F2

Formal Methods in System and MpSoC Performance Analysis and Optimisation

DATE 08 TUTORIAL NOTES

Formal methods in system and MpSoC performance analysis and optimization

Part 1 - introduction to formal platform performance analysis

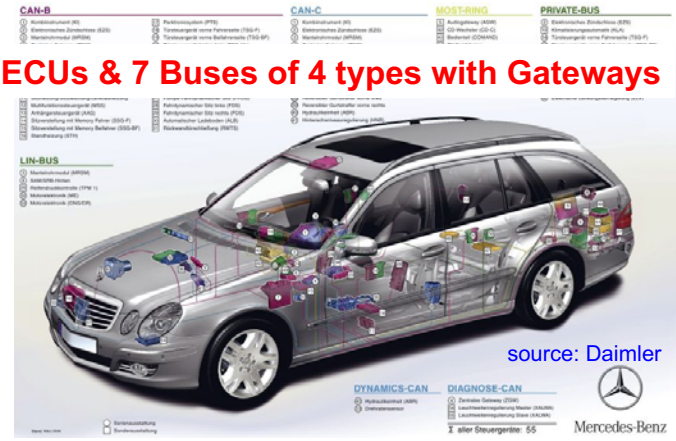
Rolf Ernst, TU Braunschweig

Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

Formal Model Applications

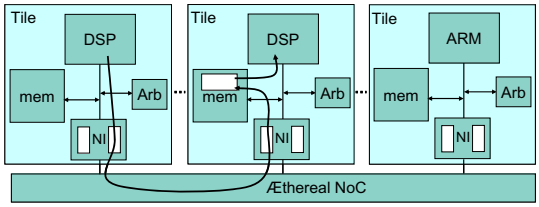
- formal models for performance analysis optimization are in use for very different types of embedded system
  - distributed networks



55 ECUs & 7 Buses of 4 types with Gateways

source: Daimler

  - MpSoC



source: Bekooj – this tutorial

Applications of Formal Methods for Performance

- architecture design in early design phases
- design verification
- control and optimize design „robustness“ throughout the design process

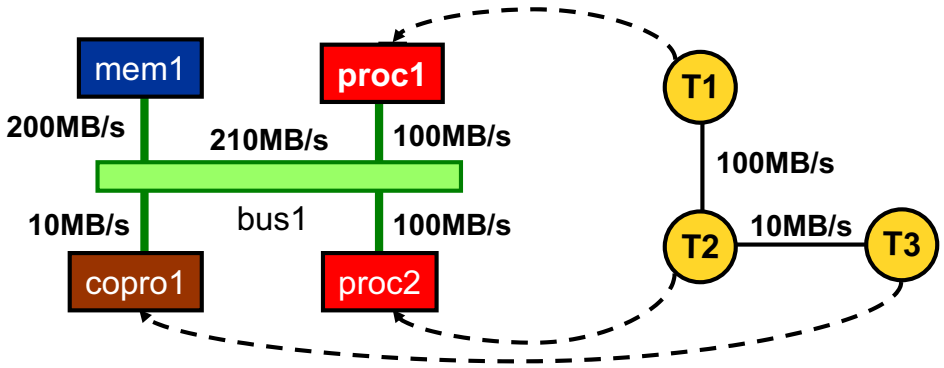
Current Practice in Early Design Space Exploration

- early design phases
  - determine architecture and communication
  - executable code not yet (completely) available
  - preliminary estimations on task structure and communication volume derived
    - from application models
    - reused SW and HW components
    - and standard component data sheets
- several simple spreadsheet models in practical use
  - e.g. textbook formula for computer networks (Hennessy/ Patterson)

$$\text{Effective bandwidth} = \min(2 \times \text{BW}_{\text{LinkInjection}}, 2 \times \text{BW}_{\text{LinkReception}}) = \frac{2 \times \text{Packet size}}{\max(\text{Overhead}, \text{Transmission time})}$$

Current Practice - A more Elaborate Simple Model

- reduction of dynamic effects to average or integral values



- allows application of weighted graph algorithms → *fast*
- frequently used in architecture optimization for distributed networks
- *does not reflect dynamic effects of transient loads, jitter, deadlines, buffer memory*

Current Practice in Design Verification

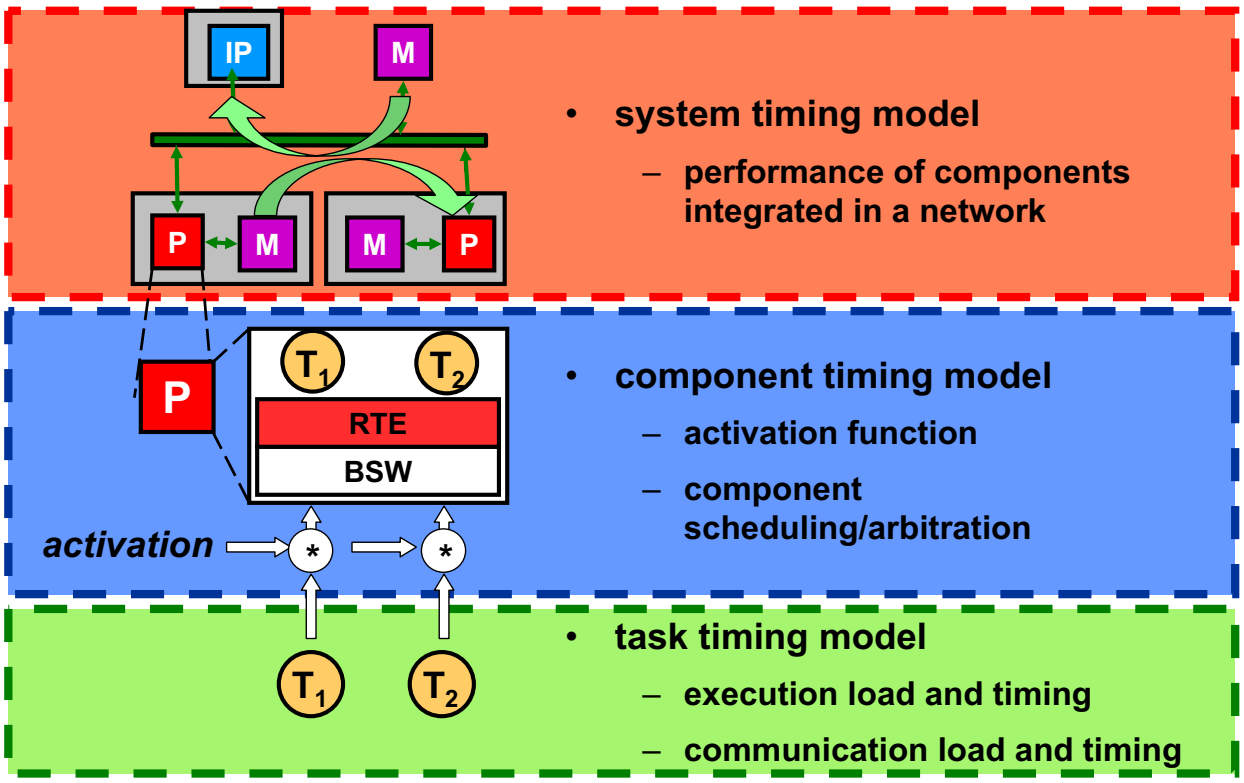
- extensive simulation
  - „programmers view“ simulation models based on the ISA
    → function test
  - TLM models to cover HW platform component interaction
    → platform function + approximate timing
  - RTL models
    → platform function + clock cycle accuracy
  → model hierachy to cover function as well as timing

- limitations
  - high modeling and computation cost for accurate models
  - simulation is always incomplete „case study“
  - concurrency requires accurate timing modeling to cover all effects
  - does not reflect design „flexibility“ to changes, updates, …

Overview

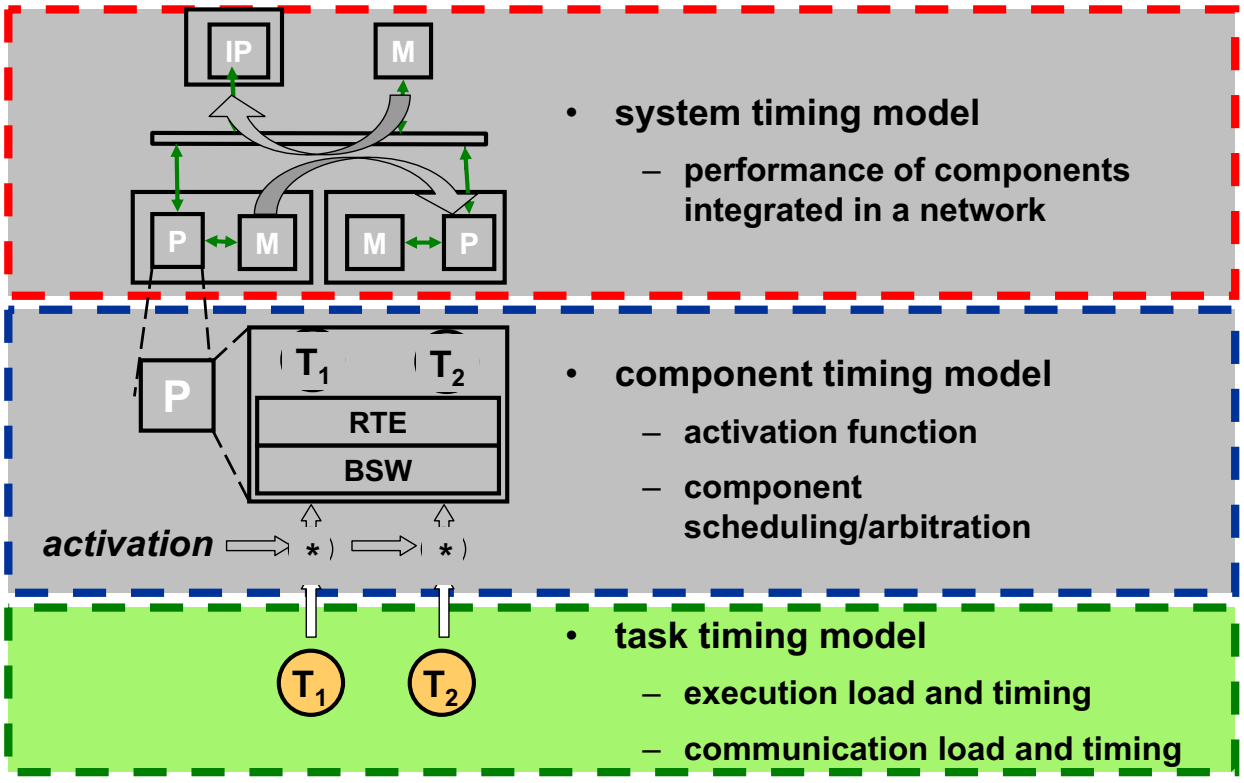
- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

Timing Model Hierarchy



- system timing model
  - performance of components integrated in a network
- component timing model
  - activation function
  - component scheduling/arbitration
- task timing model
  - execution load and timing
  - communication load and timing

Timing Model Hierarchy - Task



- system timing model
  - performance of components integrated in a network
- component timing model
  - activation function
  - component scheduling/arbitration
- task timing model
  - execution load and timing
  - communication load and timing

Formal Modeling Fundamentals – Task Execution

- *task core execution time* is the time needed to execute a given task when running alone on a processor
- task core execution time does *not* include
  - operating system overhead
  - the influence of other tasks
  - waiting and synchronization times for global resources
  - shared cache and memory access times (L1 cache often included)
- task core execution time is determined in different ways
  - estimated in early design phases
  - measured with a cycle accurate simulator (e.g. CoWare, Vast)
  - measured with instrumented code on a prototype (e.g. dspace)
  - formally analyzed using program path analysis in particular for high safety requirements (e.g. absint)

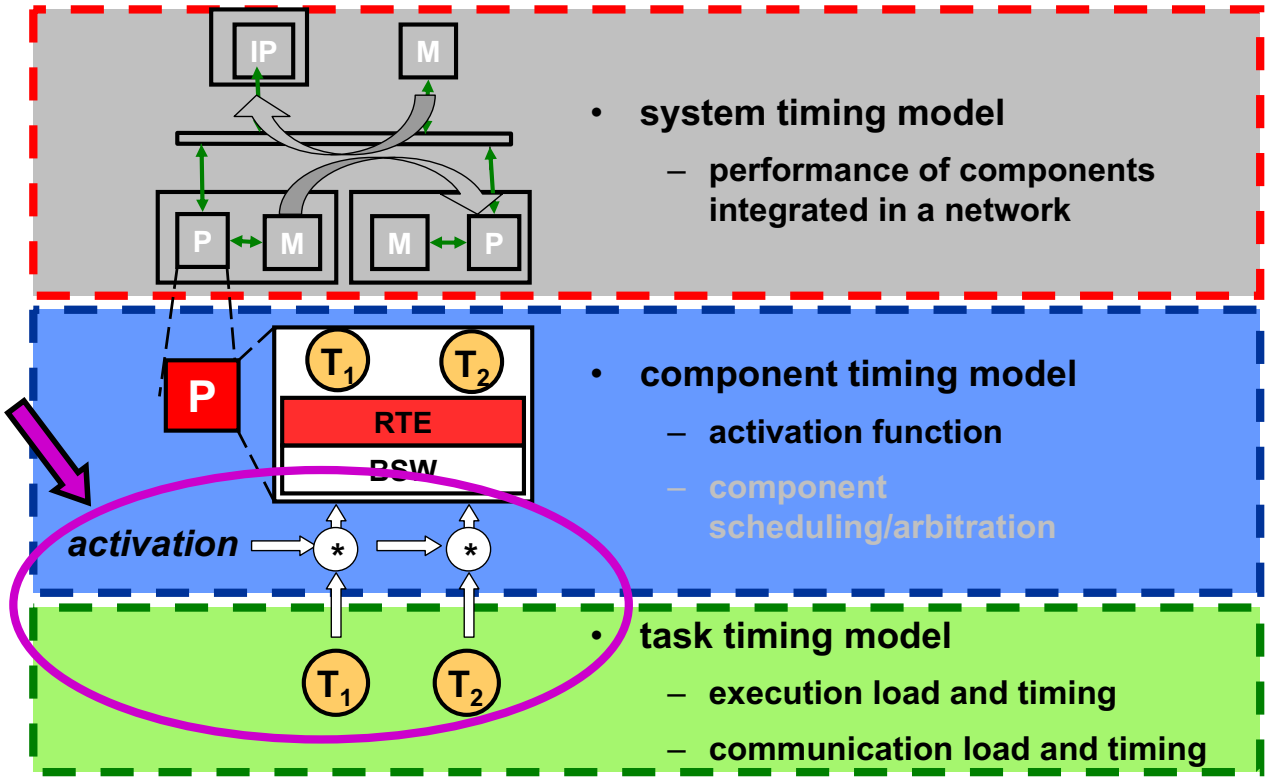
Formal Modeling Fundamentals – Communication

- *core communication time* is the transmission time for a given message to be communicated over a link when no other communication is active
- core communication time does *not* include
  - arbitration (scheduling)
  - buffering
  - gateway, multi-hop or MIN timing overhead
- core communication time is determined in different ways
  - simulation or prototyping
  - formal model of communication protocol (e.g. Symtavision)

Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

Timing Model Hierarchy - Activation



Formal Modeling Fundamentals – Activation

- **total task load, also called utilization of task i, U_i, depends on activation function**

  total task load = load/task execution * task activation requency

  = task core execution time * task activation frequency

  – example: periodic task i with core execution time C_i and period T_i

  $U_i = C_i/T_i$

- **what defines the task activation function ?**
  - **application model (Simulink, SPW, LabView, …)**
  - **environment model (reactive systems)**
  - **service contracts (max no of requests per time, …)**

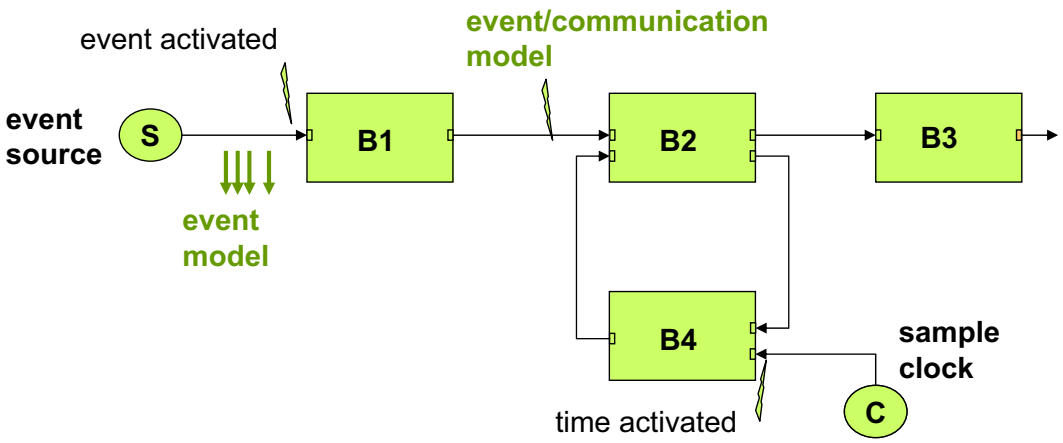
  → typically application rather than platform dependent

  → platform can „modulate“ activation timing to avoid malfunction (e.g. traffic shaping, back pressure)

- **two classes of activation – time activation, event activation**

Activation Functions

- **two classes of activation**
  - **time activation – tasks are periodically activated by clock**
    - **example: periodic sample in signal processing / control eng.**
  - **event activation – tasks are activated when event arrives**
    - **example: automata**



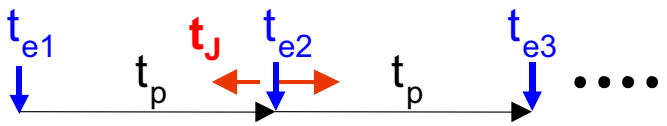
activation functions - example

Modeling Events as streams

- in formal performance models, events are modeled as *streams* rather than as sequences of individual events
- examples
  - a clock is given by its period rather than as a sequence of clock ticks
    → clock can be modeled as an event stream
  - a sampled sensor signal is modeled by the sample period and the sample jitter
- the event streams are defined as functions or as parameter tuples

Popular Event Stream Models – PJD

- standard event model used in real-time systems
  - event sequences are modeled by three parameters, period p, jitter j, and minimum time interval between 2 events
  - important models that can easily be decribed
    - strictly periodic events (typically clock released)
    - periodic events with jitter
    - sporadic events
    - sporadically periodic events

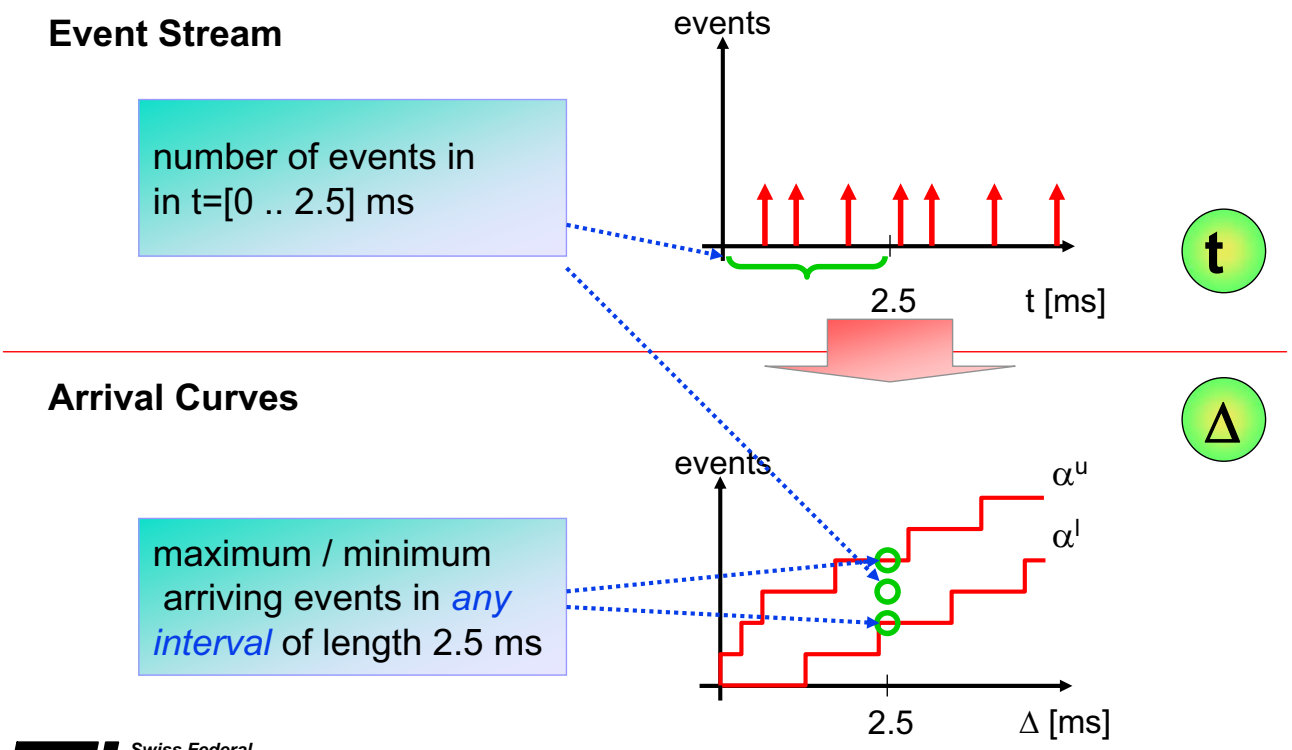


  - covers a large class of applications
  - conservatively approximates more complex functions

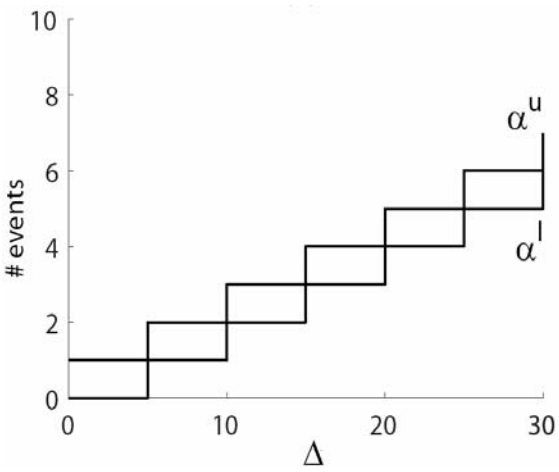
Popular Event Stream Models - Arrival Curves

- arrival curves of the network calculus
  - captures the no. of event in a time interval Δt
  - $\alpha^l(\Delta t)$ is lower bound
  - $\alpha^u(\Delta t)$ is upper bound
- can be used to describe the standard event models
- reaches infinite values for $\Delta t \rightarrow \infty$
  - must be approximated or extended by periodic function for $\Delta t \rightarrow \infty$
- is when event sequences become very complex, e.g.
  - as a result of operations on event sequences

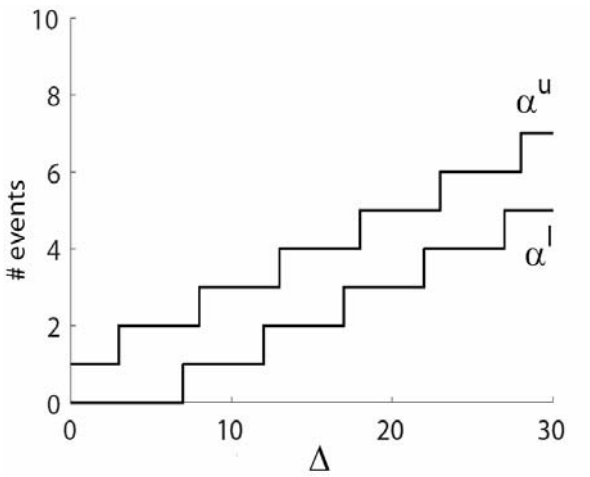
Arrival Curves - Example



Example 1: Periodic with Jitter



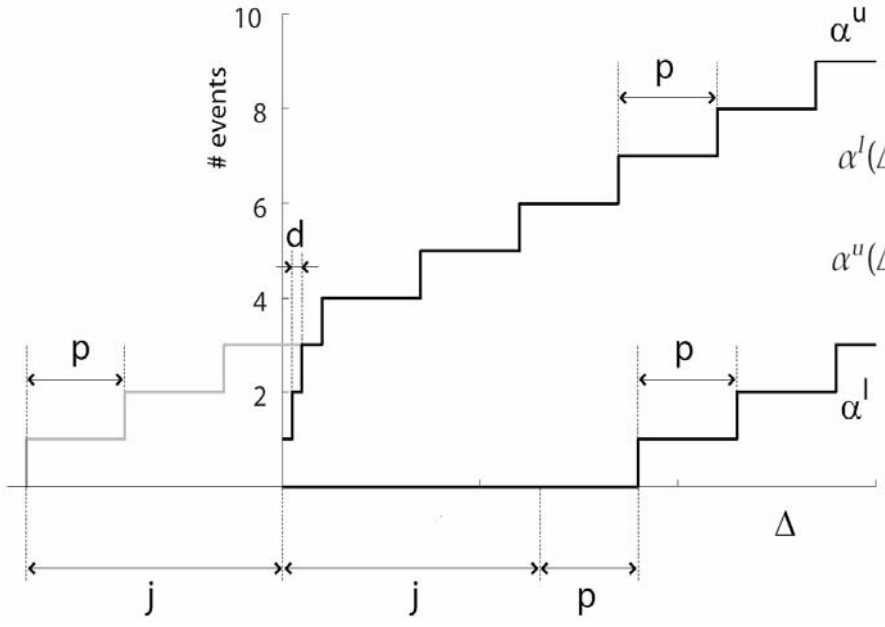
periodic



periodic with jitter

Example 2: Periodic with Jitter and Minimum Distance d

- ***Arrival curves***:

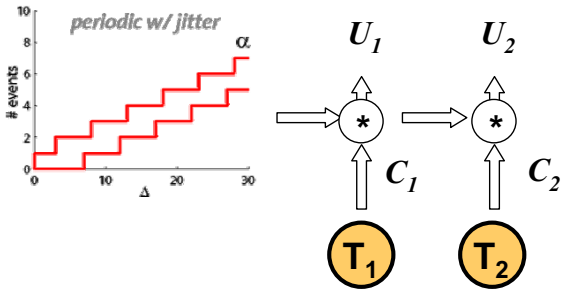


$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor$$

$$\alpha^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil \right\}$$

Total Load of a Task

- with activation model and core execution time or (core communication time), we can now derive the total load of a task

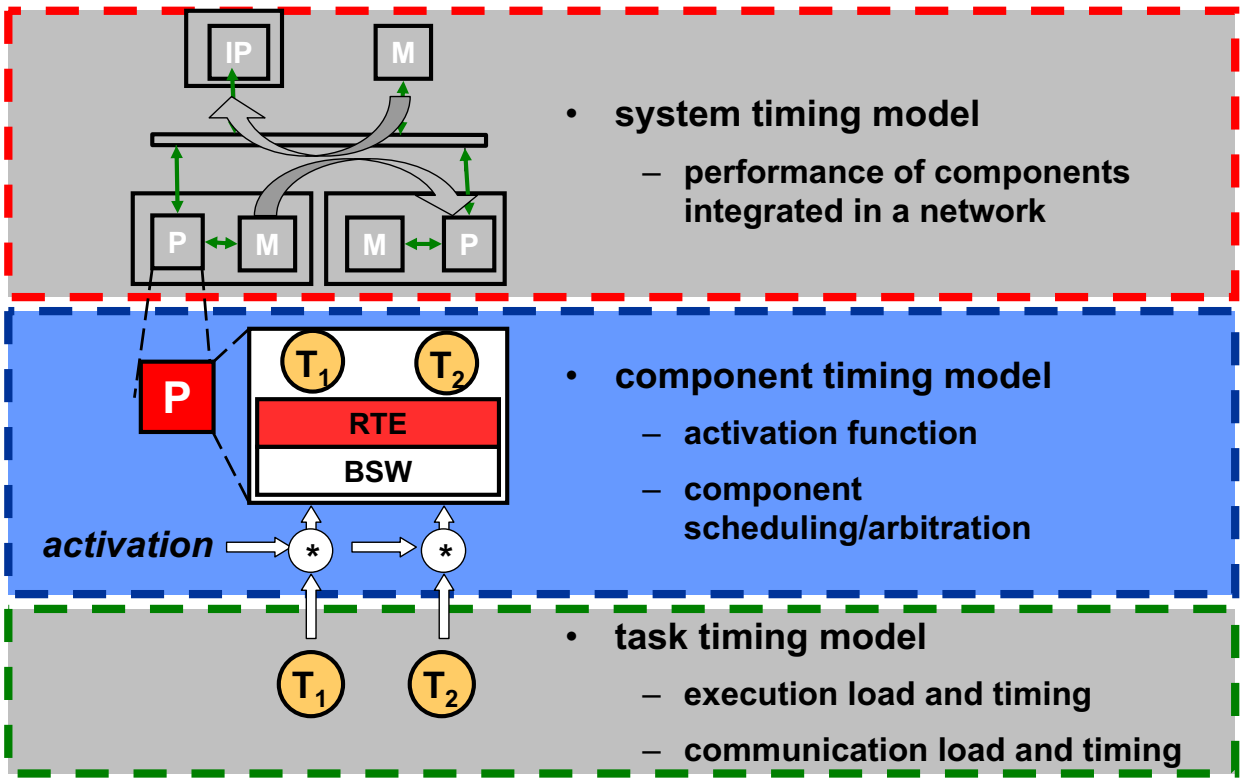


- the resource is not fully available to one task or communication, but is shared with others

Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

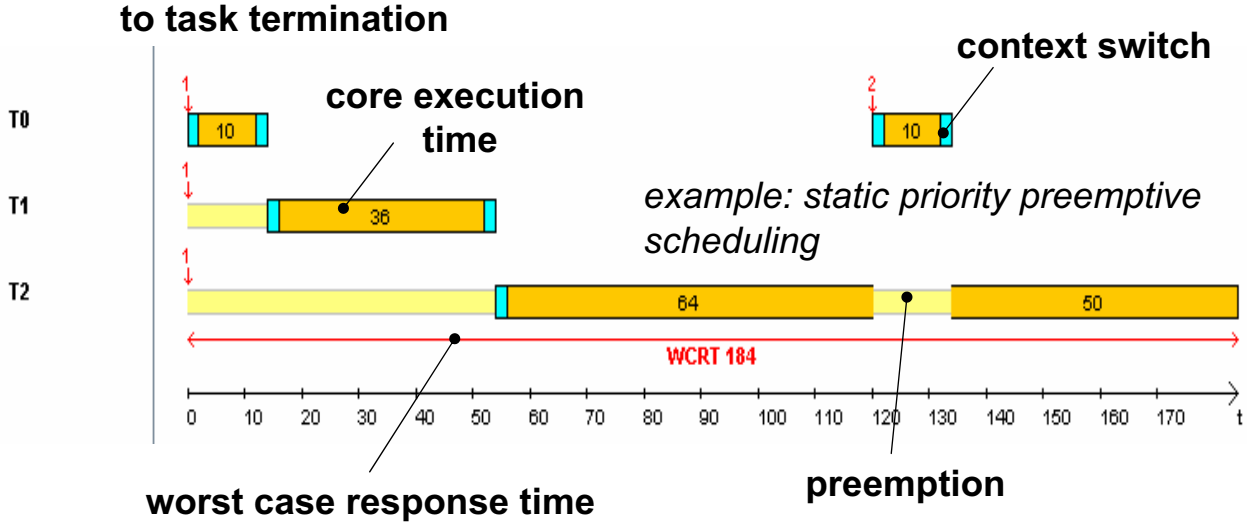
Timing Model Hierarchy - Component Timing



- system timing model
  - performance of components integrated in a network
- component timing model
  - activation function
  - component scheduling/arbitration
- task timing model
  - execution load and timing
  - communication load and timing

Timing Effects of Scheduling/Arbitration

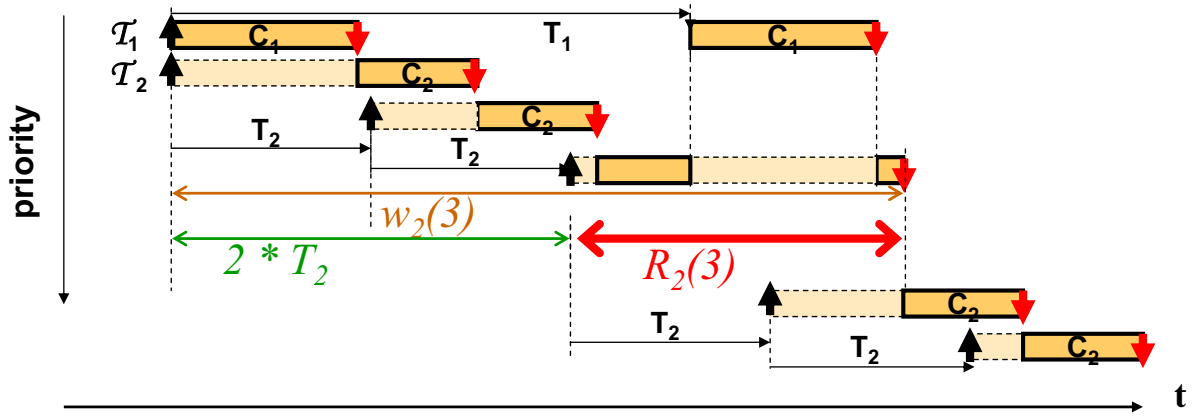
- tasks execute longer than their core execution time
  - time assigned to other tasks
  - operating system overhead
  - context switch, blocking, …
- response time of a task is maximum from time of activation to task termination



Scheduling Analysis

- different analysis algorithms
  - generalization of busy window algorithm (Lehoczky, Tindell) to fit general event model (Richter, Jersak, Henia, Racu, Ernst, Schliecker, et al.)
    - Tool SymTA/S
  - extension of Network Calculus to Real-time Calculus (Chakraborty, Wandeler, Künzli, Thiele, et al.)
    - Tool MPA

Analysis uses "Busy Window" approach

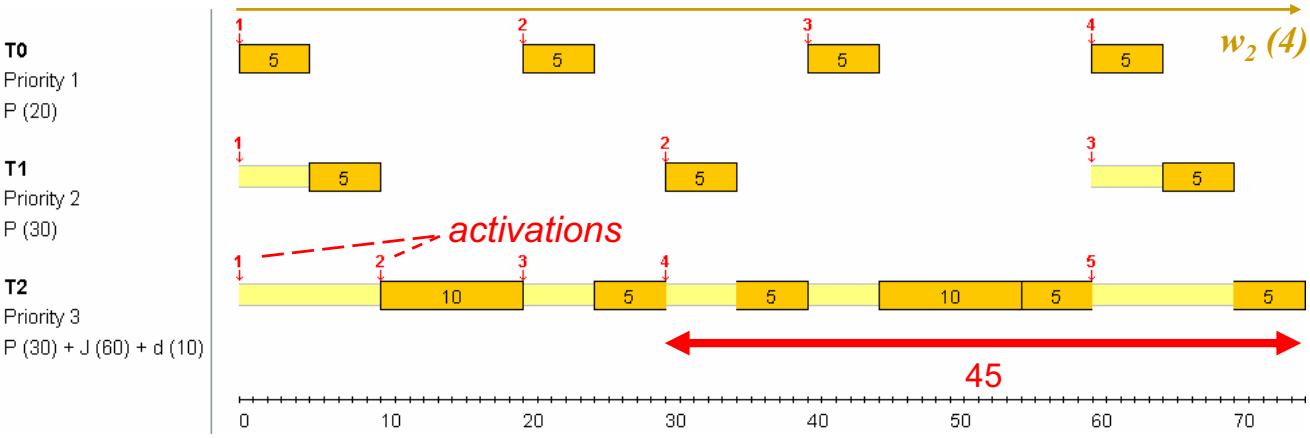


$$w_i(q) = q\,C_i + \sum_{j \in \mathrm{hp}(i)} C_j \left\lceil \frac{w_i(q)}{T_j} \right\rceil$$

$$R_i(q) = w_i(q) - (q-1)\,T_i$$

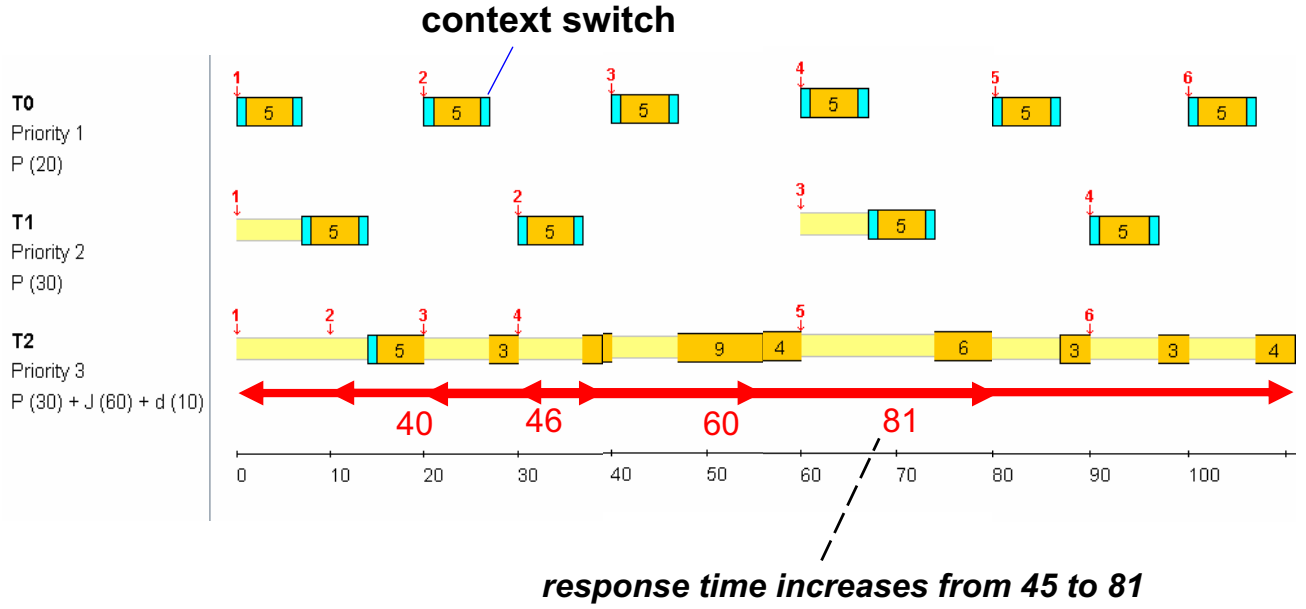
increase w_i until fix point found where equations hold!

Busy Window Analysis



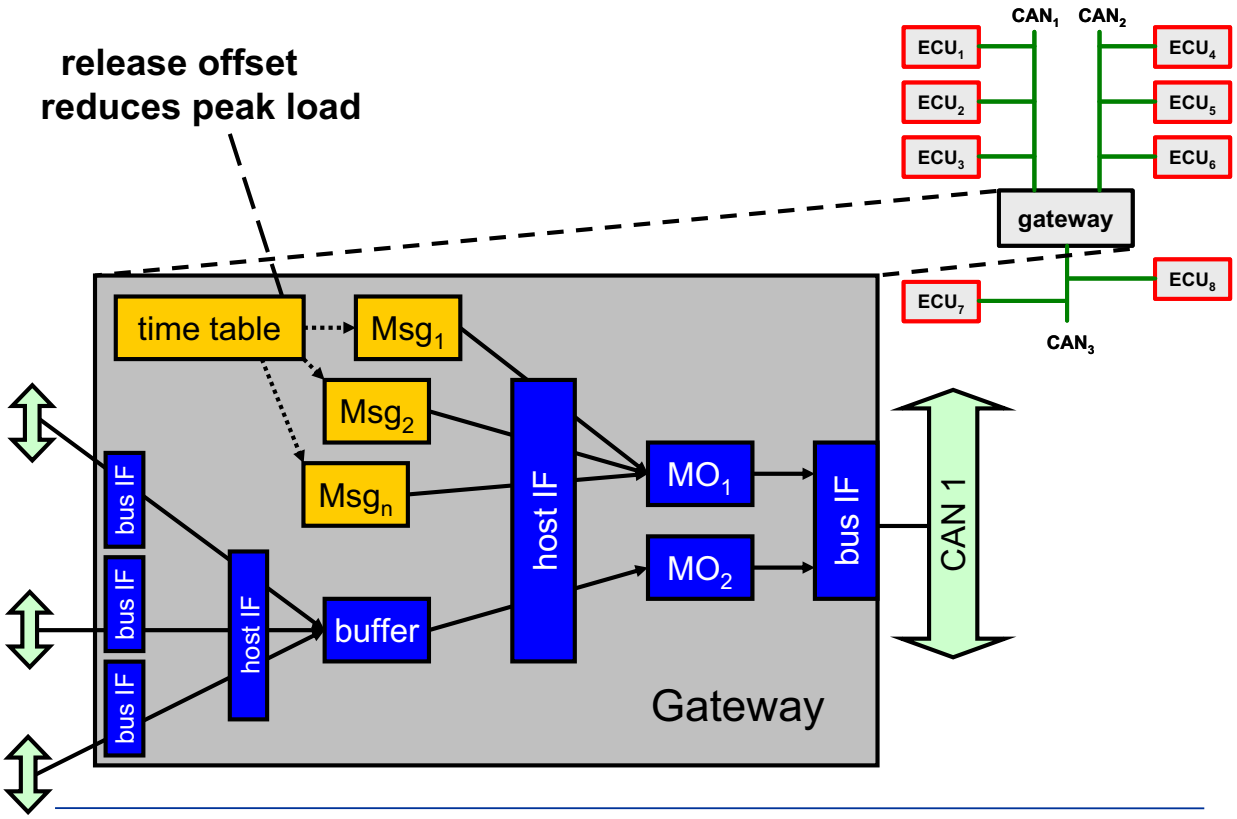
- very versatile approach
- has been extended to analyze even difficult scheduling strategies
  - round-robin, non preemptive, collaborative processes (e.g. OSEK), …
- can handle unkown worst case (e.g. release offsets – time table)
- can handle stream queues and register communication
- window size increases with load (limited by deadline)
- this window „unrolling“ processes can be considered as symbolic simulation

Importance of Context Switch Consideration

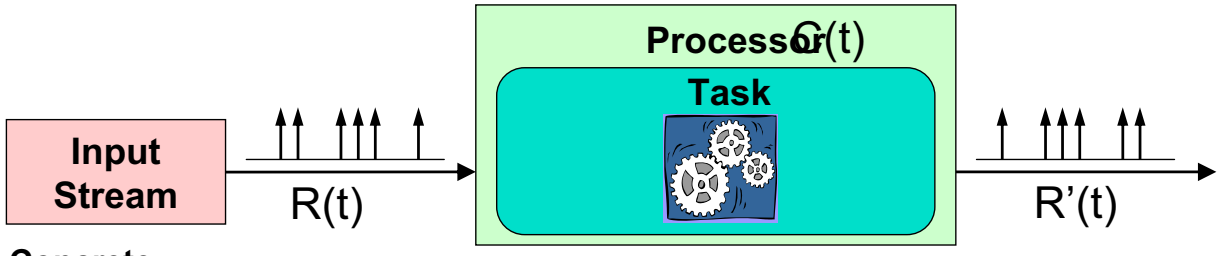


- context switch increases load → non load preserving

Time Table for Release Offset

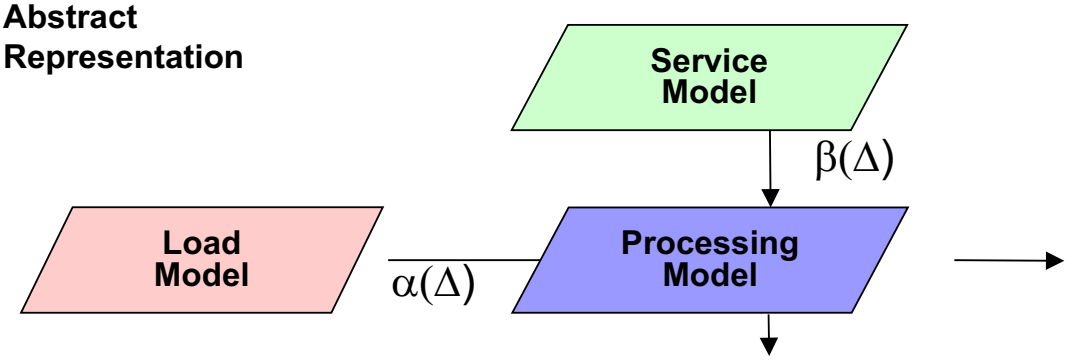


Real-time Calculus



Concrete Instance

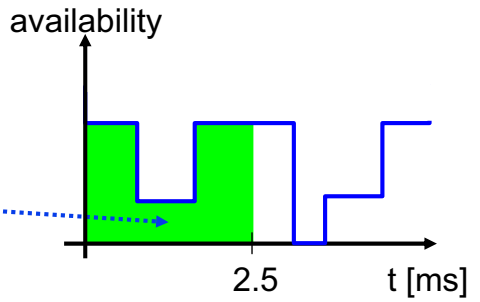
Abstract Representation



Service Model (Resources)

Resource Availability

available service
in t=[0 .. 2.5] ms

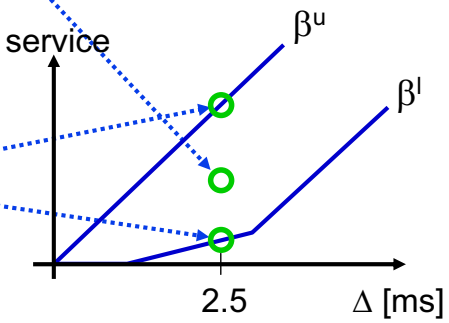


t

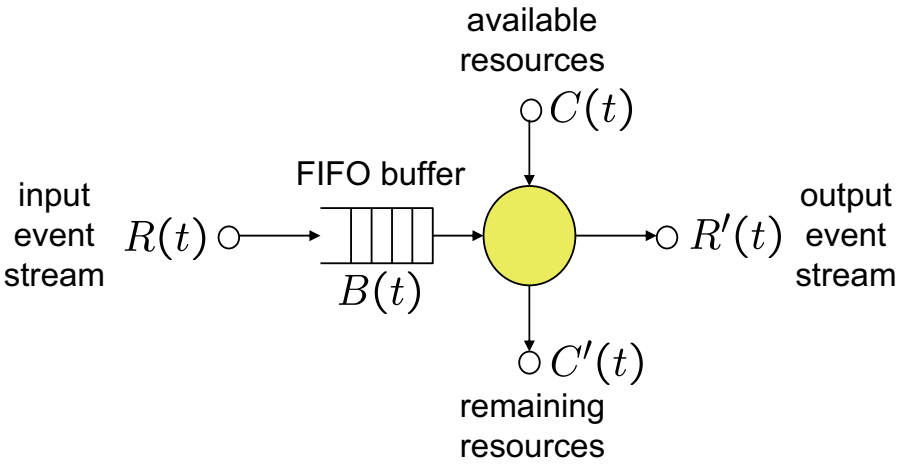
Service Curves

Δ

maximum/minimum
available service in *any interval* of length 2.5 ms

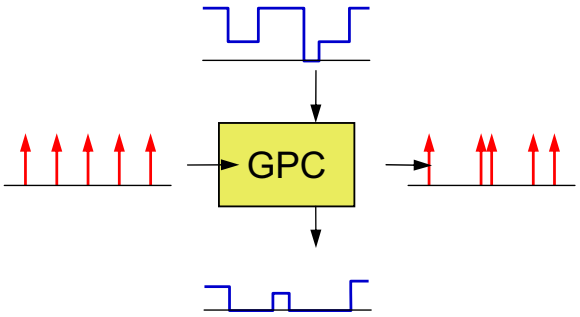


Greedy Processing Component (GPC)



- ***Examples:***
  - computation (event – task instance, resource – computing resource [tasks/second])
  - communication (event – data packet, resource – bandwidth [packets/second])

Greedy Processing Component



Behavioral Description

- Component is triggered by incoming events.
- A fully preemptable task is instantiated at every event arrival to process the incoming event.
- Active tasks are processed in a greedy fashion in FIFO order.
- Processing is restricted by the availability of resources.

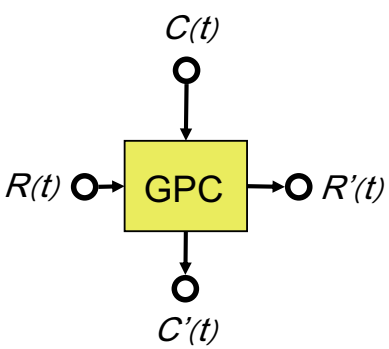
Greedy Processing Component (GPC)

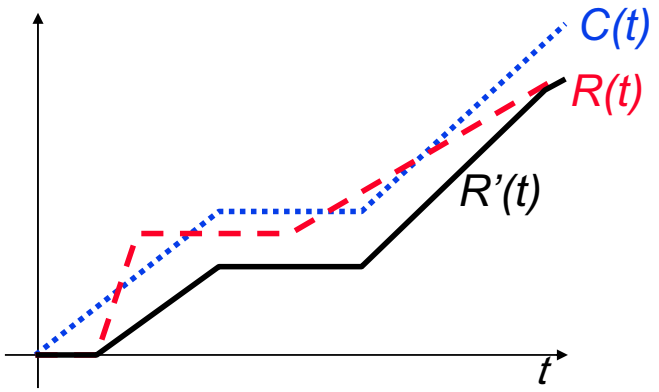
If the resource and event streams describe available and requested units of processing or communication, then

$$C(t) = C'(t) + R'(t)$$
$$B(t) = R(t) - R'(t)$$

} Conservation Laws

$$R'(t) = \inf_{0 \le u \le t} \{R(u) + C(t) - C(u)\}$$

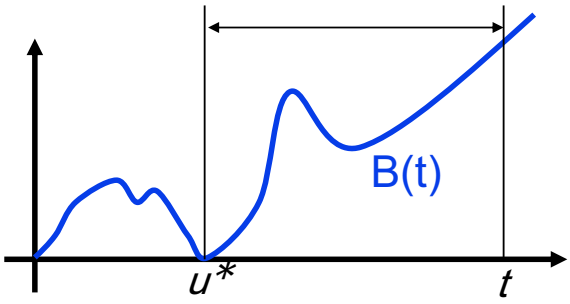




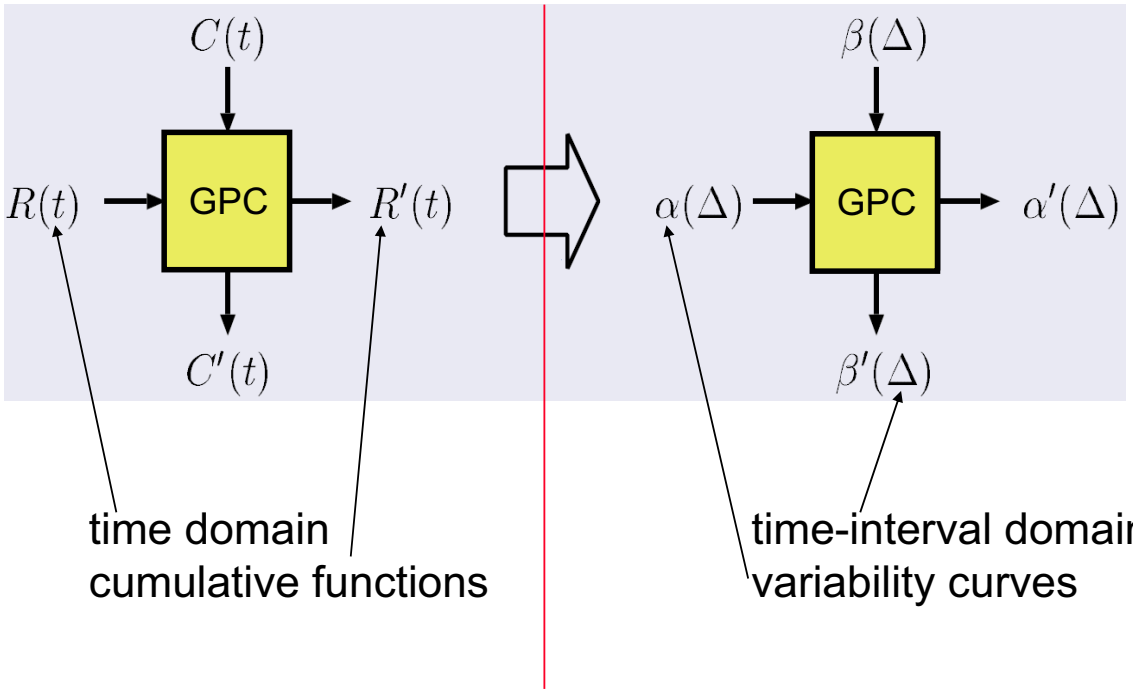
Greedy Processing

- For all times $u \le t$ we have $R'(u) \le R(u)$ (conservation law).
- We also have $R'(t) \le R'(u)+C(t)-C(u)$ as the output can not be larger than the available resources.
- Combining both statements yields $R'(t) \le R(u) + C(t) - C(u)$.
- Let us suppose that u^* is the last time before t with an empty buffer. We have $R(u^*) = R'(u^*)$ at u^* and also $R'(t) = R'(u^*) + C(t) - C(u^*)$ as all available resources are used to produce output. Therefore, $R'(t) = R(u^*) + C(t) - C(u^*)$.
- As a result, we obtain

$$R'(t) = \inf_{0 \le u \le t} \{R(u) + C(t) - C(u)\}$$



Abstraction



Some Definitions and Relations

- $f \otimes g$ is called ***min-plus convolution***

$$(f \otimes g)(t) = \inf_{0 \le u \le t} \{f(t-u) + g(u)\}$$

- $f \oslash g$ is called ***min-plus de-convolution***

$$(f \oslash g)(t) = \sup_{u \ge 0} \{f(t+u) - g(u)\}$$

- For ***max-plus convolution and de-convolution***:

$$(f \overline{\otimes} g)(t) = \sup_{0 \le u \le t} \{f(t-u) + g(u)\}$$

$$(f \overline{\oslash} g)(t) = \inf_{u \ge 0} \{f(t+u) - g(u)\}$$

- Relation between ***convolution and deconvolution***

$$f \le g \otimes h \Leftrightarrow f \oslash h \le g$$

The Most Simple Relations

- The *output stream* of a component satisfies:

$$R'(t) \ge (R \otimes \beta^l)(t)$$

- The *output upper arrival curve* of a component satisfies:

$$\alpha^{u\prime} = (\alpha^u \oslash \beta^l)$$

- The *remaining lower service curve* of a component satisfies:

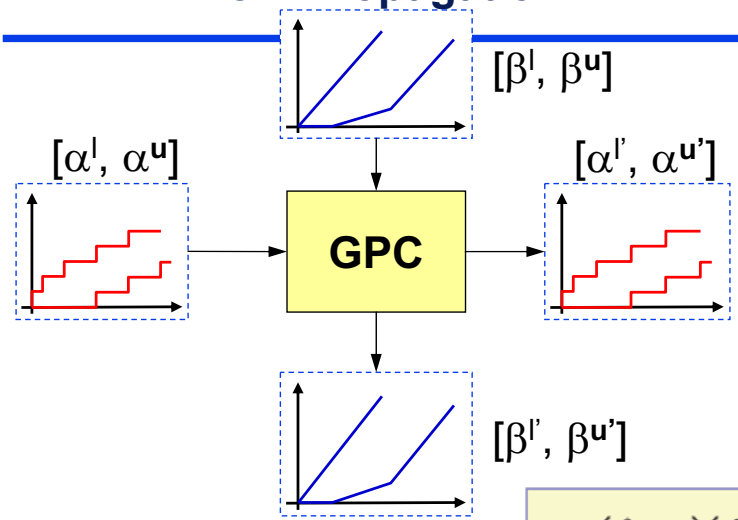
$$\beta^{l\prime}(\Delta) = \sup_{0 \le \lambda \le \Delta} (\beta^l(\lambda) - \alpha^u(\lambda))$$

Two Sample Proofs

$$
\begin{aligned}
R'(t) &= \inf_{0\le u\le t}\{R(u)+C(t)-C(u)\}\\
&\ge \inf_{0\le u\le t}\{R(u)+\beta^l(t-u)\}\\
&= (R\otimes\beta^l)(t)
\end{aligned}
$$

$$
\begin{aligned}
C'(t)-C'(s) &= \sup_{0\le a\le t}\{C(a)-R(a)\}-\sup_{0\le b\le s}\{C(b)-R(b)\}=\\
&= \inf_{0\le b\le s}\{\sup_{0\le a\le t}\{(C(a)-C(b))-(R(a)-R(b))\}\}\\
&= \inf_{0\le b\le s}\{\sup_{0\le a-b\le l-b}\{(C(a)-C(b))-(R(a)-R(b))\}\}\\
&\ge \inf_{0\le b\le s}\{\sup_{0\le \lambda\le t-b}\{\beta^l(\lambda)-\alpha^u(\lambda)\}\}\ge \sup_{0\le\lambda\le t-s}\{\beta^l(\lambda)-\alpha^u(\lambda)\}
\end{aligned}
$$

MPA-RTC – Propagation



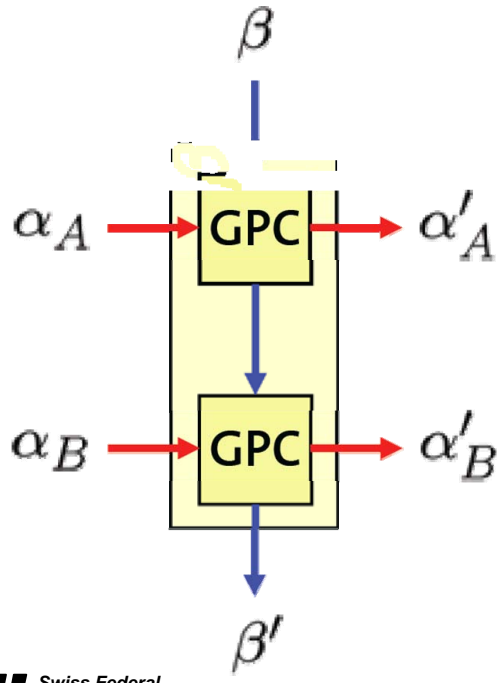
Real-Time Calculus

$$
\begin{aligned}
\alpha'^u &= \min\{(\alpha^u\otimes\beta^u)\oslash\beta^l,\beta^u\}\\
\alpha'^l &= \min\{(\alpha^l\oslash\beta^u)\otimes\beta^l,\beta^l\}\\
\beta'^u &= (\beta^u-\alpha^l)\mathbin{\overline{\oslash}}0\\
\beta'^l &= (\beta^l-\alpha^u)\mathbin{\overline{\otimes}}0
\end{aligned}
$$

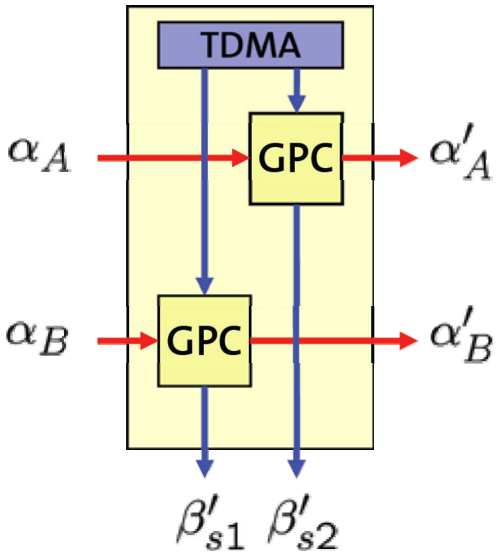
$$
\begin{aligned}
(f\otimes g)(\Delta) &= \inf_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\}\\
(f\oslash g)(\Delta) &= \sup_{\lambda\ge 0}\{f(\Delta+\lambda)-g(\lambda)\}\\
(f\mathbin{\overline{\otimes}} g)(\Delta) &= \sup_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\}\\
(f\mathbin{\overline{\oslash}} g)(\Delta) &= \inf_{\lambda\ge 0}\{f(\Delta+\lambda)-g(\lambda)\}
\end{aligned}
$$

MPA-RTC – Scheduling - Examples

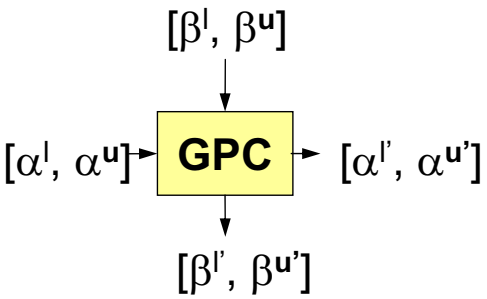
Fixed Priority Preemptive Scheduling

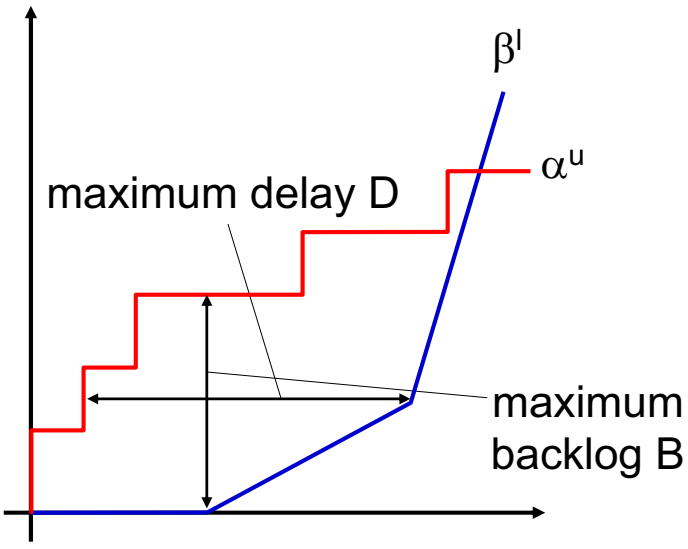


Time Division Multiple Access (TDMA)



Delay and Backlog

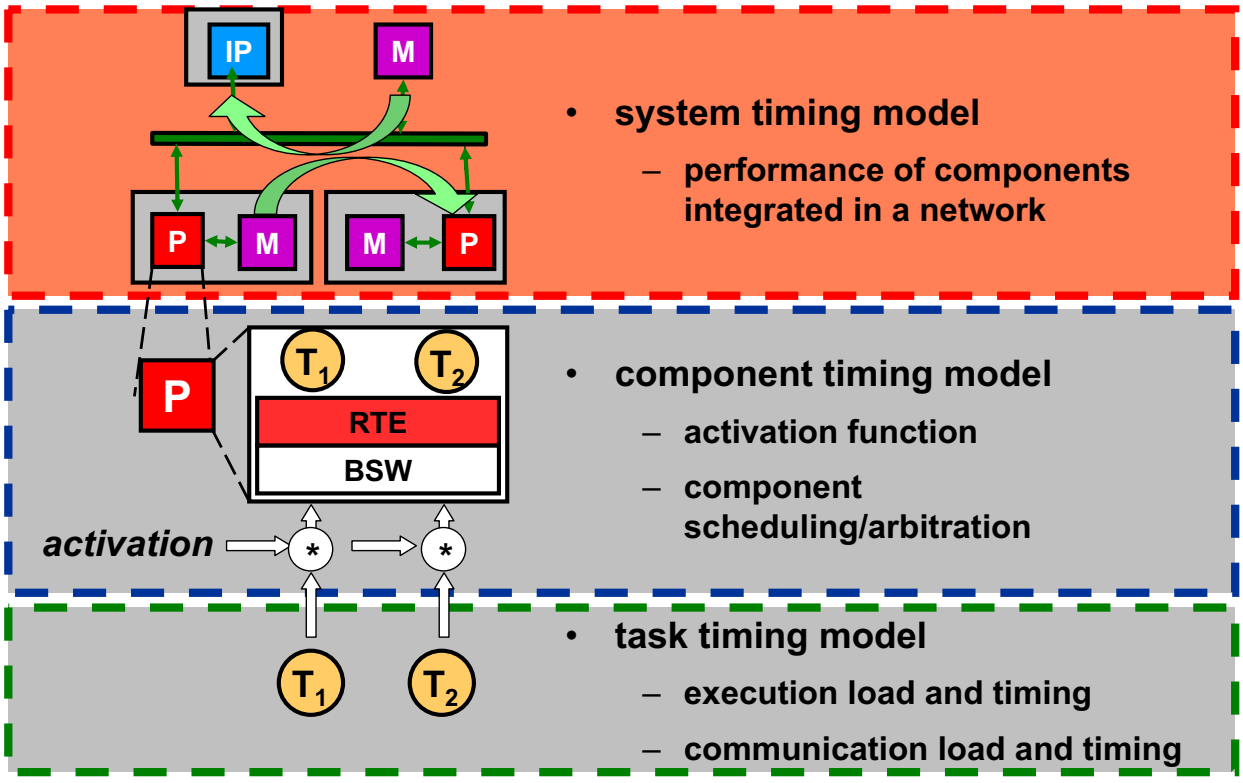




Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

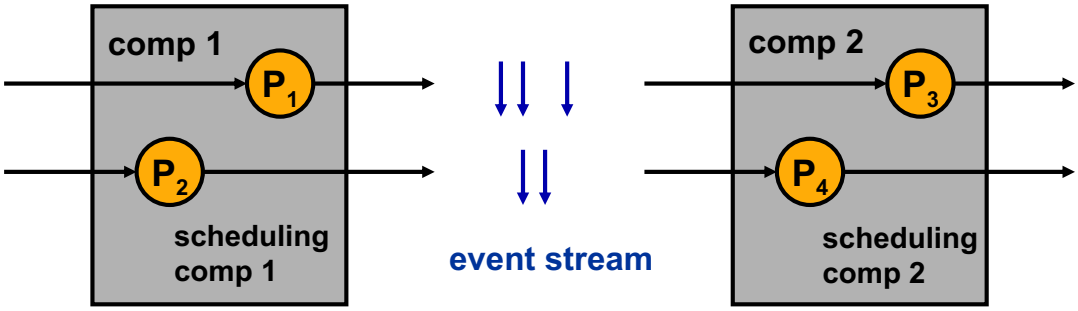
Timing Model Hierarchy – System Timing Model



- system timing model
  - performance of components integrated in a network
- component timing model
  - activation function
  - component scheduling/arbitration
- task timing model
  - execution load and timing
  - communication load and timing

System analysis using compositional approach

- independently scheduled subsystems are coupled by data flow

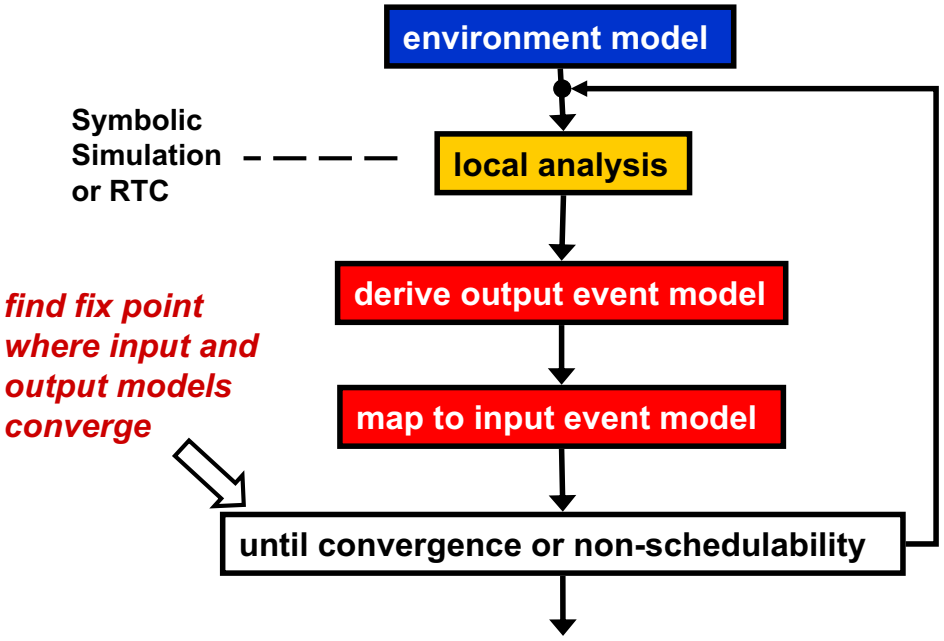


⇒ subsystems coupled by streams of data

⇒ interpreted as activating events

⇒ coupling corresponds to event propagation

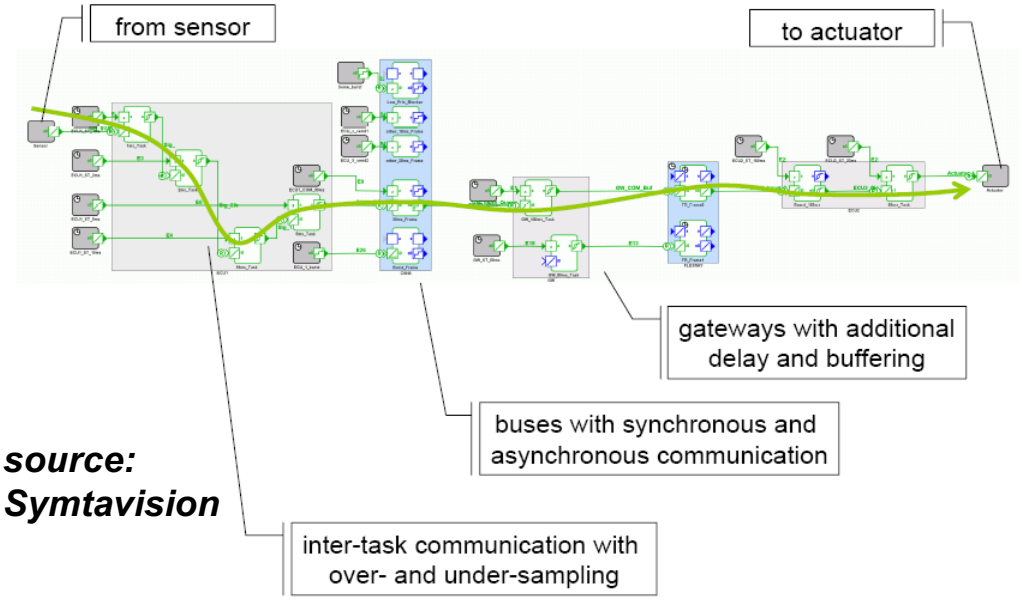
Compositional analysis principle



System-level Analysis Results

- end-to-end latencies
- buffer sizes
- system load
- ….

example: complex end-to-end latency analysis w. SymTA/S



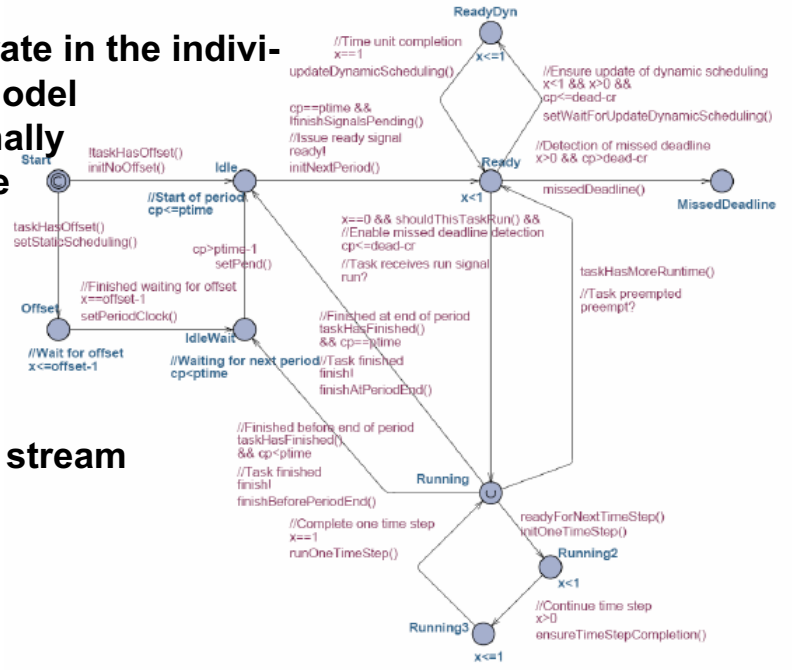
source: Symtavision

Compositional Analysis Properties

- compatible event stream models allow to couple any number of blocks for local analysis
  → scalable
- fixpoint iteration automatically adapts to platform topology
  → easy integration and extension
  → RTC and SymTA/S analysis blocks have been shown to easily work together [KHT07]
- very short analysis time (few seconds) opens new opportunities in design space and robustness optimization

Further Performance Models 1/2

- timed automata have been used to explicitely model the task scheduling algorithm and OS interactions and then apply model checking to identify deadline violations
- can be more accurate in the individual component model but is computationally far more expensive
- work e.g. Madsen or Johnson
- can potentially be linked via common event stream models



Further Performance Models 2/2

- will be discussed in separate tasks of this tutorial

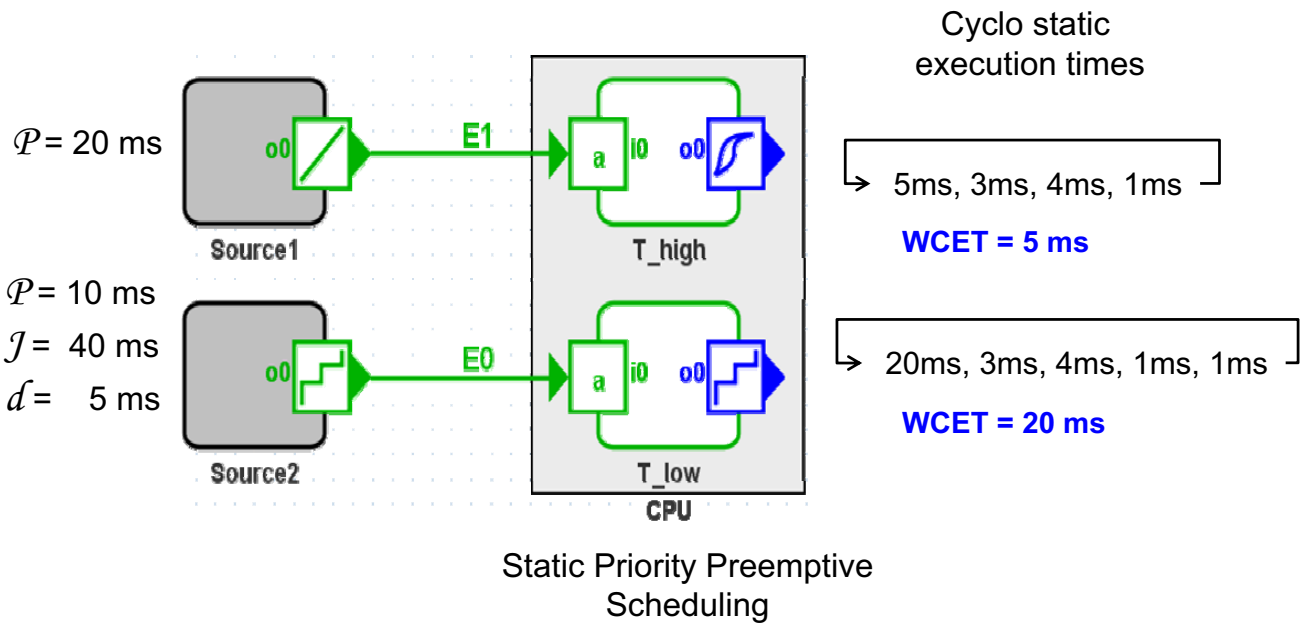
Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

Enhancements to the Basic Analysis

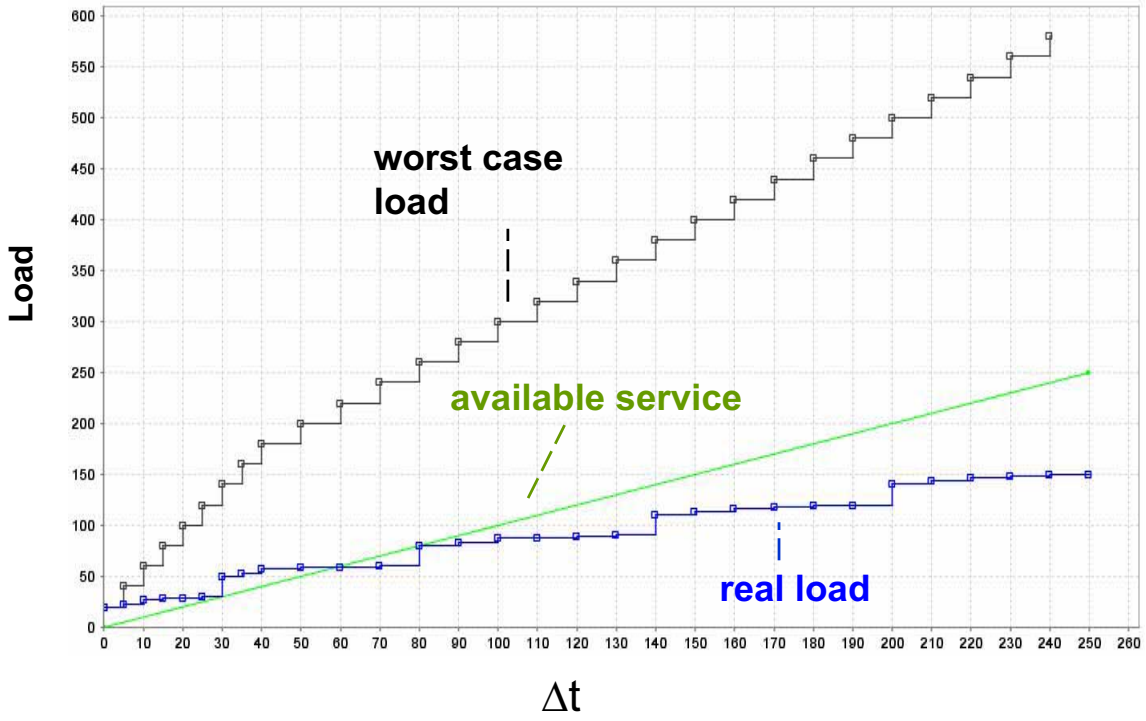
- load modeling for variyng execution times
- shared memory modeling on MpSoC
- robustness optimization

Load Modeling for Varying Execution Times

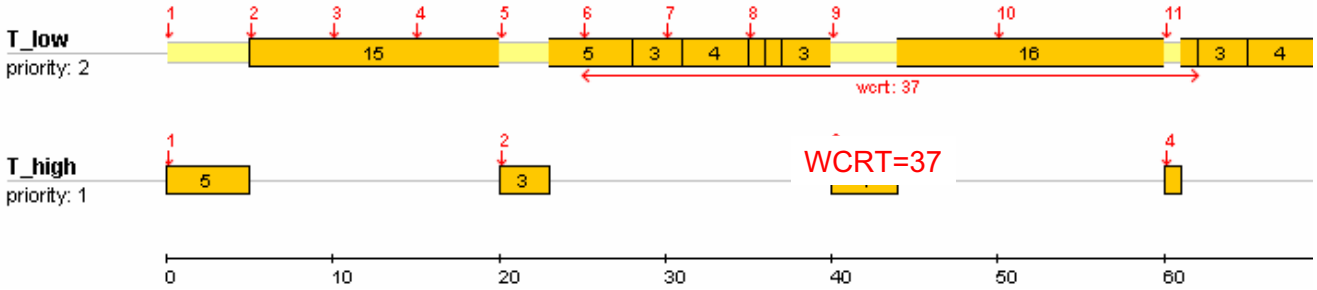


- **Simple Example – cyclo-static system**

Load for Interval Δt: T_low



Real Load Used in Improved Analysis

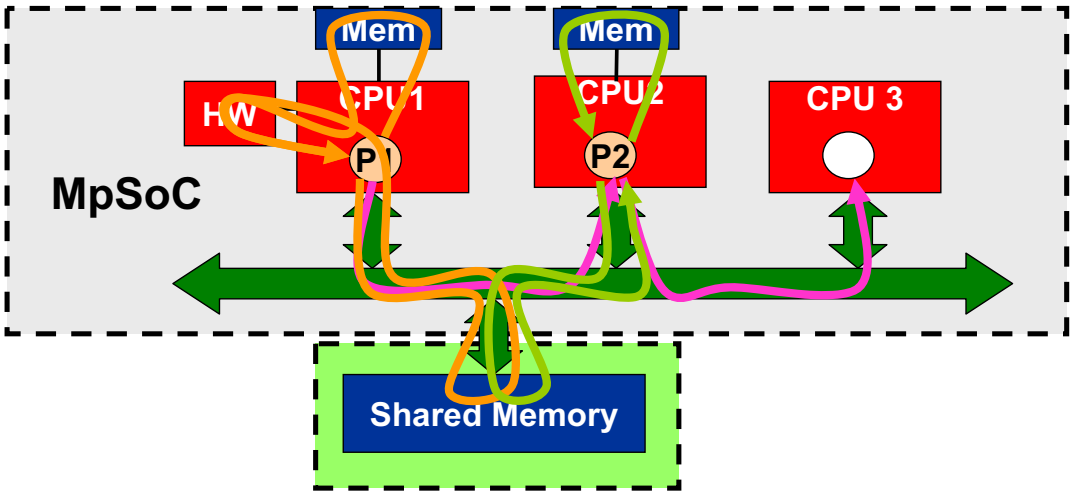


- improved analysis available for SymTA/S (see above) and MPA
- even more powerful: Scenario Analysis
  - identify different sets of tasks or deviating core execution times of tasks for different application contexts → scenarios
  - example: different use of smart phone, car: acceleration/idle, ..
  - interesting is transition between scenarios possibly leading to overloads, lost data, …
  - see literature

Enhancements to the Basic Analysis

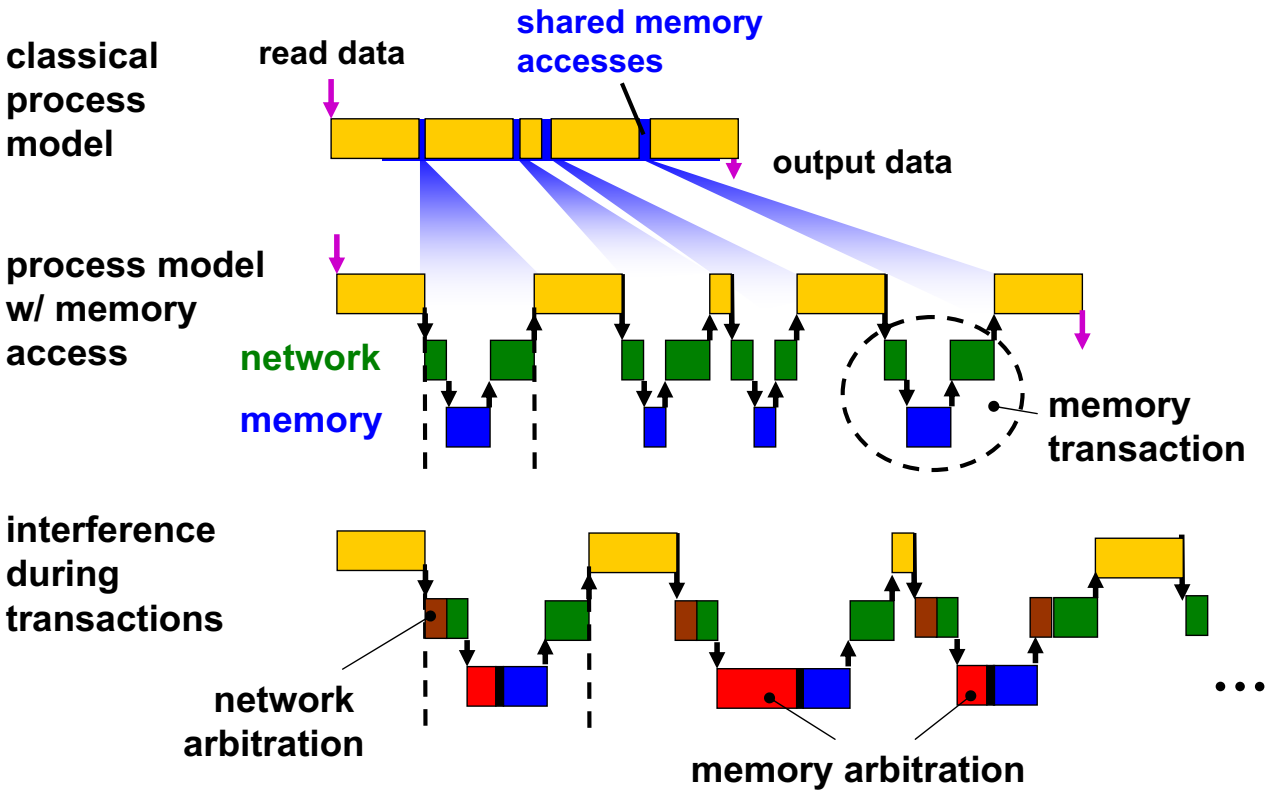
- load modeling for variyng execution times
- shared memory modeling on MpSoC
- robustness optimization

MpSoC with "Secondary" Traffic



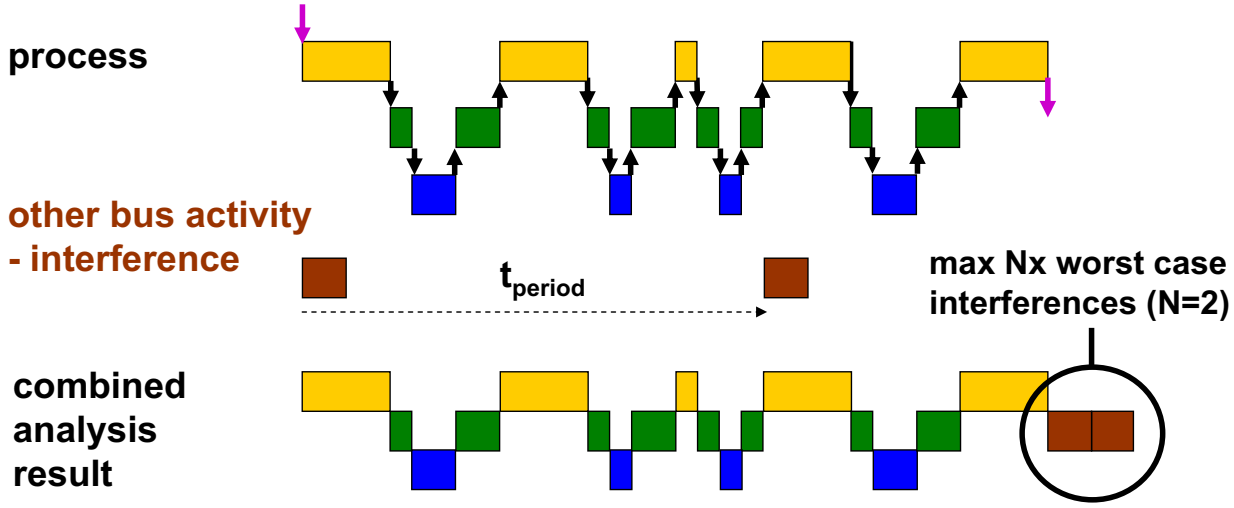
- use of shared memories or shared coprocessors
  - shared on-chip data and programs
  - larger off-chip memories
- data and program memory accesses on same network as task communication - more complex traffic

MpSoC process execution



Combined memory transaction modeling

- combined analysis of all process memory transactions
  - add all delays that can occur during all transactions of a process in the worst case
  - more realistic bus and memory timing

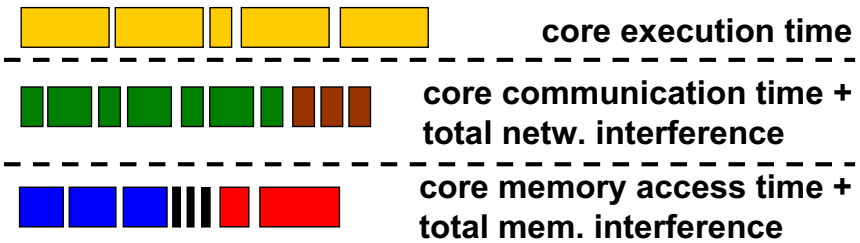


Improved combination in transaction modeling

- complex and highly dynamic interactions if memory transactions of *multiple processors* interfere
  - simple combination not sufficient

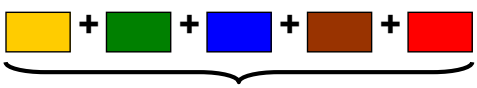
Solution:

- derive **upper total interference bound** using formal analysis



- **superimpose** on each core

  → total execution time (conservative)



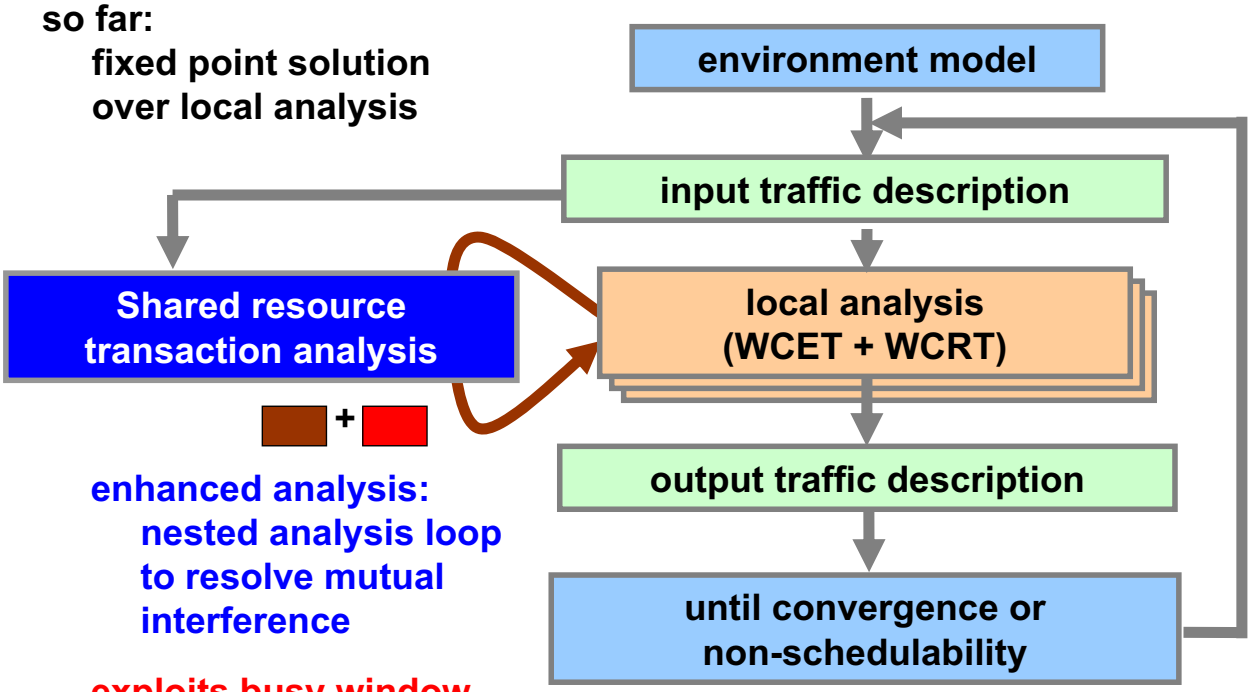
≥ *total worst case execution time*

- couple single core analysis with iterative nested scheme

  → enhanced SymTA/S analysis

Enhanced SymTA/S compositional analysis engine

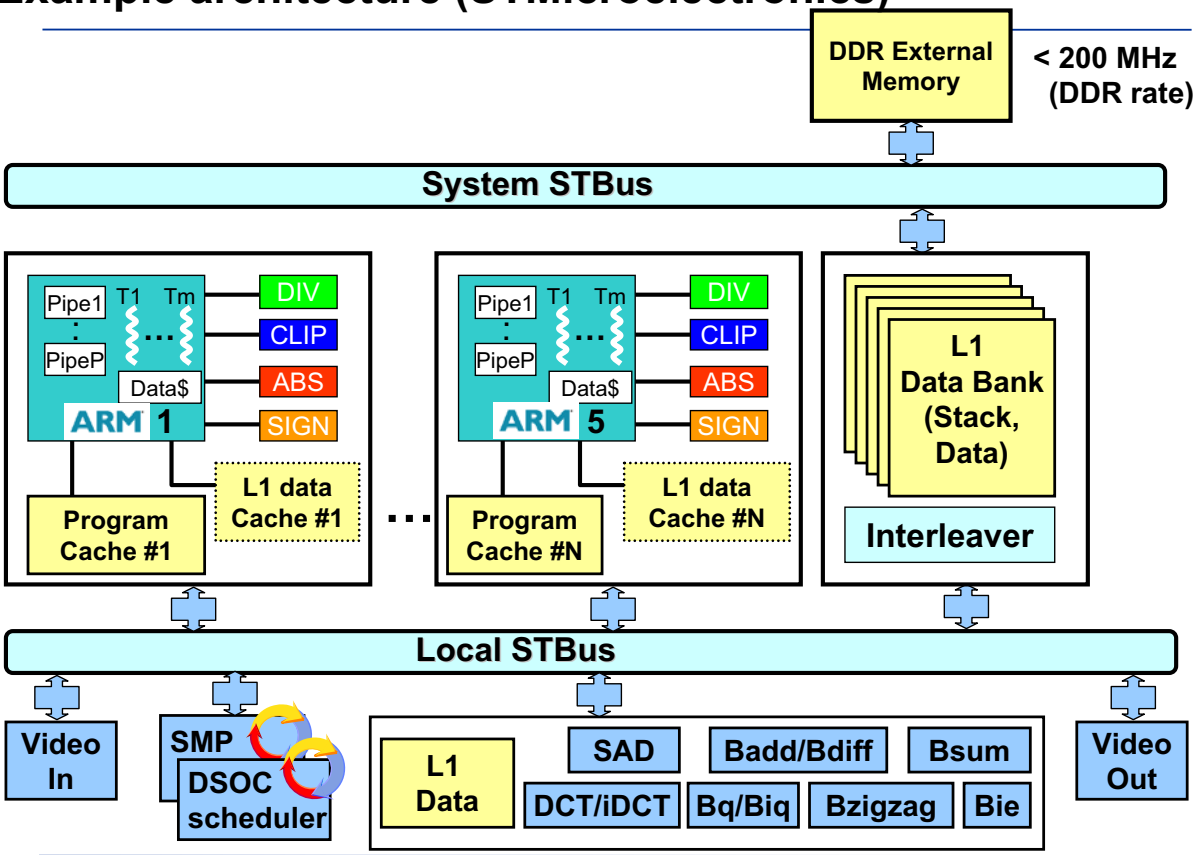
so far:
fixed point solution
over local analysis



enhanced analysis:
nested analysis loop to resolve mutual interference

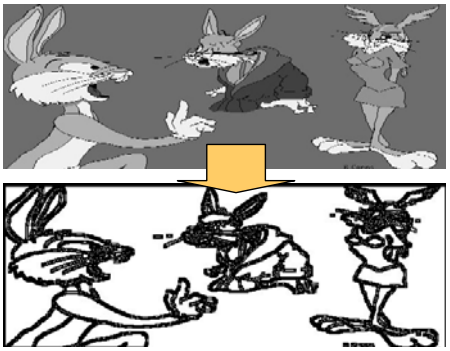
exploits busy window approach!

Example architecture (STMicroelectronics)



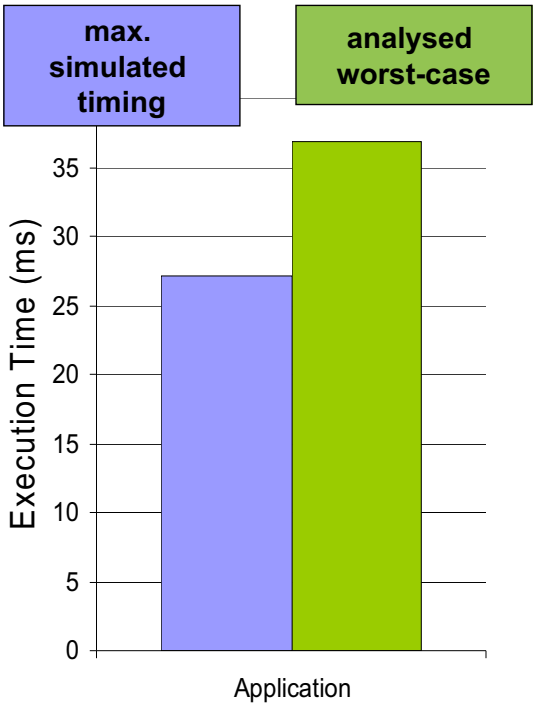
Application: MPEG 4 contour detection

- contour detection algorithm from École Polytechnique de Montreal (Gabriela Nicolesu)
- 2 – 4 processor architecture
- 2 threads per processor
  - round-robin scheduling
- StepNP simulator available
- input data aquisition using simulation
  - simulation results of subtasks on single core



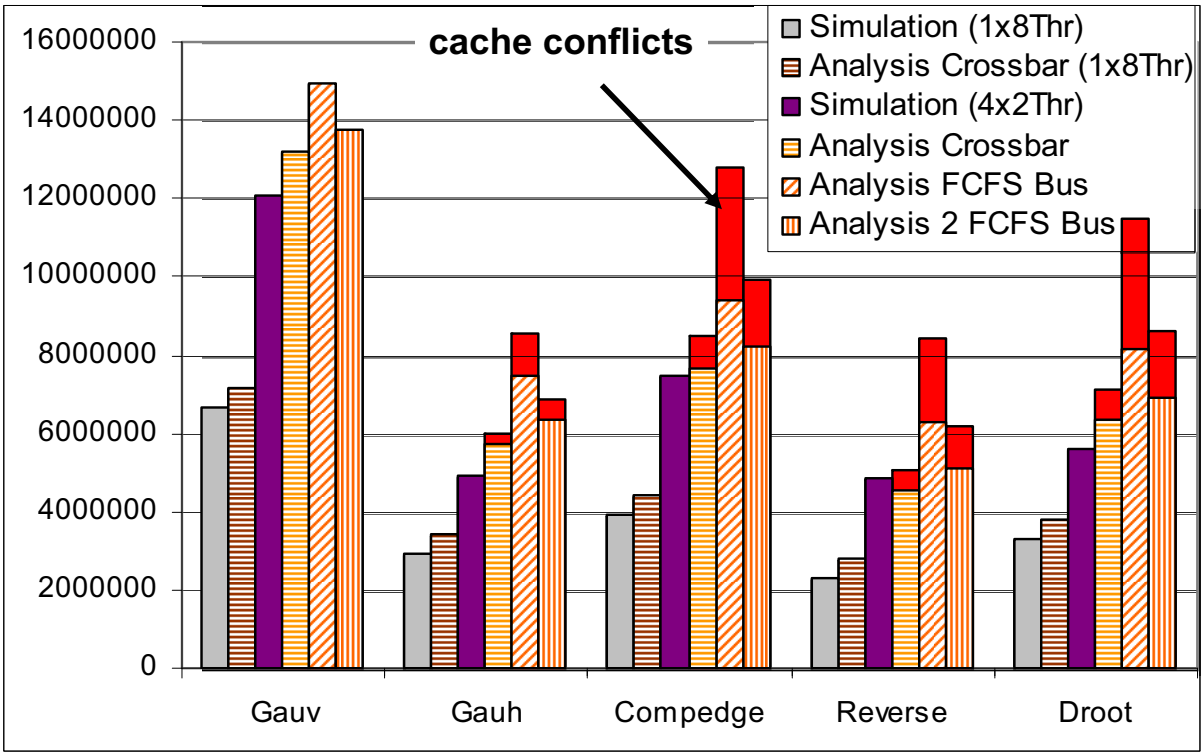
System level analysis

- very fast analysis of worst case behavior considering
  - bus/network congestion (if any)
  - memory congestion
  - multithreading
  - coprocessors ...
- 35% larger analyzed timing than maximum simulation result
  - símulation uses simplified crossbar communication model
- planned for investigation of processor sharing, degree of parallelism / pipelining, …



concurrent execution

Detailed Analysis



Enhancements to the Basic Analysis

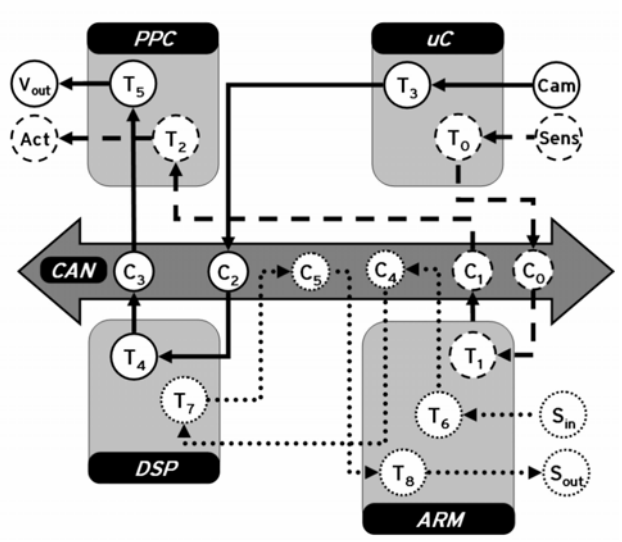
- load modeling for variyng execution times
- shared memory modeling on MpSoC
- robustness optimization

Robustness Optimization

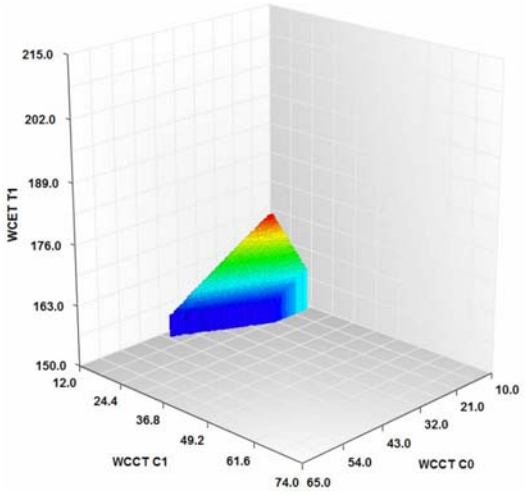
- **goal: improve slack in architecture, such that a single or a combination of load data can change later on without affecting end-to-end deadlines and other constraints**
- **approach**
  1. **analyze remaining slack in architecture (1 or multidimensional)**
     - **uses binary search or evolutionary algorithms**
  2. **optimize system parameters to maximize slack**
     - **uses evolutionary algorithm**
- **search parameters and controlled by designer**

Example System

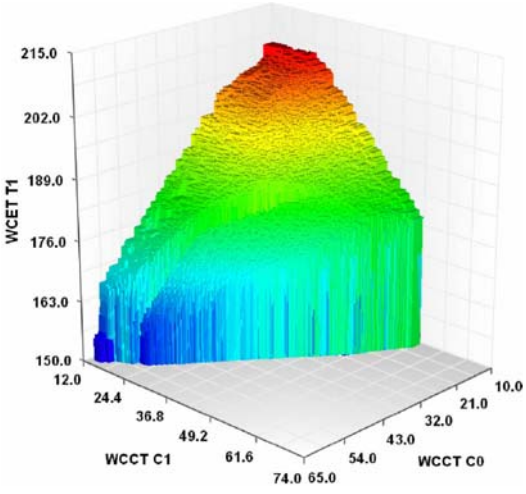
- Distributed embedded system
- 4 priority scheduled computational resources
- connected via CAN bus
- 3 sub-applications
  - Sens→Act
  - S_{in}→S_{out}
  - Cam→V_{out}



Improving slack for T1 + C0 + C1 (1)



Org. Config.

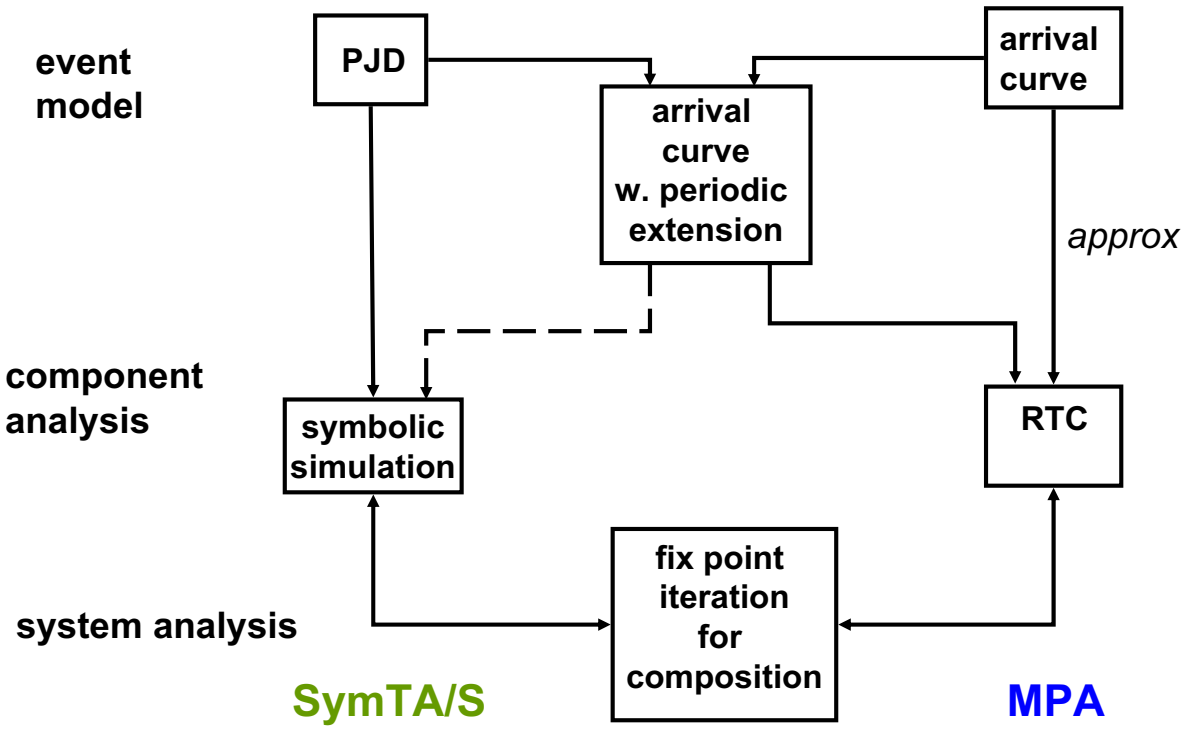


Optimized

Overview

- applications for formal performance analysis methods
- formal performance modeling and analysis principles
- modeling activation and event streams
- component analysis
- system analysis
- enhancements to the basic analysis
- summary and comparison
- conclusion

The Compositional Analysis „Landscape“



Summary and Comparison

- the core difference between MPA and SymTA/S is the component analysis engine and not so much the event model (any more)
- RTC focuses on a closed notation of event and service model with a closed and intuitive formalism as a result
  - hierarchy is easier modeled in this formalism
  - very good results are achieved for the covered design space
- SymTA/S uses a generalisation of the busy window approach by Lehoczky/Tindell that develops the task sequence over a busy window
  - very versatile approach that covers complex features such as release offsets, mutual dependencies (round robin), context switching with blocking (non-preemptive) and „secondary“ memory access models
  - very good results shown for a wide range of industrial systems

Conclusion

- several performance analysis and optimization approaches have been proposed for heterogeneous embedded multiprocessor systems which have brought the technology far beyond the stage of toy examples
- the cost of a predictable design has been reduced by higher modeling and analysis precision
- applications range from early design stages when no executable code is available to verification of design integration
- the technology is applicable both to large scale distributed systems and for MpSoC

References Symta/S 1/4

[RRE03] K. Richter and R. Racu and R. Ernst. "Scheduling Analysis Integration for Heterogeneous Multiprocessor SoC." In *IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, December 2003.

[HHJ05] R. Henia and A. Hamann and M. Jersak and R. Racu and K. Richter and R. Ernst. "System Level Performance Analysis - the SymTA/S Approach." In *IEE Proceedings Computers and Digital Techniques*, 2005.

[RER07] R. Racu and R. Ernst and K. Richter and M. Jersak. "A Virtual Platform for Architecture Integration and Optimization in Automotive Communication Networks." In *SAE World Congress*, Detroit, USA, April 2007.

[RHE07] R. Racu and A. Hamann and R. Ernst and K. Richter. "Automotive Software Integration." In *Proc. of the 44th Design Automation Conference*, San Diego, CA, USA, June 2007.

[HRE06] A. Hamann and R. Racu and R. Ernst. "Formal Methods for Automotive Platform Analysis and Optimization." In *Proc. Future Trends in Automotive Electronics and Tool Integration Workshop (DATE Conference)*, Munich, March 2006.

[HRE07] A. Hamann and R. Racu and R. Ernst. "Multi-Dimensional Robustness Optimization in Heterogeneous Distributed Embedded Systems." In *Proc. of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium*, April, 2007.

[HJR06] A. Hamann, M. Jersak, K. Richter, R. Ernst. "A framework for modular analysis and exploration of heterogeneous embedded systems." In *Real-Time Systems Journal*, Volume 33, pp 101-137, July 2006.

References Symta/S 2/4

[RHE06] R. Racu and A. Hamann and R. Ernst. "A Formal Approach to Multi-Dimensional Sensitivity Analysis of Embedded Real-Time Systems." In *Proc. of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, July 2006.

[RE06] R. Racu and R. Ernst. "Scheduling Anomaly Detection and Optimization for Distributed Systems with Preemptive Task-Sets." In *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, USA, April 2006.

[HE07] R. Henia, R. Ernst. "Scenario Aware Analysis for Complex Event Models and Distributed Systems." In *Proc. Real-Time Systems Symposium*, December 2007.

[SHR07] S. Schliecker and A. Hamann and R. Racu and R. Ernst. "Formal Methods for System Level Performance Analysis and Optimization." In *Proc. of the Design Verification Conference (DVCON)*, San Jose, CA, February 2008.

[SIE06] S. Schliecker and M. Ivers and R. Ernst. "Integrated Analysis of Communicating Tasks in MPSoCs." In *Proc. 3rd International Conference on Hardware Software Codesign and System Synthesis (CODES)*, Seoul, Korea, October 2006.

[SHE06] S. Stein and A. Hamann and R. Ernst. "Real-time Property Verification in Organic Computing Systems." In *Proc. of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, November 2006.

[RE08] J. Rox and R. Ernst. "Modeling Event Stream Hierarchies with Hierarchical Event Models." In *Proc. Design, Automation and Test in Europe (DATE),* Munich, 2008.

[KHT07] S. Künzli, A. Hamann, L. Thiele, R. Ernst. "Combined Approach to System Level Performance Analysis of Embedded Systems". Codes+ISSS 2007, Salzburg, 2007.

References MPA 3/4

- E. Wandeler. Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems. PhD Thesis ETH Zurich, 2006

[WT07a], E. Wandeler and Lothar Thiele, Workload correlations in multi-processor hard real-time systems. Journal of Computer and System Sciences, Mar. 2007

[WTVL06] E. Wandeler and Lothar Thiele and Marcel Verhoef and Paul Lieverse, System Architecture Evaluation Using Modular Performance Analysis - A Case Study, Software Tools for Technology Transfer (STTT), Oct. 2006}

[TWS06] Lothar Thiele and Ernesto Wandeler and Nikolay Stoimenov, Real-time interfaces for composing real-time systems, International Conference On Embedded Software EMSOFT 06, 2006.

[TWC05] Lothar Thiele and Ernesto Wandeler and Samarjit Chakraborty, A Stream-Oriented Component Model for Performance Analysis of Multiprocessor DSPs, IEEE Signal Processing Magazine, special Issue on Hardware/Software Co-design for DSP, May 2006

[WMT05] Ernesto Wandeler and Alexandre Maxiaguine and Lothar Thiele,Quantitative characterization of Event Streams in Analysis of Hard Real-Time Application}, Real-time Systems. Mar, 2005

[CKT03] Samarjit Chakraborty and Simon K\"unzli and Lothar Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs},. DATE 2003.

[TCN99] Lothar Thiele and Samarjit Chakraborty and Martin Naedele, Real-time Calculus for Scheduling Hard Real-Time Systems, International Symposium on Circuits and Systems ISCAS 2000, Mar. 2000.

References other 4/4

- J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In Proceedings Real-Time Systems Symposiom, pages 201–209, 1990.
- K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time systems. Journal of Real-Time Systems, 6(2):133–152, Mar 1994.
- F. Baccelli, G. Cohen, G. J. Olster, and J. P. Quadrat, Synchronization and Linearity --- An Algebra for Discrete Event Systems, Wiley, New York, 1992.
- J.-Y. Le Boudec and P. Thiran, Network Calculus - A Theory of Deterministic Queuing Systems for the Internet, Lecture Notes in Computer Science, vol. 2050, Springer Verlag, 2001.

Formal Methods in System and MpSoC Performance Analysis and Optimisation

Hans Sarnowski
EF-611, BMW Group

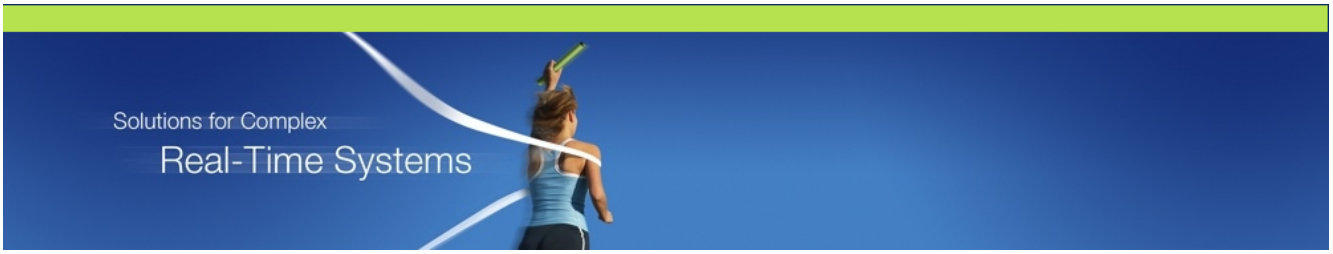
Marek Jersak
Symtavision GmbH

Tutorial, DATE 08 Conference
March 10, 2008

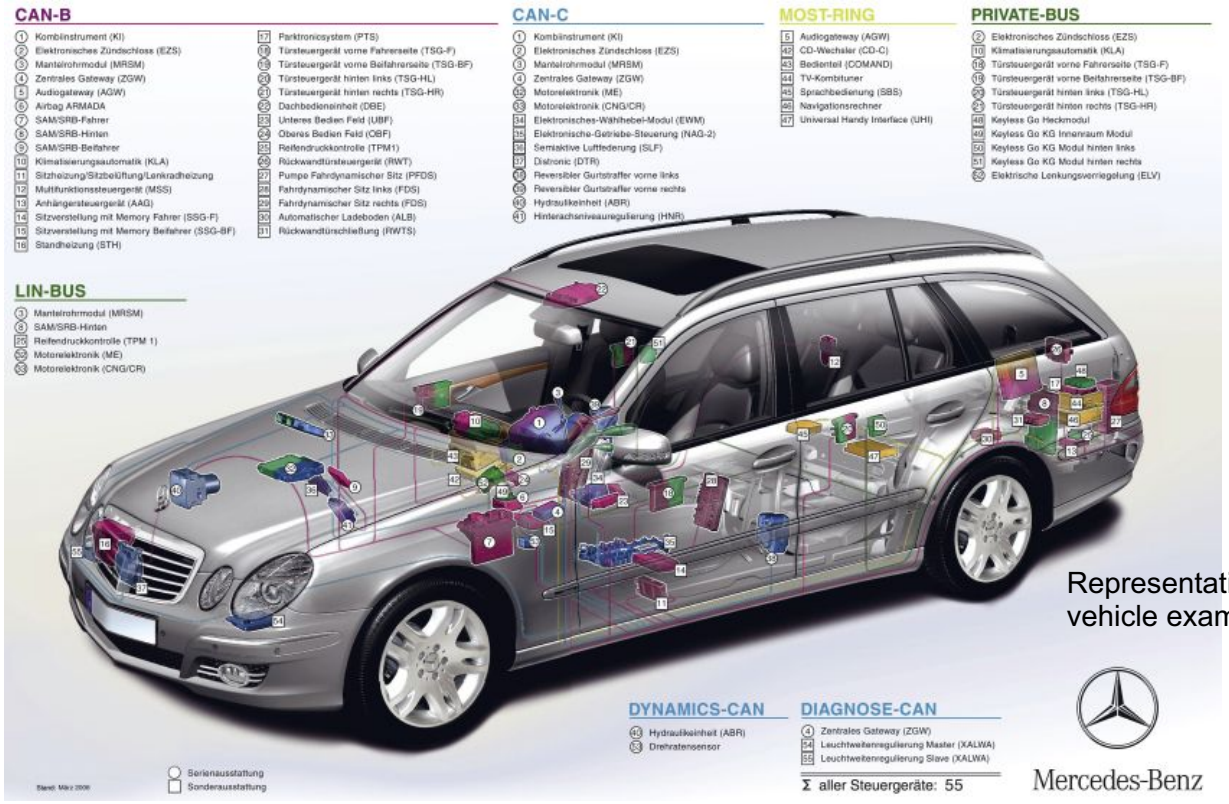
SYMTA VISION

Performance Analysis and Optimisation – industrial applications in automotive design

Tutorial, DATE 08 Conference
March 10, 2008



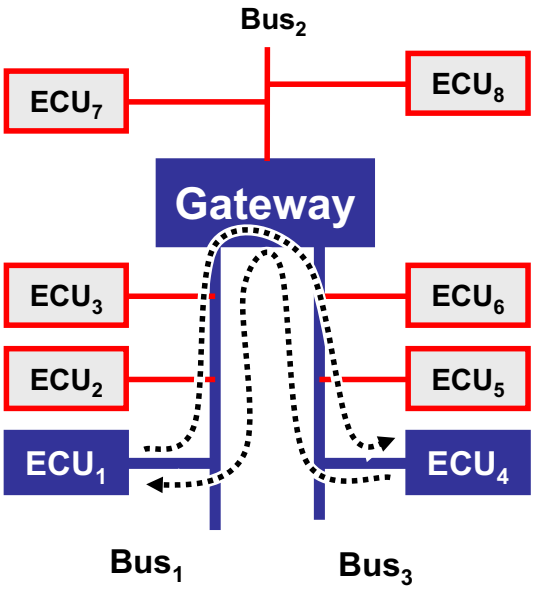
Integration Challenges: 5 buses, 55 ECUs, hundreds of messages, thousands of functions



Representative vehicle example

Mercedes-Benz

Typical Automotive Architecture Today



Bus protocols

- CAN
- FlexRay
- Lin
- MOST
- Proprietary

ECU (electronic control unit)

- Single-Core CPU (but moving to dual-core)
- OSEK RTOS

Local + end-to-end timing / performance are important

Many functional problems are in fact *timing* problems

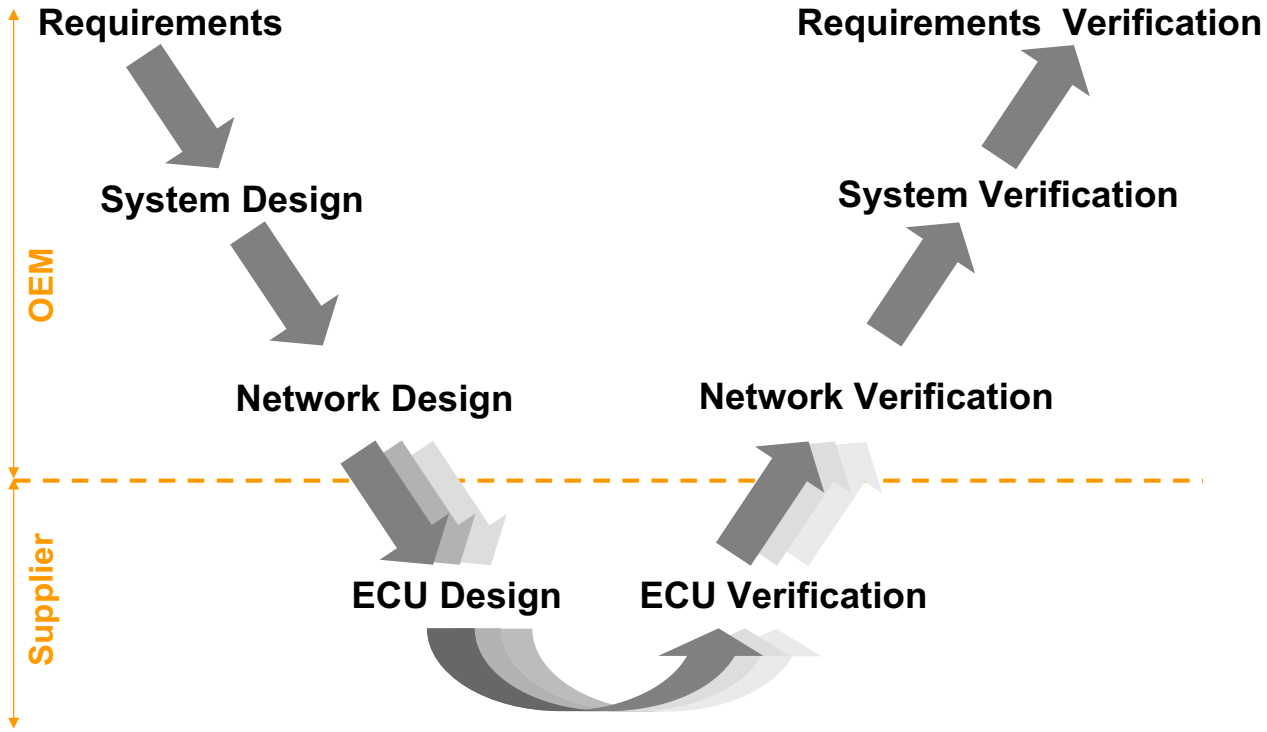
- ECUs (temporarily) overloaded
- tasks not always schedulable
- deadlines are missed

- network (temporarily) overloaded
- messages arrive "too late“ or with “too large” jitter
- messages are lost (buffer overflow)

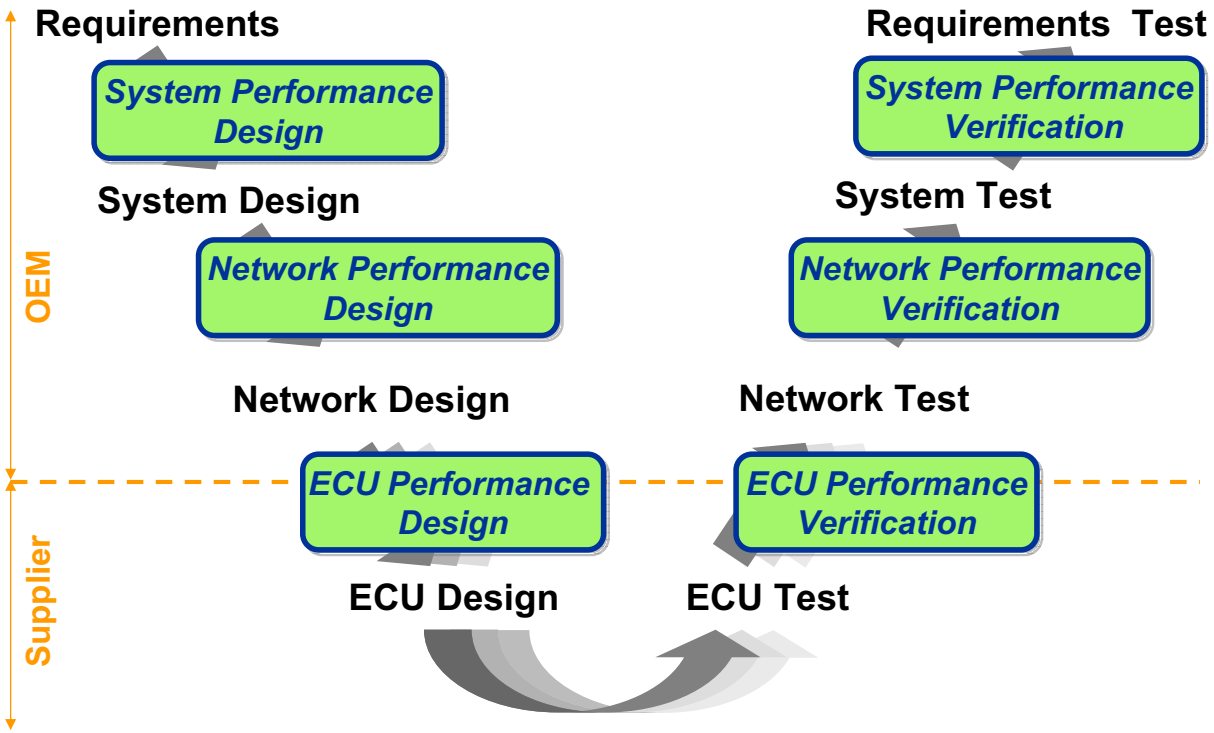
- end-to-end deadlines of car function are missed
- stability of distributed control is compromised

➔ Carefully monitor performance and timing during design and integration

Established Design Process



Adding Performance Design and Verification



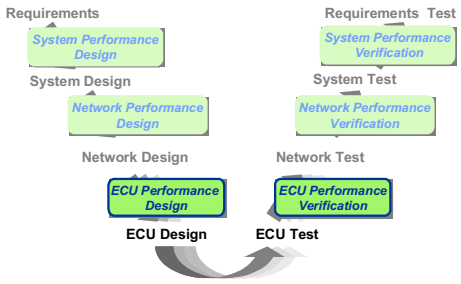
Selected Automotive Use Cases

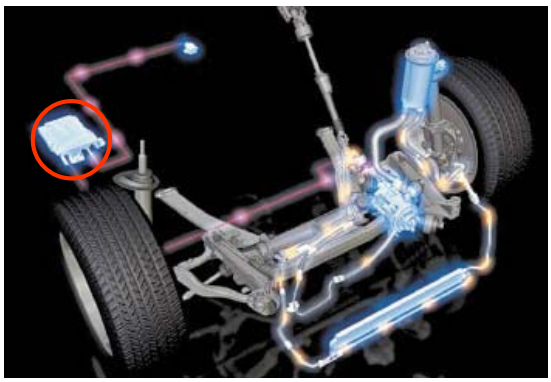
SYMTA VISION
Performance Analysis and Optimisation – industrial applications in automotive design
© Symtavision GmbH, Germany

Example 1: Safety-Critical ECU

Chassis domain: Active Front Steering

- Verifying Performance and Timing for all critical cases
- Safeguarding against liability claims
- Optimizing ECU performance and cost (use of cheaper CPU)

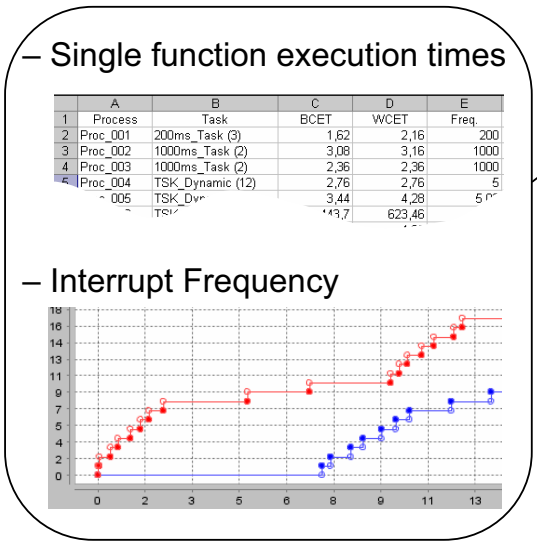




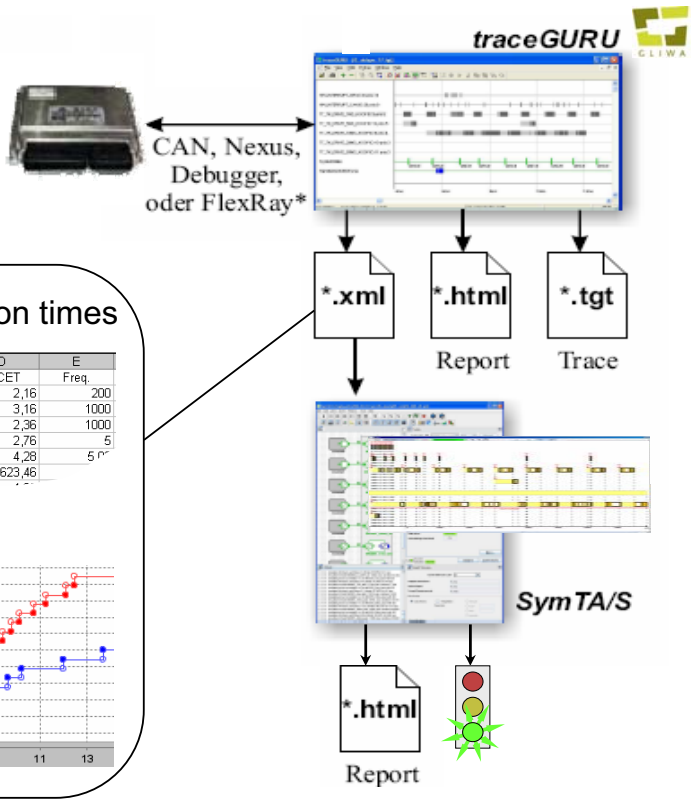
Source: BMW

Hans Sarnowski, responsible BMW Engineer: „You really get to know your system and can detect real-time errors in a fraction of time“

Integration: Tracing + SymTA/S

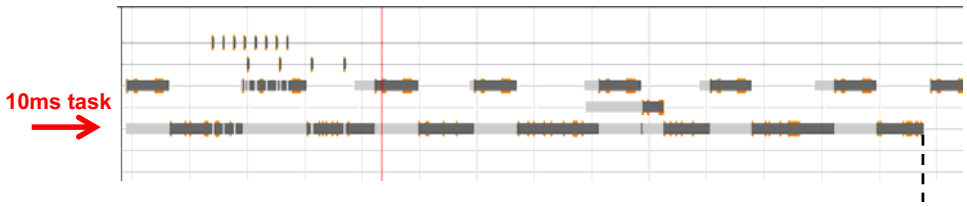


	A	B	C	D	E
1	Process	Task	BCET	WCET	Freq.
2	Proc_001	200ms_Task (3)	1,62	2,16	200
3	Proc_002	1000ms_Task (2)	3,08	3,16	1000
4	Proc_003	1000ms_Task (2)	2,36	2,36	1000
5	Proc_004	TSK_Dynamic (12)	2,76	2,76	5
	005	TSK_Dyn	3,44	4,28	5
			143,7	623,46	

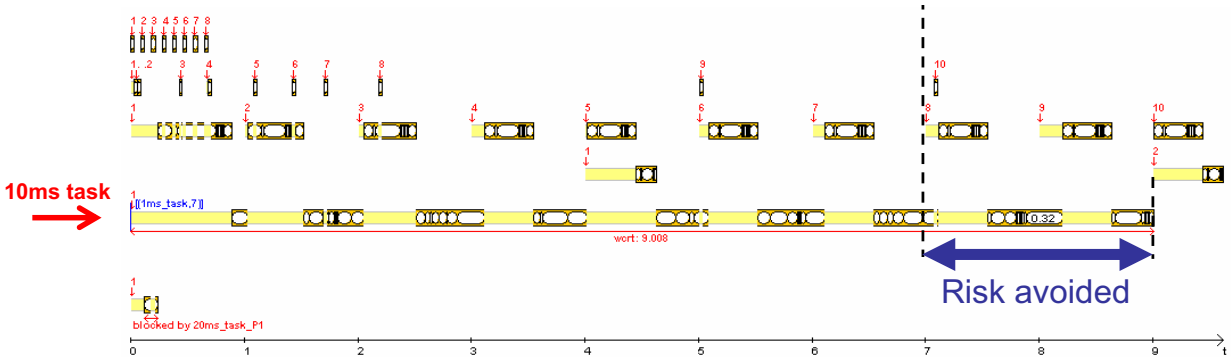


Focus: Tracing vs. SymTA/S Analysis

- Measured 10ms task: Response time 6,9ms
  - 4 CAN, 8 SPI interrupts, 7 preemptions by 1ms task



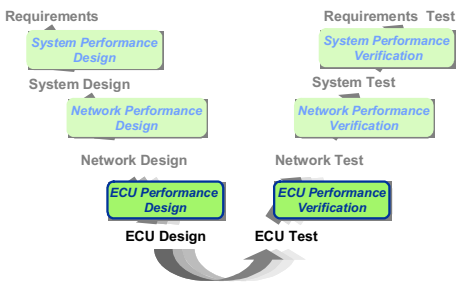
- SymTA/S Analysis of 10ms task: Worst-case response time 9ms
  - 10 CAN, 8 SPI interrupts, 9 preemptions by 1ms task, blocking

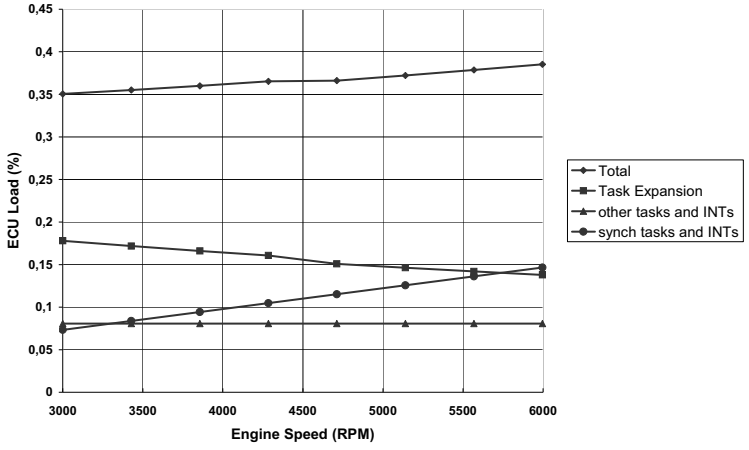


Example 2: High-Performance ECU

Powertrain domain: Engine Control

- Verifying Performance and Timing for all engine speeds (RPM)
- Avoiding Deadline Overruns (would lead to ECU reset)
- Optimizing ECU performance and cost for different markets

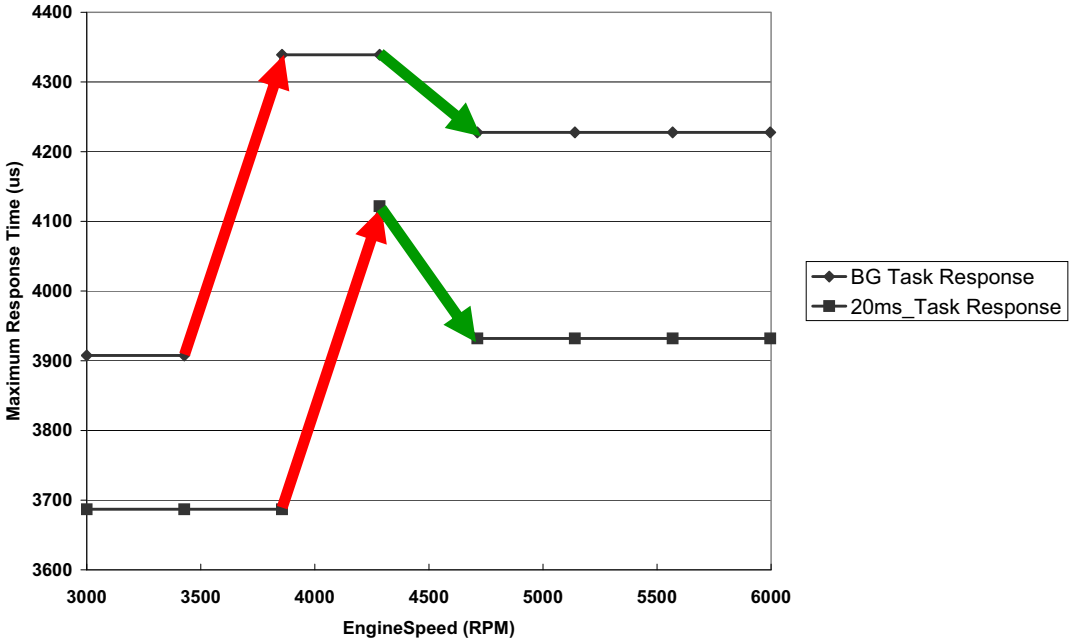




Detecting "Anomalies"

Additional preemption by RPM-synchronous tasks (increases task interference)

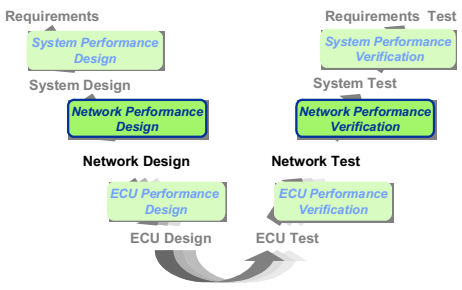
Task cut-off (reduces core execution time)

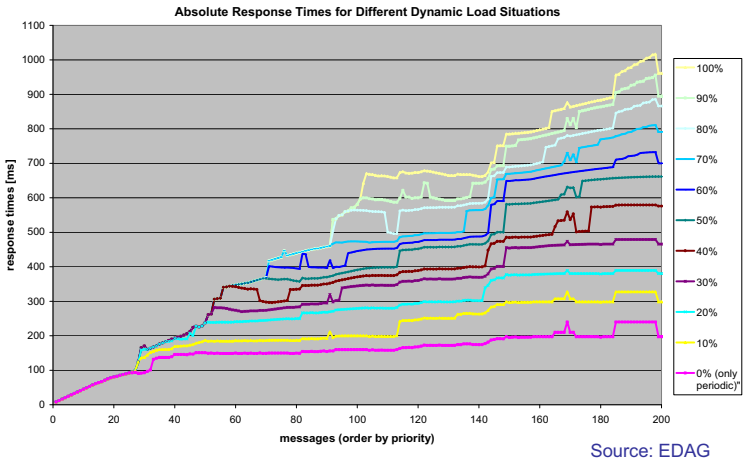


Example 3: Bus Configuration

Bus / Network : CAN-Protocol

- Balancing Periodic load
- Calculating limits for dynamic load
- Configuring existing networks to handle additional traffic

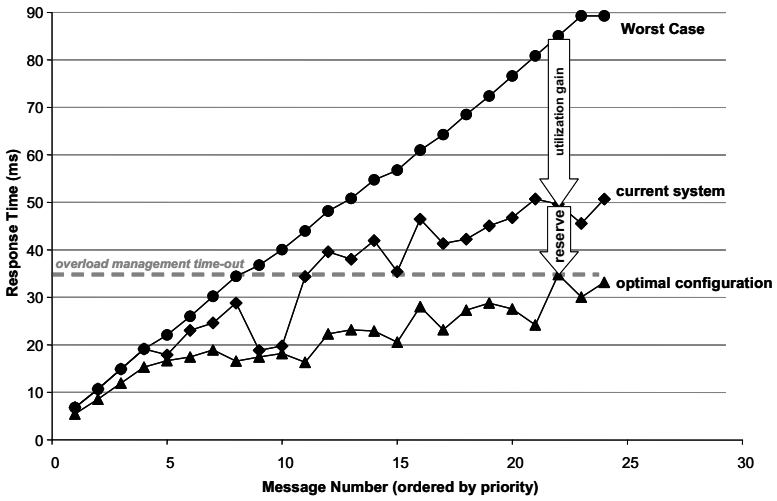




Source: EDAG

Focus: Reliable CAN Bus Configuration

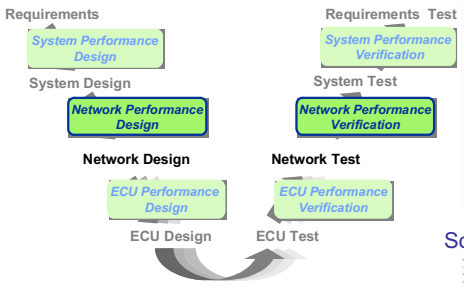
- Optimized COM-Task Offsets
- Optionally: Optimized signal to frame mapping, CAN IDs
- Result: Reliable and optimized bus extension

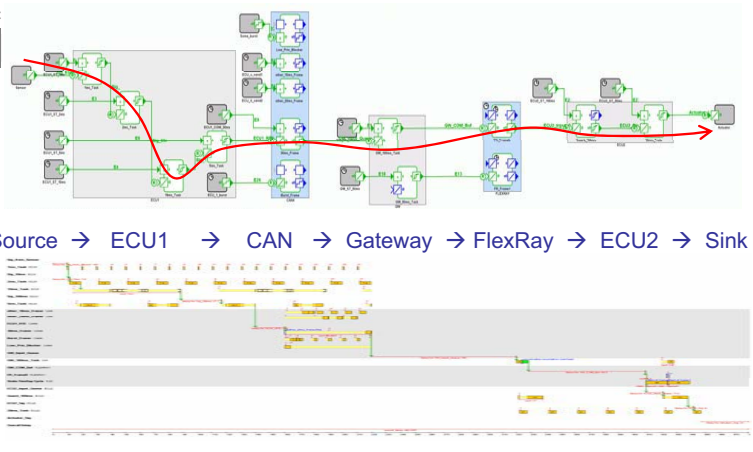


Example 4: Network Extension

Bus / Network : Gated Network

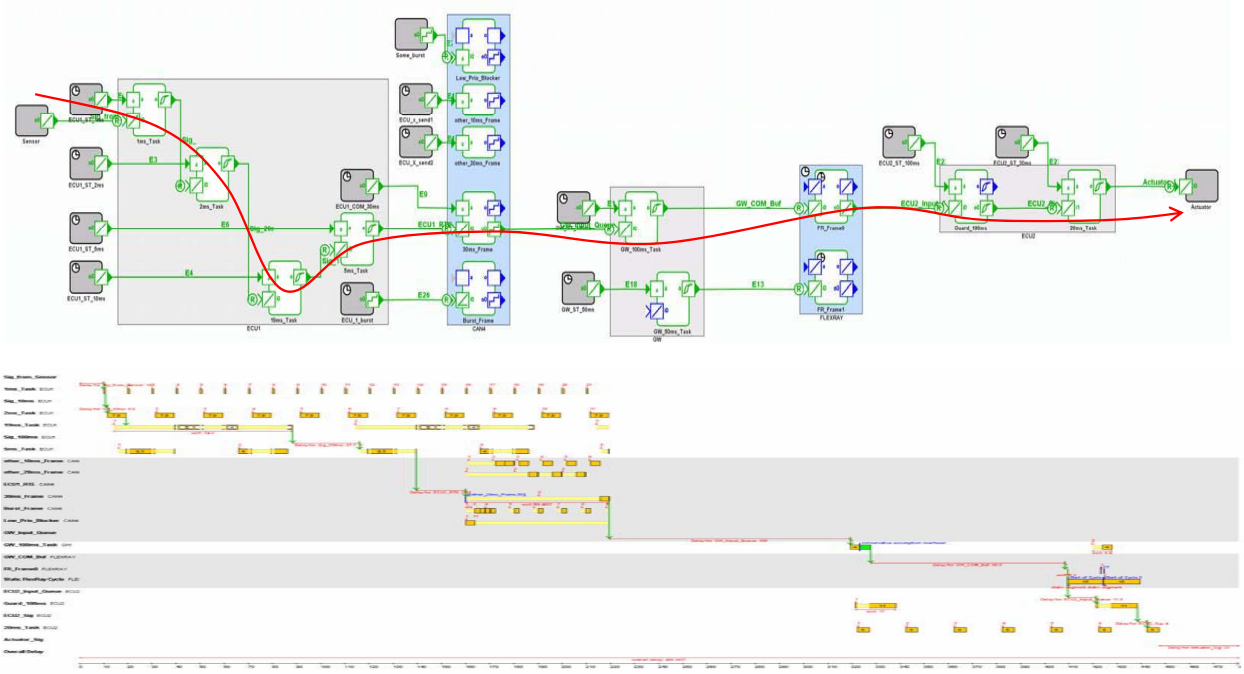
- Verifying end-to-end Timing
- Gateway dimensioning
- Optimizing synchronization to reduce end-to-end latency





Focus: End-to-end Timing Analysis

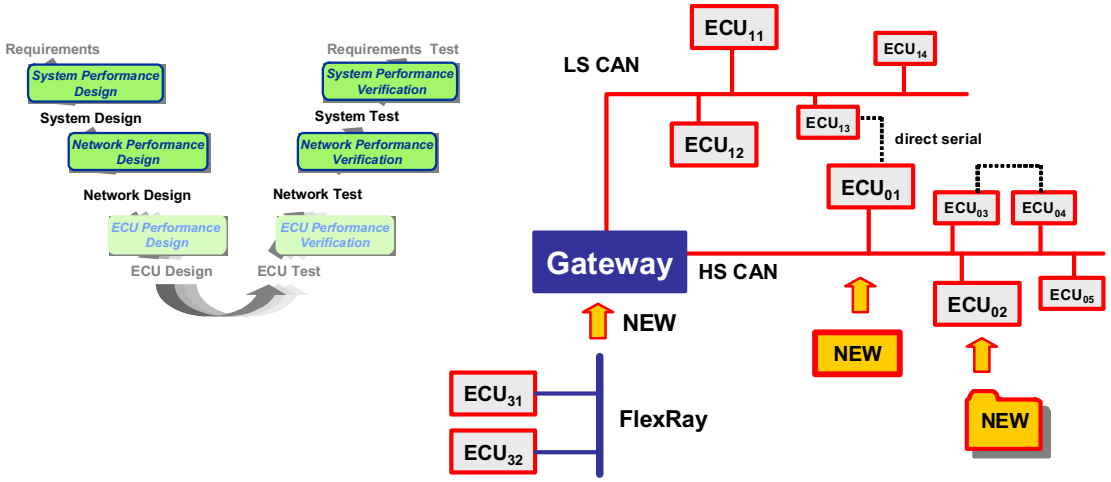
e.g.: Source → ECU1 → CAN → Gateway → FlexRay → ECU2 → Sink



Example 5: System Extension

Automotive System: Many ECUs and Protocols

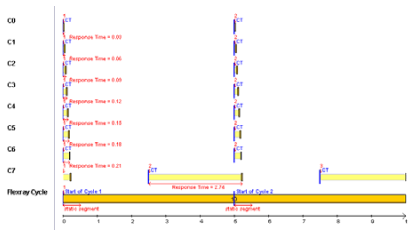
- Complete System-level analysis of alternative configurations
- Migration to FlexRay and AUTOSAR
- Timing contracts between Integrator and Supplier



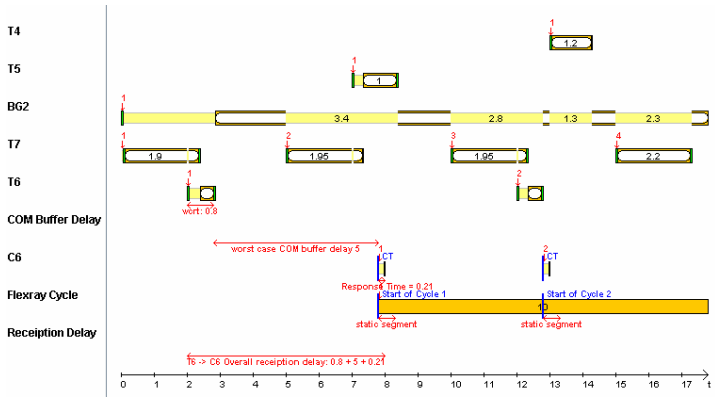
Timing Challenges in FlexRay and AUTOSAR

FlexRay : A challenge for ECU integration

- FlexRay does not solve timing problems in general
- A good FlexRay design requires timing effects to be understood
- Sync / async ECU integration can make a huge difference



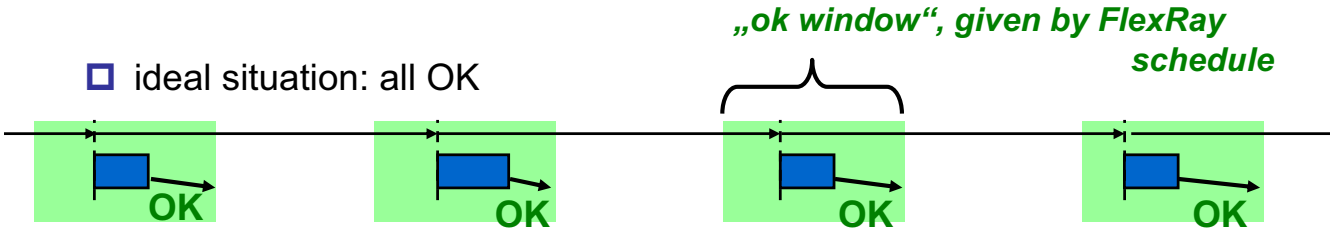
simple, static FlexRay schedule



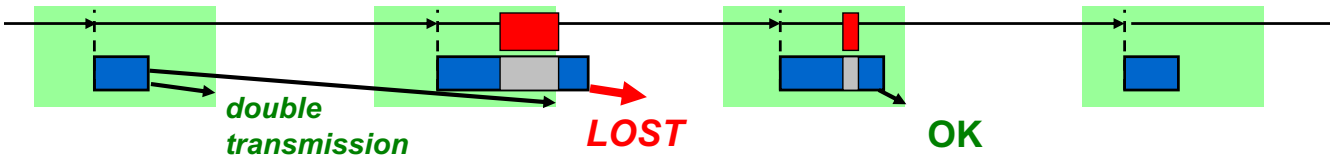
complex, dynamic system schedule

OSEK (preemptive OS) synchronized with FlexRay

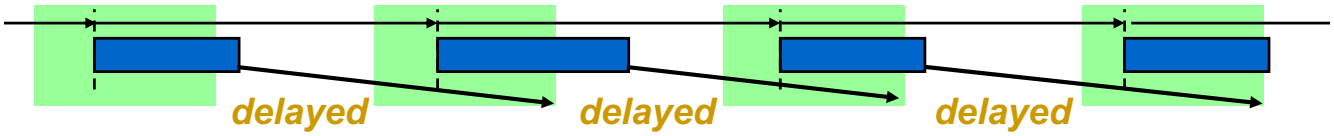
- ideal situation: all OK



- typical situation: some ok, some double, some lost

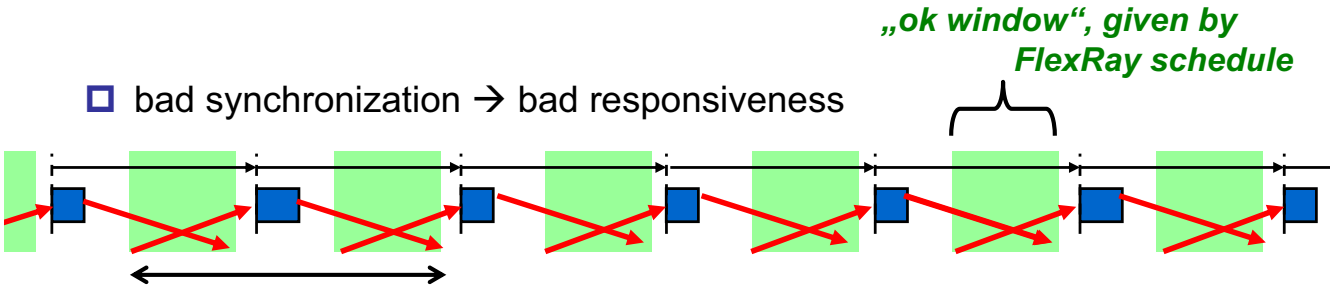


- no loss but all delayed by one full FlexRay cycle

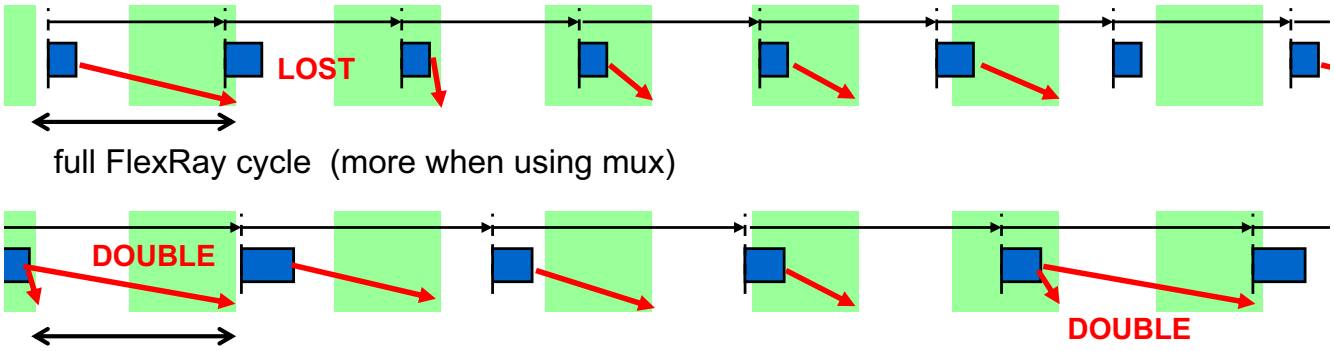


OSEK ECU Asynchronous to FlexRay

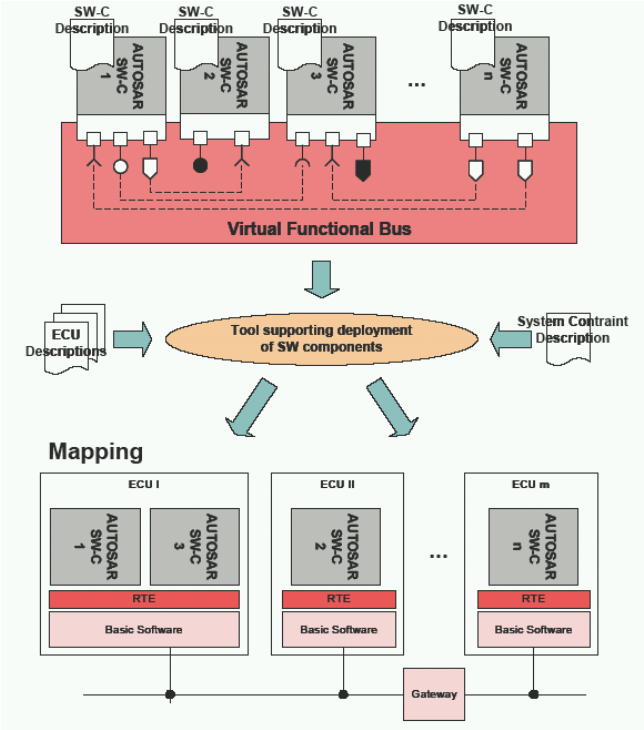
- bad synchronization → bad responsiveness



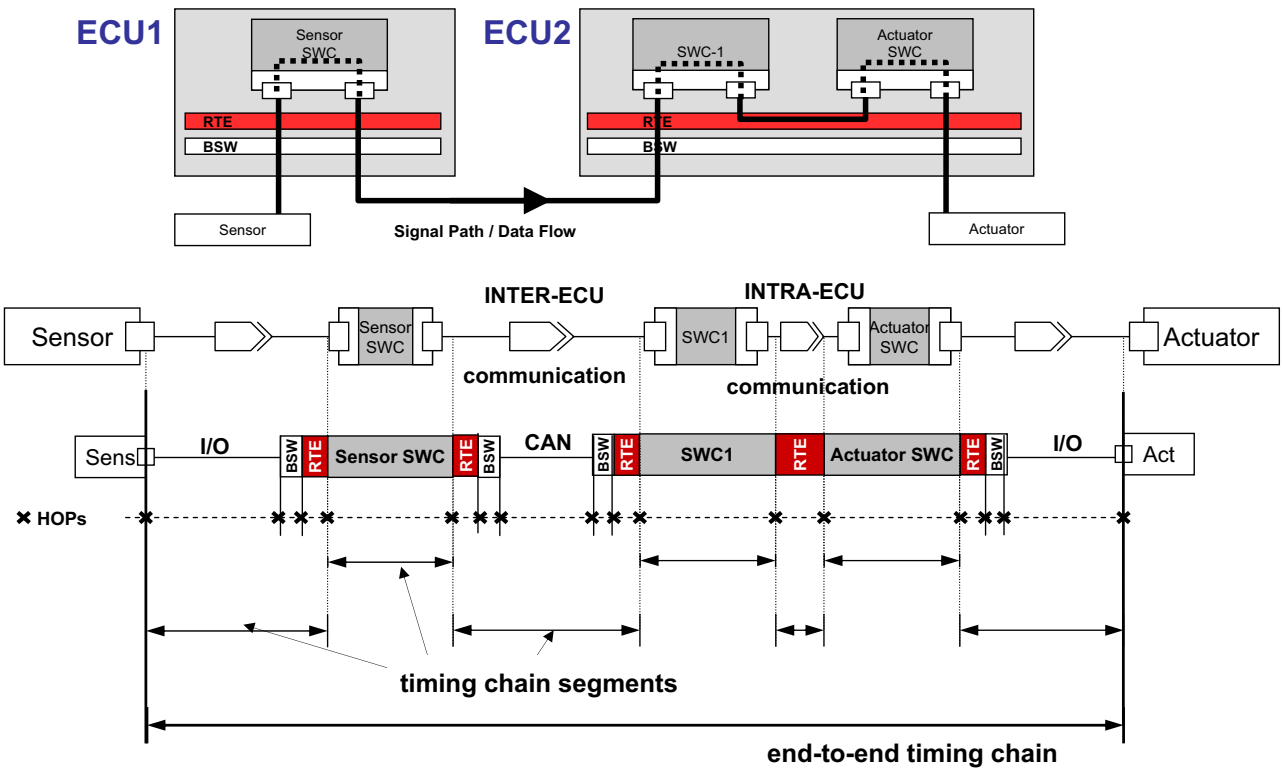
- clock skew effects → large send & receive signal jitters



AUTOSAR Vision

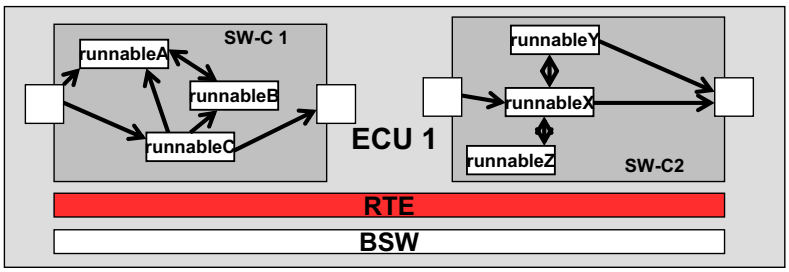


AUTOSAR System Timing Aspects

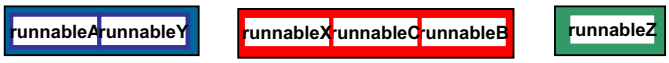


AUTOSAR SW-C vs. "Runnables" and Tasks

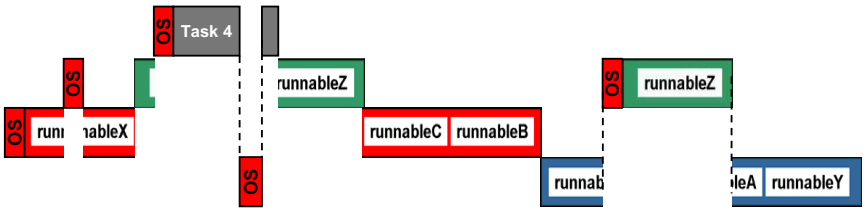
- SW architecture:
  2 SW components,
  6 runnables

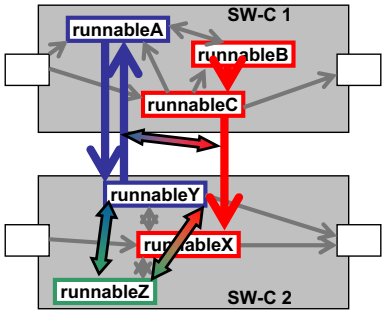


- Implementation: 3 Tasks



- Schedule and timing dependencies





Ongoing work

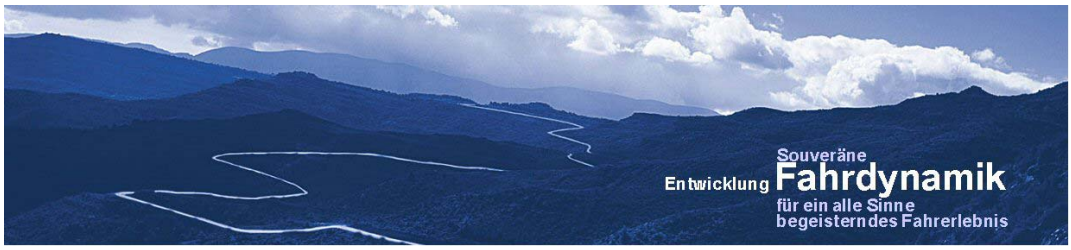
- INTEREST – FlexRay Methodology
  http://www.interest-strep.eu/index.html

- TIMMO – AUTOSAR Methodology
  https://www.timmo.org/

INTEREST is a project funded by the Sixth European Framework Programme (FP6) - including the Information Society Technologies (IST) priority. The responsibility for the content rests with the authors.

TIMMO is a project in the framework of the ITEA 2, EUREKA cluster programme Σ! 3674. The work of the German partners is funded by the German Ministry for Education and Research (BMBF) under the funding IDs 01IS07002(B-I,K). The responsibility for the content rests with the authors.

Time is Money – Real-Time is a lot of Money
System Optimization for BMW-Active Front-Steering



Hans Sarnowski
EF-611

DATE'08 Conference, 10. March 2008, Munich

BMW Group

Steering with Pleasure.
Agenda.

- Introduction
- System overview Active Front-Steering
- Application of timing tools for Active Front-Steering design
- Evaluation of results

Introduction.
Design goals.

- Conventional steering systems have no further potential as the diversification/variation of hand moments (Servotronic) is largely exhausted.
- Driving performance is increasing continuously. A rigid steering system can hardly provide both stability and handling.
- Stability control through breaking of individual wheels is experienced as more and more uncomfortable.

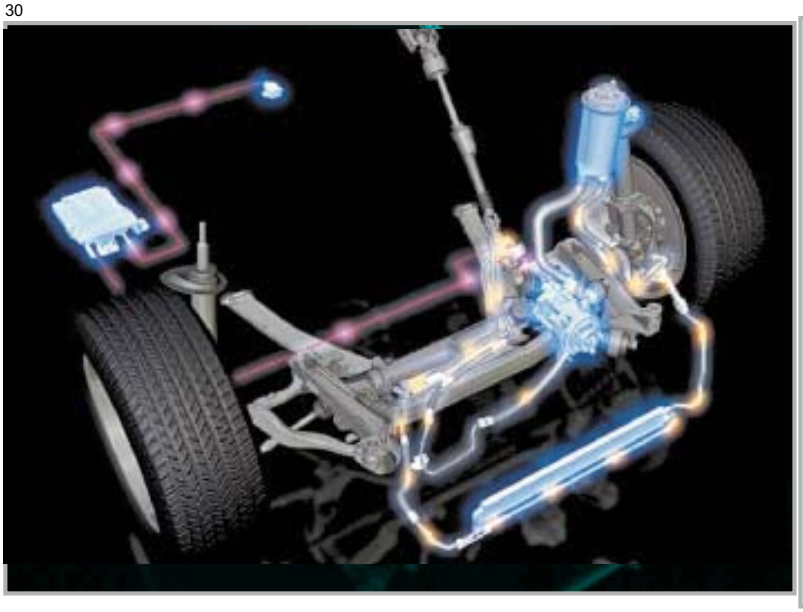
↓

Superposition steering offers both the advantages of conventional steering systems and the functions of steer-by-wire-systems.

↑

- Pure by-wire-systems are (still?) too complex, too costly and show a synthetic steering sensation. Their market acceptance is questionable.

System Overview
Components of Active Front-Steering

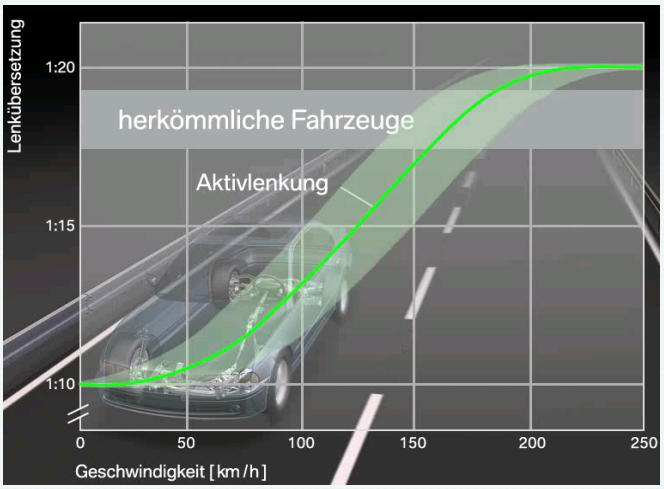


Components of Active Front-Steering:

- Rack steering
- E-Motor
- Superposition gearbox
- Steering column
- Control unit
- Sensor cluster
- Power steering pump, oil tank
- Tubes, cooler

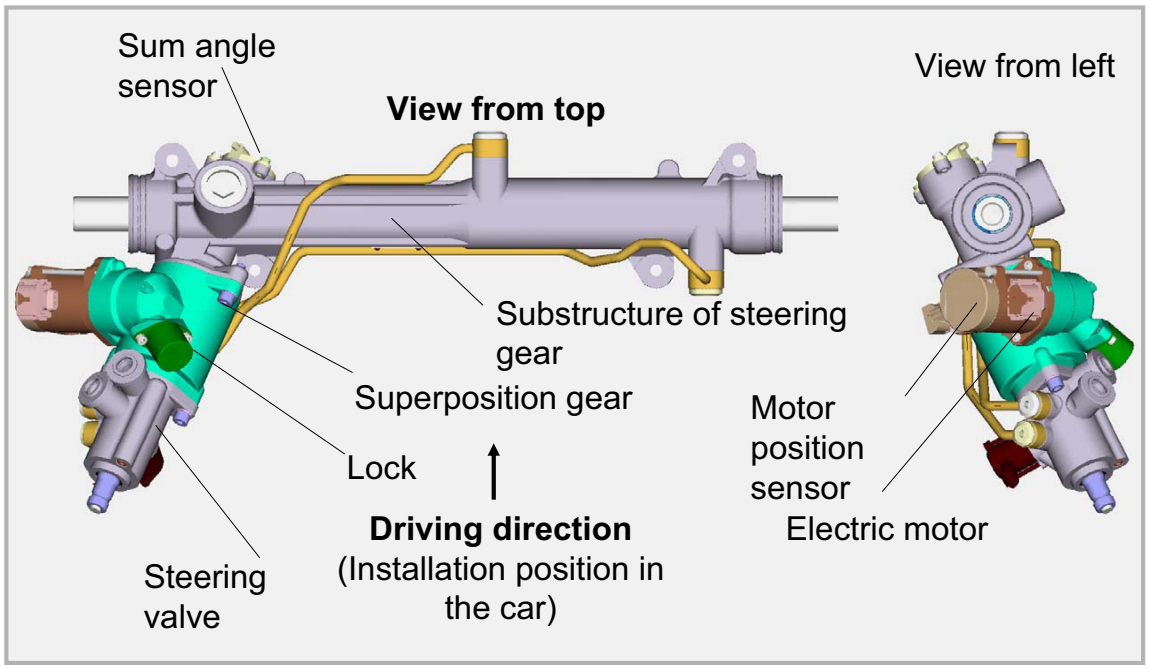
Function Overview
Steering Functions

- The steering ratio depends on steering wheel angle and vehicle velocity
- Adjusting objective target-settings according to the driving dynamics of the vehicle
- Coupling with Servotronic (Steering transmission and the level of steering power are adjusted to each other)



Superposition steering offers both the advantages of conventional steering systems with the functions of steer-by-wire-systems.

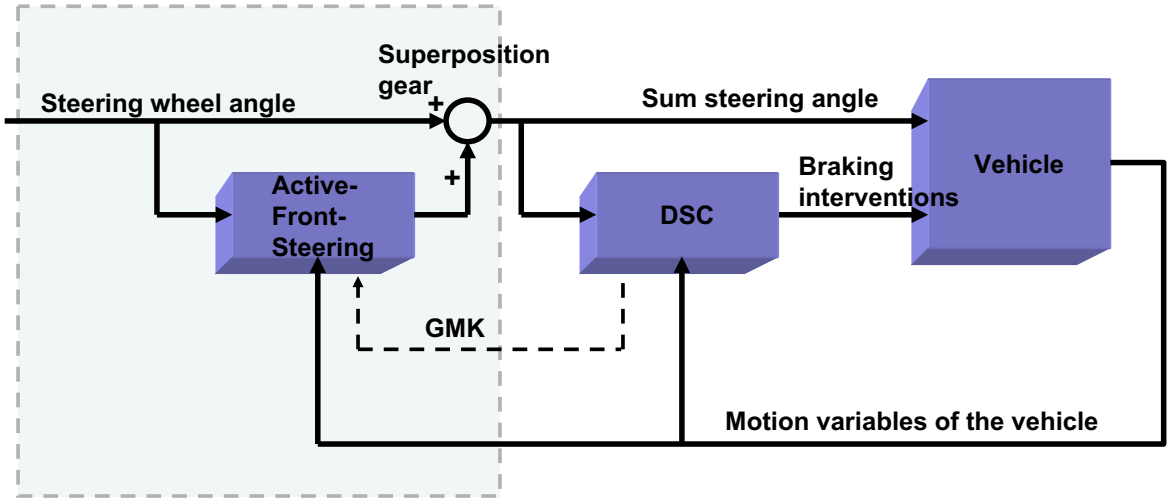
Mechanics.
Mechanical Design.



Quelle: ZFLS GmbH

Function Overview

Integration of Active Front-Steering - DSC.



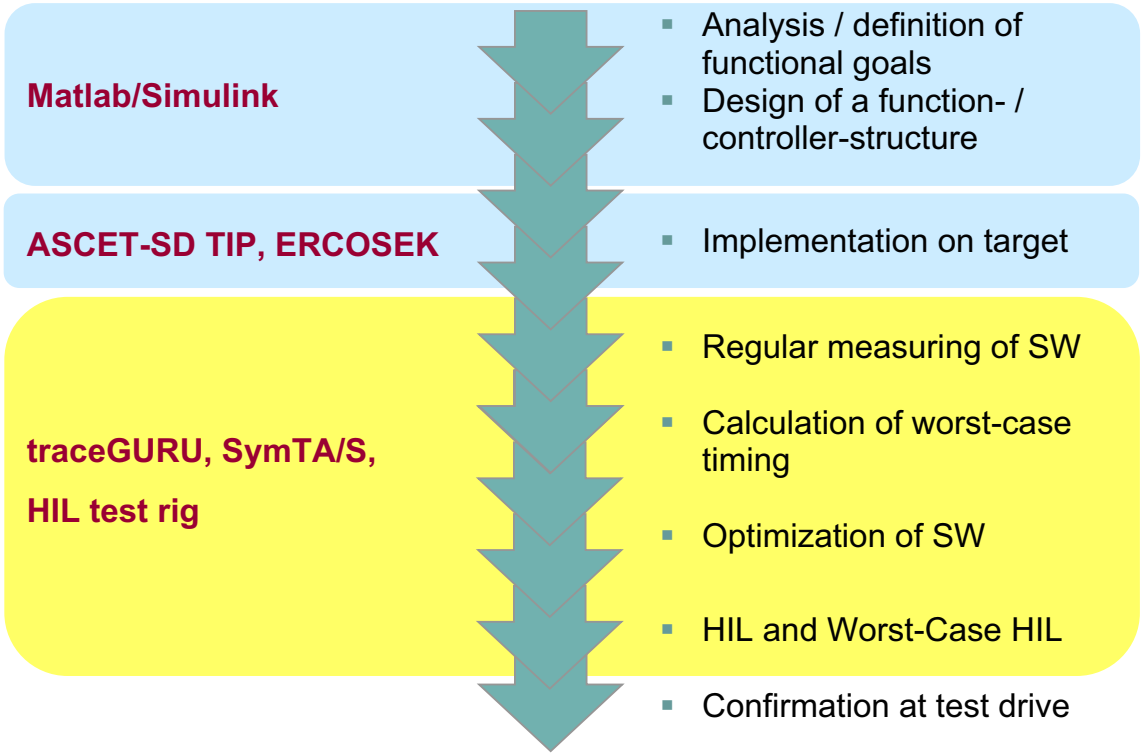
Goal Conflict

Use of a considerably cheaper, simpler and slower processor hardware at a higher functionality

Measures:

- Constant control with measuring and visualization tools **traceGURU** (Gliwa GmbH)
- Use of Scheduling Analysis tool **SymTA/S** (Symtavision)
- Setup of networked "hardware in the Loop" test rig with "worst-case" configuration

Development Process
Tool Chain



Development
Use of timing tools

Timing tools were used for the following actions:

- Processor selection
- Execution time measurements for Integration stages (releases)
- Design of timing layout
- Relaxation of run time situation
- Measurement of individual functions
- Optimization of run times
- Detection of timing problems
- Verification of timing corner-cases

Use of Timing Tools

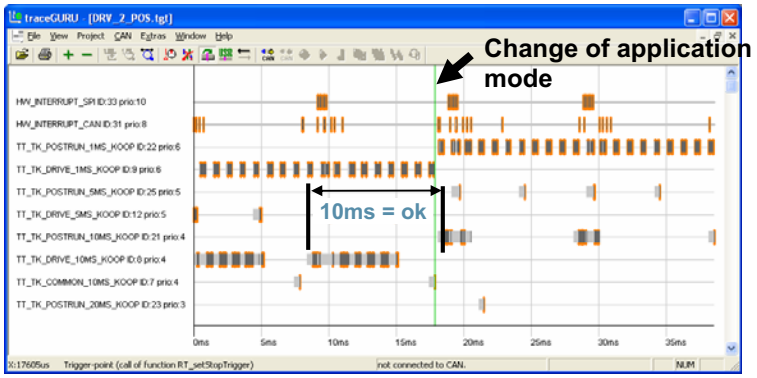
Debugging of timing problems I

Use of ***traceGURU*** as debugging tool for specific timing problems

Example: A significant deviation from the given 10 ms period occurs occasionally for CAN messages

Cause: timers were reset during "application mode changes", so that the required 10 ms pattern was violated

Solution: Adjusting the delays of periodic tasks which operate the CAN driver



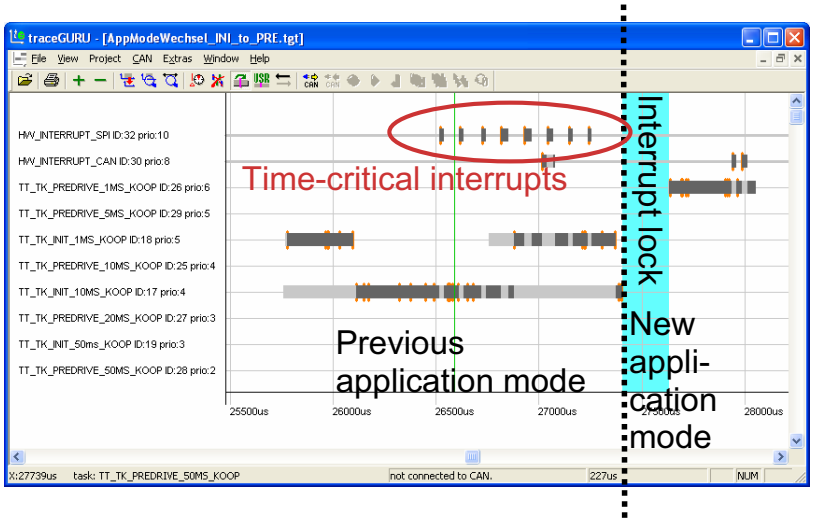
Use of Timing Tools

Debugging of timing problems II

Example: Infrequent error occurring at the calculation of the delay angle

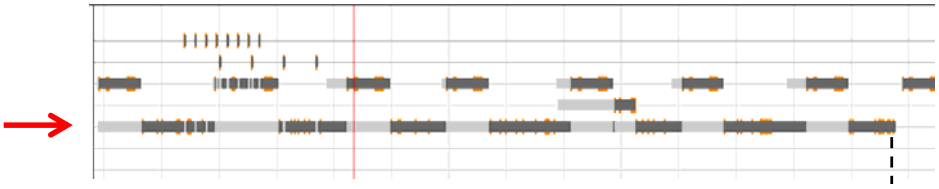
Cause: Coincidence of a series of time-critical interrupts and application mode change. This causes a longer disabling of interrupts at a particularly **short(!)** run time of the 10 ms task.

Solution : A modified timing layout and deterministic switch of the application mode, this provides that the interrupts cannot coincide with the lock.

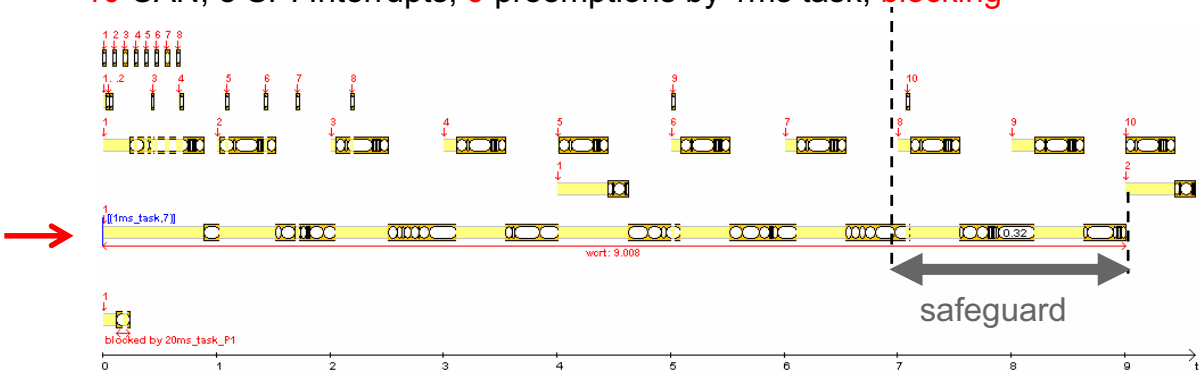


Comparison of Tracing and Scheduling Analysis

- **Measurement:** Response time 6.9ms
  4 CAN, 8 SPI interrupts, 7 preemptions by 1ms task

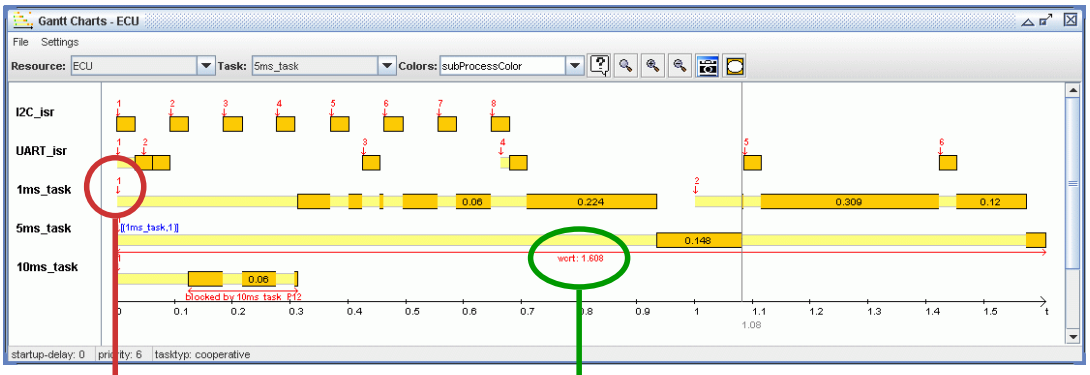


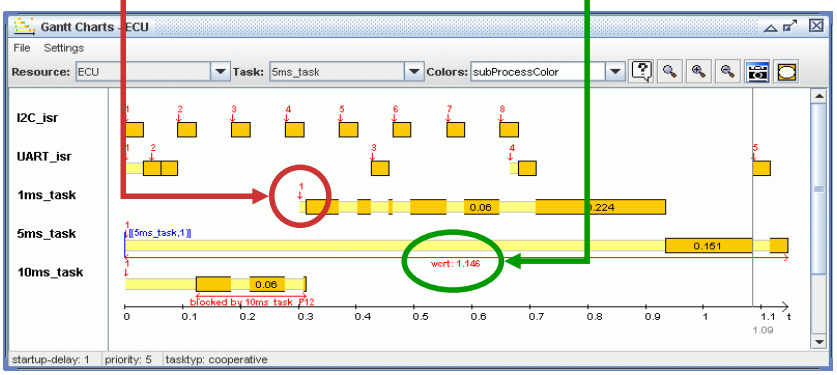
- **Worst-Case Analysis with SymTA/S:** Response time 9ms
  10 CAN, 8 SPI Interrupts, 9 preemptions by 1ms task, blocking



Optimization:
Delayed task activation

Use of **SymTA/S** for delay optimization





Response time 5ms Task
Previous: 1.608 ms
New: 1.146 ms

Use of timing tools
Processor selection

Goal:

Cost reduction at higher functionality

Situation: MPC Processor family is set, but which derivative to choose?

- a derivative providing internal flash (as used in the previous project) or

- a derivative without internal flash (slower but lower-priced).

After comprehensive delay analyses (***traceGURU***/***delayGURU*, SymTA/S***) and an estimation of future functionality the lower-priced alternative without internal flash was selected.

With hindsight the right decision was made: the processor load of the meanwhile completed series software just differs insignificantly from the prediction.

****delayGURU***: targeted and run time scalable (can be automated if required) delay of the application

Safety requirements
Understanding the system
Predictable, time-stable software behavior

Similar to the software function design ("what is happening?"), the timing behavior is to be defined precisely ("when does it happen?")

Conclusion: The graphic display showing the execution of tasks, processes, interrupts and arbitrary pieces of code provides completely new insights into the software.

„You really get to know your system and this enables you to discover runtime errors in a fraction of time."

Summary

- **Strengths of Tracing**
  - Visualization enables a fast understanding of the system
  - Debugging of special, clearly identifiable timing problems
  - Easily connectable to a real ECU and real buses
  - Fast and efficient input of timing data
- **Strengths of Scheduling Analysis**
  - Safeguarding against hard-to-find worst-cases
  - Analysis of end-to-end timing
  - Fast optimization of the system configuration
  - Architecture exploration already in early design stages
- **The combination enables a fast and effective development.**

Dataflow analysis for predictable multiprocessor design

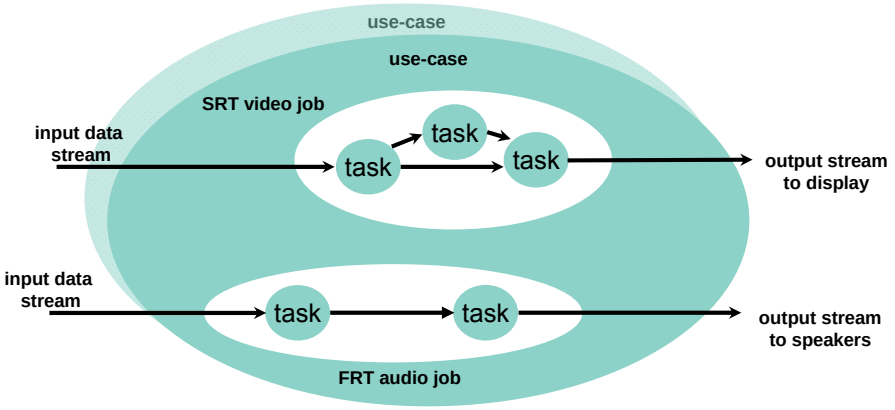
Marco Bekooij

Outline

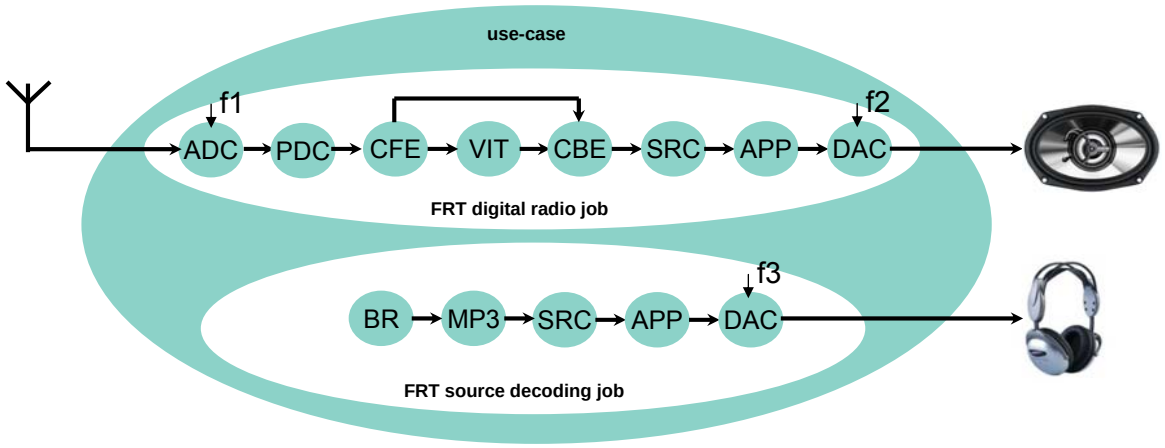
- Context
  - Stream processing applications
  - Application characteristics
  - Architecture characteristics
  - Design requirements
- Dataflow analysis techniques
  - Latency-rate characterization of schedulers
  - Cyclic dependencies between tasks
  - Single rate dataflow analysis
- Summary

Application model

- Jobs are *composed of tasks*
- Simultaneously running jobs *together form use-cases*
- Jobs often have *real-time requirements*
  - Firm (FRT) if deadline misses are *highly undesirable: steep quality degradation*
  - Soft (SRT) if occasional *deadline misses are tolerable*



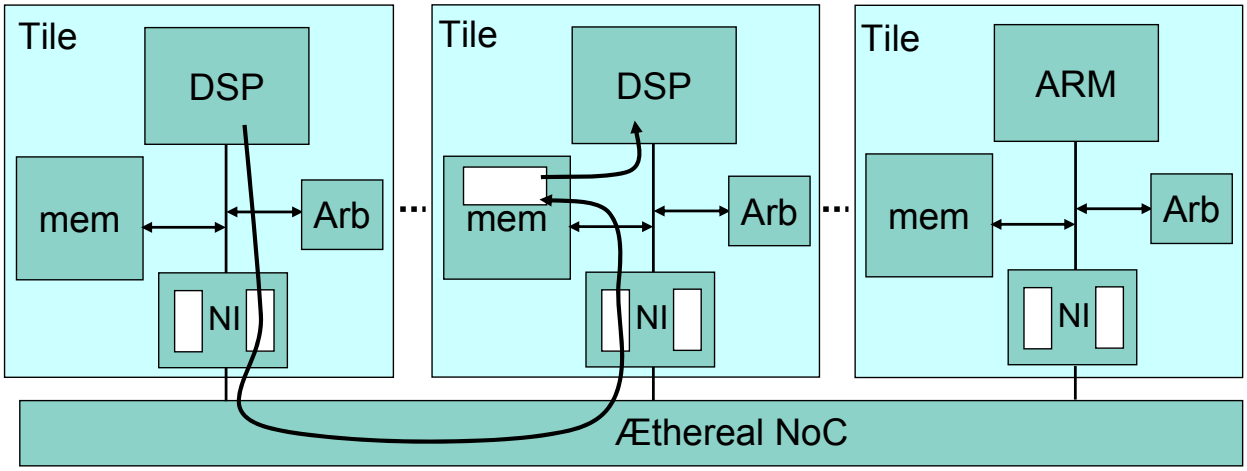
In-car infotainment use-case



- Observations:
  - Reactive system because stream from transmitter cannot be slowed down
  - Firm real-time jobs because deadline misses are highly undesirable but not catastrophic
  - Both streams are equally important
  - Throughput constraints dominate latency constraints

Architecture evolution

[A. Moonen et. al., GSPx2005]

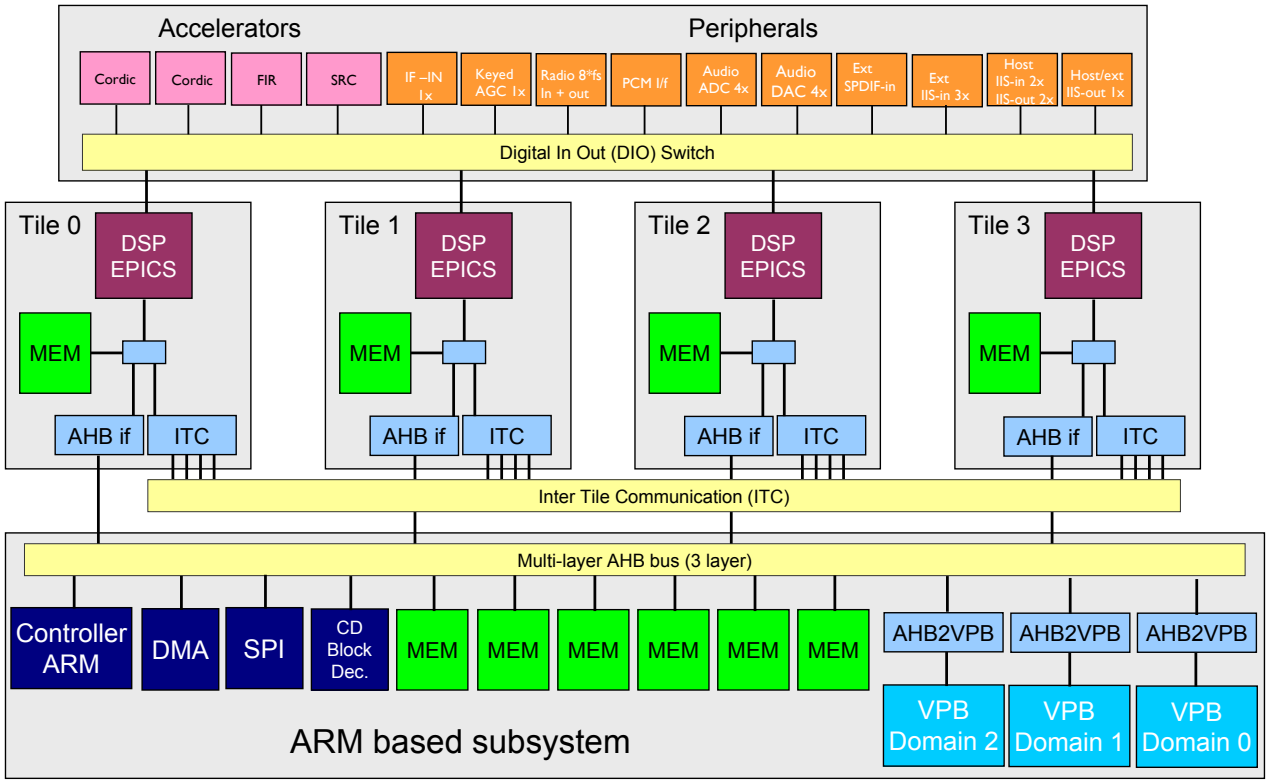


- Statically configured connections + communication *with* addresses
- Processors write in destination memory
- On-chip memories preferably small (<256kByte)

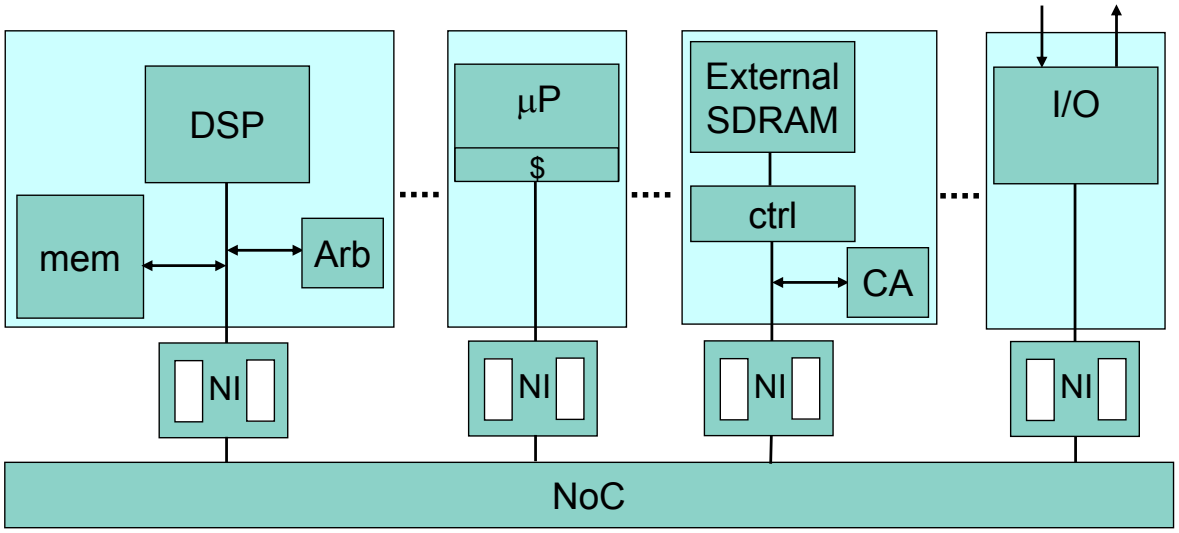
FPGA demo [A. Hansson, University Booth, DATE 2008]

Car-radio IC of NXP

[A. Moonen et. al., GSPx2005]



Architecture evolution

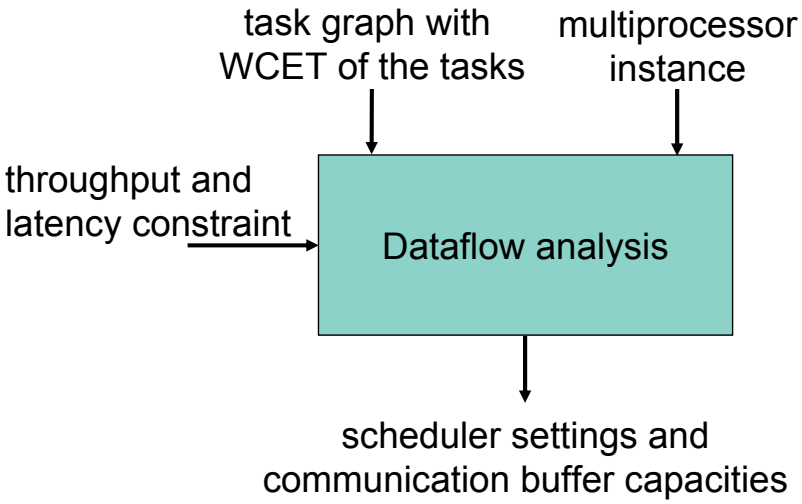


- Statically configured connections + communication with addresses
- Large external memory is needed for digital radio channel decoders

Use of WCET of the tasks is not cost effective!

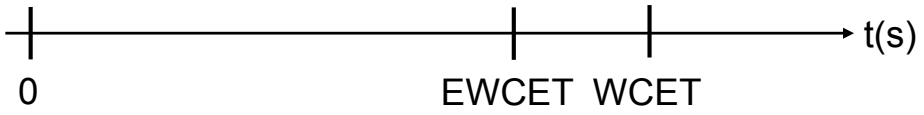
[M. Bekooij et. al.: Bits & Chips 2007]

Dataflow analysis



Guarantee: no deadline misses for all possible input streams

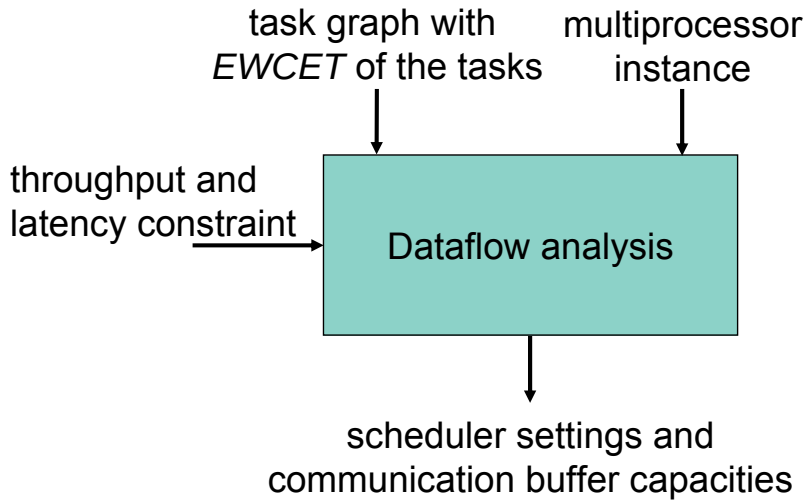
Estimated worst-case execution time (EWCET) of tasks



EWCET= measured upper-bound for the set of test streams
- requires knowledge input data to obtain a tight conservative WCET
- caches and external SDRAM ⇒ large difference average and worst-case

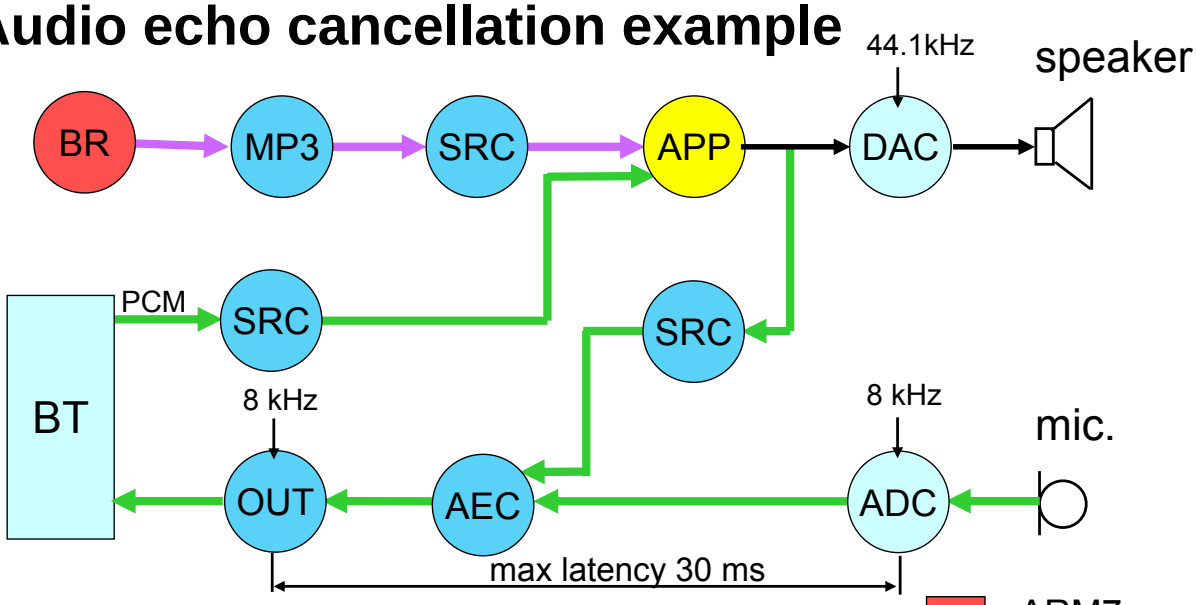
Design requirement:
- guarantee that there are no deadline misses for the set of test streams
  - impractical to do better!

Dataflow analysis for firm real-time jobs



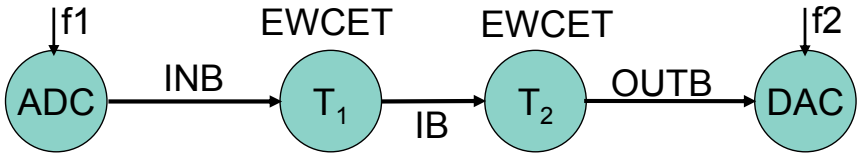
Guarantee: no deadline misses for ***the set of test streams***

Audio echo cancellation example



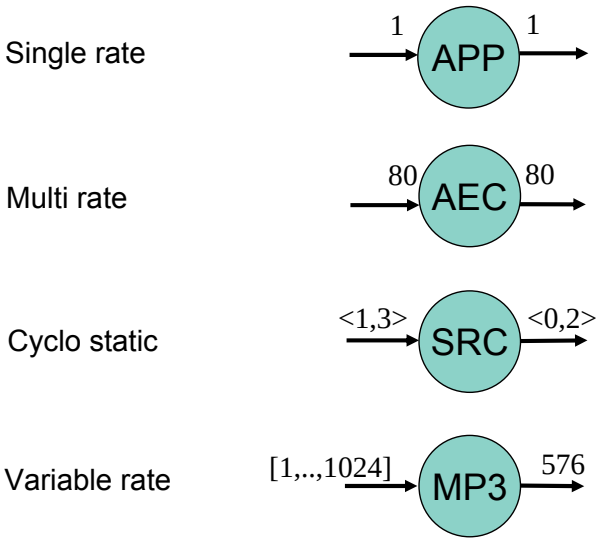
- multiple jobs and streams
- real-time constraints: throughput + latency
- PCM stream has external clock
- starting & stopping streams without clicks
- ***EWCET of MP3 task***

Overload behavior



- Execution time (ET) > EWCET ⇒ deadline misses
- Deadline misses:
  - Under-run output buffer (OUTB) or overflow input buffer (INB)
  - Prevent internal buffer (IB) overflow because difficult to handle by tasks
- Design requirements:
  - Overflow and under-run should not occur for the set of test-streams
  - Compensation for ET>EWCET should be supported
  - Worst-case temporal behavior other jobs should not be affected
    - Design and verify each job in isolation
    - Use only WCETs in case of analogue radio jobs

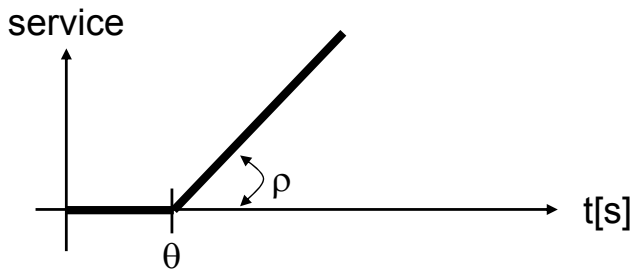
Advanced task models are needed in the audio echo cancellation application



Recent extensions dataflow analysis

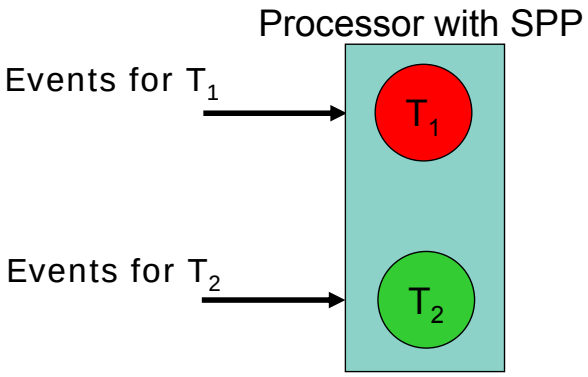
1. Express effects arbitration in dataflow model
2. Throughput analysis techniques with a low computational complexity
3. Interfacing with environment
4. Latency constraints
5. Data-dependent I/O

Requirement dataflow analysis: latency-rate (LR) server characterization of arbiters



All starvation free arbiters can be characterized as an LR-server
- Latency θ
- Rate ρ

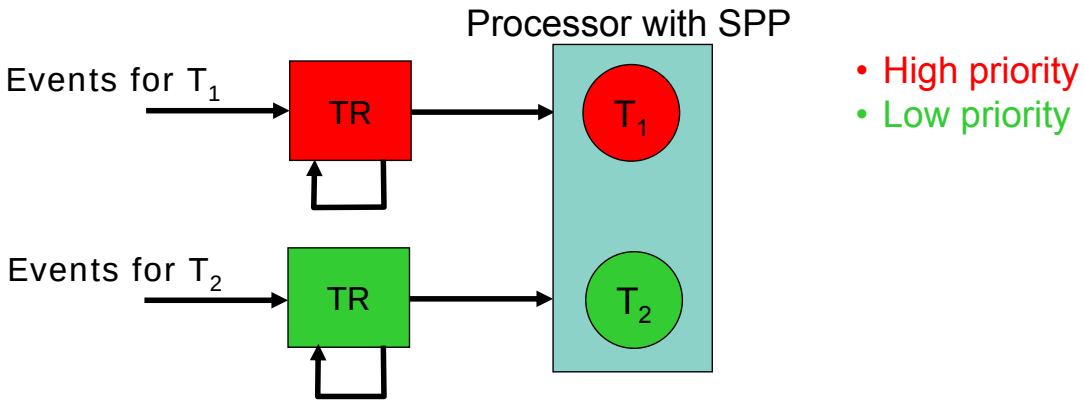
Static priority preemptive (SPP)



- High priority
- Low priority

Unknown minimum interval between events ⇒ not an LR- server
- not starvation free; high priority task 1 can prevent execution task 2

Static priority preemptive with traffic regulator

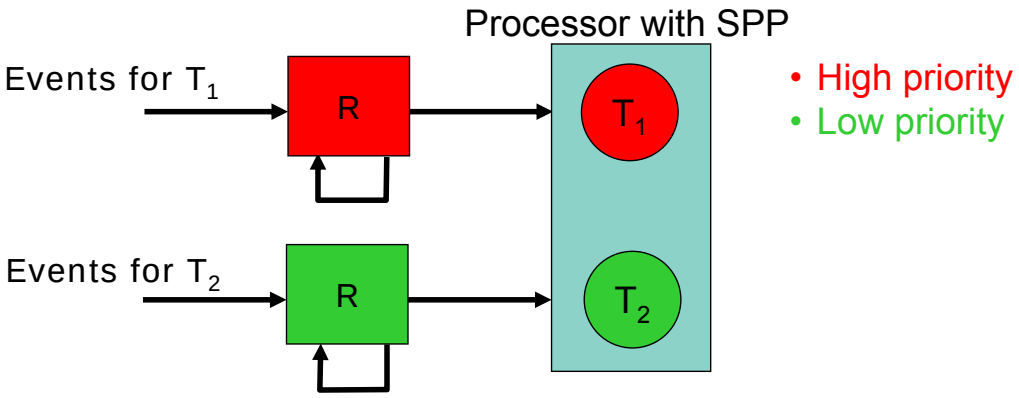


Known minimum time between events + known WCET ⇒ LR- server
• bounded latency θ and minimum rate ρ

Starvation free arbitration examples

- Given known WCET
  - Round-robin, weighted round-robin
  - With traffic shapers
    - Static priority preemptive (SPP)
    - EDF
- Does not require known WCET or minimum distance between events
  - Time division multiplex (TDM)
  - Constant bandwidth server (CBS)
  - Polling server

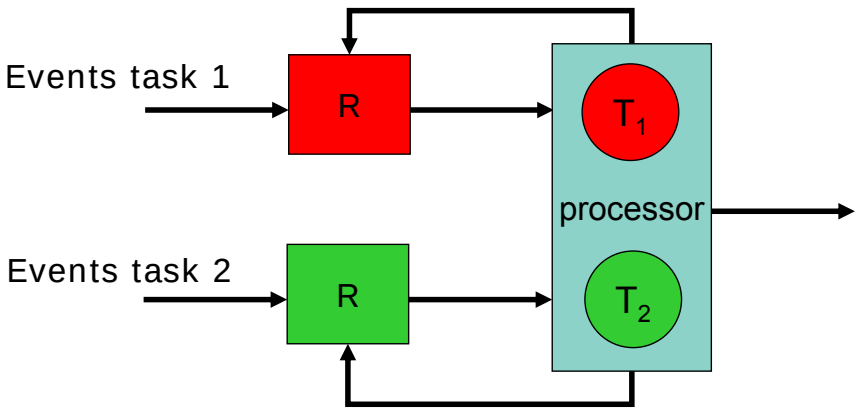
Characteristics traffic regulation



Traffic regulation does not maximize progress and slack of tasks

- no events for $T_1 \Rightarrow$ no earlier starts of T_2
- $ET_x < WCET_x \Rightarrow$ no earlier starts of T_x

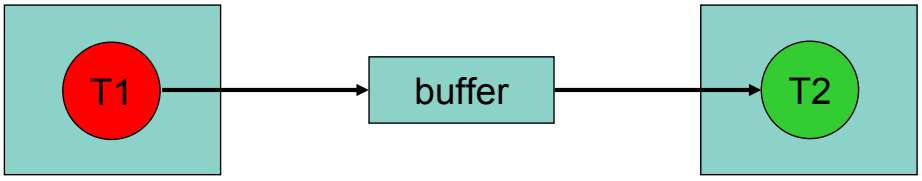
Characteristics service regulation



Feedback loop used to maximize progress & bound interference

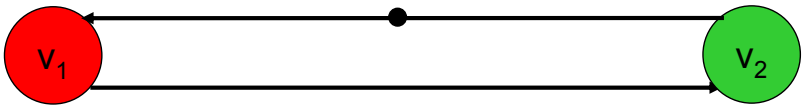
- no events for $T_1 \Rightarrow$ earlier starts of T_2
- $ET_x < WCET_x \Rightarrow$ earlier starts of T_x
- feedback loop is often implicit TDM, CBS

Backpressure



1. Functional determinism ⇒ data may not be lost ⇒ buffer may not overflow
2. WCET of T2 is not known (only EWCET is known) ⇒ unknown consumption rate ⇒ T1 checks for space
3. Check for space ⇒ backpressure & cyclic dependency

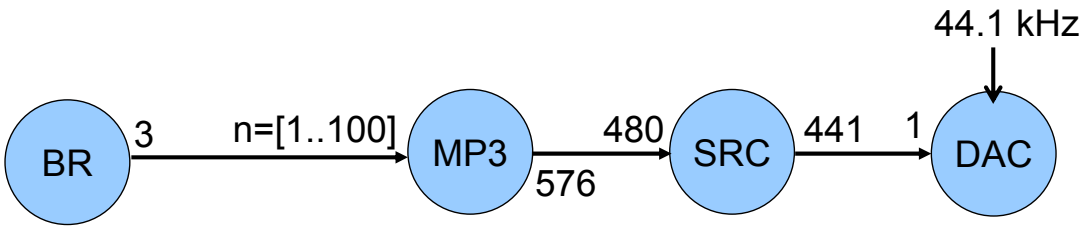
Dataflow model



Cyclic dependencies

- Cyclic dependencies must be taken into account:
  - Functional cyclic dependencies (e.g. previous frame in video decoder)
  - Maximum buffer capacities can be a constraint (e.g. buffers in interface of the communication network)
  - Consumption or production rate of a task can be data-dependent
- Trade-off buffer capacity and budget:
  - Higher processor cycle budget for a task ⇒ smaller buffers
- Tightness of the analysis:
  - Checking space bounds jitter ⇒ smaller buffers
- Requires analysis techniques with a low computational complexity
  - Analysis of the complete job is done at once

Variable consumption rate

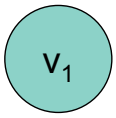


- Digital to analog converter (DAC) determines throughput constraint
- MP3 decoder task consumes variable amount of data
- Block-reader (BR) task must “know” consumption speed MP3 task
  - Implies cyclic dependency that affects the temporal behavior!

[M. Wiggers et.al.,DATE 2008]

Dataflow analysis

Elements of a single-rate dataflow graph



Nodes denote actors

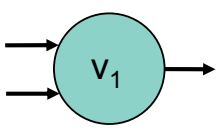
Edges denote unbounded queues

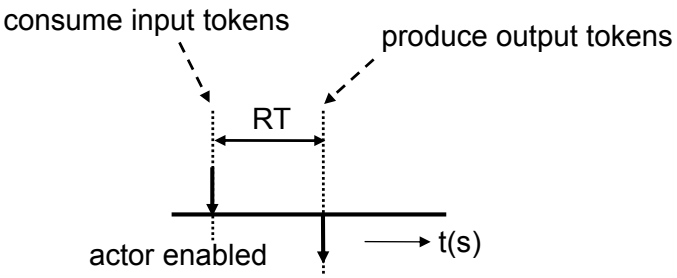
Dots denote tokens

Firing rule = enabling condition

Response time (RT) = interval between enabling and finish

Characteristics of a dataflow actor



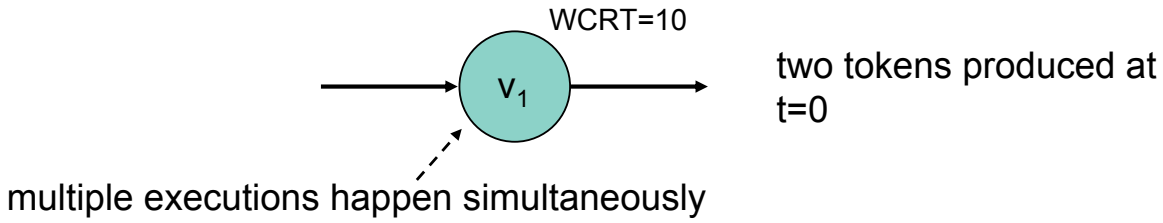


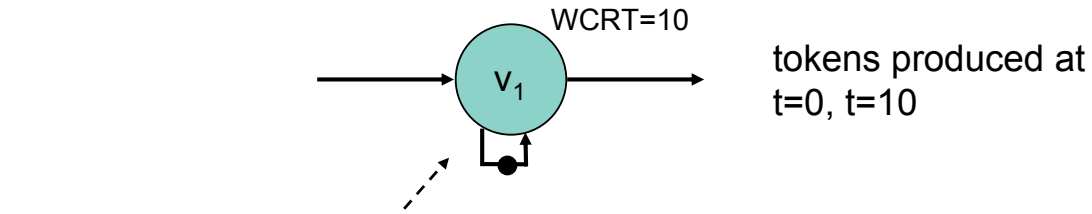
An actor:

- can represent a quantum of work
- has a firing rule (e.g. a token on each input)
- is enabled if the firing condition is satisfied
- is stateless
- consumes input tokens when actor starts
- produces output tokens in zero time when actor finishes its execution

Auto-concurrency (overlapping execution)

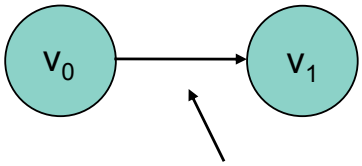
Assume two tokens arrive at t=0





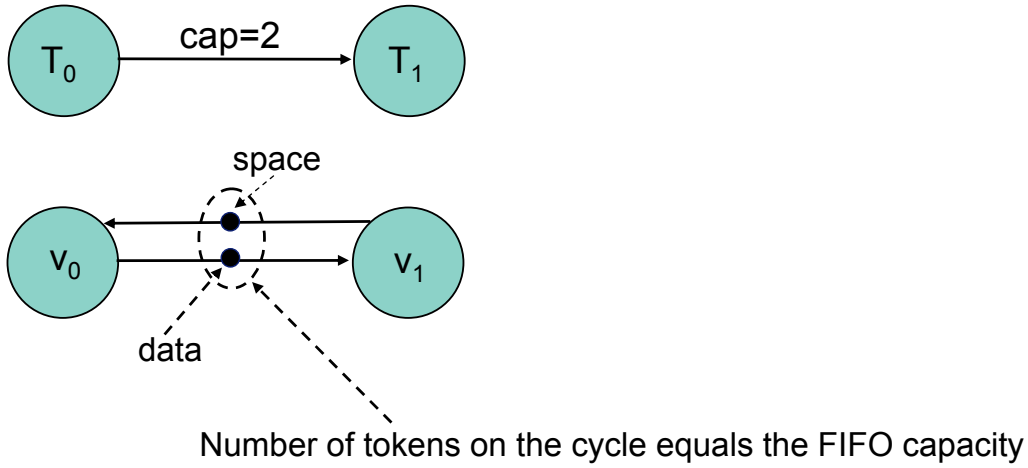
next execution cannot start before previous has finished

Unbounded queue



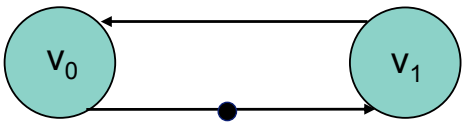
Edge represents a queue with unbounded capacity

Bounded FIFO model



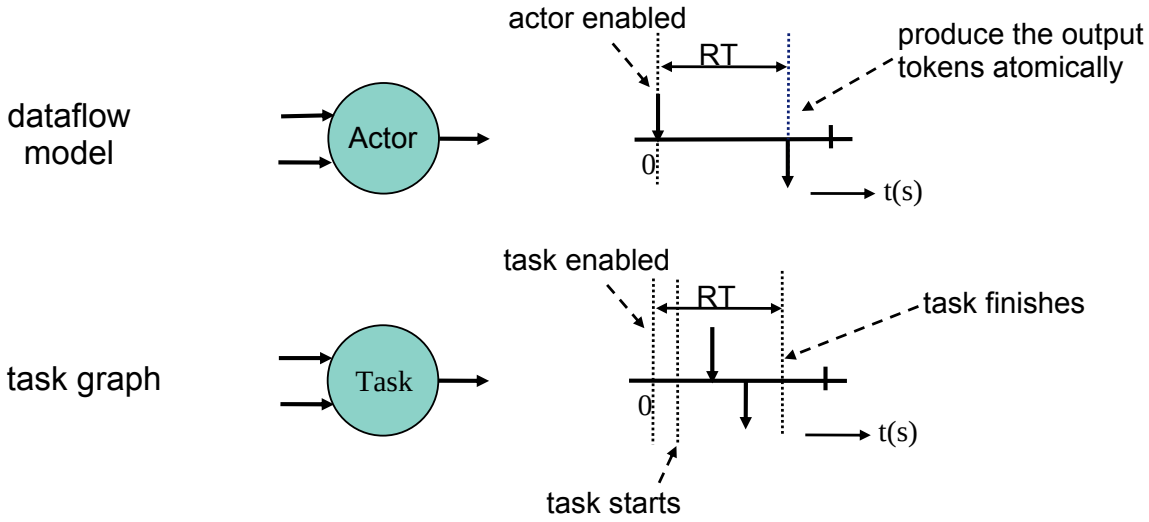
Monotonicity

- Monotonic temporal behavior:
  - An earlier production of a token cannot result in a later start of an actor during self-timed execution
- Consequence:
  - Sufficient to show that a schedule exist that satisfies the throughput and latency constraints given worst-case response times
  - Smaller response time result in earlier arrival tokens
    - Scheduling anomalies do not occur during self-timed execution of a dataflow model
- Requires sequential firing rules

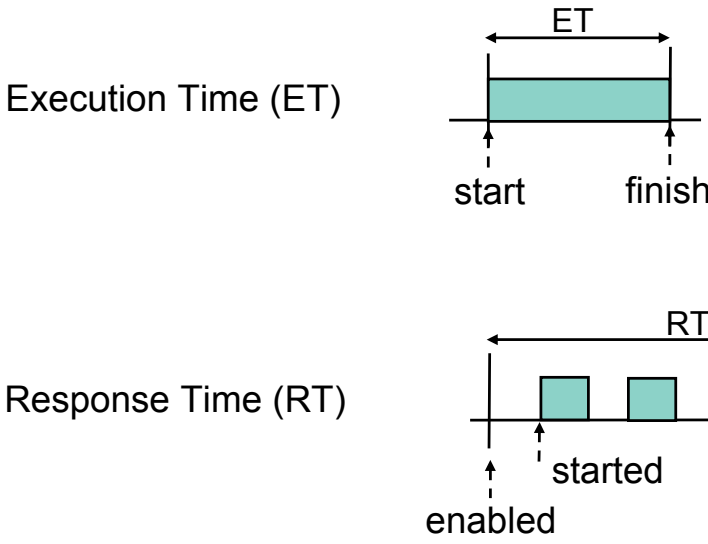


Earlier arrival token results in earlier start v_1 and v_0

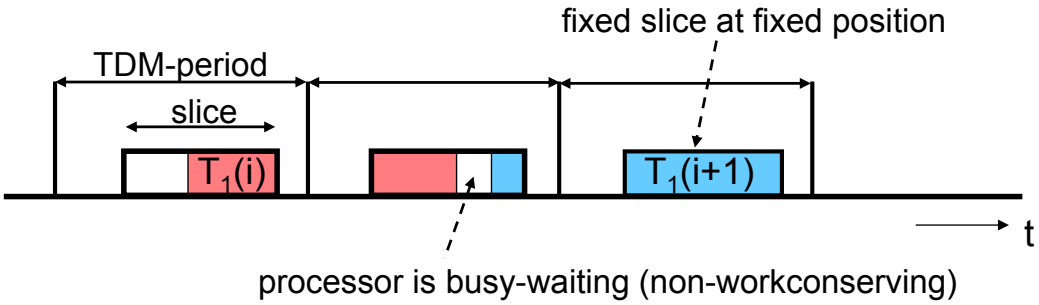
Tasks versus dataflow actors



Response time

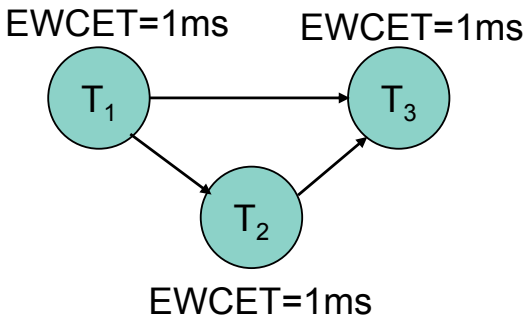


Time Division Multiplex (TDM)



Throughput analysis

Task-graph

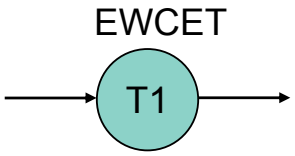


Assume:
- T_1 and T_2 share one processor, each task get a TDM-slice of 4 ms every 8 ms
- Infinite buffer capacity

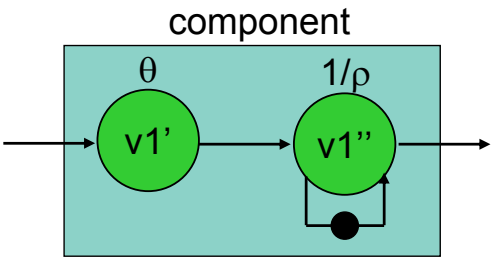
What is the minimum throughput?

Dataflow model after latency-rate characterization

Task-graph

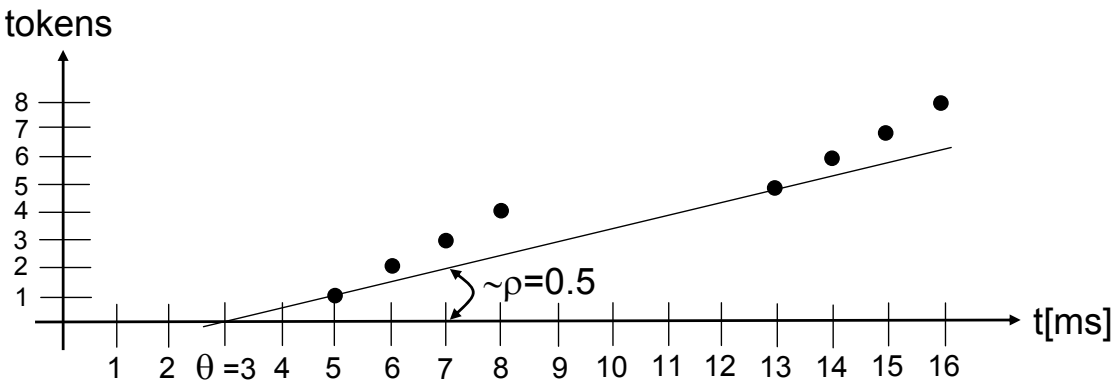


Dataflow model
(including effects arbitration)



[M. Wiggers et.al., Scopes 2006]

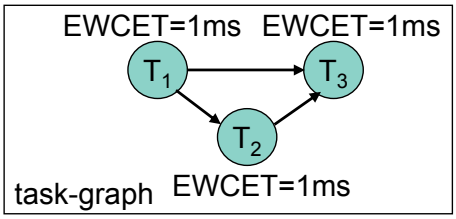
Latency-rate characterization



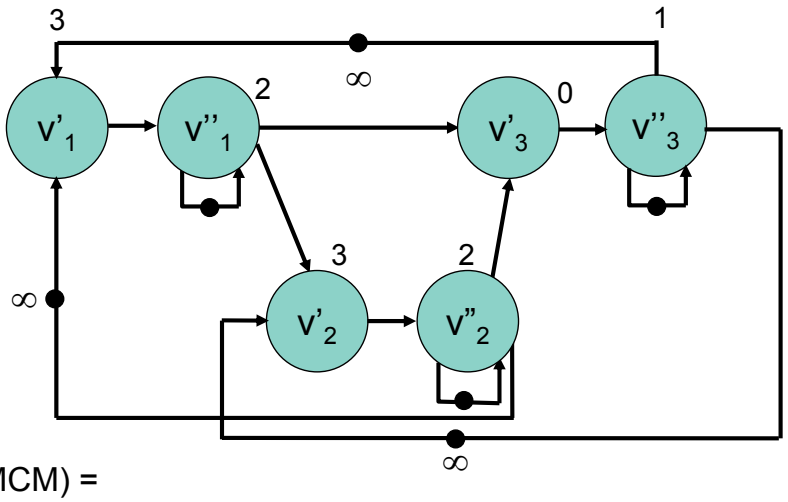
- Latency θ=3ms
- Rate ρ=4/8=0.5 executions/ms

Throughput analysis

TDM: ρ=0.5 execution/ms, θ=3 ms



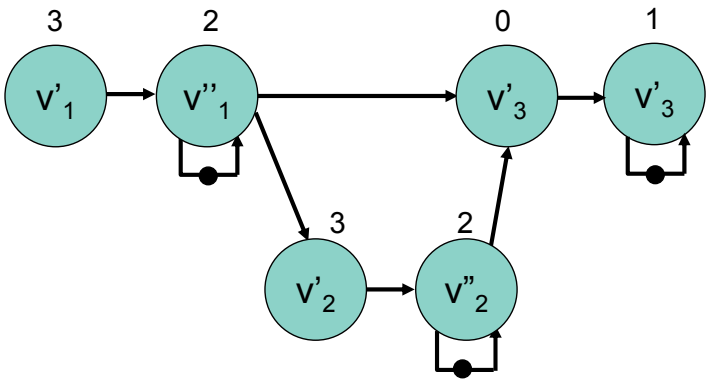
Dataflow model



Maximum cycle mean (MCM) =
$\max_{c \in Cg} (\sum_{v \text{ on } c} \text{EWCRT}(v))/ \text{tokens}(c) = 2/1 = 2$ ms/execution

Throughput analysis

Dataflow model

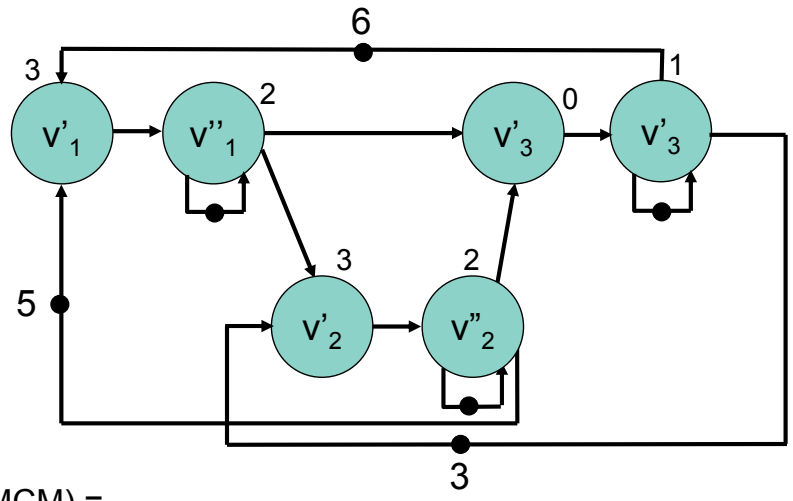


Maximum cycle mean (MCM) =
$\max_{c \in Cg} (\sum_{v \text{ on } c} \text{EWCRT}(v))/ \text{tokens}(c) = 2/1 = 2$ ms/execution

Buffer capacity computation

TDM: ρ=0.5 execution/ms, θ=3 ms

Dataflow model



Maximum cycle mean (MCM) =
$\max_{c \in Cg} (\sum_{v \text{ on } c} \text{EWCRT}(v))/ \text{tokens}(c) = 2/1 = 2$ ms/execution

Summary

- Introduced dataflow analysis techniques to compute buffer capacities and scheduler settings given throughput and latency constraints
- Dataflow analysis techniques are applicable if starvation free arbiters are applied
  - SPP is not starvation free
- Service regulation: bound interference + reduce miss-probability
  - Compared to traffic regulation
  - Space must be available before task can start $\Rightarrow$ cyclic dependencies
- Sufficient to show at design-time that a schedule exist
  - Due to monotonic temporal behavior of self-timed dataflow graphs
- Multiprocessor architecture includes processors with caches
  - Use of WCETs is not cost-effective, use EWCET instead

References

- B. Akesson, K. Goossens, and M. Ringhofer. Predator: A Predictable SDRAM Memory Controller. In *Proc. Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, 2007.
- M. Bekooij, A. Moonen, and J. van Meerbergen. Predictable and Composable Multiprocessor System Design: A Constructive Approach**,** In *Proc. Bits&Chips Symposium on Embedded Systems and Software*, October 2007, Eindhoven, The Netherlands.
- M. Bekooij, M.Wiggers, J. van Meerbergen, Efficient Buffer Capacity and Scheduler Setting Computation for Soft Real-Time Stream Processing Applications, In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, April 2007.
- A. Kumar, A. Hansson, J. Huisken and H. Corporaal. An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip. *In Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, April 2007.
- A. Moonen et.al. A Multi-Core Architecture for In-Car Digital Entertainment, GSPX publication 24-27 October 2005: In *Proc. Int'l Conference on Global Signal Processing (GSPx)*, October 2005.
- O. Moreira and M. Bekooij. Self-Timed Scheduling Analysis for Real-Time Applications, In *EURASIP Journal on Advances in Signal Processing, 2007*

References

- O. Moreira and M. Bekooij. Scheduling Multiple Independent Hard Real-Time Jobs on a Heterogeneous Multiprocessor, In *Proc. Int'l Conference on Embedded Software (EMSOFT), September* 2007
- S. Sriram and S. S. Bhattacharyya. Embedded Multiprocessors: Scheduling and Synchronization. Marcel Dekker Inc., 2000
- D. Stiliadis and A. Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. In *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.
- S. Stuijk, T. Basten, M.C.W. Geilen and H. Corporaal. Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs, In *Proc. Design Automation Conference (DAC)*, June 2007.
- M. Wiggers, M. Bekooij, and G. Smit. Modelling Run-Time Arbitration by Latency-Rate Servers in Data Flow Graphs. In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, April 2007.
- M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2007.

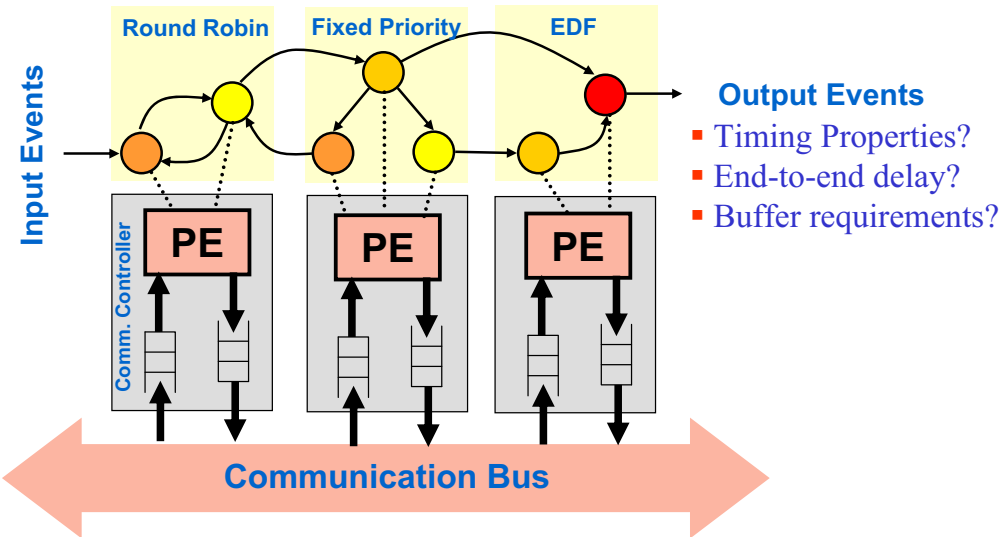
Formal Methods in System and MpSoC Performance Analysis and Optimization

-- Combining State-based and Functional Models --

Samarjit Chakraborty
National University of Singapore

NUS National University of Singapore

Performance Analysis Problem



- Tasks have different activation rates and execution demands
- Each computation/communication element has a different scheduling/arbitration policy

Performance Analysis Challenges

- Heterogeneous processing elements
- Different scheduling policies (e.g. EDF, Rate Monotonic, Round Robin, etc.)
- Wide variation in task execution times
- Different bus protocols (e.g. TDMA, FCFS etc.)
- Irregular event arrival patterns

Solution: Use Simulation

- SimpleScalar (www.simplescalar.com) Instruction Set Simulator and SystemC (www.systemc.org) for transaction-level modeling and simulation

- Disadvantages
  - Excessive running times
  - Insufficient corner case coverage
  - No formal guarantees
  - Difficulty in integration

Solution: Analytical Methods

- Standard event models [Richter *et al.* 2002, Yi. *et al.* 2004, etc.]
  - Periodic, sporadic events
  - Using classical scheduling theory from the real-time systems literature
- Real-time Calculus [Chakraborty, Kunzli and Thiele 2003]
  - Models general event streams and resource availability
    - Bursty arrival patterns, irregular resource availability
  - Represents arrival/service patterns as functions
  - Uses min/max-plus algebraic framework for analysis

Solution: Analytical Methods (cont.)

- Event Count Automata
  [Chakraborty, Phan and Thiagarajan 2005]
  - Represents event streams and resource availability as automata
  - Accepts integer sequences, which represent all possible permissible arrival/service patterns
  - Uses automata verification techniques (e.g. model checking) for analysis

Different Possibilities: Comparison

Criteria \ Methods	Standard Event Models (SEM)	Real-time Calculus (RTC)	Event Count Automata (ECA)
Design scope	✪	✪ ✪	✪ ✪ ✪
Efficiency	✪ ✪ ✪	✪ ✪ ✪	✪
Accuracy	✪	✪ ✪	✪ ✪ ✪

A modeling technique that is:
- Expressive enough
- Efficiently analyzable

?

Combinations of two or more models

Integrating Multiple Performance Models

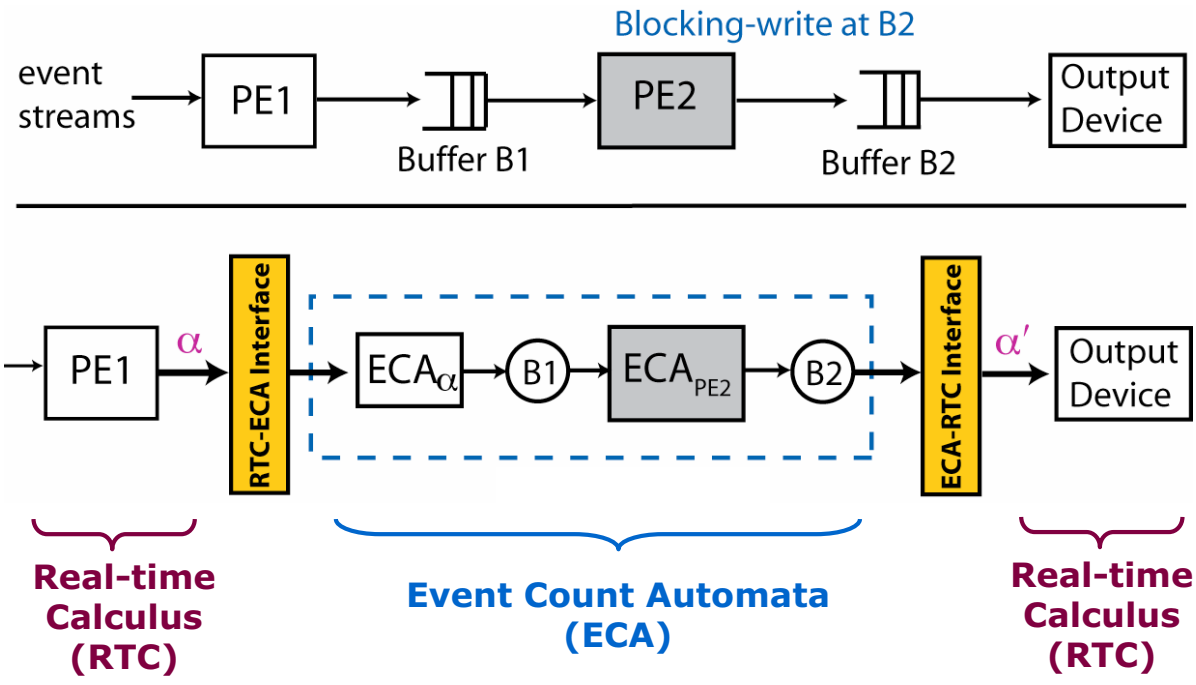
- Simulation + Real-time Calculus [Kunzli *et al.* DATE'06]
- SymTA/S + Real-time Calculus [Kunzli *et al.* CODES + ISSS'07]
- Synchronous Data Flow graphs + Standard Event Models [Schliecker *et al.* DATE'07]

- **Provides the required amount of modeling power**
- **Without incurring excessive analysis complexity**

This Talk

- Composing a functional and state-based model
  - Real-time Calculus (RTC): efficient
  - Event Count Automata (ECA): expressive
- Technical Challenge: An ***interfacing*** technique for composing RTC and ECA
  - RTC: functional, requires algebraic techniques for analysis
  - ECA: state-based, requires more expensive state space exploration techniques
- Advantages
  - Formal performance guarantees
  - Expressive, but analysis is not expensive

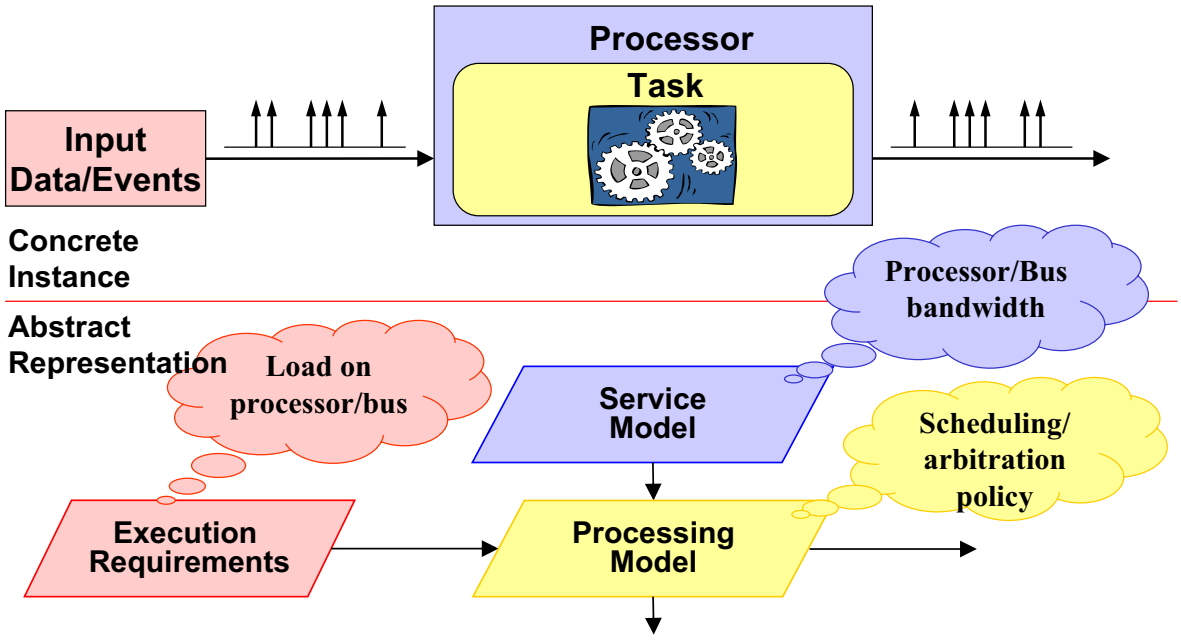
ECA-RTC Interfacing



System Modeling

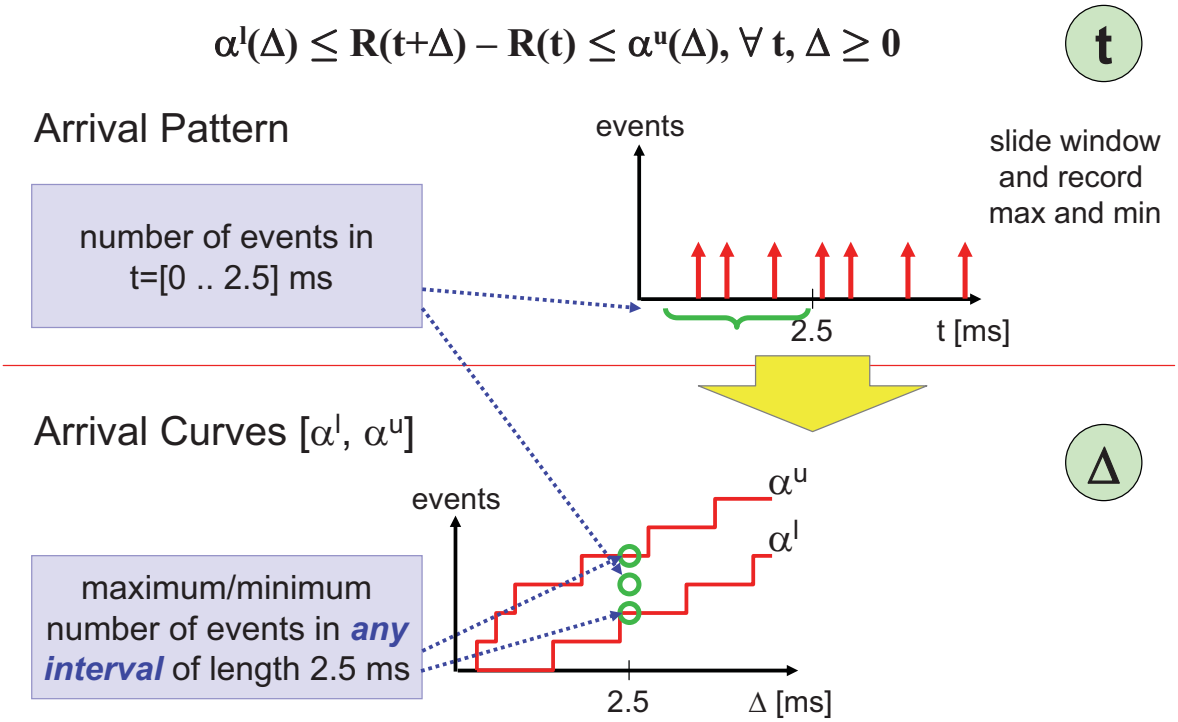
- Both RTC and ECA rely on a ***count-based abstraction*** for modeling workload and service
  - *How many* events can arrive over any time window of length Δ?
  - *How many* events may be processed with any time window of length Δ?
- RTC models such information as functions and uses algebraic techniques for analyzing them
- ECA models such information as automata and relies on state-space exploration techniques

RTC-Based Modeling: Overview

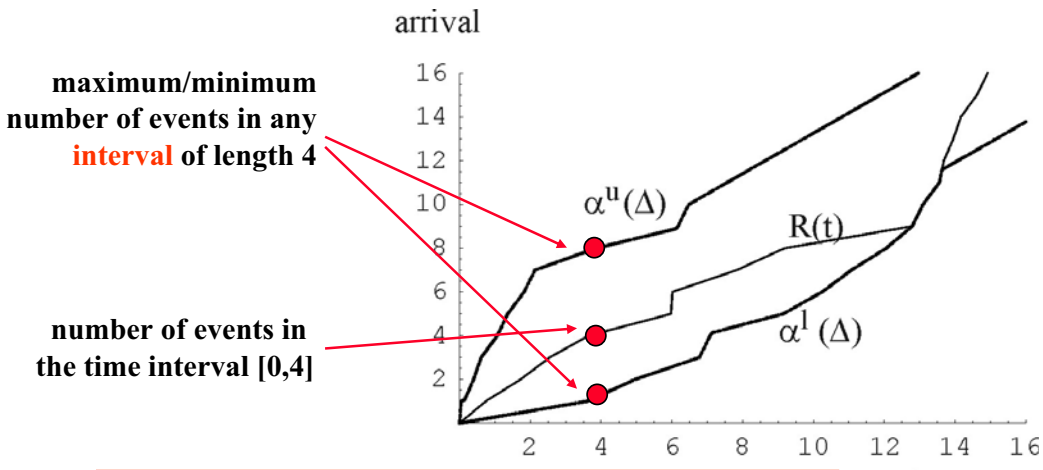


Event Model – Modeling Execution Requirements

$$\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta), \forall\, t, \Delta \geq 0$$



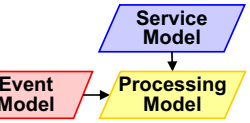
Event Model – Modeling Execution Requirements

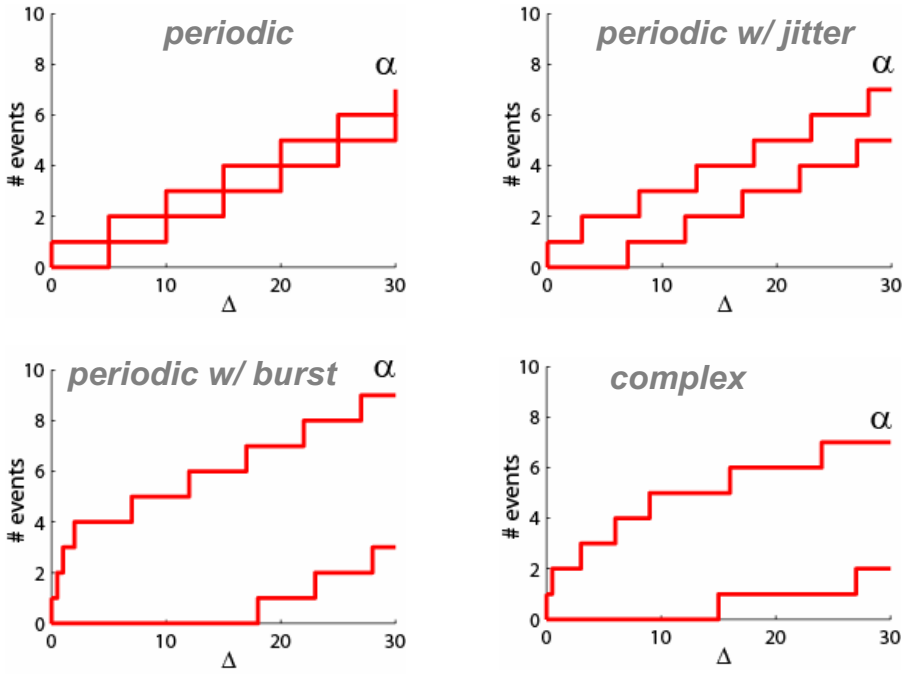


$$\alpha^l(\Delta) \leq R(t+\Delta) - R(\Delta) \leq \alpha^u(\Delta), \forall\, t, \Delta \geq 0$$

- Max/Min number of events arriving over different time interval lengths

Event Model - Examples





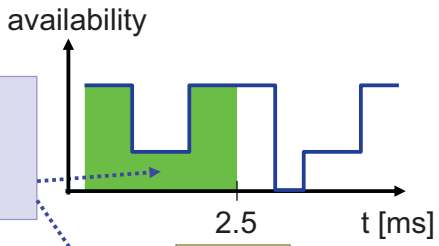
Service Model – Modeling Resource Availability

$$\beta^l(\Delta) \leq S(t+\Delta) - S(t) \leq \beta^u(\Delta), \forall\, t,\Delta \geq 0$$

t

Resource Availability

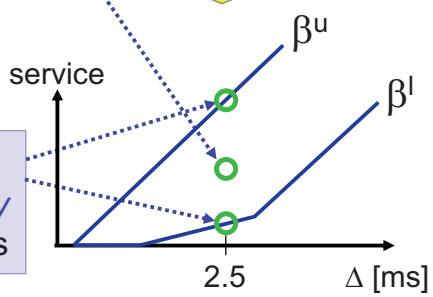
available service in t=[0 .. 2.5] ms



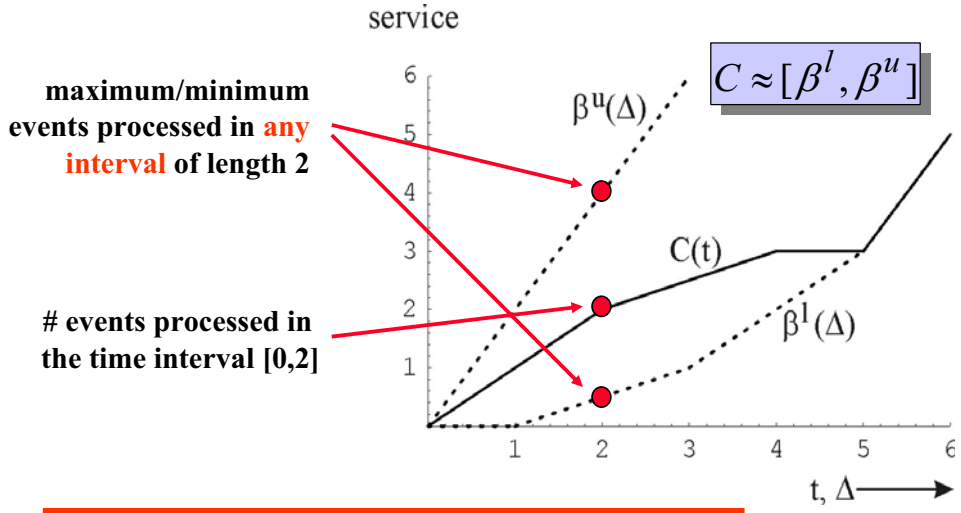
Δ

Service Curves [β^l, β^u]

maximum/minimum available service in *any interval* of length 2.5 ms

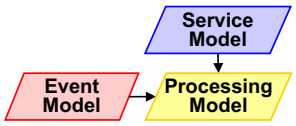


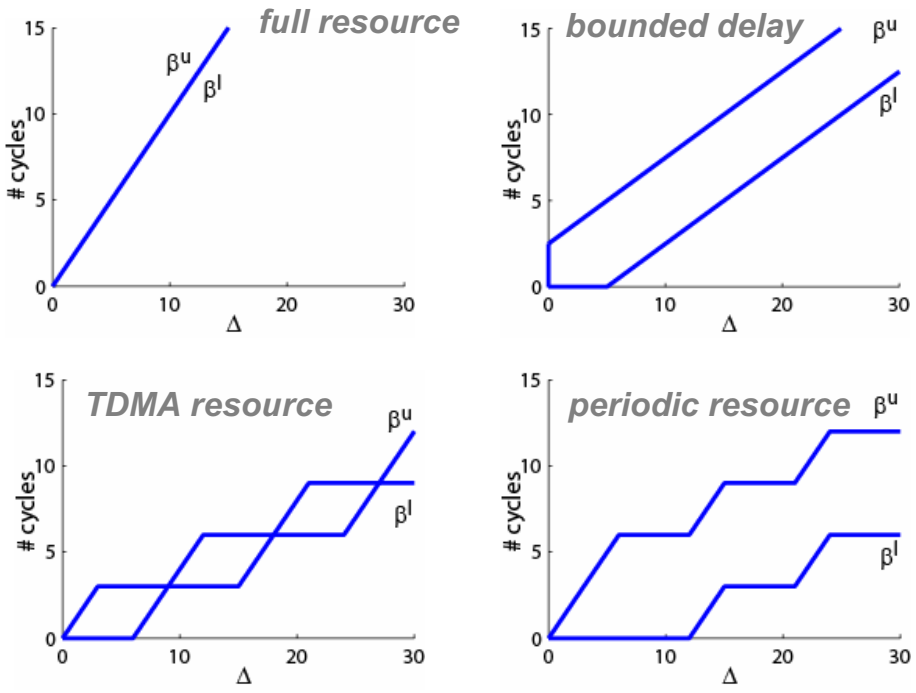
Service Model – Modeling Resource Availability



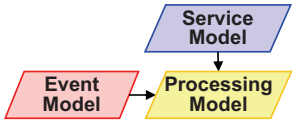
$$\beta^l(\Delta) \leq S(t+\Delta) - S(t) \leq \beta^u(\Delta), \forall\, t,\Delta \geq 0$$

Service Model - Examples



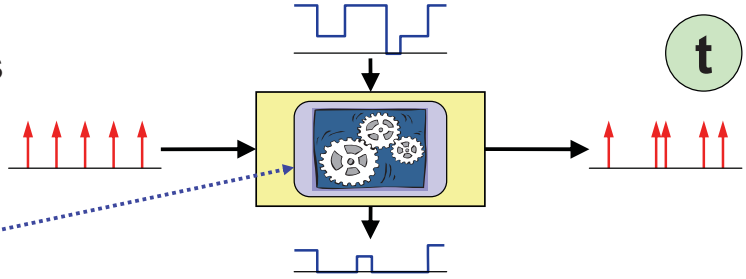


Processing Model

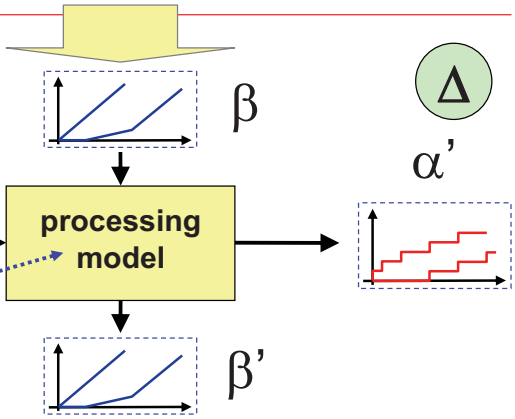


HW/SW Components

Tasks mapped onto the processor and the scheduling policy

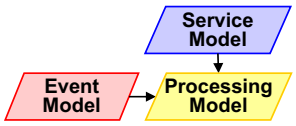


Model



$$\alpha'(\Delta) = f_\alpha(\alpha, \beta)$$
$$\beta'(\Delta) = f_\beta(\alpha, \beta)$$

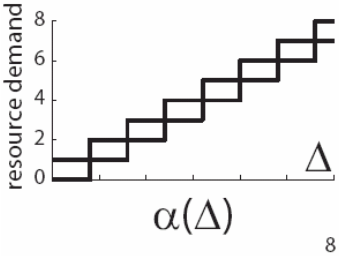
Processing Model



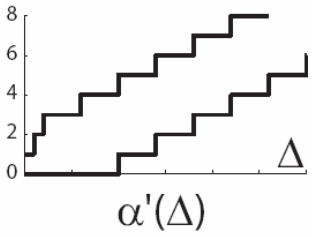
Abstract representations of:

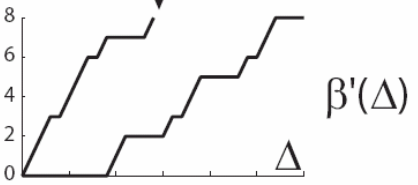
available resource supply

input events



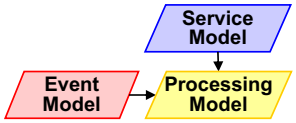
processed events





remaining supply

Processing Model

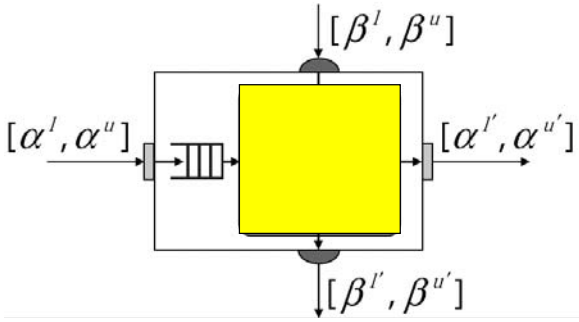


- Computation of
  - Timing properties of the processed events
  - Bounds on the remaining resource

$$v(\Delta) \wedge w(\Delta) = \min\{v(\Delta), w(\Delta)\}$$

$$v(\Delta) \underline{\oplus} w(\Delta) = \inf_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\} \qquad v(\Delta) \underline{\otimes} w(\Delta) = \inf_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$

$$v(\Delta) \overline{\oplus} w(\Delta) = \sup_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\} \qquad v(\Delta) \overline{\otimes} w(\Delta) = \sup_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$



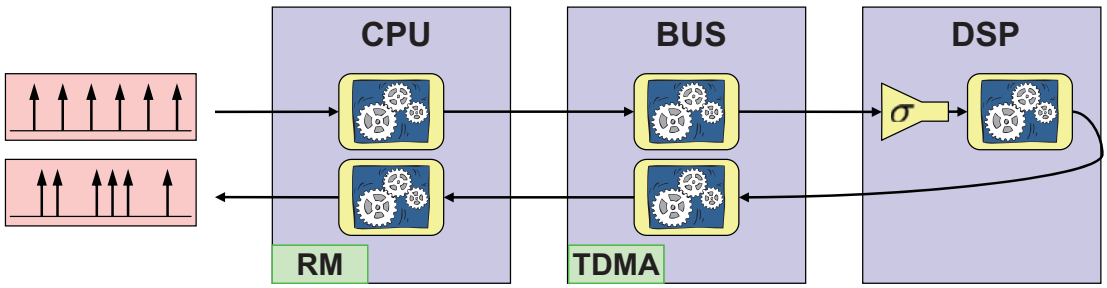
$$\alpha^{u'} = ((\alpha^u \underline{\oplus} \beta^u) \overline{\otimes} \beta^l) \wedge \beta^u$$

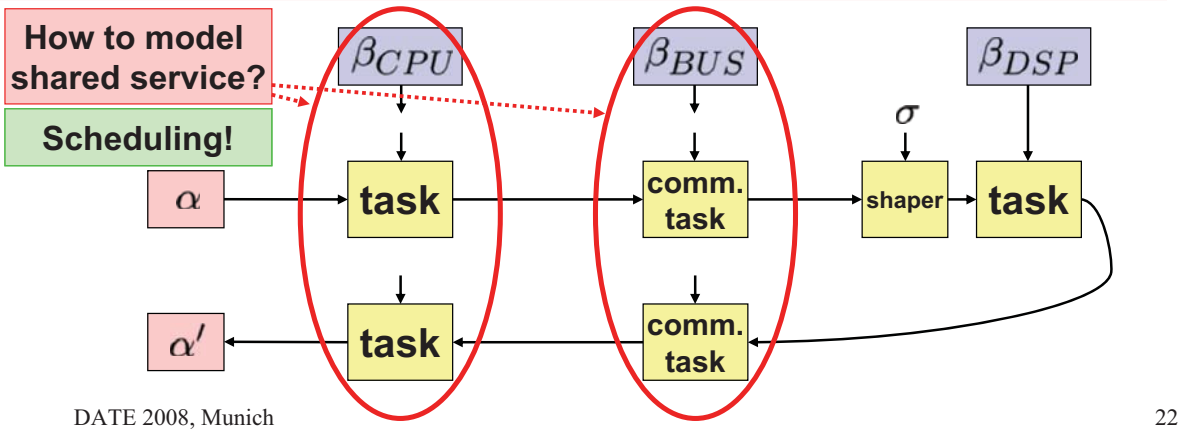
$$\alpha^{l'} = ((\alpha^l \overline{\otimes} \beta^u) \underline{\oplus} \beta^l) \wedge \beta^l$$

$$\beta^{u'} = (\beta^u - \alpha^l) \underline{\otimes} 0$$

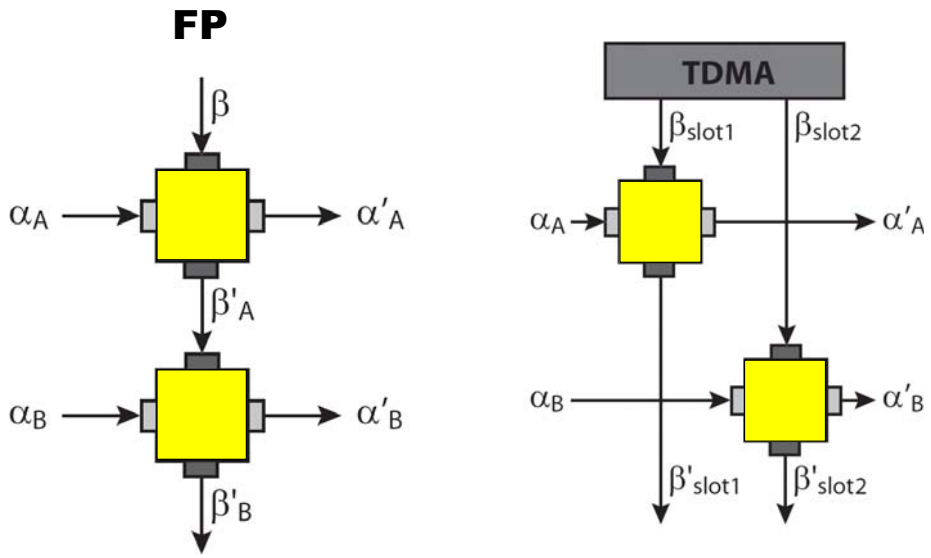
$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\oplus} 0$$

Compositional Analysis





Modeling Schedulers

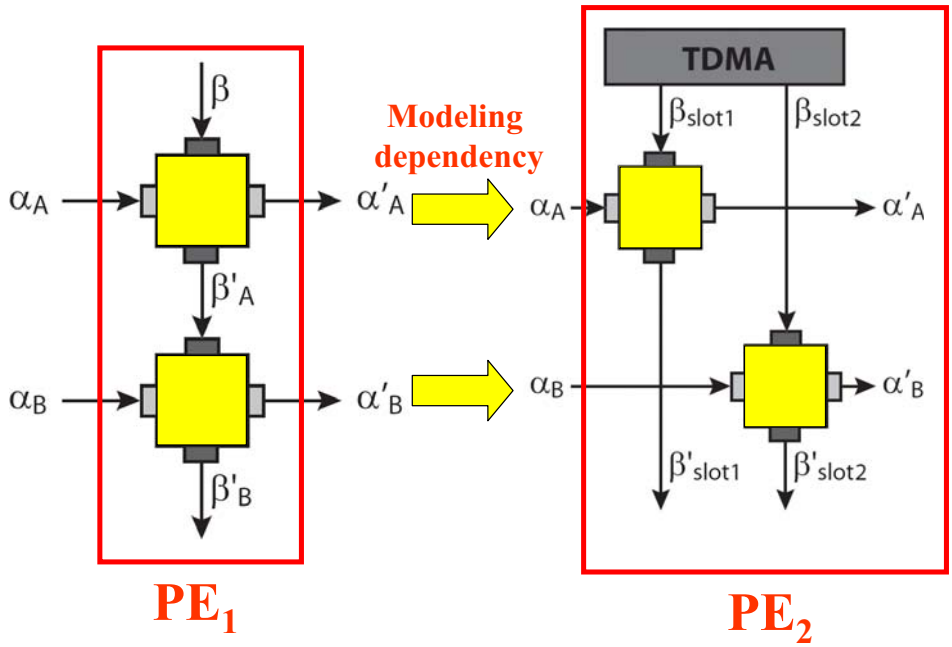


Using Scheduling Networks

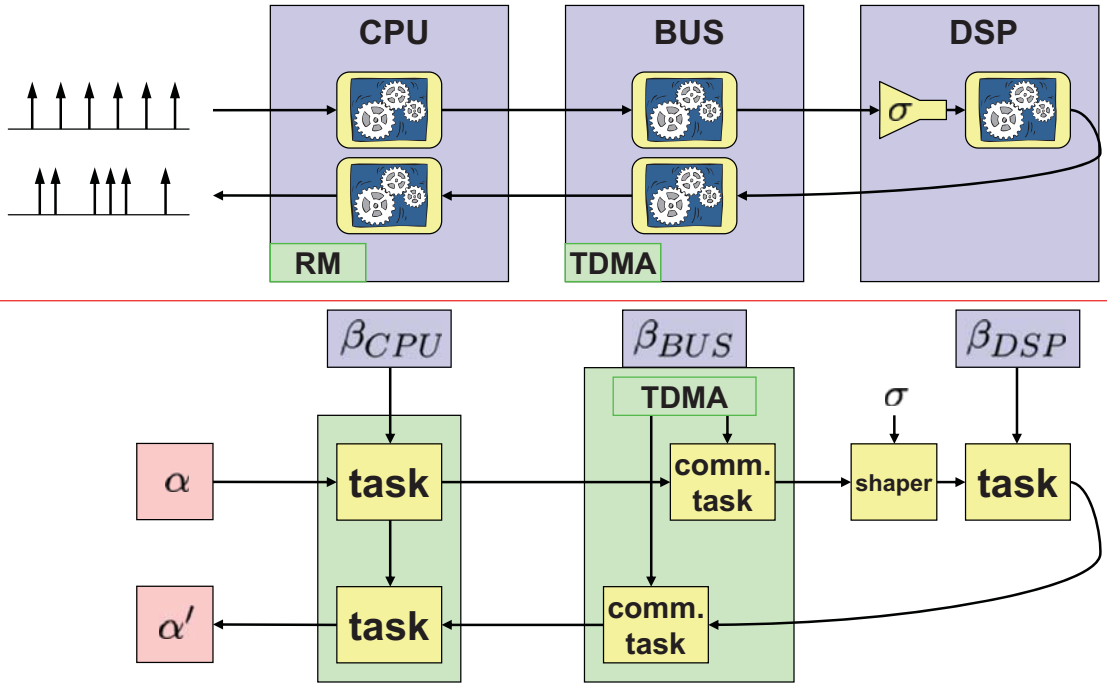
Compositional Analysis

Compositional Schedulability/Timing Analysis

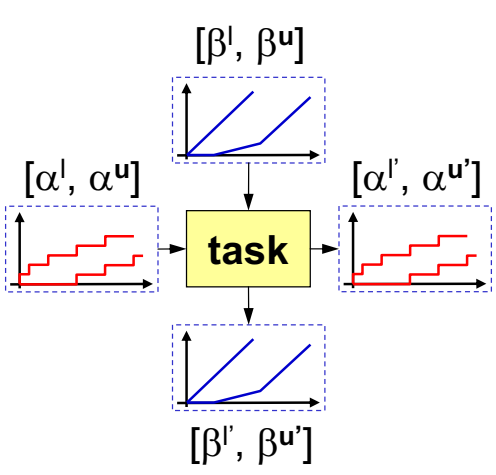
- **E.g. How much does the jitter increase?**

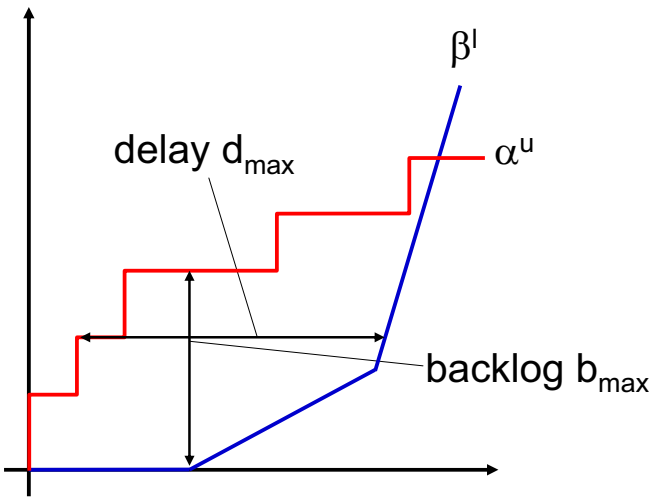


Compositional Analysis – Complete Example



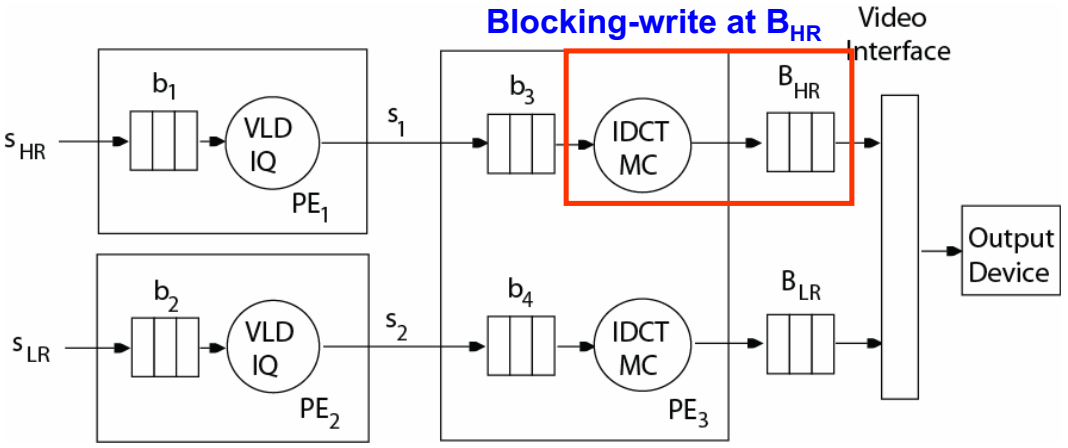
Analysis: Delay and Backlog





- **Max/Min buffer fill level**
- **Max/Min delay**
- **Utilization**
- **….**

But, We Need to Model ***State!***



Processing two video streams in a PiP application

- The service offered by PE_3 depends on the ***state*** of the buffer B_{HR}
  - Cannot be modeled easily in a functional setting

But, We Need to Model *State*

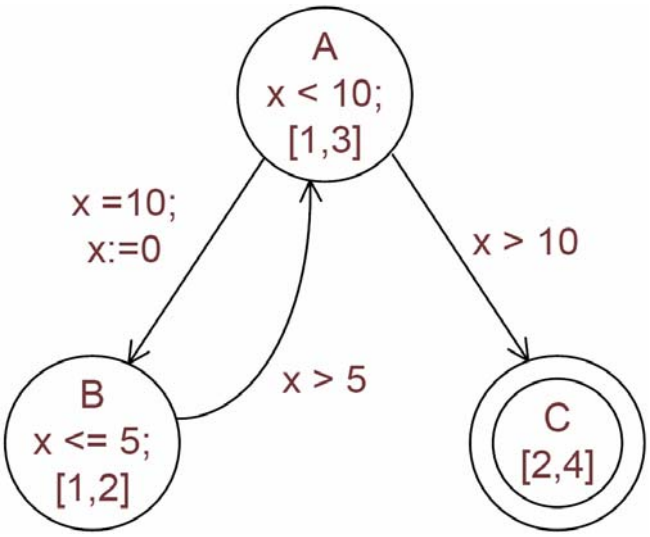
- We would like to retain the *count-based abstraction* to specify arrival and service
- But at the same time be able to model systems where the arrival and service depend on the state of the system
- Recently timed automata has been used as a generalization of traditional real-time event models
- But timed automata explicitly records/specifies timing information. Analysis/verification might be expensive …
- Exact arrival times of data items is not relevant. Rather the number of items that can arrive within a certain time interval is of relevance to us

Event Count Automata (ECA)

- Event stream /resource = ECA
- ECA = FSM + *Count variables (CVs)*
  - Records the arrival patterns of a stream and decides to accept or reject the stream
- CV: records number of events observed on a stream over a time interval

Event Count Automata (ECA)

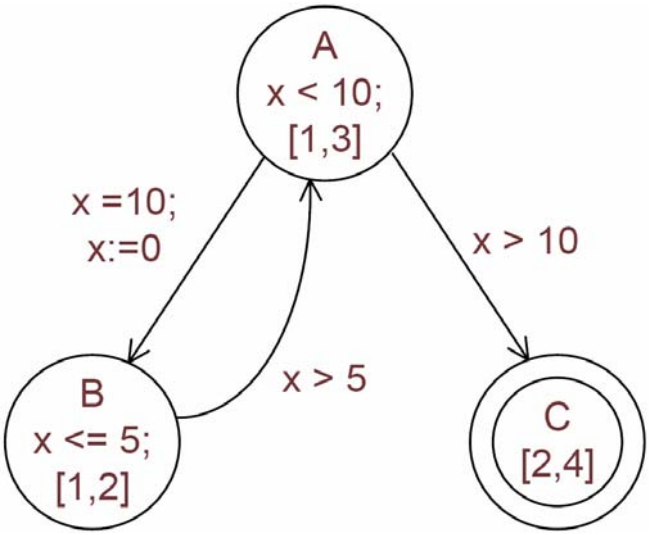
State-based modeling of stream processing systems



A = (S, s_{in}, X, Inv, ρ, →, F)

S = set of states
s_{in} is the initial state
X = set of count variables
Inv = function that assigns to each state an invariant constraint
ρ = rate function
→ = transition relation
F = set of final states

Event Count Automata (ECA)



Arrival pattern =
3 2 2 3 2 2 2 3 2 4

Corresponding run of the automaton =
(A, 0) → (A, 3) → (A, 5)
→ (A, 7) → (A, 10)
→ (B, 0) → (B, 2)
→ (B, 4) → (B, 6)
→ (A, 6) → (A, 9)
→ (A, 11) → (C, 11)
→ (C, 15)

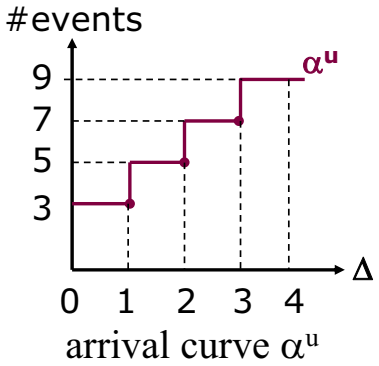
ECA - Properties

- ECA describes arrival patterns of the form $n_1, n_2, \ldots, n_k$ with n_i denoting the number of events arriving during the time $[i\text{-}1, i)$
- The dynamics of an ECA is captured by an associated behavioral automata, whose states are configurations of the form (s, V). s is a state of the ECA and V is a valuation of the variables
- The basic theory of ECAs follows from the fact that the associated behavioral automata can be quotiented into a finite state automata which accepts the same set of sequences. Arguments are similar to those used for constructing the regional automata for a timed automata

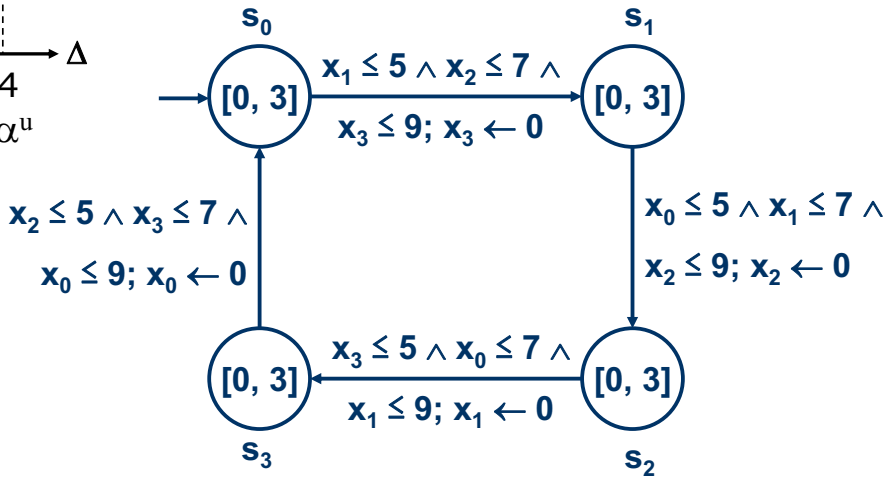
ECA - Properties

- If *L(A)* is the language defined by an ECA *A*, then *L(A)* is regular
- ECA-definable languages are closed under boolean operations
- It is possible to effectively determine if *L(A)* is empty

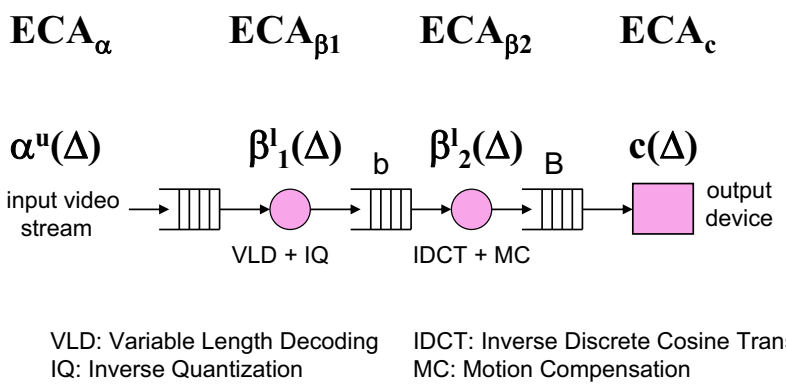
ECAs Representing Arrival & Service



ECA corresponding to α^u



System Modeling using ECAs



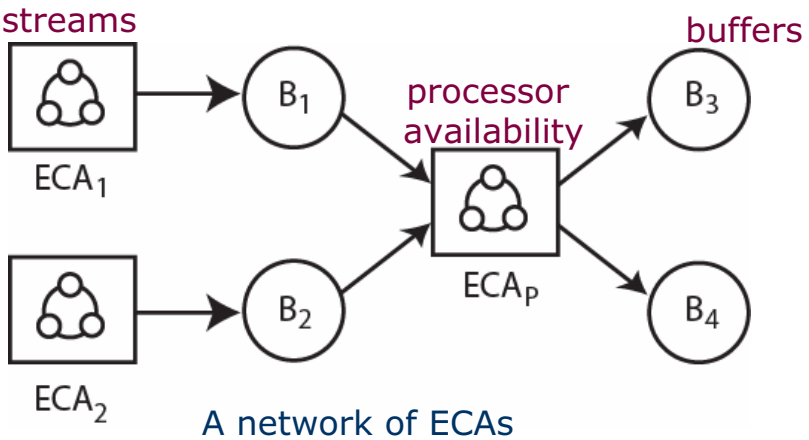
VLD: Variable Length Decoding
IQ: Inverse Quantization

IDCT: Inverse Discrete Cosine Transform
MC: Motion Compensation

MPEG-2 video decoder mapped onto two processors communicating via FIFO buffers

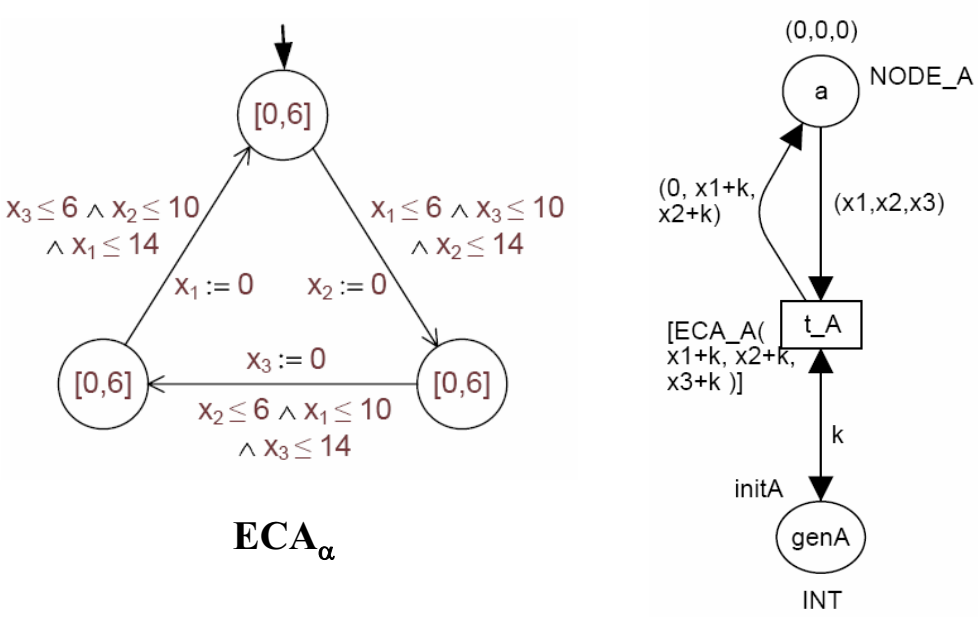
Networks of ECAs

- Each ECA is augmented with
  - I/O buffers
  - Update function: implements scheduling policy

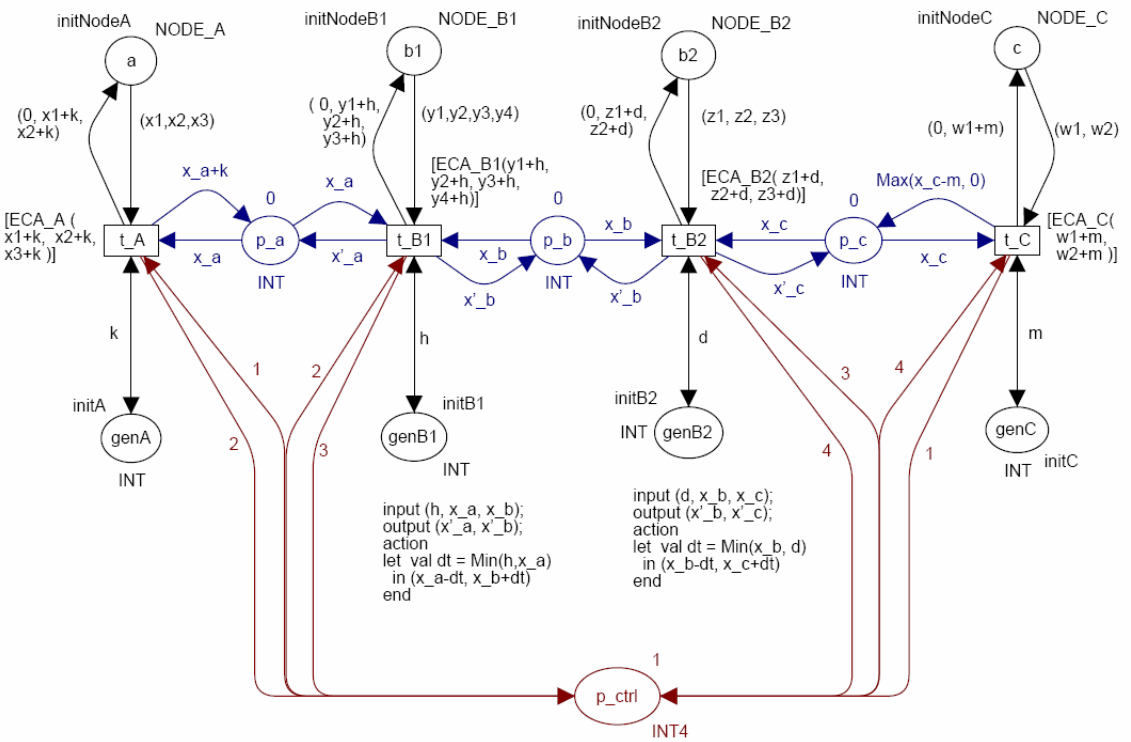


A network of ECAs

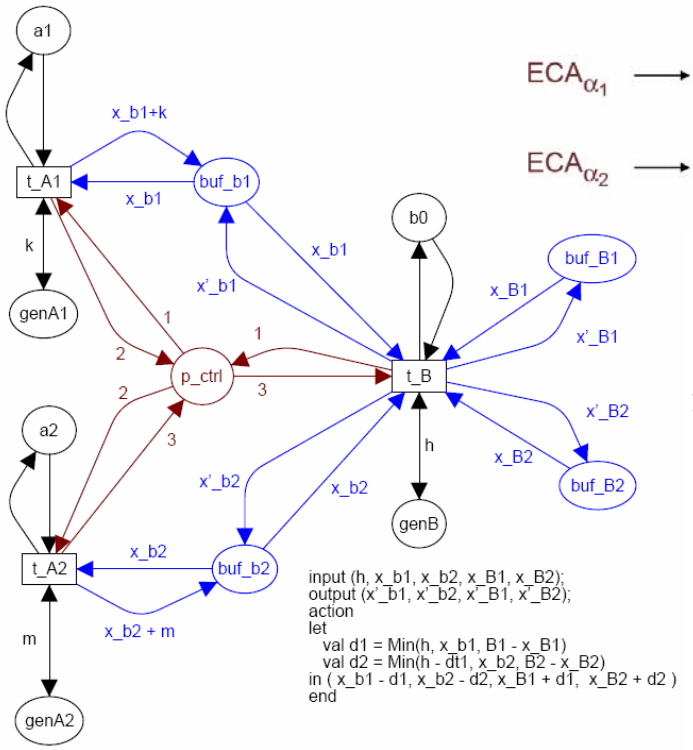
Encoding ECAs as CPNs

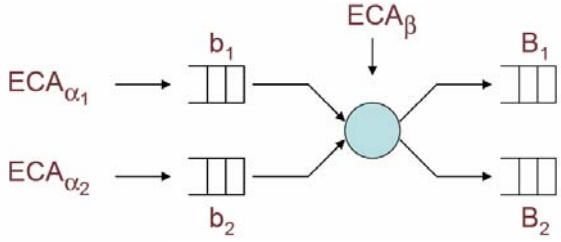


Networks of ECAs



Scheduling Multiple Streams





Complex schedulers can now be easily specified

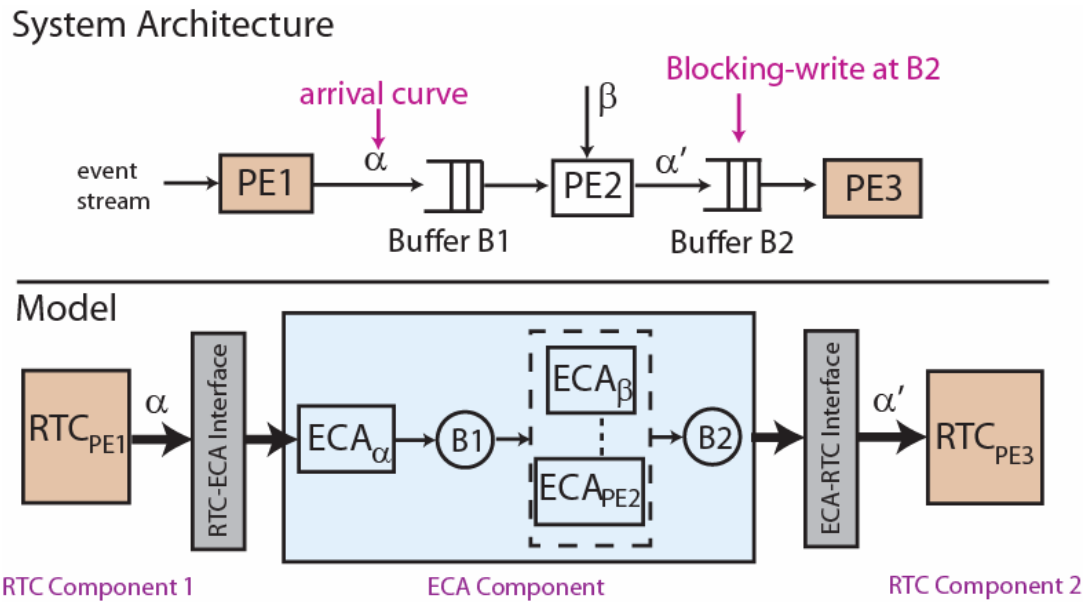
RTC vs. ECA

Real-time Calculus	Event Count Automata
▪ Functional	▪ State-based
▪ Cannot model state information	▪ Able to model state information naturally
▪ Efficient Analysis	▪ Complexity increases as the system grows

Integrating Performance Models

- **RTC + ECA**
  - State-based components: ECA
  - Non state-based components: RTC
  - Using interfaces to connect RTC and ECA components
- Technical Challenge
  - RTC → ECA interface
  - ECA → RTC interface

RTC+ECA

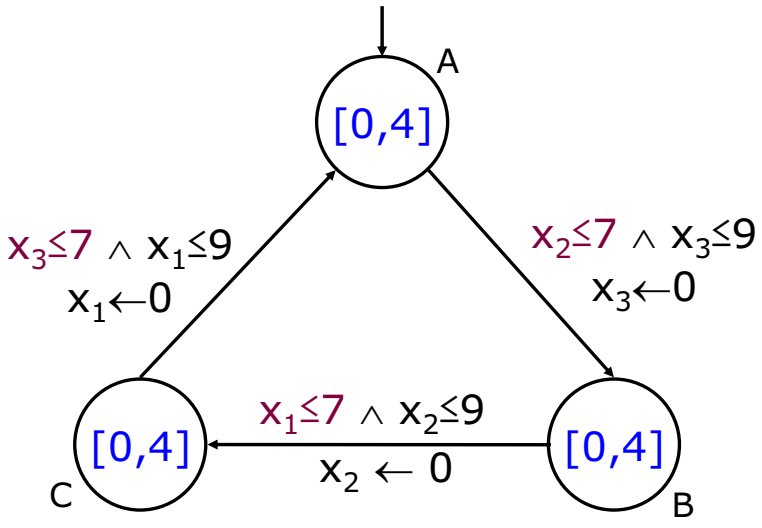


RTC→ECA Interface

- Converts an arrival curve *v* into an ECA *A*
  - An arrival pattern satisfies *v* if and only if it is accepted by *A*
- Finds a suitable set of count variables *CV*
- Necessary and sufficient conditions on *CV* under which an arrival pattern satisfies *v*
- Appropriate resets of *CV* to capture all the time intervals constrained by *v*

RTC to ECA

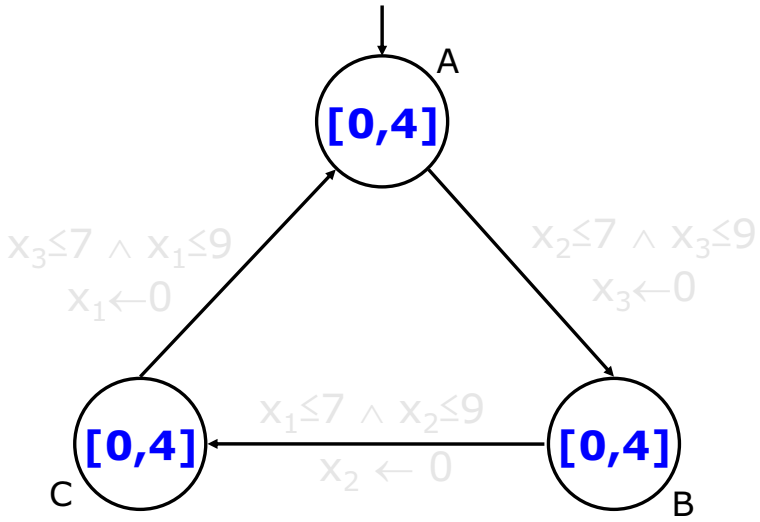
Δ	$\alpha^{upper}(\Delta)$
1	4
2	7
3	9



x_1, x_2, x_3 : count variables of the ECA

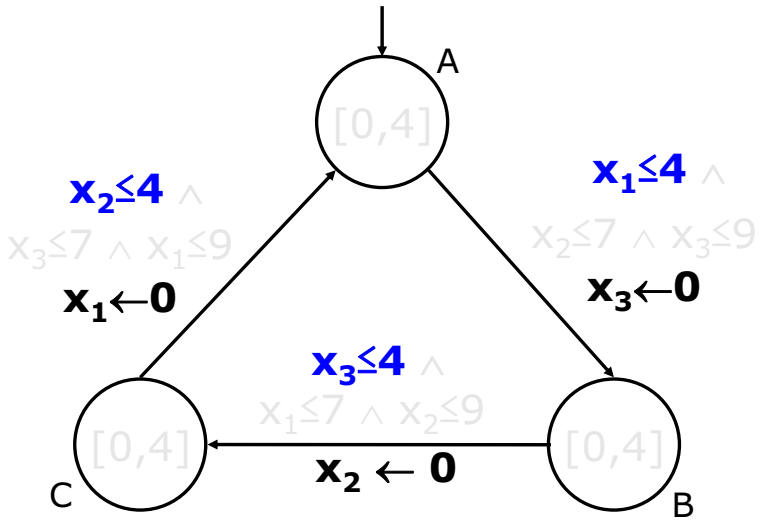
RTC to ECA

Δ	$\alpha^{upper}(\Delta)$
1	4
2	7
3	9



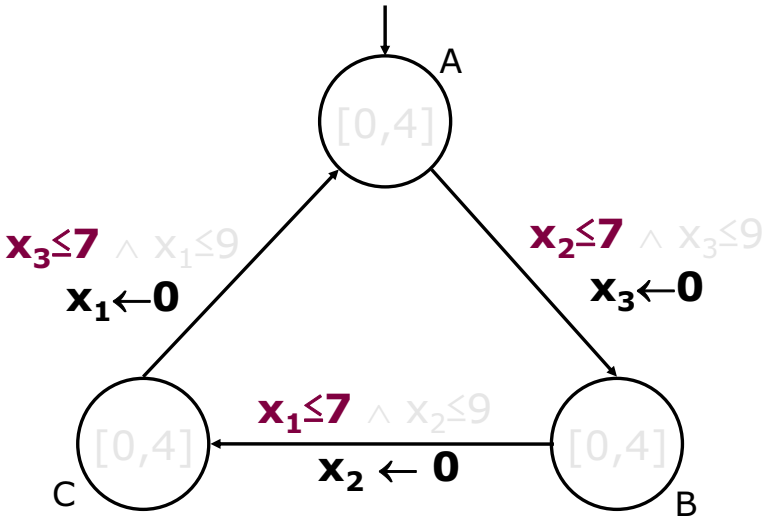
RTC to ECA

Δ	$\alpha^{upper}(\Delta)$
1	4
2	7
3	9



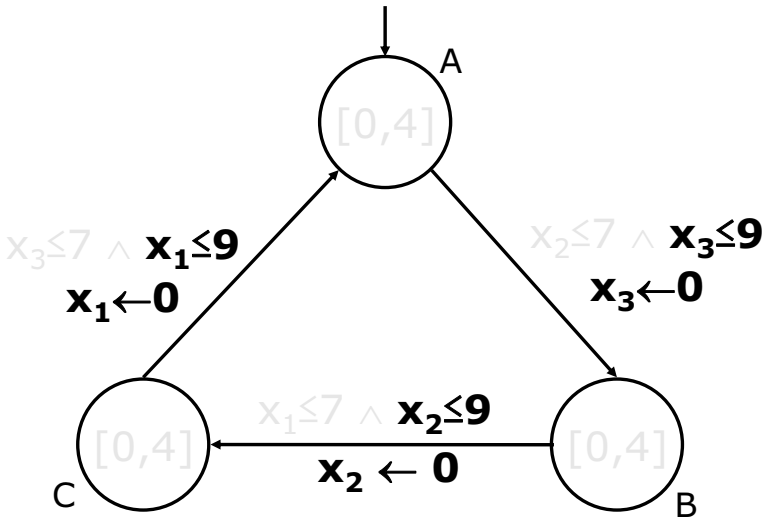
RTC to ECA

Δ	$\alpha^{upper}(\Delta)$
1	4
2	**7**
3	9



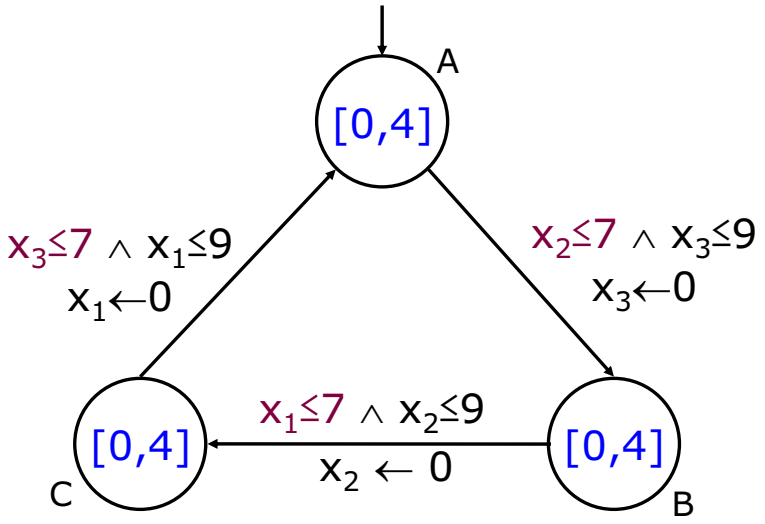
RTC to ECA

Δ	$\alpha^{upper}(\Delta)$
1	4
2	7
3	**9**



RTC to ECA

Δ	$\alpha^{upper}(\Delta)$
1	4
2	7
3	9



ECA→RTC Interface

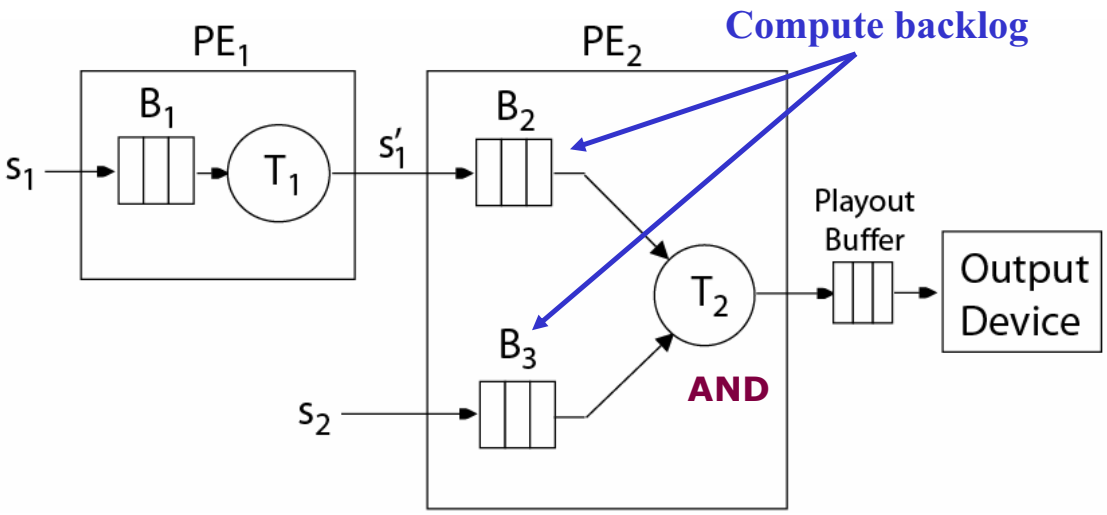
- Given an ECA component, construct the output arrival and remaining service curves
- Idea
  - Time interval : use different window sizes Δ
  - For each Δ, compute $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$
    - $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$: max. and min. number of output events observed in any interval of length Δ
  - Using automata verification to compute the bounds

Experimental Evaluation

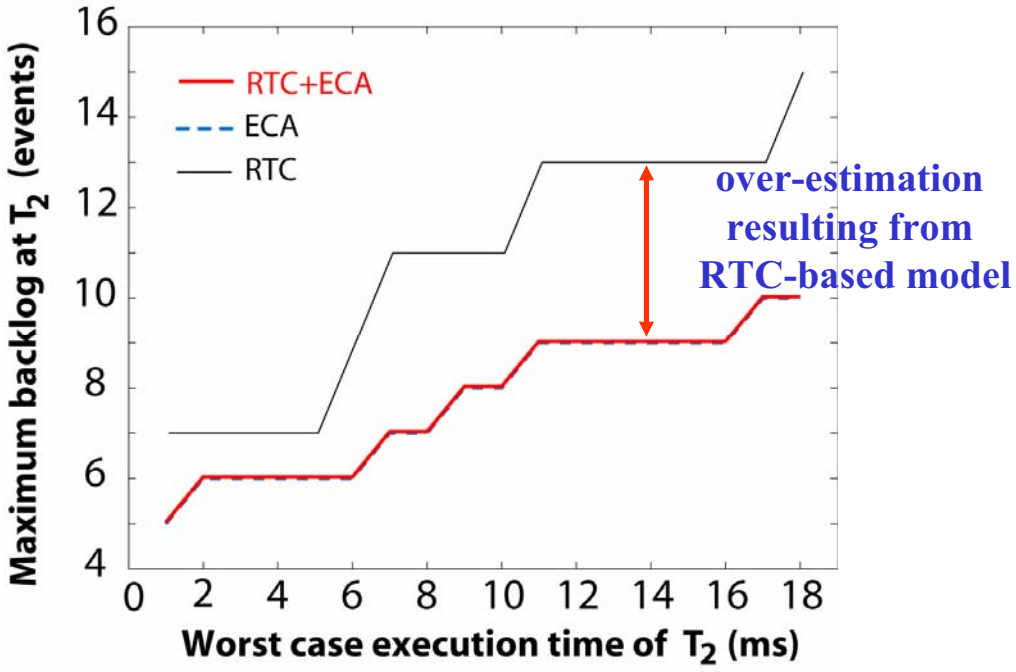
- Based on:
  - Real-time Calculus Toolbox (for RTC)
    - http://www.mpa.ethz.ch/Rtctoolbox
    - Developed at ETH Zürich
  - Symbolic Analysis Laboratory (for ECA)
    - http://sal.csl.sri.com/
    - Developed in collaborations of SRI, Stanford and Berkeley

Exp 1: AND-task Activation

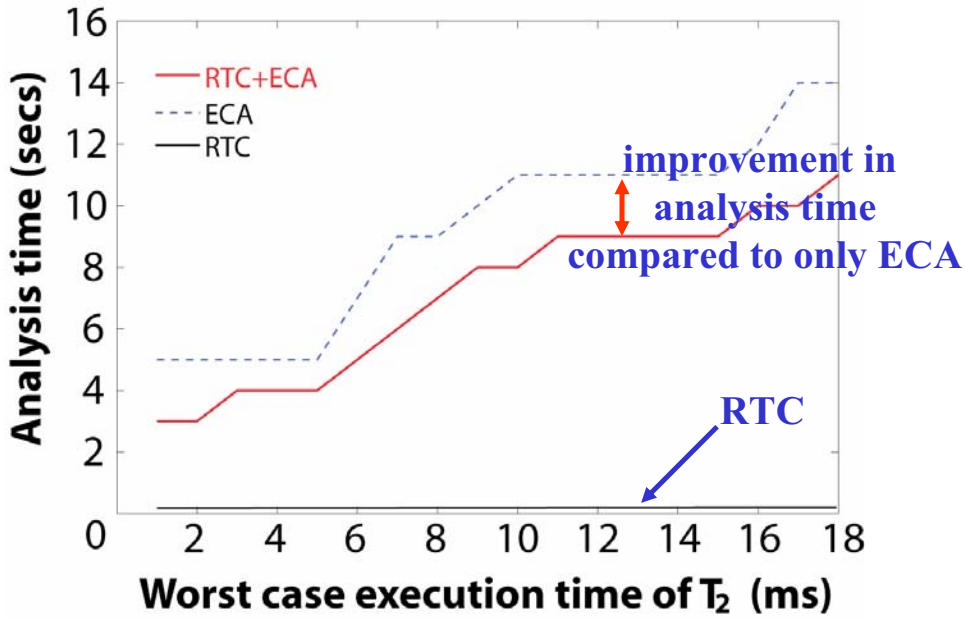
Combine 2 streams and process them



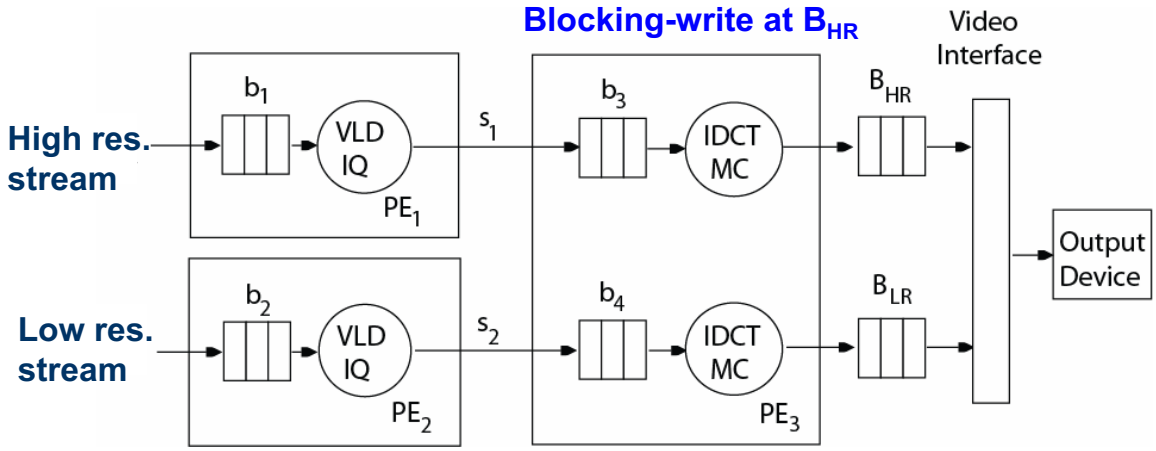
Exp 1: Backlog Results



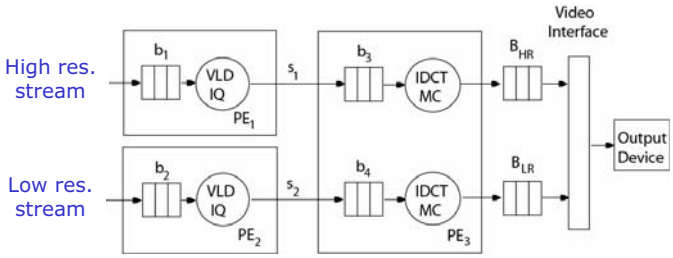
Exp 1: Analysis Time



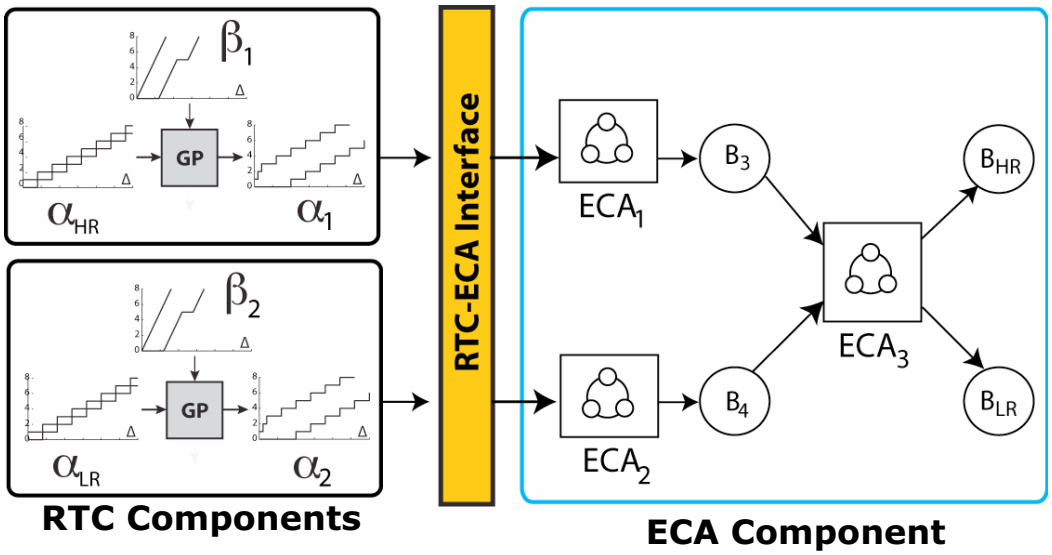
Exp 2: Picture-in-picture (PiP) Application



B, b: buffers, PE: Processing Element

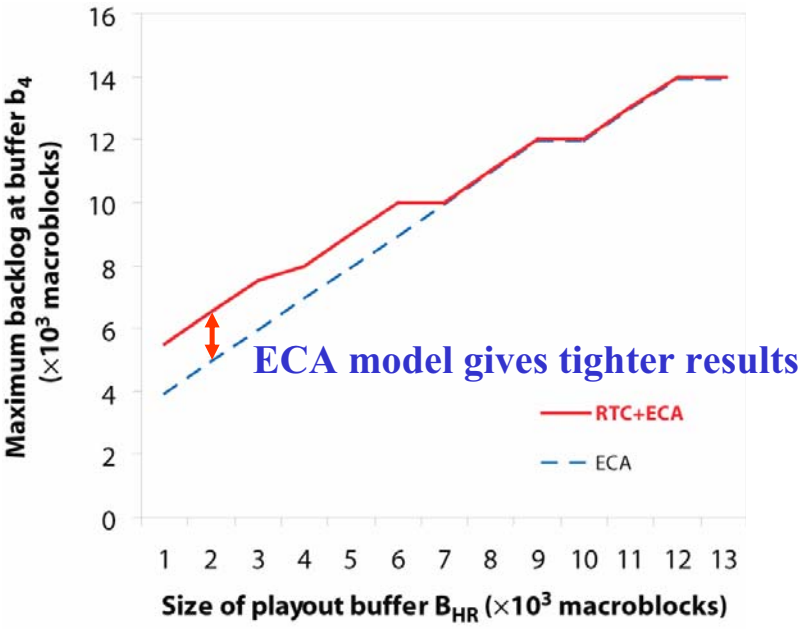


Modeling the PiP Application

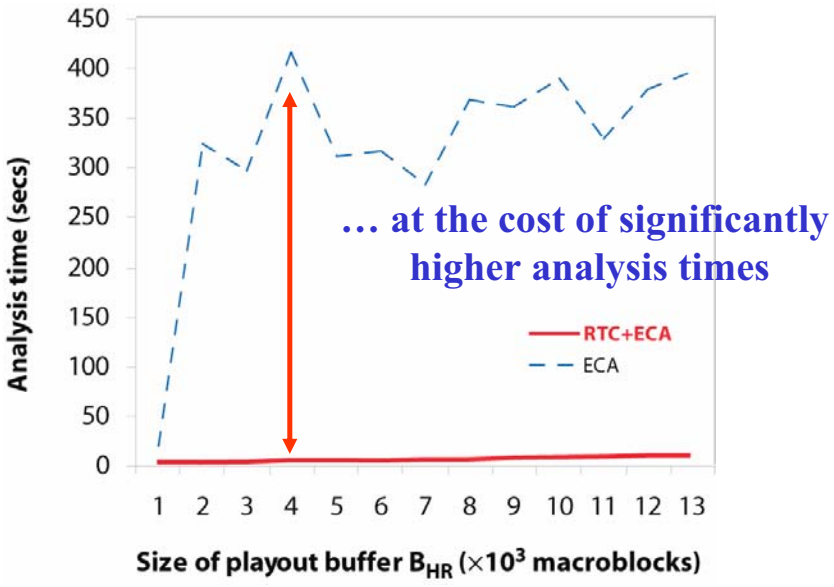


RTC Components **ECA Component**

Exp 2: Backlog Results



Exp 2: Analysis Times



Experiments

- Experiment 1
  - RTC is not accurate
  - ECA is slow
- Experiment 2
  - RTC cannot model
  - ECA is slow
- Both experiments
  - Our proposed method is more accurate than RTC and much more efficient than ECA

Concluding Remarks

- MpSoCs are increasingly becoming complex and heterogeneous
  - Single modeling/analysis paradigm might no longer be sufficient
  - Using multiple specification formalisms has been studies before (SPI: A System Model for Heterogeneously Specified Embedded Systems, Ziegenbein *et al*., TVLSI 2002)
  - However, the use of multiple performance models is relatively new and is open for research

Acknowledgements

- Thanks are due to
  - Linh T.X. Phan and P. S. Thiagarajan from the National University of Singapore
  - Lothar Thiele, Simon Kuenzli, Ernrsto Wandeler and Alexander Maxiaguine from ETH Zurich

Selected References

- S. Künzli, F. Poletti, L. Benini, L. Thiele, Combining simulation and formal methods for system-level performance analysis, *Design Automation and Test in Europe (DATE)*, 2006
- S. Chakraborty, L. T. X. Phan, P. S. Thiagarajan, Event Count Automata: A State-Based Model for Stream Processing Systems, *IEEE Real-Time Systems Symposium (RTSS)*, 2005
- S. Schliecker, S. Stein, R. Ernst, Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis, *Design Automation and Test in Europe (DATE)*, 2007
- Simon Künzli, Arne Hamann, Rolf Ernst, Lothar Thiele, Combined Approach to System Level Performance Analysis of Embedded Systems, *Intl. Conf. on Hw/Sw Co-design and System Synthesis (CODES+ISSS)*, 2007
- L. T. X Phan, S. Chakraborty, P. S. Thiagarajan, L. Thiele, Composing Functional and State-based Models for Analyzing Heterogeneous Real-Time Systems, *IEEE Real-Time Systems Symposium (RTSS)*, 2007