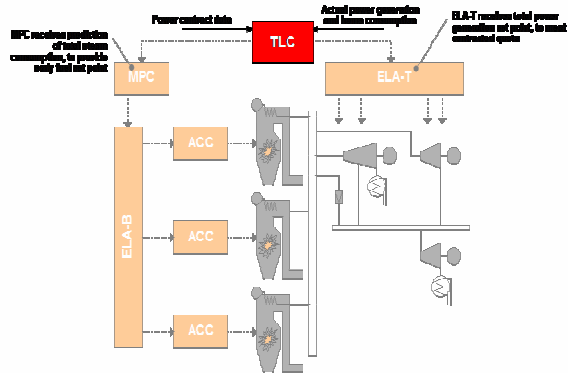
Power delivery to the grid

Work flow

- Prepare 24x7 h contract (sales)
 - Optimize (predicted demand)
 - Validate against technology limits
- Approve/download to database
- Upload / Execute (operators)
- Modify by spot market (via Web)

Algorithmic solution

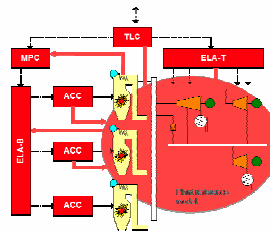
- Diminishing horizon MPC (target optimization)
- Risk sensitive solution



TLC Information links

Reference plant topology model

- Technology configuration
 - Driven by "live" and planned process data
 - What if analysis
- Generation range, costs (ELA)
- Unit commitment optimization



Predictor – trajectories of

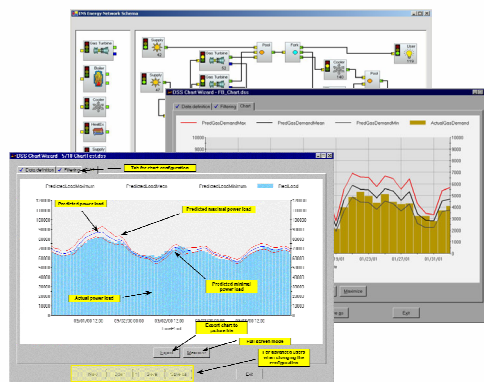
- Process steam demand
- Heating steam demand
- Home consumption

Based on

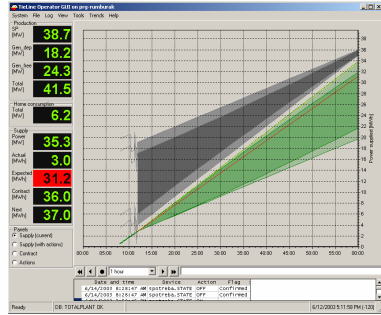
- Historical data (incl. categorical)
- Climatic data (and predictions)

Used in

- Contract optimization – 1 week/1 hour
- Contract execution – 2-3 hours/1-5 min (controlled/dependent resources)



TLC Execution

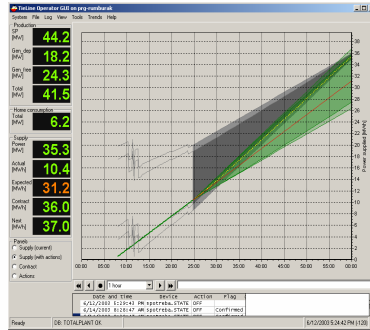


Execution

- Diminishing horizon MPC, cautious strategy
- Minimize risk not to meet contract

Operator intervention

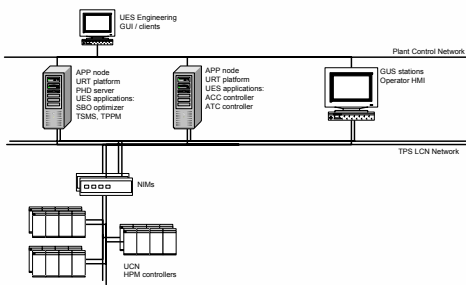
- "Early warning" features
- Start-up/shut-down scenarios
- Decision support - what-if analysis



Uncertainty Sources

- Heating/process steam demand (predictions)
- Home consumption
- Equipment trips

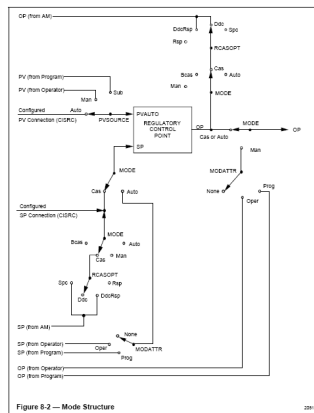
UES integration with DCS



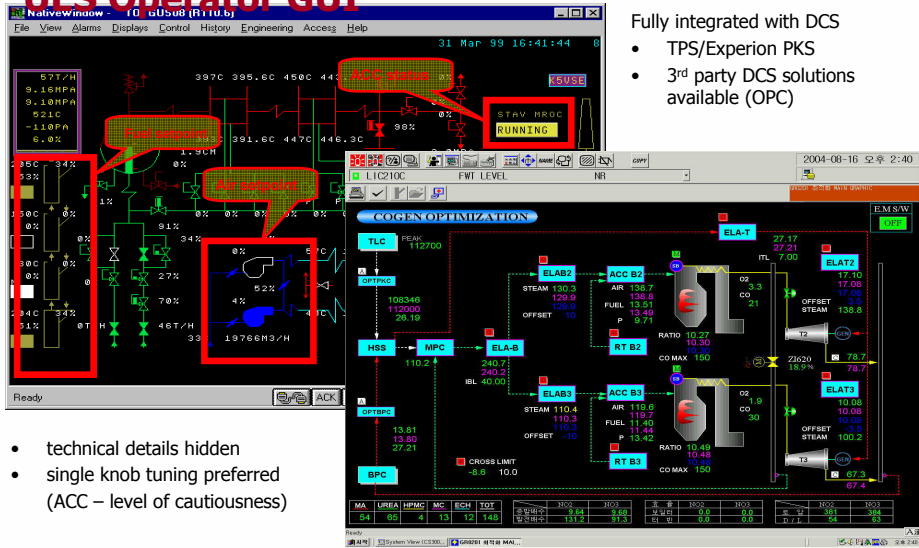
- PC node with URT platform hosts UES controllers
- URT communicates with DCS via transparent OPC server
- GUI for Engineers – PC station
- GUI for Operators – integrated to DCS stations

Security on DCS level

- Local/remote setpoint (AUTO/CAS)
- Shed-time / shed mode concept (CAS / BCAS)



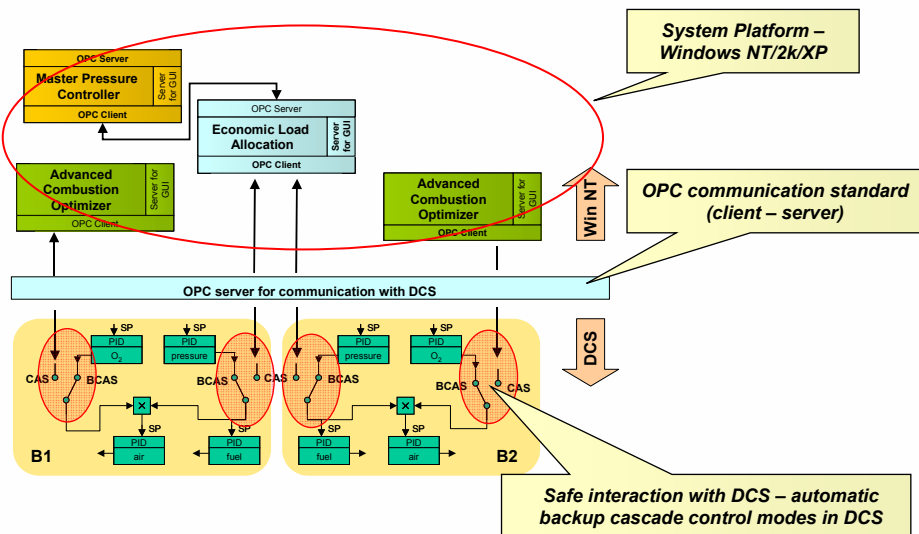
UES Operator GUI



- Fully integrated with DCS
- TPS/Experion PKS
 - 3rd party DCS solutions available (OPC)

- technical details hidden
- single knob tuning preferred (ACC – level of cautiousness)

UES data interface – URT/DCS link

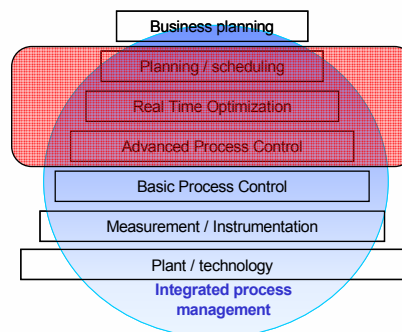


Agenda

- Introduction
 - Advanced Process Control scope, status, drivers
- Unified Energy Solutions (UES) portfolio
 - Architecture, components, technology
- Unified Real Time (URT) platform for advanced control applications
 - Objectives, architecture, benefits
 - Demo of key features
- Conclusions

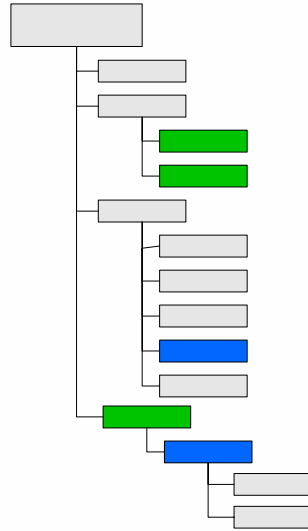
Unified Real Time (URT) Platform - Objectives

- Environment for hosting advanced process-control, real-time optimization and planning/scheduling applications that
 - Are hybrid, large and/or complex
 - Use any DCS for underlying process measurements and regulatory control
 - Involve dynamic configuration, flexible scheduling, complex organization, etc.
- Build on experience with DCS applications
- Provide tight integration with DCS and business control level



URT- Architecture

- Component-based environment built on DCOM technology
- Components are organized in a tree structure
- URT consists of data items, function blocks and schedulers
 - Data items – hold application data or another components
 - Function blocks – provides user-defined functionality
 - Schedulers – units of execution, execute function blocks
- The URT platform is a one process in OS (Windows 2000 or later versions)
- OPC DA and A&E server



URT – Data Subsystem

- Fixed set of elementary data items (leaves)

- Scalars

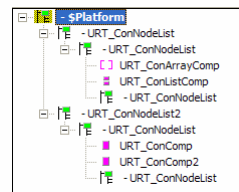
- double, float, time
- short, int, long, enumeration
- string, bool, variant, link

Name	Type	Value	Description
URT_ConFloat	float	-1.40000E12	Float value
URT_ConEnum	enum	ADD	Enumeration
URT_ConString	string	TextText	String value
URT_ConLong	long	54365465	Long value
URT_ConDouble	double	1.2324540E234	Double value
URT_ConTime	time	2005/07/10 20:00.123	Date-time value

- Containers of scalars – array, list, (queue, stack)

- Data items for building structures (nodes)

- Component – keeps a reference to any type of URT component
- Containers of components – array, list



URT – Data Subsystem

- Connections – data item may have input and/or output connection
 - Input connection reads data from remote source (data item)
 - Output connection writes data to remote target (data item)
 - Internal connection – uses native URT protocol
 - External connection – uses OPC protocol

Name	Type	Value	Description	Connection	OutConnection	Timestamp
INOUT	double	0:0	Input/output value	.././PV	.././OP	2005Jul06 22:23:38.441
< end >						

- Buffering – data items have optional buffers
 - Data can be transferred between schedulers (threads) without blocking the execution

URT – Execution and Scheduling

- Basic unit of execution and scheduling is a scheduler component
- Scheduler owns a thread in which all child components are executed
- Schedulers support:
 - Periodical execution
 - Asynchronous on-demand execution
 - Synchronous on-demand execution

Name	Type	Value	Description
urtExecState	enum	INACTIVE	Module execution state
urtPriority	enum	NORMAL	Module execution priority
urtNumOverlaps	long	0	Number of execution overlaps
urtStarTime	time	00:00:00.000	Last time module executed
urtEndTime	time	00:00:00.000	Time to end module execution
urtElapsedTime	time	00:00:00.000	Last module execution duration
urtIntervalCount	long	0	Number of intervals run since platform startup
urtIntervalActual	time	-1:00:00:0...	Module execution interval (actual)
urtInterval	time	-1:00:00:0...	Module execution interval (desired)
urtOffset	time	00:00:00.000	Module execution offset
urtDemand	long	0	Demand module unsynch execution
urtSyncDemand	long	0	Demand module synch execution
urtMaxOverlaps	long	1	Max permitted execution overlaps
urtMaxOverlaps1	long	5	Max permitted execution overlaps first interval (multi...
urtSpeedUp	float	1.00000	Speed up scheduling (for simulation): < 0 run contin...
urtMsgCompID	long	0	Internal parameter - Component ID used to raise me...
PVProc	func blk		PV processing

Scheduler – Execution cycle

- Scheduler's execution cycle consists of three phases (commands)
 - Pre-execute()
 - Execute()
 - Post-execute()
- The commands are propagated through the URT tree (post-order traversal)
- Data items read data in Pre-execute() and write data in Post-execute(). They do not use the Execute() phase.
- Function blocks calculate their outputs in Execute() phase. They usually do not use Pre-execute() and Post-execute() phases.

URT – Function Blocks

- Points of customization
- Prepared base classes for C++, C# and VB.NET languages
- User class must implement OnPostBuild() method and Execute() method.
- OnPostBuild() configures the function block
 - creates all 'local' data items
 - Initializes the function block
- Execute() does the required functionality
 - QP Solver, Data switch, Resampling, ...

```
HRESULT CMulFB::OnPostBuild() {
    m_PVX.SetUp(L"PVX", L"Input X");
    m_PVY.SetUp(L"PVY", L"Input Y");
    m_OP.SetUp(L"OP", L"Output");
    m_VALID.SetUp(L"VALID", L"Valid");
    m_VALID = false;
    return S_OK;
}
HRESULT CMulFB::Execute() {
    m_OP = m_PVX * m_PVY;
    m_VALID = m_OP <= HI_LIMIT;
    return S_OK;
}
```

URT – Function Blocks Contd.

- Base class implements all necessary functionality of a URT Component
- Base class provides as set of functions for the function block
 - Finding a component in the platform
 - Locking of the sub-tree
 - Subscribing of events
 - Generating of messages and events
- URT provides API for platform management (creating platforms, creating/deleting components, browsing platform, ...)

Function Block Example

Name	Type	Value	Description
urtSetFBState	enum	STOP	Set FB active/inactive
urtFBState	enum	STOP	FB run state
urtFBExecState	enum	STOPPED_FB	FB execution state - composite of FB and sched state
urtTimeOutInterval	time	00:00:00.000	Time out interval
urtFBCritical	bool	true	Flag indicating whether FB is critical. If true FB runs ...
PV	double	0	PV to be processed
PVBACKUP	double	0	Backup value for substitution
MAXSUBS	long	0	Maximum number of substitutes
SUBS	long	0	Substitute index
FILTER	struct		Filter parameters
EQUATION	struct		Equation coefficients
OP	double	0	OP

Name	Type	Value	Description
GAIN	double	1.00000000	Equation gain: GAIN * PV + OFFSET
OFFSET	double	0	Equation offset: GAIN * PV + OFFSET
< end >			

URT – Messages and Persistency

- Messages
 - Function blocks can raise messages and events
 - URT messages and events are converted to the standard OPC Alarms and Events and published via OPC A&E server

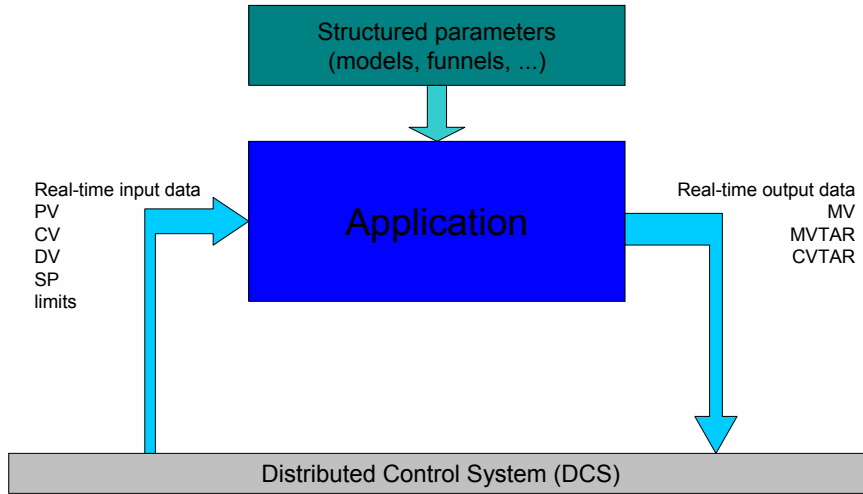
Source	Time	Message	Severity	Condition	AckFi
[All]	[All]	[All]	[All]	[All]	[All]
K4@CASCADE@CASCADE	7/6/2005 10:55:12 PM	Data for slaves are invalid.	WARN - Low	EXECCENDI...	NO
K4@CE@RTDATA@INDAT...	7/6/2005 10:55:10 PM	Could not find connection target ".../RATIO/RTPROC...	WARN - High	BADRESOL...	NO

- Persistency
 - Configuration (tree structure) and the data are stored in one file (checkpoint file) in XML format
 - URT provides built-in function block for automatic periodical saving of checkpoints

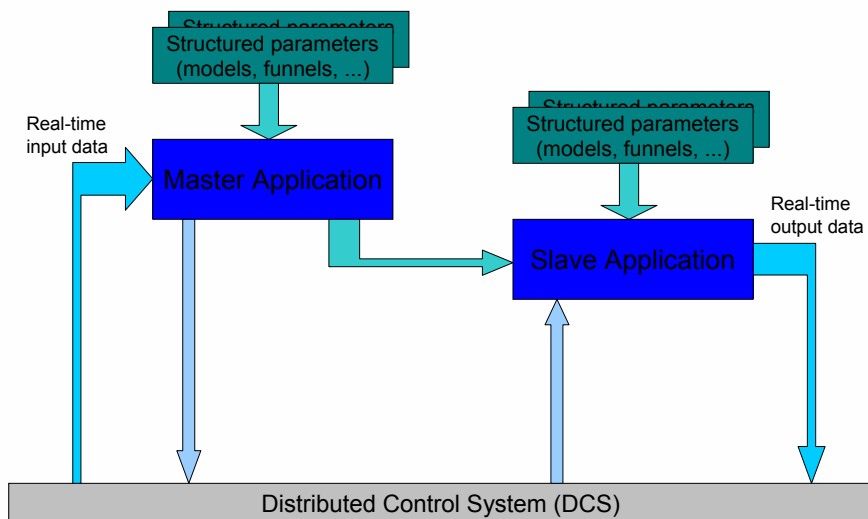
URT - Navigation

- Each component must have a name. Any two sibling components must have different names
- The tree of components can be seen as a XML document, where names of the components in URT are names of elements in XML document
- The URT component can be located using XPath-like queries
- The query can be absolute (including the name of the platform) or relative. An absolute query may point to another platform.
 - Absolute query: (Plaform1)/\$Plaform1/Unit/App/Engine/Item
 - Relative query: ../../Params/HILM
- The query may contain some attributes of the component (e.g. type)
- The queries are used in connections and links

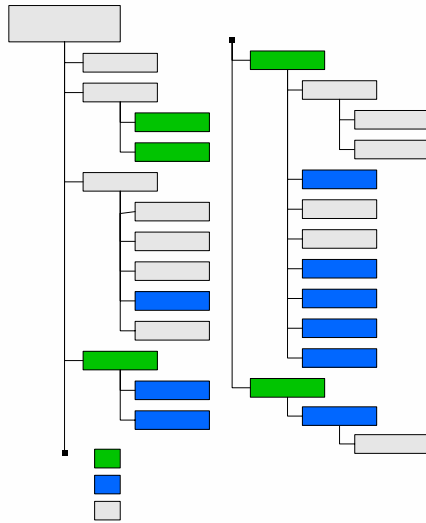
Simple Advanced Control Application



More Complex Advanced Control Application

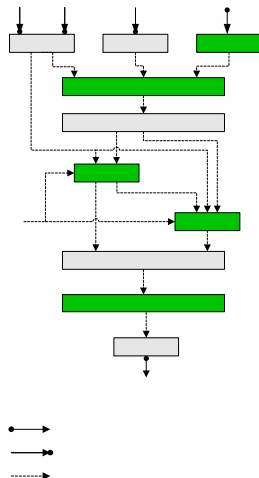


Advanced Control Application on the URT



- CONFIG – configuration data, can be changed off-line only
- RTDATA – external data (underlying DCS)
- PARAMS – application parameters, complex data structures, can be changed on-line
- LIMITS, ENIGNE, CASCADE – independent application modules

Data Flow in ENGINE Module



- Input values are read to input data structures RTIN, PARAMS, HISTORY
- Input values are converted by EUTOPCT function blocks to DATAPCT/IN structure
- Data are process by function blocks EXEC1 and EXEC2
- The results are written to DATAPCT/OUT structure
- Output values are converted by PCTTOEU function blocks and written to output data structure RTOUT

ATION

CONFIG

RTDATA

PARAMS

Application Building

Name	Type	Value
urtSetFBState	enum	STOP
urtFBState	enum	STOP
urtFBExecState	enum	STOPPED_FB
urtTimeOutInterval	time	00:00:00.000
urtFBCritical	bool	true
ROOT	link	../././APP1:B
ACTION	enum	BUILD

- Special function block for building application
- Multi-phase building process
- One function block can create several applications inside the platform

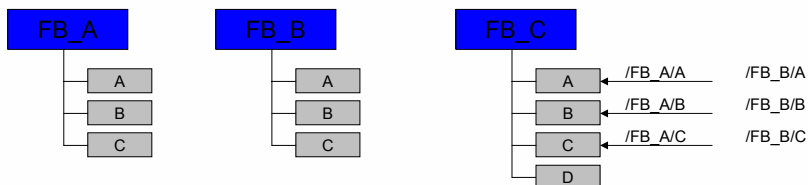
Name	Type	Value
urtSetFBState	enum	STOP
urtFBState	enum	STOP
urtFBExecState	enum	STOPPED_FB
urtTimeOutInterval	time	00:00:00.000
urtFBCritical	bool	true
ROOT	link	../././APP1
ACTION	enum	BUILD
< end >		

38

© V. Havlena, J. Fíndejs, 2006

Indirection Example - Problem

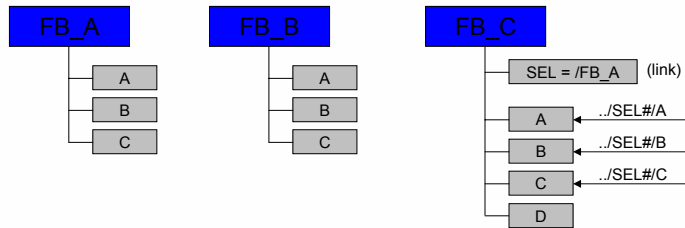
- Scenario
 - An application has two function block FB_A and FB_B, which calculate the same type of outputs and store them in parameters A, B and C
 - The function block FB_C read output values of FB_A or FB_B via input connections and calculates output value D
- Problem
 - How to put information about the source function block to one place?



39

© V. Havlena, J. Fíndejs, 2006

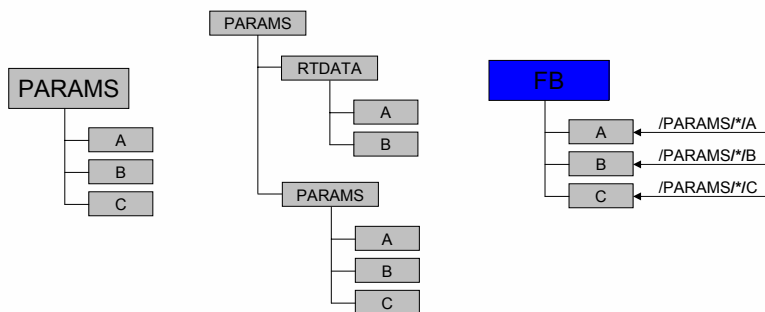
Indirection Example - Solution



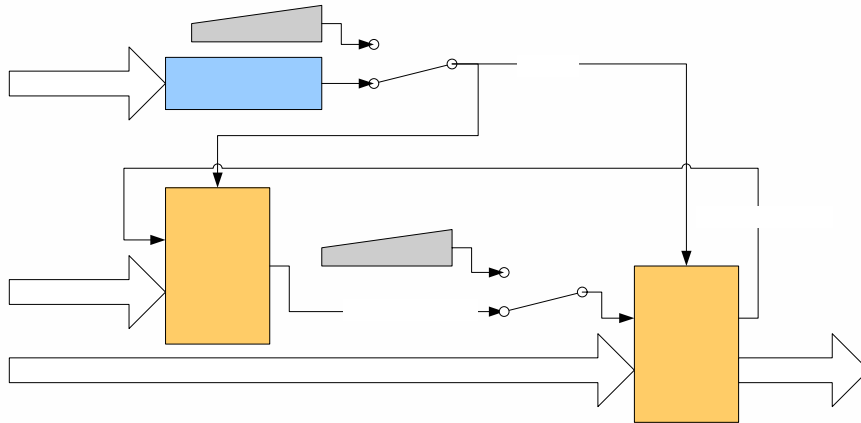
- The function block FB_C creates a link SEL
- All input connection are configured via the link SEL
- The link SEL can point to any URT component having child data items A, B and C

Value Substitution Example

- Scenario
 - An application has a set of parameters
 - The parameters are stored in the structure PARAMS and they are modified by an external application
- Problem
 - How to allow substitution of some parameters by real-time data without modifying the application and the parameter set



URT Demo - Advanced Combustion Controller



Agenda

- Introduction
 - Advanced Process Control scope, status, drivers
- Unified Energy Solutions (UES) portfolio
 - Architecture, components, technology
- Unified Real Time (URT) platform for advanced control applications
 - Objectives, architecture, benefits
 - Demo of key features
- Conclusions

Process Data

Conclusions

- Advantages of component-oriented approach already proven by software community
- Operating systems and general frameworks does not meet the needs of control and optimization applications
- Unified Real Time platform
 - Fully componentized applications
 - Flexible, extensible and distributed environment
 - Rapid application development
 - Standard data interfaces (OPC)
 - Advanced features – real-time reconfiguration, what-if analyses, ...

ARTIST2

Real-Time Motor Controller



Michal Sojka

Czech Technical University

<http://rtlab.felk.cvut.cz>

sojkam1@fel.cvut.cz

ARTIST2

Graduate Course on Embedded Control Systems
Prague, Czech Republic, April 3-7, 2006

DC Motor Controller in Linux

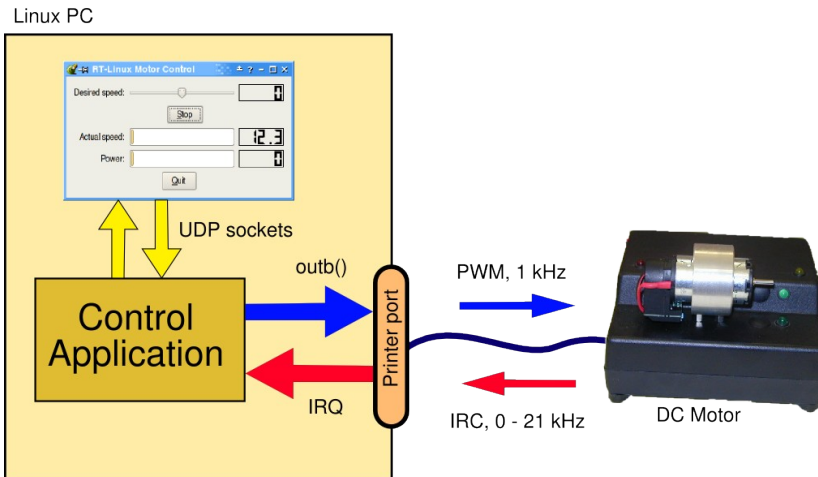
The goal is to create a controller in ANSI C language, which controls the angular velocity of the motor.



03/24/06

© Michal Sojka 2006

Description of the Model



Operating System

- We need:
 - Fast response to interrupt request
 - Accurate timers (sample period, PWM)
 - Real-Time scheduler
 - Direct access to hardware
- Possible solutions:
 - Real-Time OS (RTLinux, vxWorks, ...)
 - Writing a **device driver** for **Linux** with real-time extensions (High resolution timers, real-time preemption patch etc.)
 - Modified Linux kernel for accessing hardware from user-space

Our approach

- Modification to standard Linux kernel
 - Added high resolution timers patch
 - Allow non-root users to:
 - enable real-time scheduling for their processes
 - access I/O ports
- Interrupt handling in user-space
 - VM86 system call
 - Mainly for use by DOS Emulator
- User friendly interface through **libpos** library

Steps to Create a Controller

1. Create a basic C application.
2. Try to rev up the motor at full speed.
3. Write a thread generating PWM signal (period 1 ms)
4. Write an IRQ handler (position measuring).
5. Write a thread measuring the velocity.
6. Implement a velocity controller (PID).
7. Write a graphical interface for the controller.
8. Implement communication with GUI.

Steps to Create a Controller

1. Create a basic C application.
2. Try to rev up the motor at full speed.
3. Write a thread generating PWM signal (period 1 ms)
4. Write an IRQ handler (position measuring).
5. Write a thread measuring the velocity.
6. Implement a velocity controller (PID).
7. Write a graphical interface for the controller.
8. Implement communication with GUI.

A Basic C Application

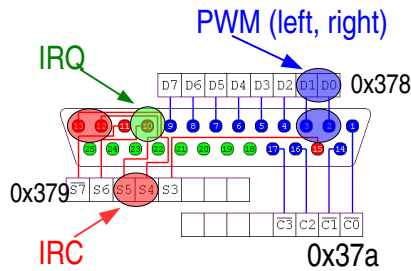
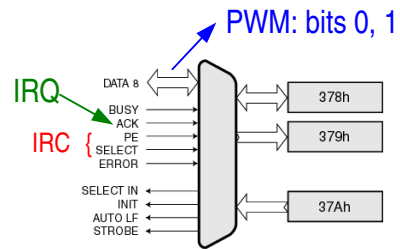
```
#include <stdio.h>

int
main(int argc, char *argv[])
{
    printf("Hello\n");
    return 0;
}
```

- In directory **~/artist2/hello**
 - Compile by command **make**
 - Executable appears in **~/artist2/_compiled/bin**
- Run the compiled application by:
~/artist2/_compiled/bin/hello

Parallel Port

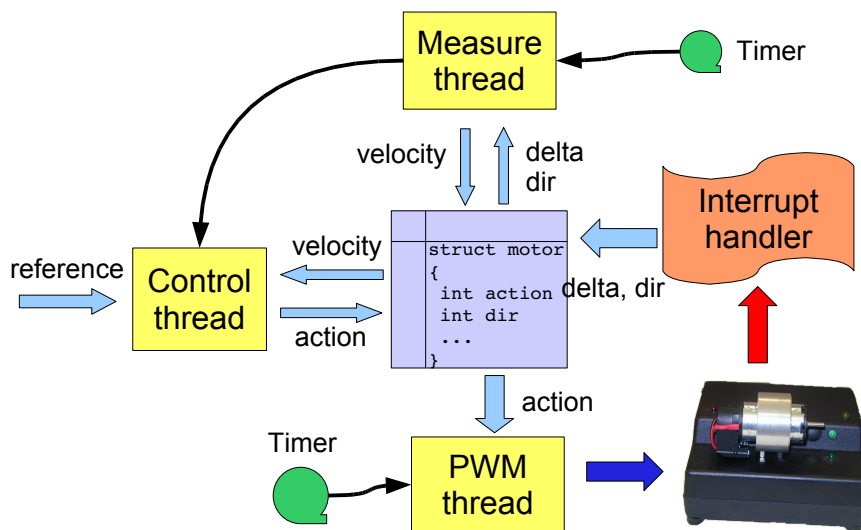
- Motor rotation:
 - left: `outb(1, 0x378);`
 - right: `outb(2, 0x378);`
- IRC signals:
 - `inb(0x379);`



03/24/06

© Michal Sojka 2006

The Structure of Control Application



03/24/06

© Michal Sojka 2006

Periodic Threads

```

#define MS (1000000)

void *thread_func(void *arg)
{
    pthread_make_periodic_np(pthread_self(), gethrtime(), 2*MS);
    while (1) {
        /* do something */
        pthread_wait_np();
    }
    return NULL;
}

int main(void)
{
    pthread_t thr;

    pthread_create(&thr, NULL, &thread_func, NULL);
    return 0;
}

```

start time (now)

period

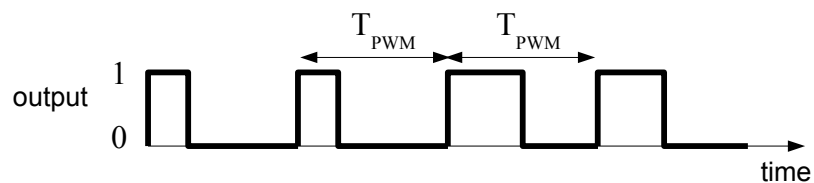
wait for the start of the next period

03/24/06

© Michal Sojka 2006

PWM Generation

- The value of the variable **action** specifies the control action.
- Use the **usleep** function to suspend the thread for a given number of microseconds.
- The PWM period should be set to 1 ms. This is due to the timer resolution (~1 us) and user-space overhead.



```

while (1) {
    set_output (1);
    usleep ( action * T_PWM );
    set_output (0);
    pthread_wait_np ();
}

```

03/24/06

© Michal Sojka 2006

PWM Generation (cont.)

- Constants:
 - **PWM_PERIOD**: a constant containing the period of PWM thread (in **nanoseconds!!!**).
 - **PWM_RESOLUTION**: the maximum value of **action** variable. If **action** equals to it, the output should be always 1.

```
while (1) {  
    set_output (1);  
    usleep ( PWM_PERIOD/1000 * action / PWM_RESOLUTION );  
    set_output (0);  
    pthread_wait_np ();  
}
```

convert to microseconds
↓

03/24/06

© Michal Sojka 2006

Thread Priorities

- Rate Monotonic Priority Assignment
 - the shorter task period the higher assigned priority
- **In Posix**: The higher number the higher priority

```
int init_module(void)  
{  
    pthread_attr_t attr;  
    struct sched_param param;  
  
    pthread_attr_init(&attr);  
    param.sched_priority = 1;  
    pthread_attr_setschedparam(&attr, &param);  
    pthread_create(&thr, &attr, &thread_func, NULL);  
    return 0;  
}
```

the priority of the thread
↙

03/24/06

© Michal Sojka 2006

IRQ Handling

- Register an interrupt handler
 - parallel port: IRQ 7
- Enable interrupt generation by setting a bit in parallel port control register:
outb(0x10, 0x37a);

```
void irq_handler(int intno, void *dev_id, void *regs)
{
    struct motor *motor = (struct motor *)dev_id;

    /* do something */

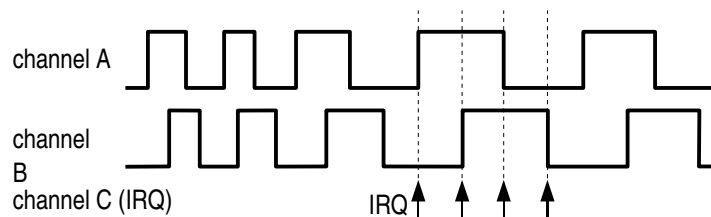
    return 0;
}

status = request_irq(motor->irq, irq_handler, 0, "motor", motor);
```

03/24/06

© Michal Sojka 2006

Signals From an IRC sensor

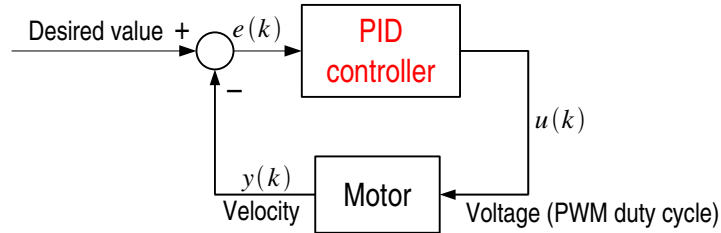


- Whenever the value of any IRC sensor channel changes, electronics in the motor generates the IRQ.
- The motor is equipped by IRC with 100 pulses per turn and there are 4 IRQs per one step. So there are 400 IRQs per turn.

03/24/06

© Michal Sojka 2006

PID Controller



- Control error:
 - `e = motor->reference - motor->velocity;`
- P controller:
 - `motor->action = PWM_RESOLUTION * P * e;`
- PID controller:
 - $$u(k) = P \cdot e(k) + I \cdot \sum_{i=0}^{k-1} e(i) + D \cdot (e(k) - e(k-1))$$

03/24/06

© Michal Sojka 2006

How to Start

1. In the boot menu chose **ARTIST2 Linux**
2. Log in as **artist<number>**, password **realtime**
3. Go to the directory: `cd ~/artist2`
4. Compile everything: `make`
5. Start GUI application: `./gui`
6. Start the controller: `~/artist2/_compiled/bin/motor`
7. Exit by **Ctrl-C**
8. Go to controller directory: `cd motor/src`
9. Open file **motor.c** in editor (Kate) and modify it.
10. Compile modified program: `make`
11. Run it: `~/artist2/_compiled/bin/motor`

03/24/06

© Michal Sojka 2006

Content of Directories

- **motor/src** – the code for the controller part
 - motor.c – the code of application (you will modify this file)
 - motor.h – common declarations for both RT and US part
 - Makefile – commands for compilation.
 - gui – script for starting the GUI application
- **motor/qtmotor** – graphical user-space interface
- **motor/curmotor** – text-based user-space interface
- **libpos** – periodic threads and interrupt emulation library

Your Tasks

- Extend the PWM thread to generate PWM signal based on the value `motor->action`.
- Implement a controller.
 - start with a P-controller which computes action as $action = K_p * (reference - velocity)$
 - Experiment to find the value of K_p
 - Extend the controller to PI. In the simplest case, you'll need to store the sum of errors.
- You may try to do other extensions – windup handling, use fixed-point arithmetic, use better implementation of PID, etc.

Debugging

- Inside the code use the `printf()` function to print the values you are interested in.
`printf("Value of action: %d\n", action);`

Friday 7th of April



ARTIST2

TORSCHÉ Scheduling Toolbox for Matlab

Přemysl Šůcha and Michal Kutil

Czech Technical University, Department of Control Engineering

Karlovo náměstí 13, 121 35 Prague 2, Czech Republic



{suchap,kutilm}@fel.cvut.cz

<http://dce.felk.cvut.cz/sucha>

<http://www.tim.cz>

Session Outline

- TORSCHÉ Introduction
- TORSCHÉ Quick Start
- Iterative Algorithms Scheduling
- Outlook

TORSCHÉ Introduction

TORSCHÉ (Time Optimization of Resources, SCHEDuling) is Matlab based toolbox for Scheduling.

Aim of the toolbox:

- rapid prototyping of scheduling algorithms
- co-design of control and scheduling problems
- repository of off-line and on-line scheduling algorithms
- open for new algorithms
- demonstration tool for education

TORSCHÉ – Problem Representation

Maltlab objects are used to represent large variety of scheduling problems.

Main objects of the toolbox:

- *task* - parameters of task
- *taskset* - object encapsulating set of tasks
- *problem* - specification of problem (Błażewicz notation)
- *graph* - representation of graph

TORSCHÉ – Algorithms

The toolbox algorithms structured into several groups.

Groups of the toolbox function:

- functions for manipulation with objects of the toolbox
- scheduling algorithms (List scheduling, EDD, ...)
- graph algorithms (Floyd's algorithm, ...)
- supplementary algorithms (ILP, MIQP, ...)
- GUIs (graphedit, ...)

Session Outline

- TORSCHÉ Introduction
- *TORSCHÉ Quick Start*
- Iterative Algorithms Scheduling
- Outlook

TORSCHÉ Quick Start

Steps to solve scheduling problems:

1. definition of tasks
2. definition of taskset
3. problem definition
4. scheduling

Example – Horn's algorithm

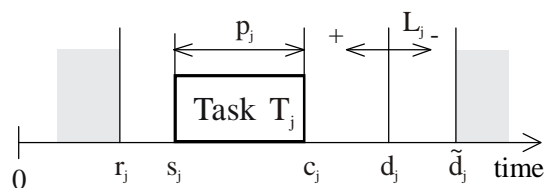
Problem: $1|pmtn,r_j|L_{max}$

$T = \{t_1, t_2, t_3\}$

$p = \{5, 2, 3\}$

$r_j = \{1, 0, 5\}$

$d_j = \{12, 11, 9\}$



Objective: minimize maximum lateness $L_{max} = \max\{L_j\}$

Algorithm: Horn's algorithm [Horn74]

Definition of Tasks

Task is defined by command *task*, for example:

```
>> t1 = task('task1', 5, 1, inf, 12)
```

```
Task "task1"
```

```
Processing time: 5
```

```
Release time: 1
```

```
Due date: 12
```

This command defines task with name "task1", processing time 5, release time 1, without deadline (inf) and due date 11. Other tasks can be defined in the same way:

```
>> t2 = task('task2', 2, 0, inf, 11);
```

```
>> t3 = task('task3', 3, 5, inf, 9);
```

Definition of Taskset

Set of tasks is created by command *taskset* :

```
>> T = taskset([t1 t2 t3])
```

```
Set of 3 tasks
```

For short:

```
>> T = [t1 t2 t3]
```

```
Set of 3 tasks
```

Problem Definition

Classification of deterministic scheduling problems:

- notation proposed by [Graham79] and [Błażewicz83]
- special problems, not specified by the notation (e.g. m-DEDICATED)

```
>> p = problem('1|rj,pmtn|lmax')  
1|pmtn,rj|lmax
```

Scheduling

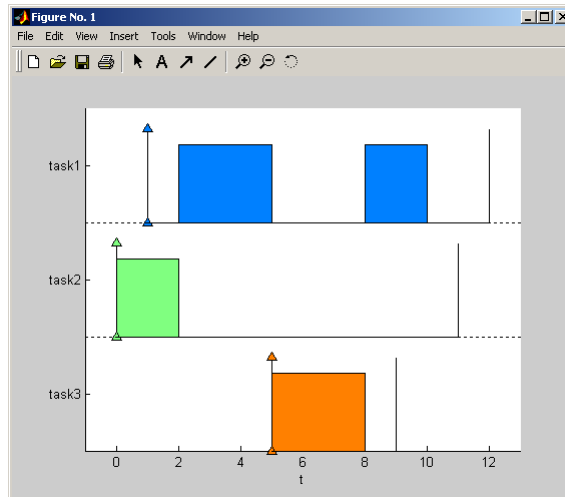
Now we can run the scheduling algorithm, for example Horn's algorithm:

```
>> TS = horn(T,p)  
Set of 3 tasks  
There is schedule: Horn's algorithm  
Solving time: 0.016s
```

Graphical representation of the schedule (Gantt chart) can be displayed using command *plot* :

```
>> plot(TS,'proc',0)
```

Schedule – Gantt Chart



Session Outline

- TORSCHÉ Introduction
- TORSCHÉ Quick Start
- *Iterative Algorithms Scheduling*
- Outlook

Iterative Algorithms Scheduling

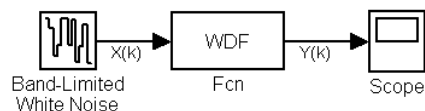
Cyclic scheduling – tasks are repeated in K iterations

Periodic schedule – tasks are repeated periodically with constant period w

Objective – to find a periodic schedule with minimal period w

Example – Cyclic Scheduling

Wave Digital Filter (WDF):



```

for k=1 to N do
  a(k) = X(k) + e(k-1)      %T1
  b(k) = a(k) - g(k-1)     %T2
  c(k) = b(k) + e(k)       %T3
  d(k) = gamma1 * b(k)     %T4
  e(k) = d(k) + e(k-1)     %T5
  f(k) = gamma2 * b(k)     %T6
  g(k) = f(k) + g(k-1)     %T7
  Y(k) = c(k) - g(k)       %T8
end

```

Hardware: one addition and one multiplication unit on a FPGA architecture with floating-point units

floating-point unit	processing time [clk]	latency [clk]
addition (+)	1	1
multiplication (*)	3	3

Problem Statement

Dedicated tasks – tasks are assigned to specified processor
(i.e. floating-point unit)

Instance representation: Cyclic Data Flow Graph (CDFG)

single operation \approx task \approx node in the CDFG

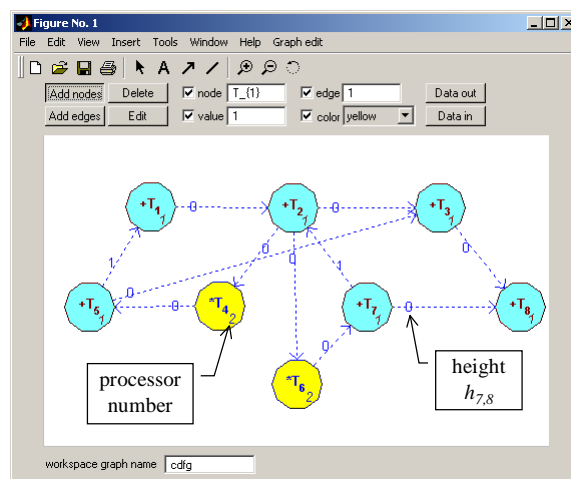
precedence constraints between operations \approx edges
weighted by height h_{ij} (dependence distance)

>> graphedit

Cyclic Data Flow Graph of WDF

```

for k=1 to N do
  a(k) = X(k) + e(k-1)  %T1
  b(k) = a(k) - g(k-1) %T2
  c(k) = b(k) + e(k)    %T3
  d(k) = gamma1 * b(k) %T4
  e(k) = d(k) + e(k-1) %T5
  f(k) = gamma2 * b(k) %T6
  g(k) = f(k) + g(k-1) %T7
  Y(k) = c(k) - g(k)   %T8
end
  
```



Parameters of Processors

Parameters of processors (floating-point units):

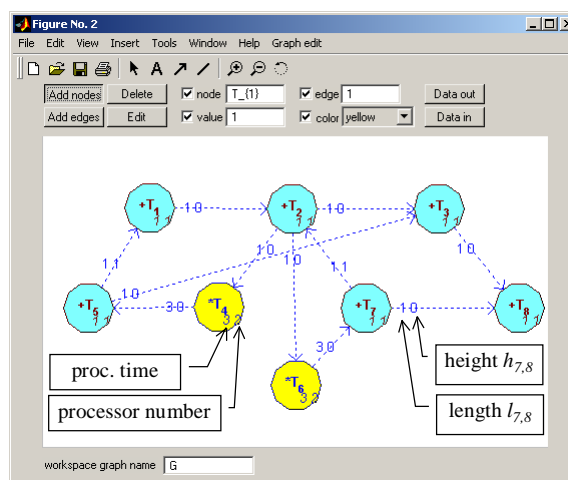
- processing time – processor occupation time
- latency – length l_{ij} (minimal distance between tasks), i.e. processing in pipeline

Scheduling problem is represented by graph G where edges are weighted by couple (l_{ij}, h_{ij}) .

```
>> UnitProcTime=[1 3];
>> UnitLatency=[1 3];
>> G = cdfg2LHgraph(cdfg,UnitProcTime,UnitLatency);
>> graphedit(G)
```

Note: Floating point units are considered non-pipelined only for simplicity reasons.

Graph G of WDF



Critical Circuit

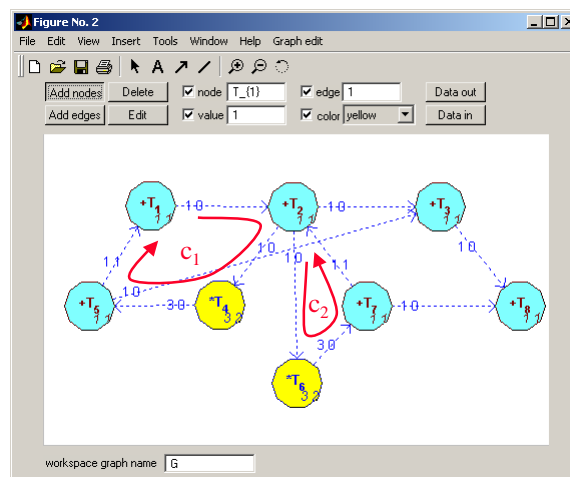
- critical circuit in graph G determines minimal feasible period w with respect to precedence constraints
- problem assumes graph G where edges are weighted by a couple of constants length l_{ij} and height h_{ij}
- objective is to find the critical circuit ratio defined as:

$$r = \max_{c \in C(G)} \frac{\sum_{e_{ij} \in c} l_{ij}}{\sum_{e_{ij} \in c} h_{ij}}$$

where C is a circuit of graph G .

- circuit C of graph G with maximal circuit ratio r is the critical circuit

Critical Circuit - WDF



Critical Circuit Ratio

Graph G contains two circuit c_1 and c_2 with circuit ratio:

$$r(c_1) = (1+1+3+1)/(0+0+0+1) = 6$$

$$r(c_2) = (1+3+1)/(0+0+1) = 5$$

Critical circuit is c_1 , therefore period $w \geq 6$. Critical circuit ratio can be evaluated in the toolbox using command:

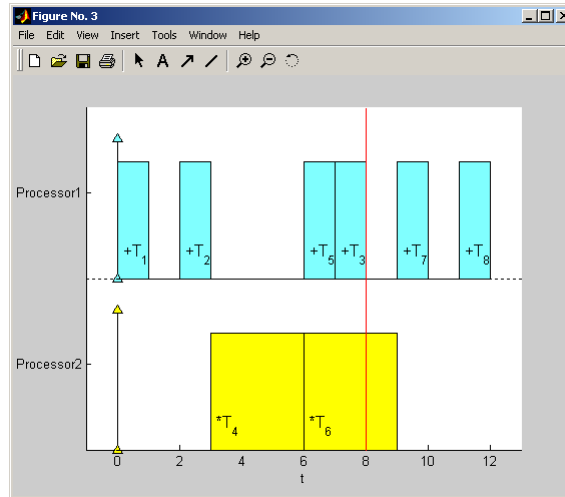
```
>> critical_circuit_ratio(G)
ans =
     6.0000
```

Solution

Graph G can be directly transformed to the taskset:

```
>> T=taskset(G)
Set of 8 tasks
There are precedence constraints
>> prob=problem('m-DEDICATED');
>> schoptions=schoptionsset('ilpSolver','glpk');
>> TS=mdcycsch(T, prob, 1, schoptions)
Set of 8 tasks
There are precedence constraints
There is schedule: MONOCYCSCCH-ILP based algorithm (integer)
Tasks period: 8
Solving time: 0.126s
Number of iterations: 4
>> plot(TS,'prec',0)
```

Resulting Schedule



Solution Summary

```
>> graphedit
>> UnitProcTime=[1 3];
>> UnitLattency=[1 3];
>> G = cdfg2LHgraph(cdfg,UnitProcTime,UnitLattency);
>> T=taskset(G);
>> prob=problem('m-DEDICATED');
>> schoptions=schoptionsset('ilpSolver','glpk');
>> TS=mdcycsch(T, prob, 1, schoptions);
>> plot(TS,'prec',0);
```

Session Outline

- TORSCHÉ Introduction
- TORSCHÉ Quick Start
- Iterative Algorithms Scheduling
- *Outlook*

Outlook

Currently we are working on:

- automatic code generation for Handel C and TrueTime
- real-time schedulability analysis
- new graph and optimization algorithms

More Materials

TORSCHÉ Scheduling Toolbox for Matlab with a complete documentation can be downloaded at:

<http://rttime.felk.cvut.cz/scheduling-toolbox/>

ARTIST2

Implementing Floating-Point DSP and Control with PicoBlaze Processors

Jiří Kadlec

CTU Prague

Pod vodárenskou věží 4

www.zs.utia.cas.cz kadlec@utia.cas.cz

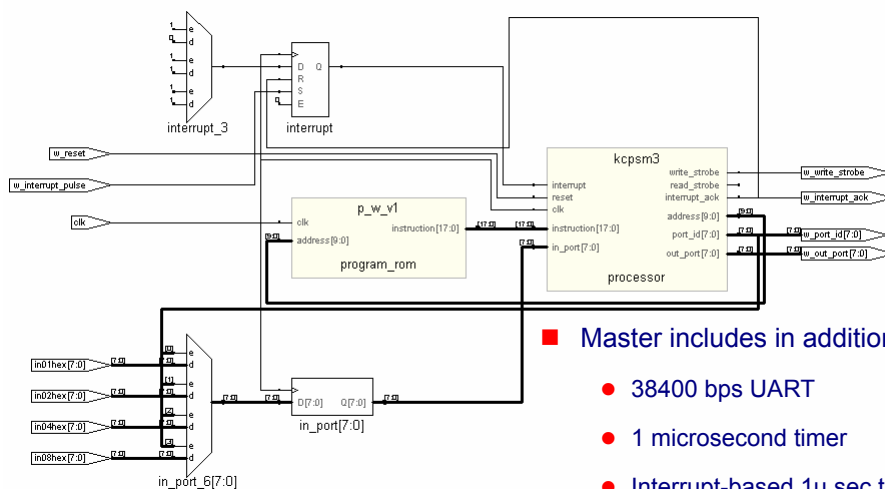
Presentation outline

- **PicoBlaze KCPSM3 processor from Ken Chapman Xilinx**
- **One PicoBlaze Master and four Workers connected by DP BRAMs**
- **Demo1: Four Bouncing Ball on a VGA – 5 PicoBlaze on XC3S200**
- **Scalable pipelined Floating point**
- **Bit-exact high level simulation in Simulink**
- **Demo2: 400 M Flop (18-bit FP) parallel vector products**
- **Power/area: Virtex2, Spartan3, Spartan3L, Spartan3E**
- **Conclusions**

PicoBlaze KCPSM3 processor

- Author: Ken Chapman, Xilinx; chapman@xilinx.com
- VHDL Core with assembler: free from www.xilinx.com
- Main parameters:
 - 8 bit CPU, 1 BRAM 1024x18 for program, only 96 slices (5% of xc3s200)
 - 16 registers, Scratch pad memory 64 byte, 8bit I/O bus, 8bit port address
 - all instructions take constantly 2 clock cycles, 1 level of interrupt
 - KCPSM3 includes Assembler, RS232 macros and uart_clock demo.
 - Optimized for Virtex E, Virtex 2, and Spartan 3
- Our design is reusing parts of Ken's uart_clock demo
- We add inter-processor connect, VGA support and Floating point HW
- We add hazard free access to DP BRAM from Master and Worker PicoBlaze

Worker: 4 input ports, 8 (max 256) output ports

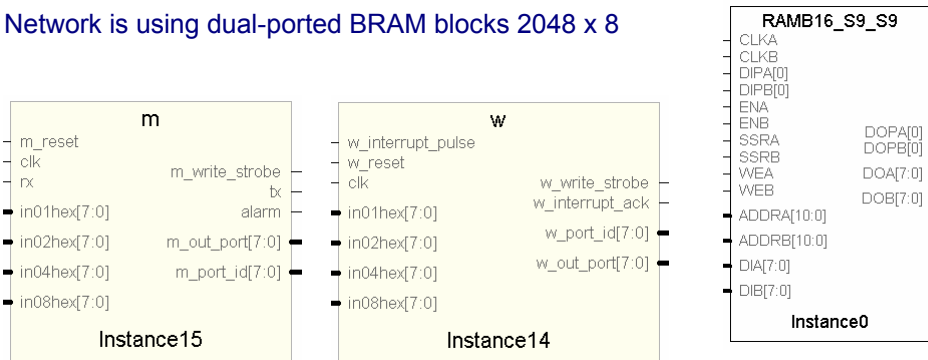


- Master includes in addition:

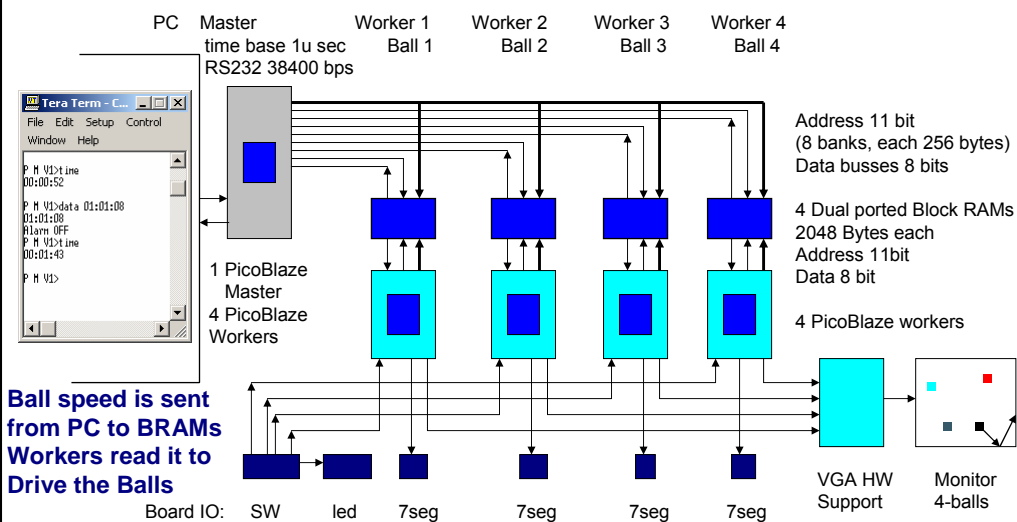
- 38400 bps UART
- 1 microsecond timer
- Interrupt-based 1u sec time base

Master and Worker macros and connectivity:

- Workers provide asynchronous interrupt input and interrupt_ack output
- Master encapsulates serial 38400 bps UART with basic Ken's SW support
- Each processor includes 1 BRAM with local program
- Network is using dual-ported BRAM blocks 2048 x 8

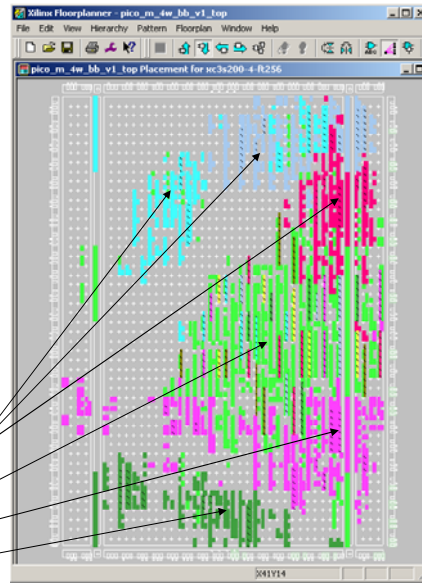


Demo 1: Four Bouncing Balls, VGA, 5 PicoBlaze net



Demo1: 4 Bouncing Balls

- Spartan3 xc3s200-4-ft256
 - Slices 962 out of 1920 50%
 - BRAMS 9 out of 12 75%
 - System clock 50 MHz
 - Up to 5x25MIPs 125 MIPs
 - Interrupt latency 4 clk
 - Master DP SRAM 4 x 2048 Byte
 - Worker DP SRAM 2048 Byte
 - HW avoids write conflicts (No hazard in case of parallel WR to same address).
-
- Power estimate:
 - Dynamic 56 mW
 - Quiescent 61 mW
 - Total 117 mW
 - Worker 1,2,3
 - VGA support
 - Master
 - Worker 4



7

© Jiří Kadlec 2006

Generic Short Latency Floating Point Macros

- Based on Celoxica DK 1.1 Handel-C Floating Point Library
- Precisions <total_length>m<mantissa>: 18m11, 24m17, 32m23, 36m27
 - ADD/SUB, MUL 2 stage pipelined (retimed)
 - FIXPT2F, F2FIXPT 4 stage pipelined (retimed)
 - DIV, SQRT Sequential. No of cycles = mantissa width + 2

Used in
Final FP
Vector
Product
Demo 2

32 bit Pipelined Floating Point Macros

- Based on Celoxica DK 3.1 Handel-C Pipelined Floating Point Library
- 32 bit Precisions <total_length>m<mantissa>: 32m23
 - ADD/SUB 10 stage pipelined MUL 7 stage pipelined
 - FIXPT2F 12 stage pipelined F2FIXPT 14 stage pipelined
 - DIV 28 stage pipelined SQRT 27 stage pipelined

8

© Jiří Kadlec 2006

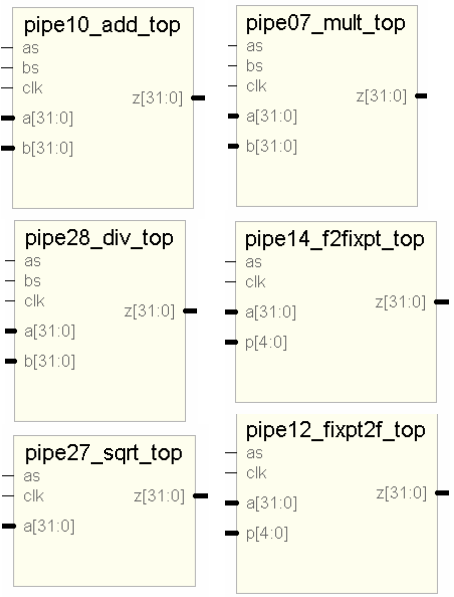
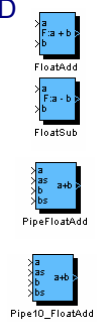
Modeling & RTL

Source code
in Handel C

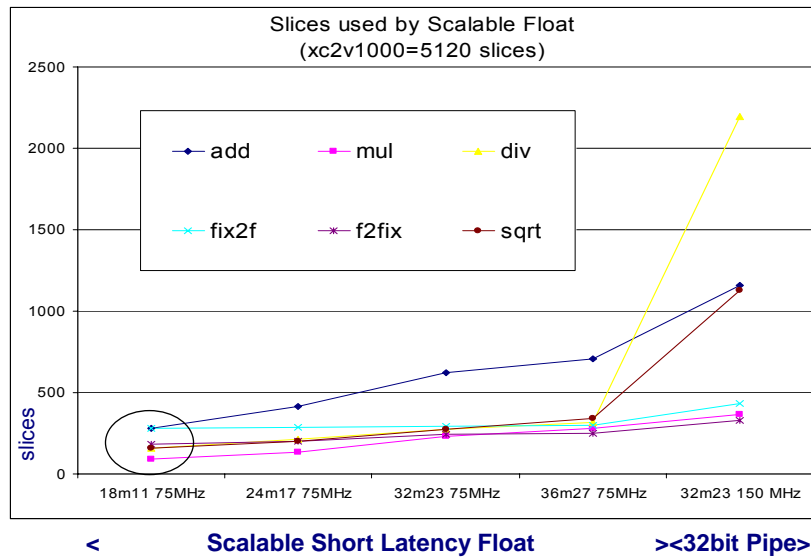
RTL level
DK4
Simulator
VHDL
C++ Simulink
S-functions

■ Example for 32bit FP ADD

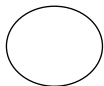
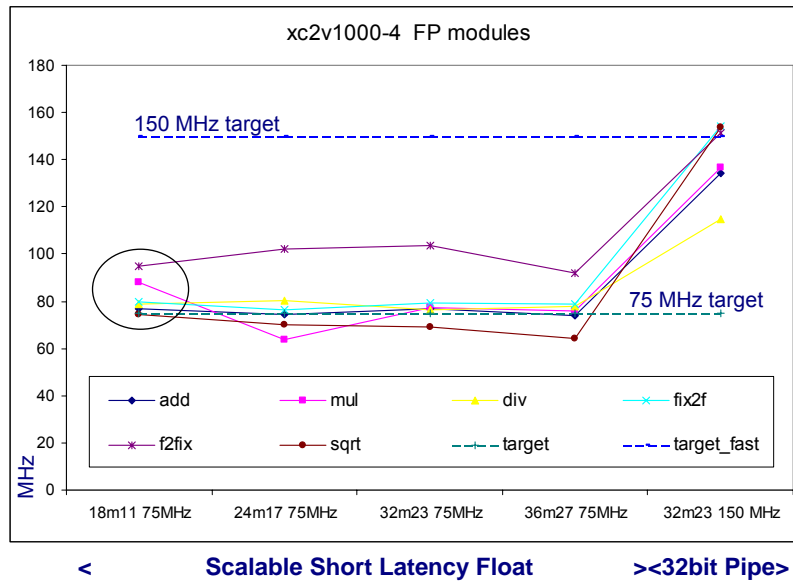
- Simple FP ADD and SUB (bit exact)
- ADD/SUB one block (bit exact)
- ADD/SUB one block (bit and cycle exact)



Area



Speed

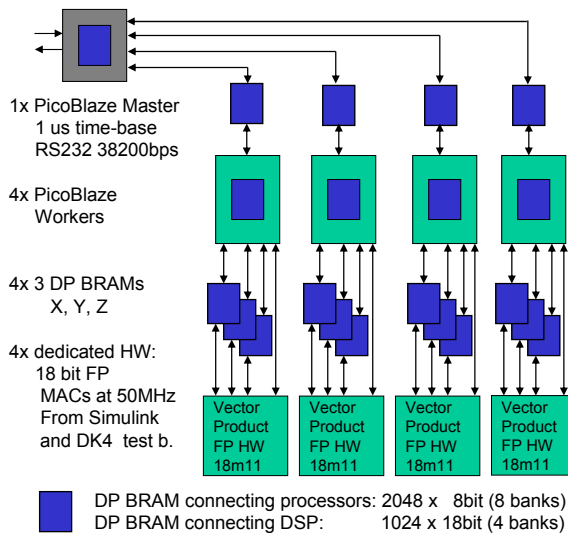


Used in Final FP Vector Product Design

< Scalable Short Latency Float >>32bit Pipe<

Demo 2: 400 M Flop (18-bit FP) vector product

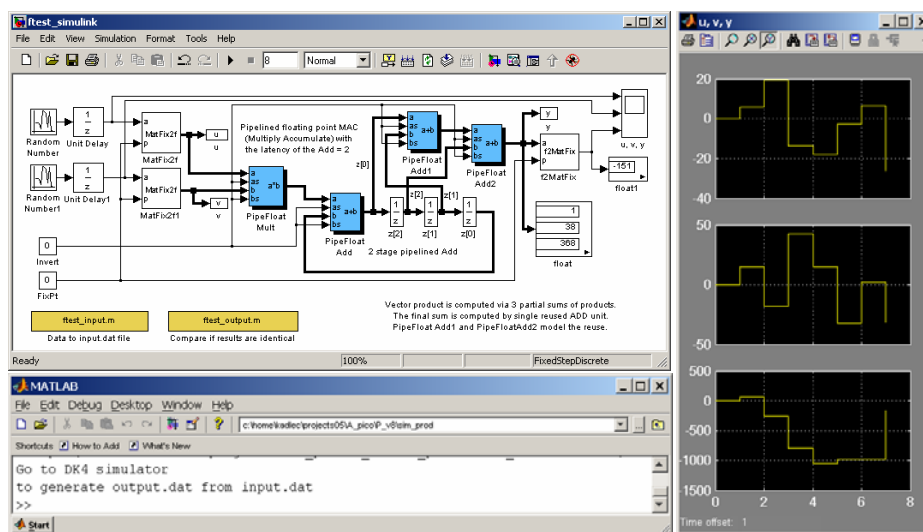
- Spartan3 xc3s1000-4, L, E
- Virtex2 xc2v1000-4
- 50 MHz clock
- 125 MIPs
- 4x 100 M FLOP
FP Mantissa 11 bit,
FP Exponent 6 bit,
FP Sign 1 bit
- DSP program:
Wait for signal;
 $Z[0]=X'[0:255]*Y[0:255];$
interrupt worker;



Design and verification strategy for FP DSP modules

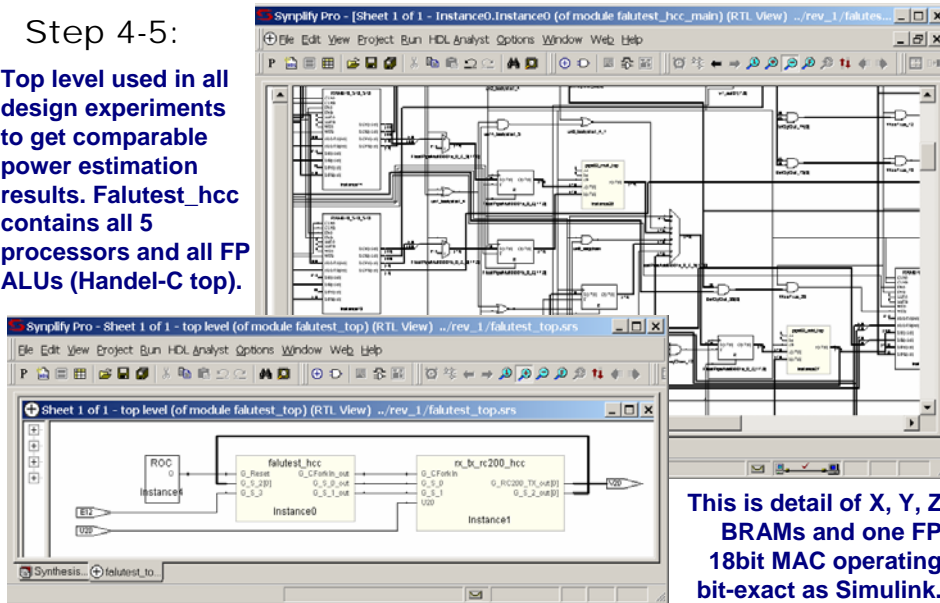
- **Step 1: Bit exact model in Simulink. Verification with Double. Create test data.**
- **Step 2: Simulation of identical HW (hand coded in Handel-C) in DK4 Software simulator (I/O functions automate connection to Matlab)**
- **Step 3: Compilation from DK4 to HW kit to verify on real HW. Kit specific versions of same I/O functions automate connection of the HW kit to Matlab without the need to modify code (parallel port in the case of RC200E).**
- **Step 4: Isolate debugged DSP design (BRAM -> do DSP -> BRAM) as module.**
- **Step 5: Attach these verified DSP modules with PicoBlaze.**
- **Step 6: Verify the DSP module first on one PicoBlaze worker with mem dump support from the Master. Use test data from Step 1**
- **Step 7: Extend your DSP design to multiple workers, large data sets and real time constrains. Concentrate on SW to manage combinations of DSP blocks.**

Step1-3: Bit exact model in Simulink and debugging.



Step 4-5:

Top level used in all design experiments to get comparable power estimation results. Falutest_hcc contains all 5 processors and all FP ALUs (Handel-C top).



This is detail of X, Y, Z BRAMs and one FP 18bit MAC operating bit-exact as Simulink.

Step 6: Integrate and test with PicoBlaze on HW

Simulink test bench generates data which can be used by DK4 simulator, HW board for verification on the HW kit (RC200E with XC2V1000-4 in our case). Finally to target PicoBlaze network, data are generated in format compatible with 18-bit wide BRAMS X,Y and Z:

This is Mem Dump managed by PicoBlaze net on rc200e hw.

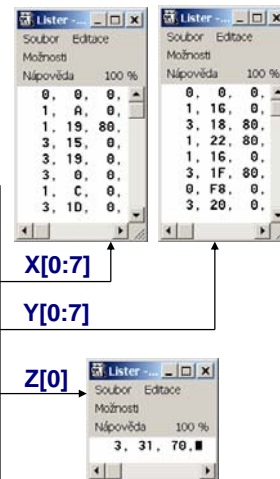
It prints test vector data and result of vector prod. identical with Simulink.

```

Tera Term - COM1.VT
File Edit Setup Control Window Help
w4:010: 00 00 00 01 0a 00 01 19 80 03 15 00 03 19 00 03
w4:020: 00 00 01 0c 00 03 1d 00 00 00 00 00 00 00 00
w4:030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
w4:040: 00 00 00 01 16 00 03 18 80 01 22 80 01 16 00 03
w4:050: 1f 80 00 f8 00 03 20 00 00 00 00 00 00 00 00
w4:060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
w4:070: 03 31 70 00 00 00 00 00 00 00 00 00 00 00 00
w4:080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
w4:090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

P M U1>time
00:04:27

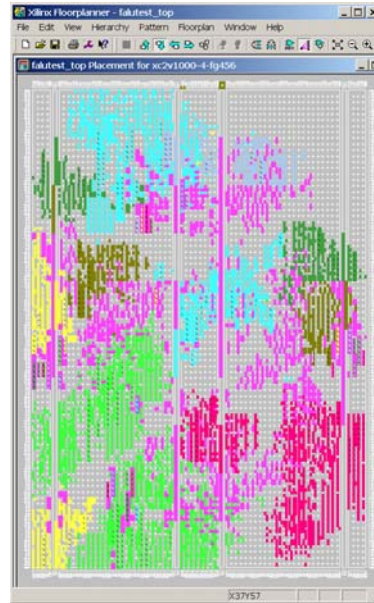
P M U1>
    
```



Finally, OK on HW :-)

Step 7: Real vector product 400mflop Virtex2 xc2v1000-4-fg456

■ Slice Flip Flops	2905	28%
■ 4 input LUTs	4241	42%
■ Occupied Slices	3292	64%
■ BRAMS	21	52%
■ MULT18x18s	4	10%
■ Clock 50 MHz	ISE: 53,3 MHz	
■ Power (Xpower setting has been verified by measurement of case temperature):		
■ Vccint Dynamic	666 mW	
Quiescent	18 mW	
■ Vccoux Dynamic	0 mW	
Quiescent	330 mW	
■ Vcco Dynamic	3 mW	
Quiescent	3 mW	
■ Total	1020 mW	

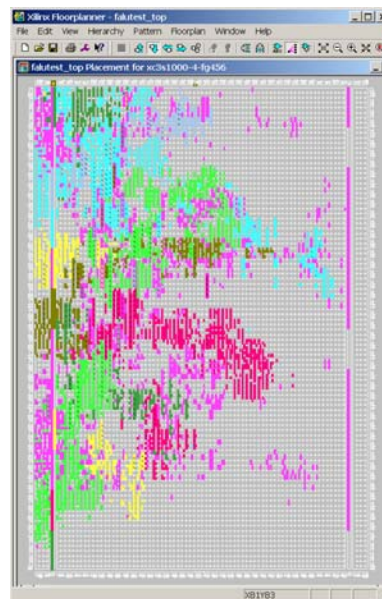


17

© Jiří Kadlec 2006

Step 7: Real vector product 400mflop Spartan3 xc3s1000(L)-4-fg456

■ Slice Flip Flops	2637	17%
■ 4 input LUTs	4424	28%
■ Occupied Slices	3097	40%
■ BRAMS	21	87%
■ MULT18x18s	4	16%
■ Clock 50 MHz	ISE: 50,6 MHz	
■ Power estimate (X_power) S3 S3L		
■ Vccint Dynamic	92,8 mW	91 mW
Quiescent	78 mW	36 mW
■ Vccoux Dynamic	0 mW	0 mW
Quiescent	62 mW	62 mW
■ Vcco Dynamic	1 mW	1 mW
Quiescent	0 mW	0 mW
■ Total	235 mW	191 mW



18

© Jiří Kadlec 2006

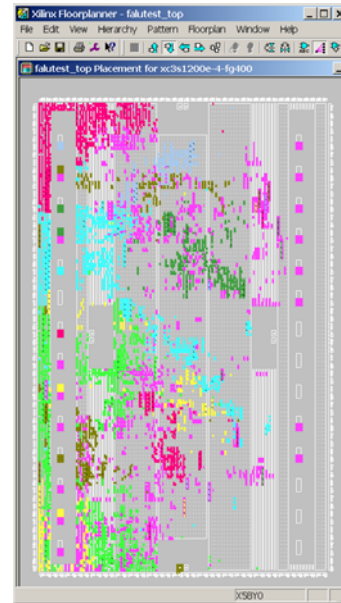
Step 7: Real vector product 400mflop Spartan3E xc3s1200E-4-fg400

■ Slice Flip Flops	2829	16%
■ 4 input LUTs	4440	25%
■ Occupied Slices	3136	36%
■ BRAMS	21	75%
■ MULT18x18s	4	14%
■ Clock 50 MHz	ISE: 50,1 MHz	

■ Power estimate is not available yet in X power tool.

■ The complete 4x 100 M FLOP Vector product with 5 PicoBlaze processors has been implemented and tested on RC200E board from Celoxica with the Virtex 2 XC2V1000-4 part, running at 50MHz.

■ Spartan 3 designs have been all compiled but not tested on real HW.



19

© Jiří Kadlec 2006

Conclusions

■ 5 PicoBlaze Architecture ++

■ It is compatible with our design strategy for DSP modules:
Simulink model -> DK4 debug -> HW debug -> Reuse in PicoBlaze net.

■ PicoBlaze is small and simple, hence manageable.

■ 5 PicoBlaze Architecture --

■ Currently implemented conversion of data formats (8bit - 18bit) is slow.

■ Spartan 3(L) power reduction ++

■ Spartan3(L) is 5x reducing power consumption comparing to Virtex2.

■ Spartan3E is most likely choice for our designs based on PicoBlaze net.

20

© Jiří Kadlec 2006