MPSOC Architecture Modeling

R. Ernst

TU Braunschweig

Overview

- introduction – MPSOC architecture trends
- implementation languages and architectures
- architecture model applications
- models for formal design methods
- summary

MPSOC architecture - trends

- system function integration
  - reactive and transformative parts
  - function IP, legacy code, new functions
- component and subsystem reuse (IP)
  - increased design productivity and reduced development cost
- programmable platforms
  - improved design productivity
  - increased volume
  - examples: network processors, multi-media platforms, automotive platforms, game platforms

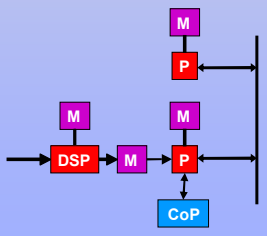
MPSOC architecture - challenges

- design specialization
  - increased performance
  - reduced power consumption
  - lower cost and size
- design flexibility
  - late changes, platforms, reuse
- HW and SW IP integration
  - result of reuse

⇒ MPSOC architectures are heterogeneous

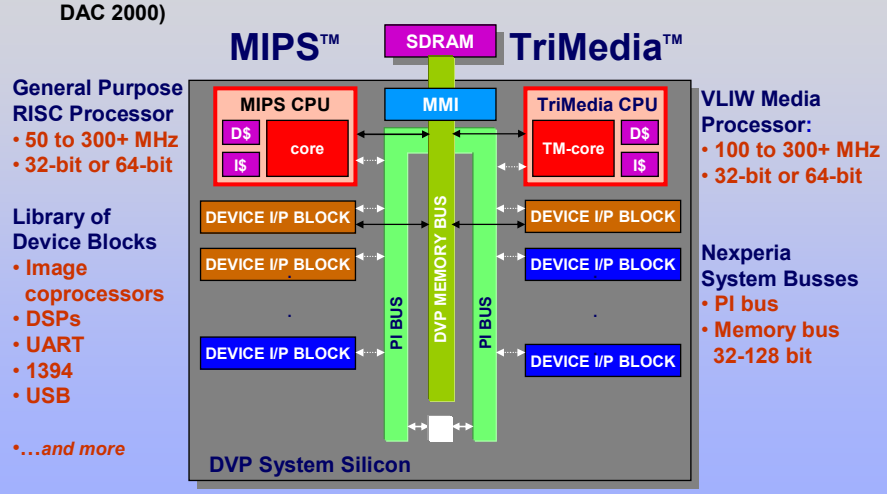
MPSOC architectures are heterogeneous

- different processing element types
  - processors, weakly programmable coprocessors, IP components
- different interconnection networks and communication protocols
- different memory types
- different scheduling and synchronization strategies

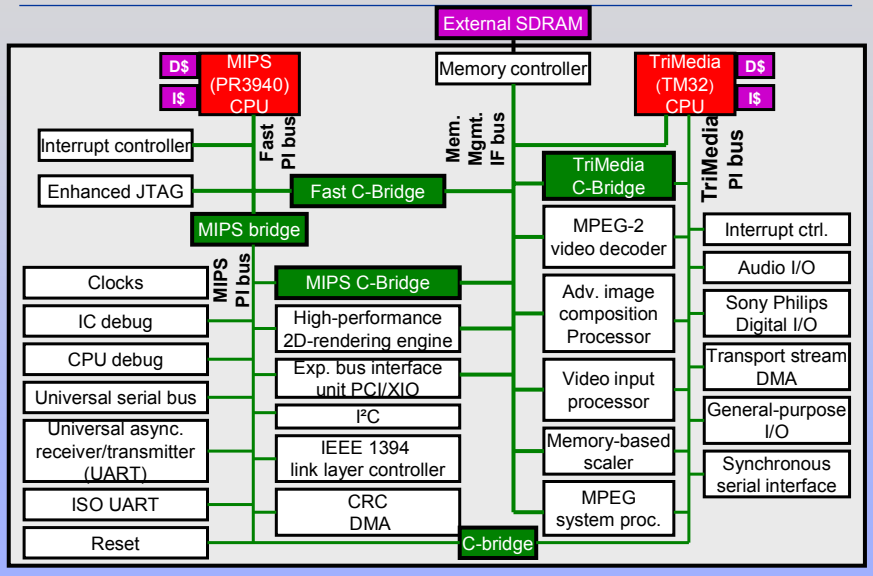


Example: Configurable platform (Nexperia™)

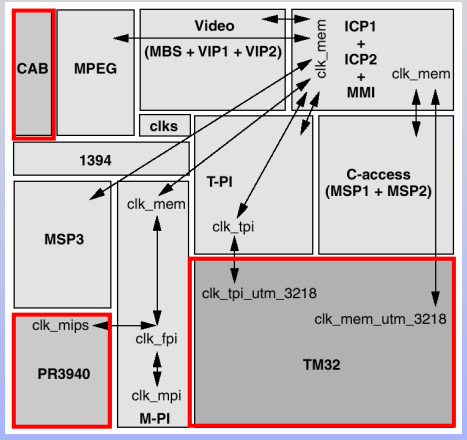
- Nexperia™ DVP hardware architecture (source: Th. Claasen, Philips, DAC 2000)



Nexperia example: Viper Setop Box



VIPER chip layout - reuse and integrate



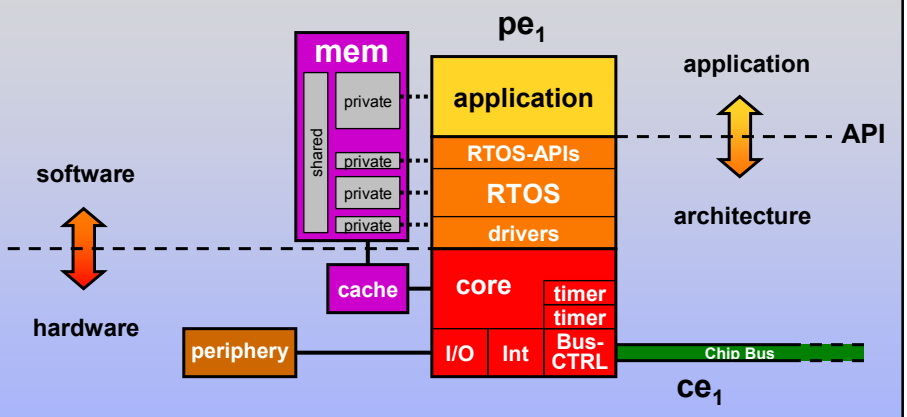
- **VIPER Hardmacros (supplied)** ☐
- **VIPER adaptable softmacros „Chiplets“** ☐

Managing HW architecture complexity

- development of application programmer interfaces (API) to hide complexity from application programmer and improve portability
- specialized RTOS to control resource sharing and interfaces

⇒ complex multi-level HW/SW architecture

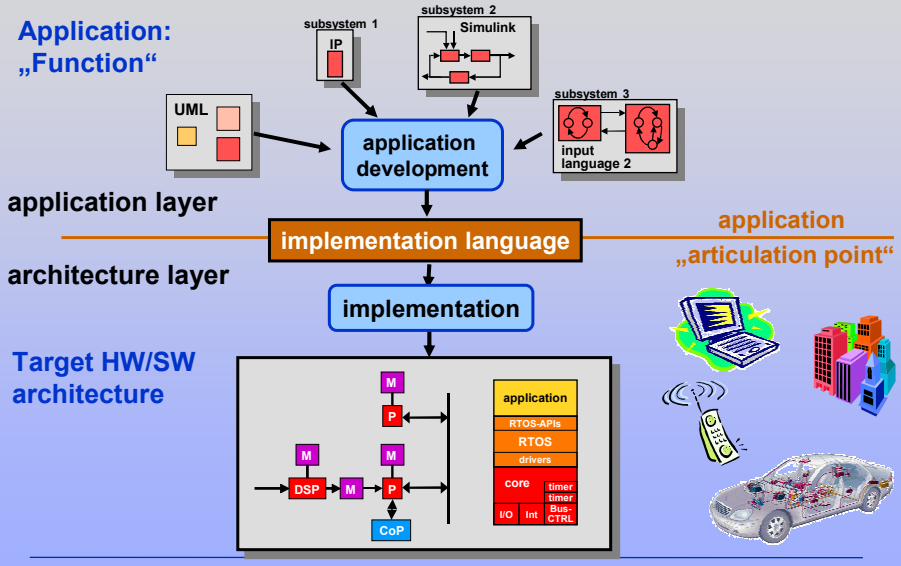
Software architecture example



- layered software architecture with HW dependent SW and API

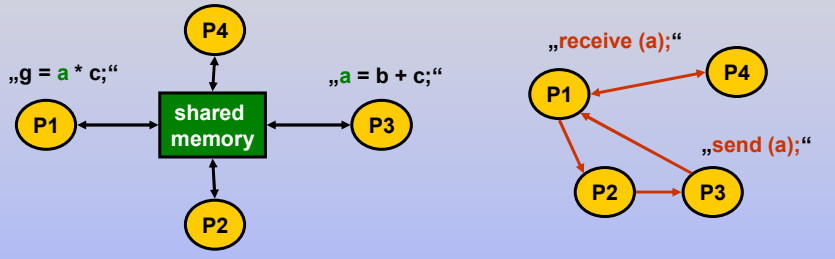
⇒ MPSOC SW is heterogeneous

Application & Architecture



Implementation Language Semantics

system of (communicating) processes



shared variable communication

C, C++, Java, (SystemC)

message passing communication

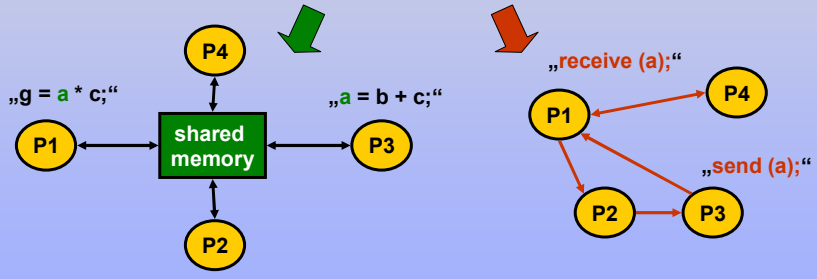
VHDL, SystemC, SpecC

Function Design → Implementation Language

system level language — SDL, Simulink, Lustre, ...

implementation language — manual implementation / code generation

examples:
- Real-Time Workshop
- Tau

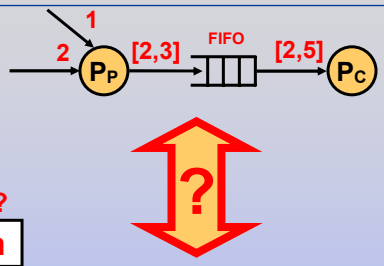


„Lossy“ function translation

- information lost in transformation
  - state dependent process behavior
    - hidden in processes
  - process coordination
    - expresses dependency and activation rules
    - important for efficient HW/SW architecture implementation
- example: data flow semantics → RTOS process activation

Token-to-event translation problem

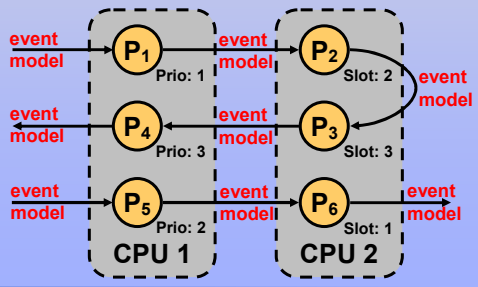
- Application view
  - complex activation dependencies
  - no timing (partial order)



What is an activating event?

➔ need transformation

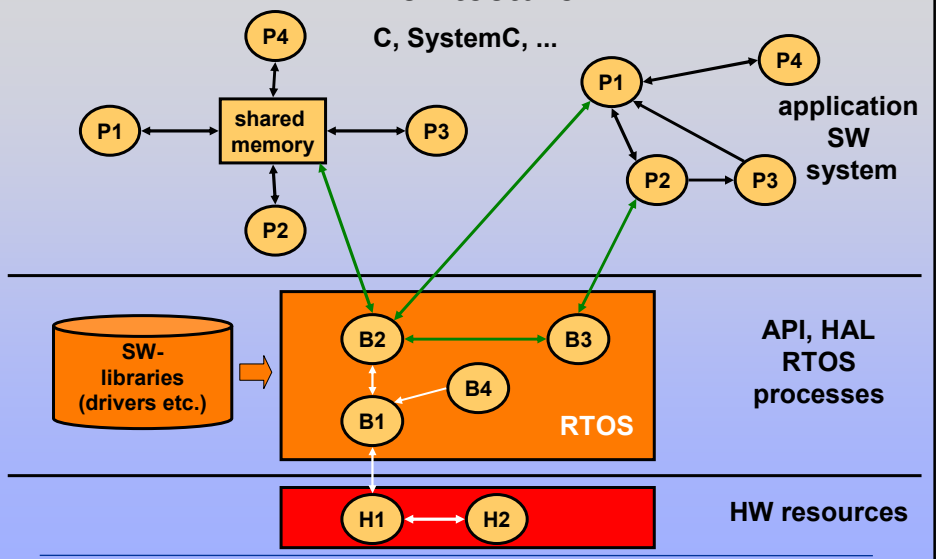
- Scheduling analysis view
  - simple activation dependencies (e.g. task graphs)
  - event timing



Coordination must be kept in translation

- solution for example e.g. using token arrival curves (Jersak/Ernst, DAC 03)
- semantic preserving models needed to keep information
  - Metropolis, Funstate, SPI

Implementation Language → HW/SW Architecture



Resource sharing

- HW/SW resources are shared
  - several SW processes mapped to one processing element
    ⇒ task scheduling
  - mapping several communications mapped to one communication path
    ⇒ communication (bus) scheduling
  - several process data mapped to one memory
    ⇒ memory assignment (space & time)

Resource sharing - 2

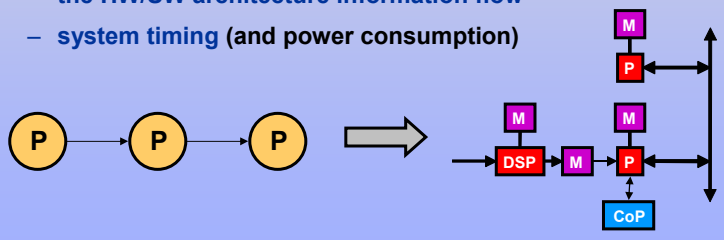
- resource sharing strategies
  - in time
    - execution sequence -> scheduling
  - in space
    - memory assignment
    - bus wire assignment

Architecture modeling applications

- implementation verification
- performance validation
  - response time
  - throughput (process exec/time unit)
  - bottleneck detection
- design optimization
  - design space exploration
  - power optimization
- cost determination (not this lecture)

MPSOC architecture modeling requirements

- given
  - an application and its environment modeled by a set of communicating processes
  - a heterogeneous HW/SW target architecture
  - an implementation of the processes on the architecture
- model
  - the HW/SW architecture information flow
  - system timing (and power consumption)

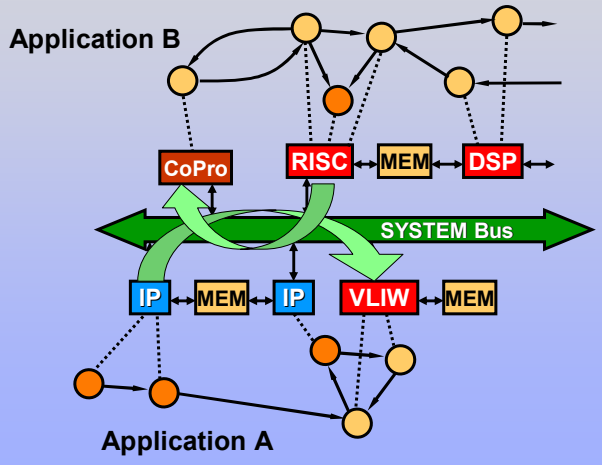


Modeling Challenges

- model complexity
  - HW/SW system state space
  - simulation run-times and analysis complexity
  - model abstraction
- activation modeling
  - simulation pattern development
  - environment modeling
- complex non-functional interdependencies
  - shared communication
  - shared components
  - shared memory

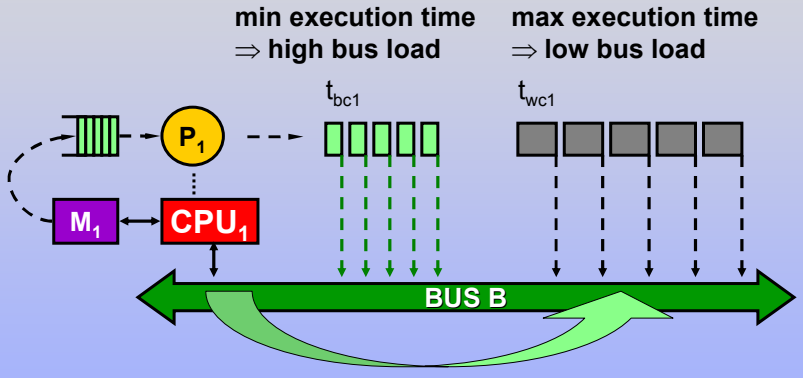
Complex non-fuctional interdependencies

- resource sharing introduces complex non-functional interdependencies („cross talk“)



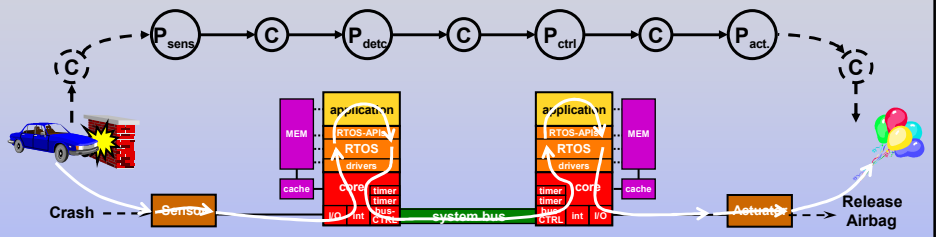
Interdependency example

- anomalies: best case can become worst case



Modeling Challenges - cont'd

- complex design objectives and constraints



Reaction time of airbag after crash ?

$$\underbrace{t_{crash} + t_{sens}}_{\text{physical delay}} + t_{csens} + t_{detc} + t_{fbus} + t_{ctrl} + t_{cact} + \underbrace{t_{act} + t_{airbag}}_{\text{physical delay}}$$

$$= t_{com} + t_{drv}$$

$$+ t_{API} + t_{process} + t_{API}$$

$$+ t_{drv} + t_{com} + t_{drv}$$

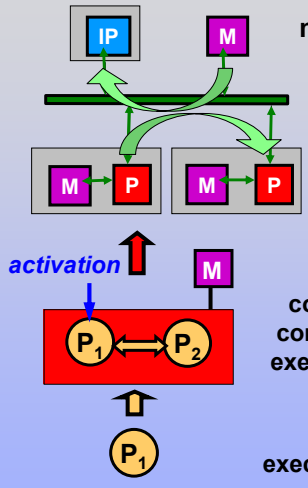
$$+ t_{API} + t_{process} + t_{API}$$

$$+ t_{drv} + t_{com}$$

MPSoC modeling - goals

- current goal: Target architecture co-simulation
  - supports system-level validation
  - uses library of component and communication models
  - requires executable code and software platform
  - extensive simulation required for complex MPSoC or distributed systems
- research goal: support formal methods for design space exploration, system-level optimization and analysis
  - different modeling approach required

Architecture model structure

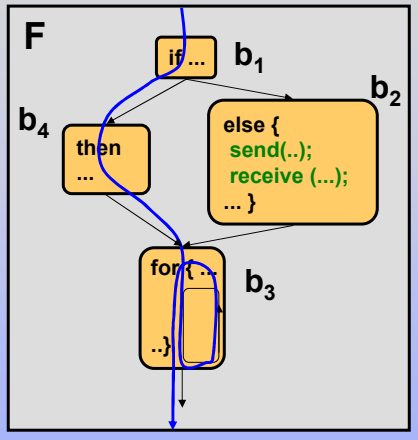


model type	influenced by
system model	• communication pattern • shared memory access • environment model
component & communication execution model	• resource sharing strategy • process activation • component state (caches)
process execution model	• execution path – data dependent • path execution – arch. dependent • communication – data & arch. dep.

Process execution model

- timing and communication depend on
  - execution path
  - architecture
  - communication mechanism and volume

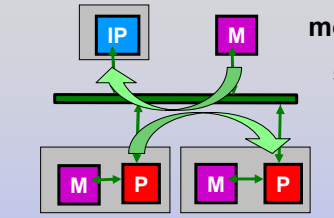
({b1, …, bn} basic blocks)



Process timing and communication

- process timing and communication can be evaluated by
  - simulation/performance monitoring
    - trigger points at process beginning and end
    - stimuli required, e.g. from component design
    - data dependent execution → upper and lower timing bounds
  - simulation challenges
    - coverage?
    - cache and context switch overhead due to run-time scheduling with process preemptions
    - influence of run-time scheduling depending on external event timing
  - formal analysis of individual process timing
    - serious progress in recent years
  - *discussion see book chapter and literature*

Architecture model structure

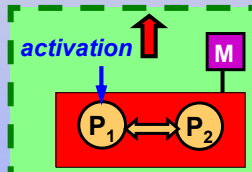


model type

system model

influenced by

- communication pattern
- shared memory access
- environment model



component & communication execution model

- resource sharing strategy
- process activation
- component state (caches)

process execution model

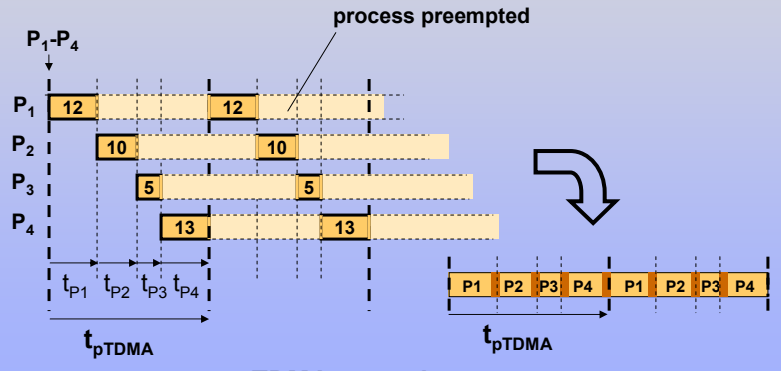
- execution path – data dependent
- path execution – arch. dependent
- communication – data & arch. dep.

Component and communication execution model

- Resource sharing strategy
  – process and communication scheduling
  - static execution order
  - time driven scheduling
    - fixed
    - dynamic
  - priority driven scheduling
    - static priority assignment
    - dynamic priority assignment
- timing depends on environment model
  - frequency of process activations or communication
  - solution for activation transformation proposed in (Jersak/Ernst DAC 03)

Ex 1: Time driven scheduling

- time division multiple access (TDMA)
  - periodic assignment of fixed time slots
  - applicable to pe or ce



TDMA example

TDMA

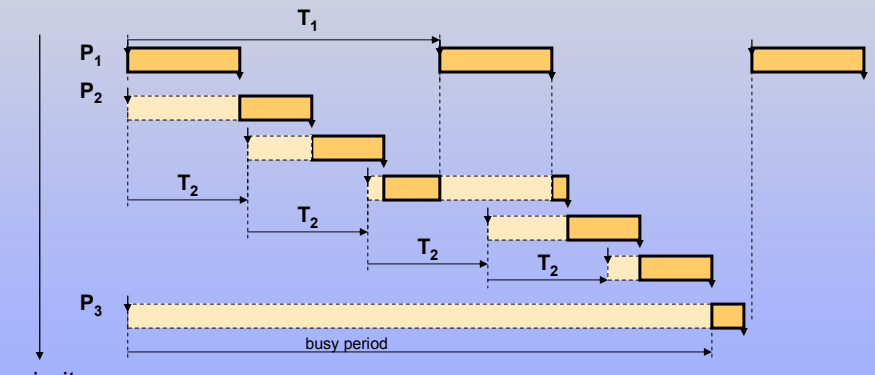
- predictable and independent performance down scaling allows to merge individual solutions

$$t_{peTDMA}(P_i, pe_i) = \left\lfloor \frac{t_{pe}(P_i, pe_i) - t_{csw}}{t_{Pi}} \right\rfloor \cdot t_{pTDMA} + t_{pe}(P_i, pe_i) \bmod t_{pi}$$

- time slot size adaptable to different service levels
- generates output jitter as a result of execution times
- problems
  - utilization
  - extended deadlines

Ex 2: Static priority with arbitrary deadlines

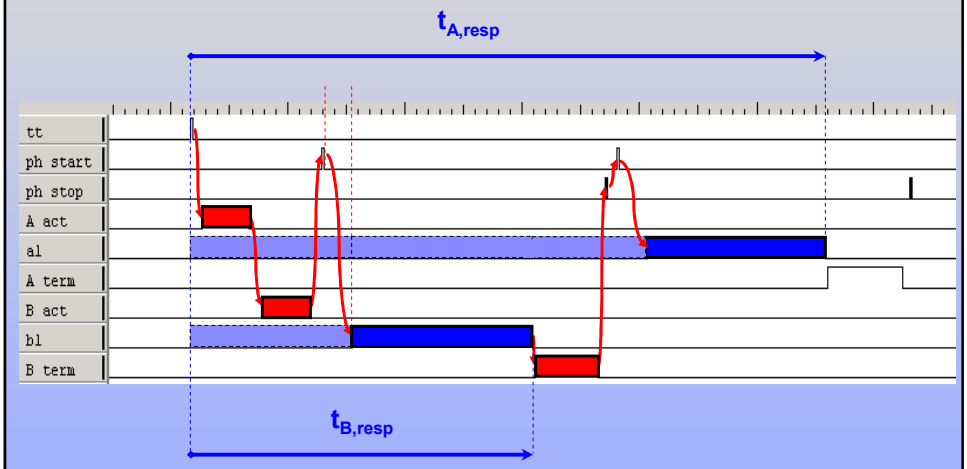
- complex execution sequence - may create output bursts
- found in communication scheduling and multiprocessing



analysis solution e.g. by Lehoczky

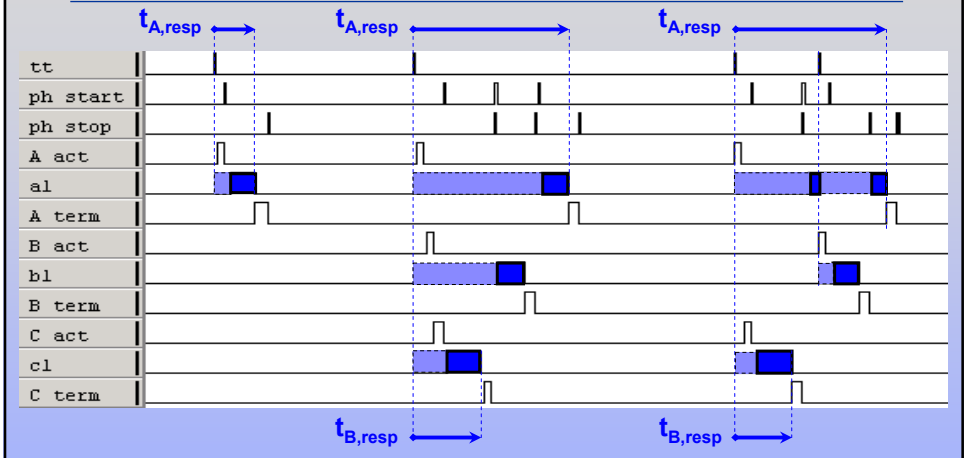
RTOS Overhead

Example: Static priority scheduling (ERCOSEK™)



RTOS overhead increases response times

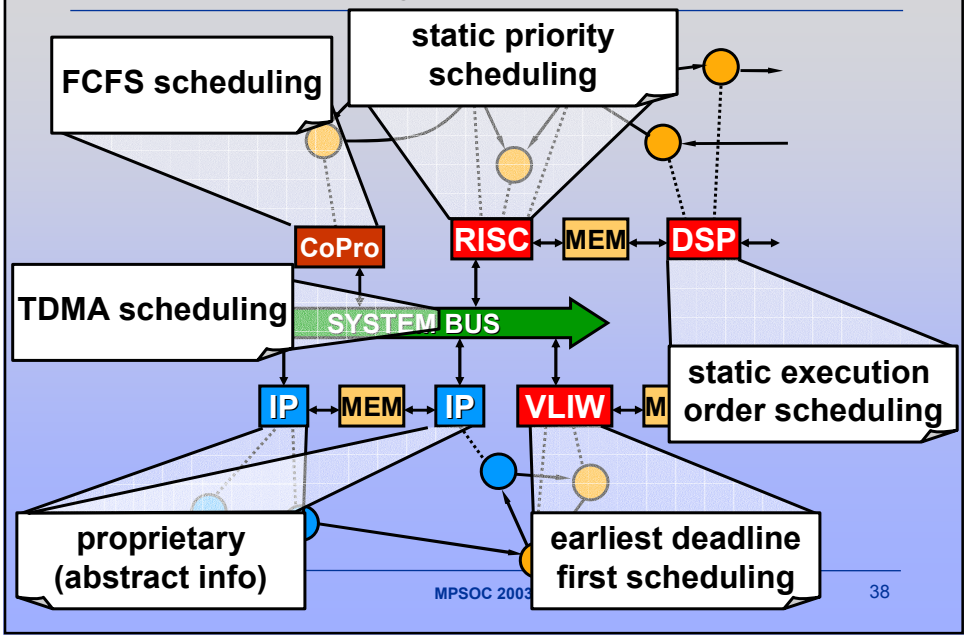
RTOS and scheduling effects combined



Cache effects

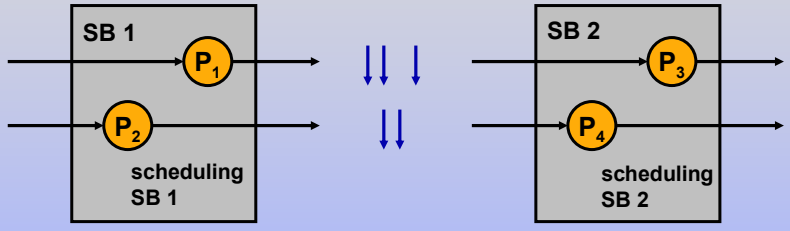
- cache contents replaced by other processes
  - → increased execution time
  - → must be considered in analysis
- scratch pad memories as alternative

System model



Component coupling

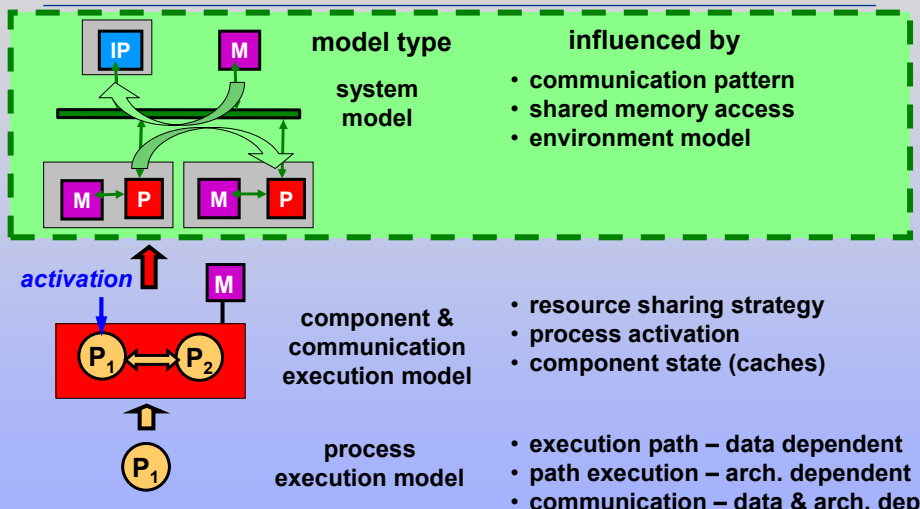
- independently scheduled subsystems are coupled by data flow



⇒ subsystems coupled by stream of events

⇒ coupling corresponds to event propagation

Architecture model structure



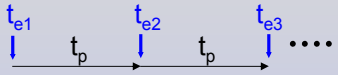
model type	influenced by
system model	• communication pattern • shared memory access • environment model
component & communication execution model	• resource sharing strategy • process activation • component state (caches)
process execution model	• execution path – data dependent • path execution – arch. dependent • communication – data & arch. dep.

System timing analysis with event propagation

- analysis scope extension to several subsystems
  - holistic approach, e.g. Tindell and Pop/Eles
  - used for automotive software
- event model generalization for a set of scheduling strategies
  - arrival and service curves Chakratborty/Thiele
  - new analysis approaches needed, e.g. Baruah
  - used for network processor design
- event stream model adaptation
  - use abstract interface stream properties to couple local analysis
  - used e.g. for automotive software

Event stream models

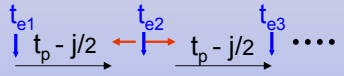
- periodic events



$$t_{e_i+1} - t_{e_i} = t_p$$

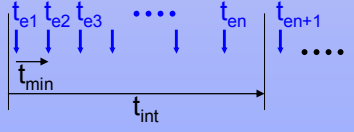
t_{ei} typically timer released

- periodic events with jitter



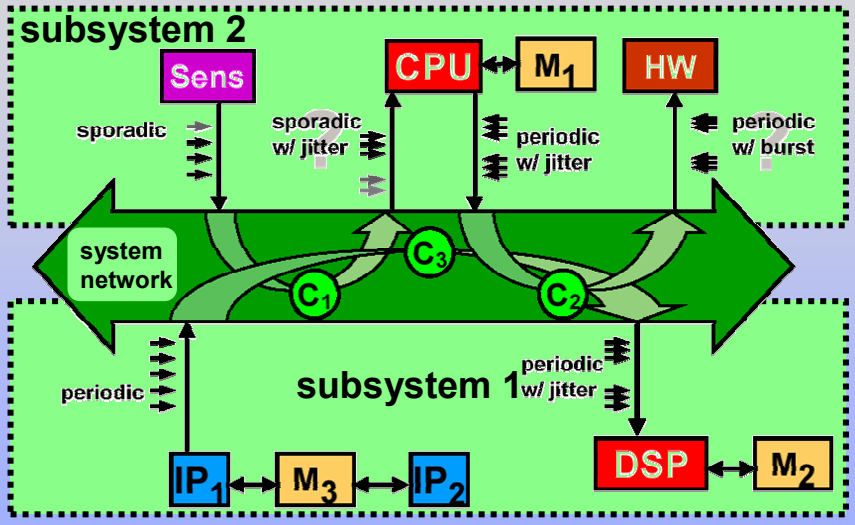
$$t_p - j \le t_{e_{i+1}} - t_{e_i} \le t_p + j;$$
$$j < t_p$$

- events with minimum inter arrival times
  - burst events, packets, sporadic events, etc.

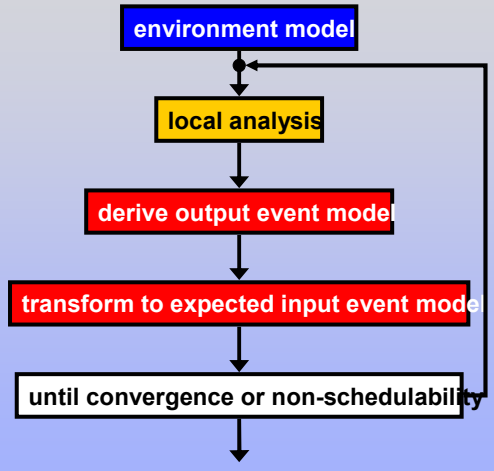


$$t_p - j \le t_{e_{i+1}} - t_{e_i} \le t_p + j;$$
$$t_{e_{i+n}} - t_{e_i} \ge t_{\text{int}}; \quad t_{e_{i+1}} - t_{e_i} \ge t_{\min}$$

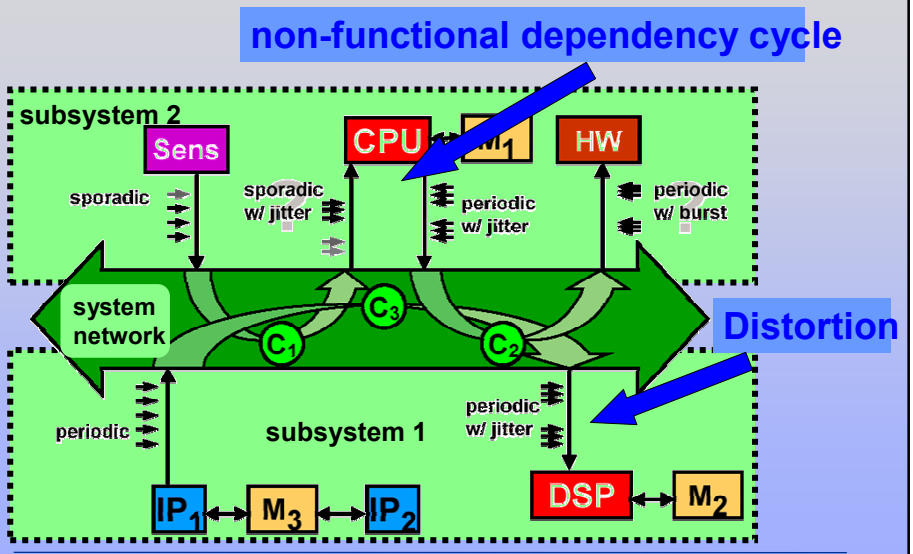
Event stream example



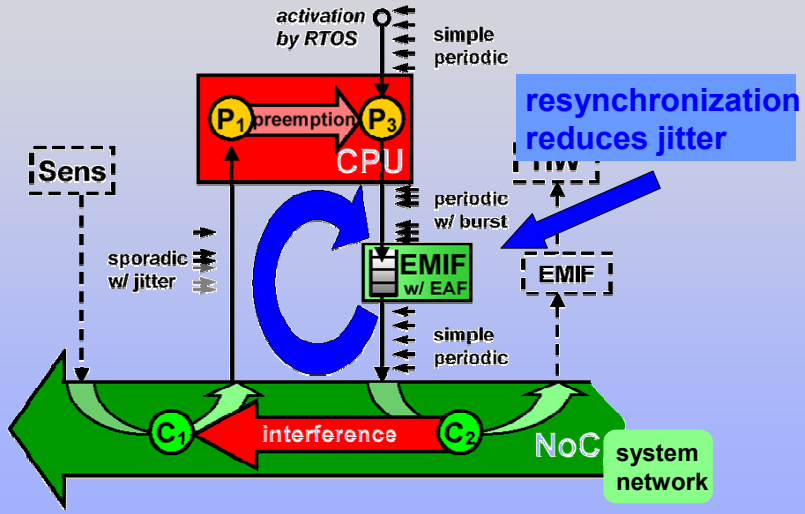
Event propagation and analysis principle



Example revisited



Dependency cycle



Some open issues

- system mode dependent timing
  - different load situations and response time requirements
  - tagged tokens -> tagged event streams
- (complex) hierarchical prcess and communication scheduling
- global scheduling optimization

Architecture model summary

- current MPSOC architecture models are primarily used for simulation
- growing complexity suggests formal methods for optimization and analysis
- emerging formal approaches supported by multi-level architecture modeling
- abstract event flow model enables heterogeneous system analysis

Literature

- see: www.spi-project.org

Acknowledgement

- The following persons contributed in developing these slides
  - Bettina Boettger
  - Jörn Braam
  - Marek Jersak
  - Kai Richter