A compositional approach to embedded system performance analysis

R. Ernst

TU Braunschweig

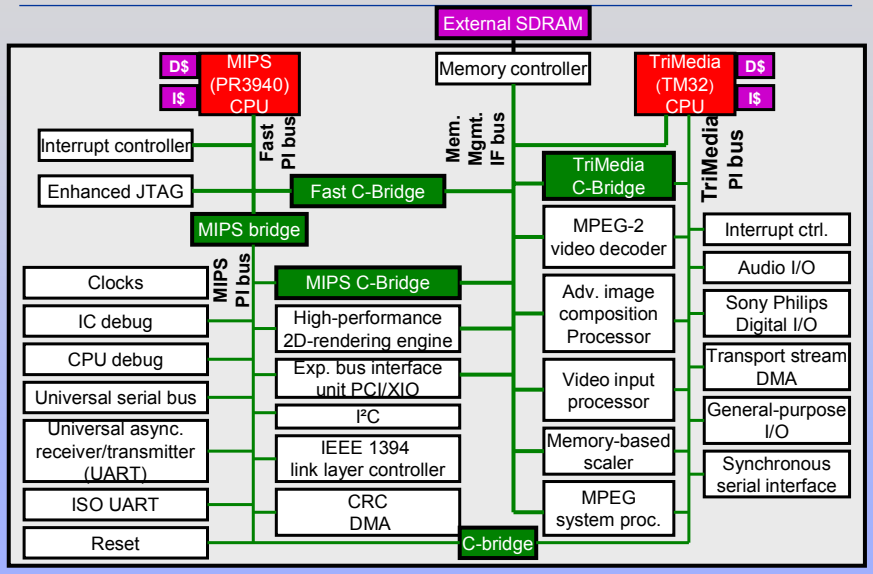
Overview

- heterogeneous embedded architectures
- formal performance analysis – holistic and flow based
- a compositional approach to performance analysis
- example
- context aware analysis
- conclusion

Embedded systems - trends

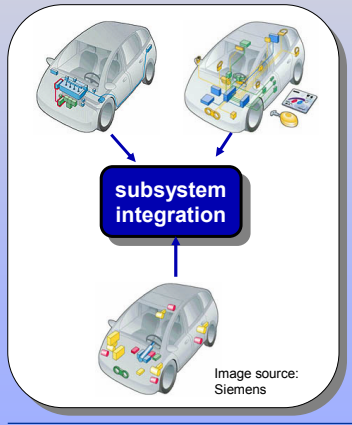
- **Trend 1: higher system integration**
  - **integration of complete programmable subsystems on a single IC – Multiprocessor-Systems-on-Chip (MpSoC)**

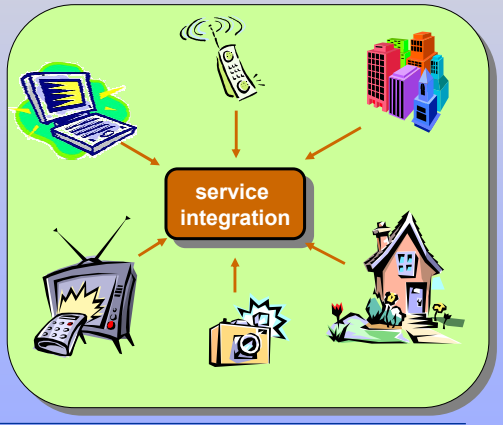
Nexperia example: Viper Setop Box



Embedded systems - trends - 2

- Trend 2: networked systems
  ubiquitous computing, telecom, automotive, avionics, space, ...





Embedded architectures - 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

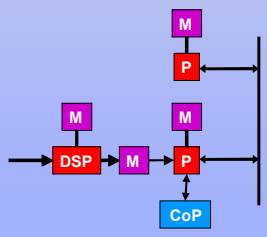
Embedded architecture - challenges

- 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

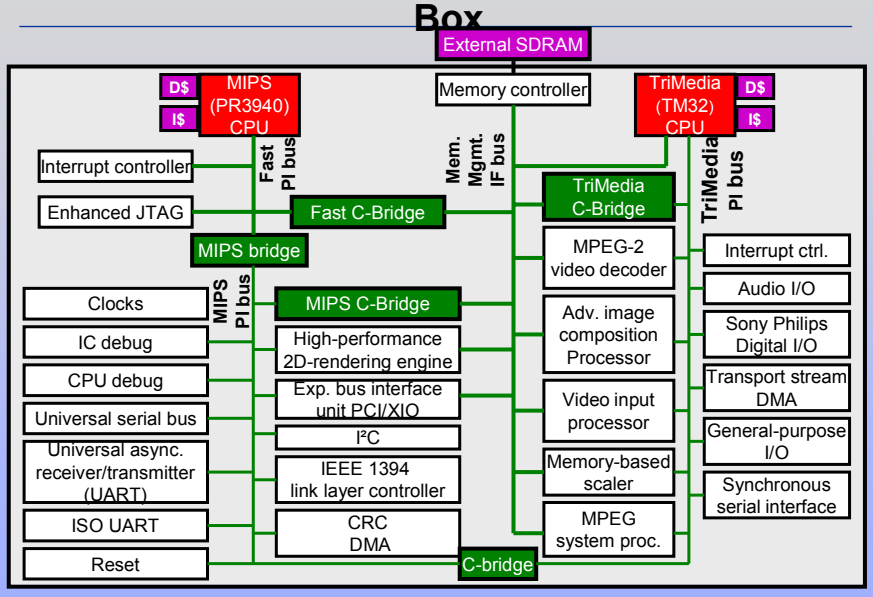
⇒ embedded HW architectures are heterogeneous

ES 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



Heterogeneous architecture: Viper Setop Box

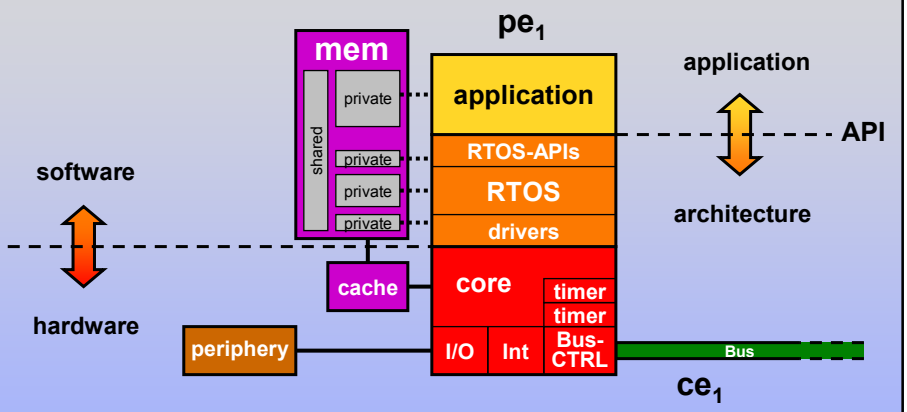


Managing HW architecture complexity

- **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

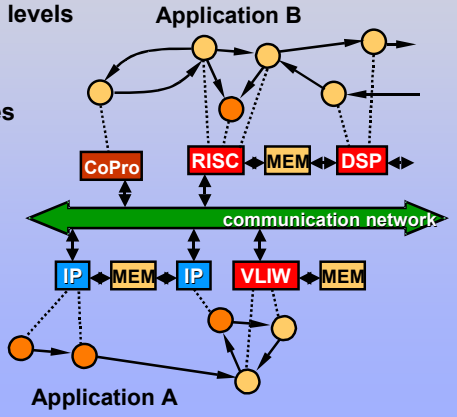


- layered software architecture with HW dependent SW and API

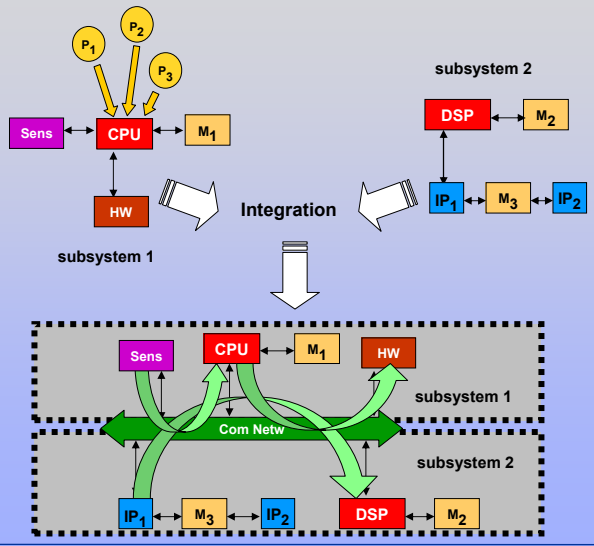
⇒ embedded SW is heterogeneous

Communication Centric Design

- communication network as a backbone for systems integration
- state of the art:
  - off chip: busses w. different protocols and performance levels
  - on chip: Proprietary or standard buses with bridges AMBA, Sonics, ...
  - future: multi-stage networks on-chip as well as off-chip (PCI Express)



Subsystem integration - example



Communication centric design - key challenges

- subsystem integration&verification
- design space exploration and optimization

Integration tasks

- resource sharing
  - 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)
- interface synthesis
  - synthesizing process communication to target system communication
- integration verification

Integration verification

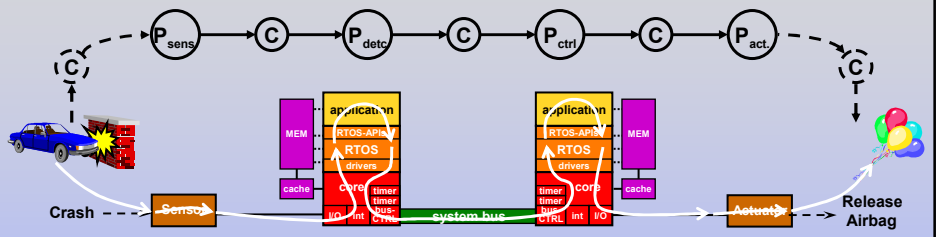
- correct implementation of specified function
  - HW/SW co-simulation, verification

} general design problem

- correct target architecture parameters
  - processor and communication performance
  - adherence to timing requirements
  - no memory over/underflow
  - no run-time dependent dead-locks

} challenge to heterogeneous system design

Complex performance objectives and constraints

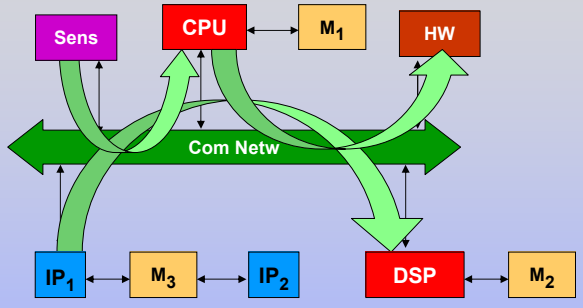


Reaction time of airbag after crash ?

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

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

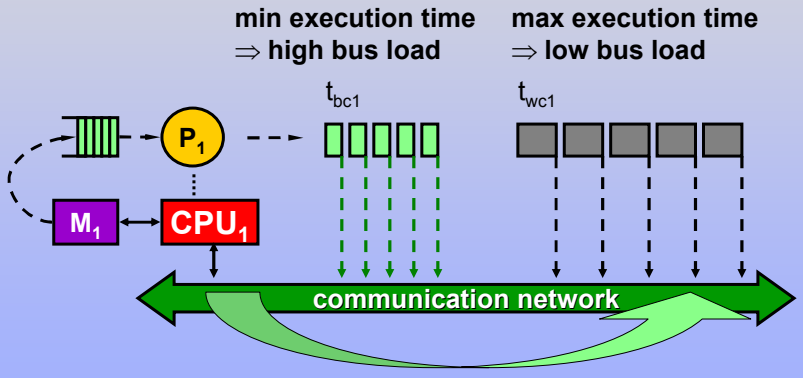
Complex run-time interdependencies



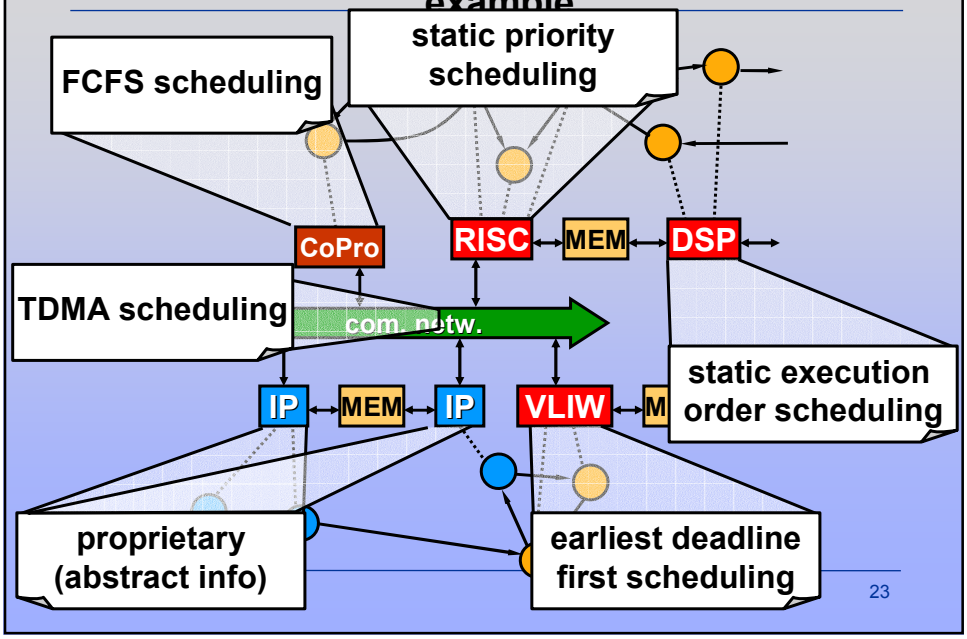
- run-time dependencies of independent components via communication
- influence on timing and power

Interdependency example

- complex non-functional interdependencies
- complex system corner cases



Heterogeneous resource sharing - example



Performance verification - state of the art

- current approach: target architecture co-simulation
  - combines functional and performance validation
  - reuse component validation pattern for system integration and function test
  - reuse application benchmarks for target architecture function validation
  - visualization of system execution
  - extensive simulation run times to include many test cases

Co-simulation limitations

- identification of *system* performance corner cases
  - different from *component* performance corner cases
  - target architecture behavior unknown to the application function developer (cp. functional HW test)
    - ⇒ test case definition and selection ?
- analysis of target architecture
  - confusing variety of run-time interdependencies
  - data dependent “transient” run-time effects
  - mixed in co-simulation
    - ⇒ limited support of design space exploration
    - ⇒ debugging challenge
- inclusion of incomplete application specifications
  - ⇒ additional performance models required

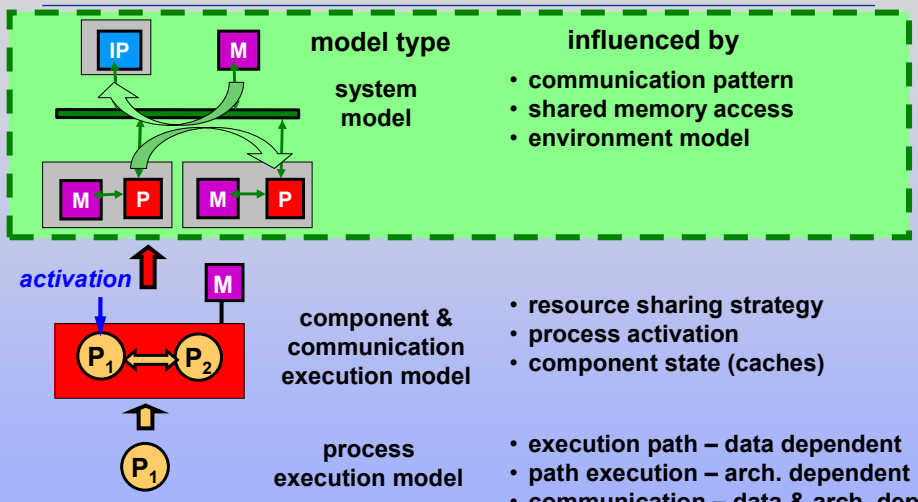
Alternatives?

- conservative design
  - install independency in resource sharing
  - example: fixed execution time slots – TDMA
- formal performance analysis

Formal performance analysis

- formal techniques known for individual components and subsystems (RMS, static scheduling etc.)
- heterogeneity is problem

Performance model structure



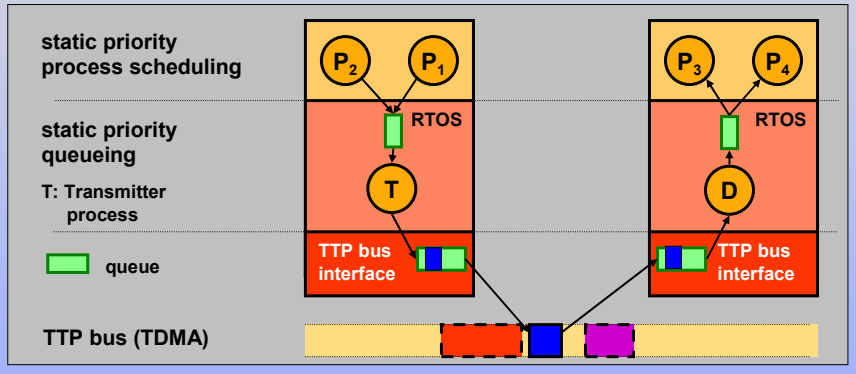
model type	influenced by
system model	• communication pattern • shared memory access • environment model
component & communication execution model	• resource sharing strategy • process activation • component state (caches)
process execution model	• execution path – data dependent • path execution – arch.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

System performance analysis approaches

- global approach
  - analysis scope extension to several subsystems
- flow based hierachical approach
  - global flow analysis combined with local scheduling analysis

Analysis scope extension

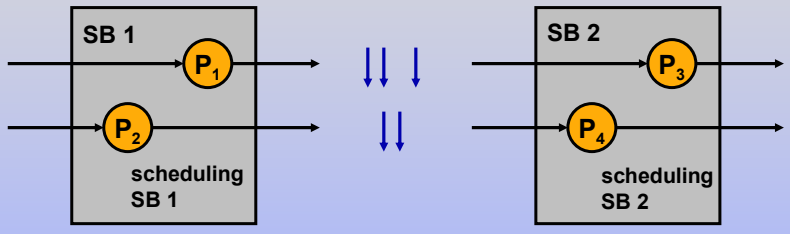
- coherent analysis („holistic“ approach)
- example: Tindell 94, Pop/Eles (DATE 2000, DAC 2002, …): TDMA + static priority – automotive applications



- problem: scalability

Hierarchical approach

- independently scheduled subsystems are coupled by data flow

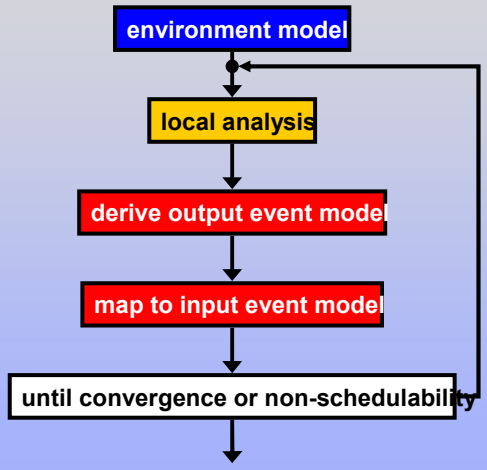


⇒ subsystems coupled by stream of data

⇒ interpreted as activating events

⇒ coupling corresponds to event propagation

Event propagation and analysis principle

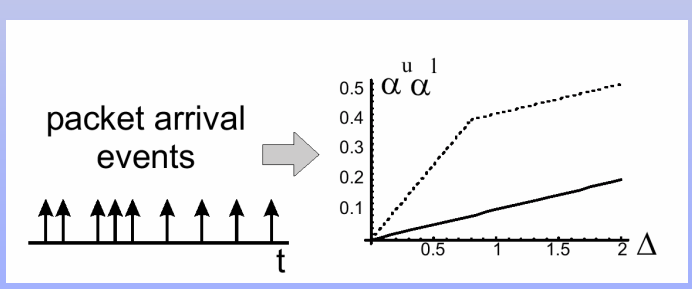


Flow based hierarchical approaches

- event model generalization for a set of scheduling strategies
  - arrival and service curves Chakraborty/Thiele/Gries/Künzli …
  - 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

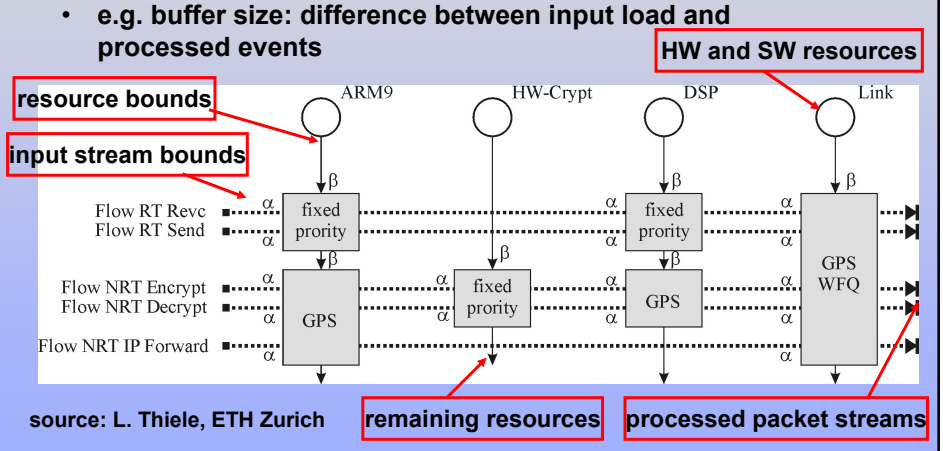
Generalized event model

- incoming events stream is integrated over time intervals and captured in arrival curves
- event consumption is captured in service curves
- 2 curves each for upper and lower bounds (interval)
- upper and lower bounds linearly approximated for analyis



Load analysis with interval event model

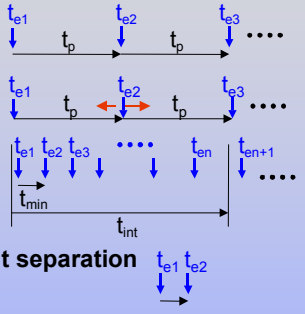
- Load analyis: Addition propagation of computation and load intervals - min/max algebra
- e.g. buffer size: difference between input load and processed events



source: L. Thiele, ETH Zurich

Compositional Approach (Ernst et al.)

- observation: real-time analysis assumes similar event models at input
  - periodic event stream
  - periodic event streams with jitter
  - periodic event streams with burst
  - sporadic events with minimum event separation
  - sporadic events with bursts



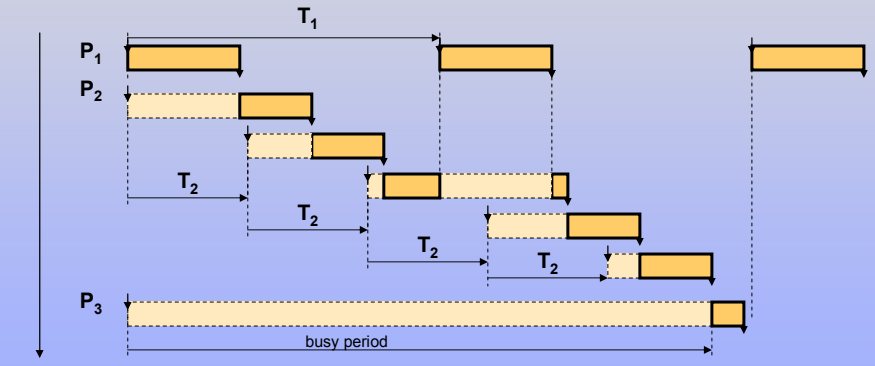
- comparable event models appear at output
- volume of generated events is fixed or interval (data dependent)

RTA event model examples

- static execution order
  - periodic → periodic w. jitter
    - Jitter: fixed part (scheduling) + variable part (data dependency)
  - sporadic → sporadic w. jitter
- static priority
  - periodic → periodic w. jitter and burst
  - sporadic → sporadic w. jitter and burst
- time driven scheduling: TDMA, round robin
  - adds jitter to any model

Example: Static priority with arbitrary deadlines

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



analysis solution e.g. by Lehoczky

Few stream models can be derived

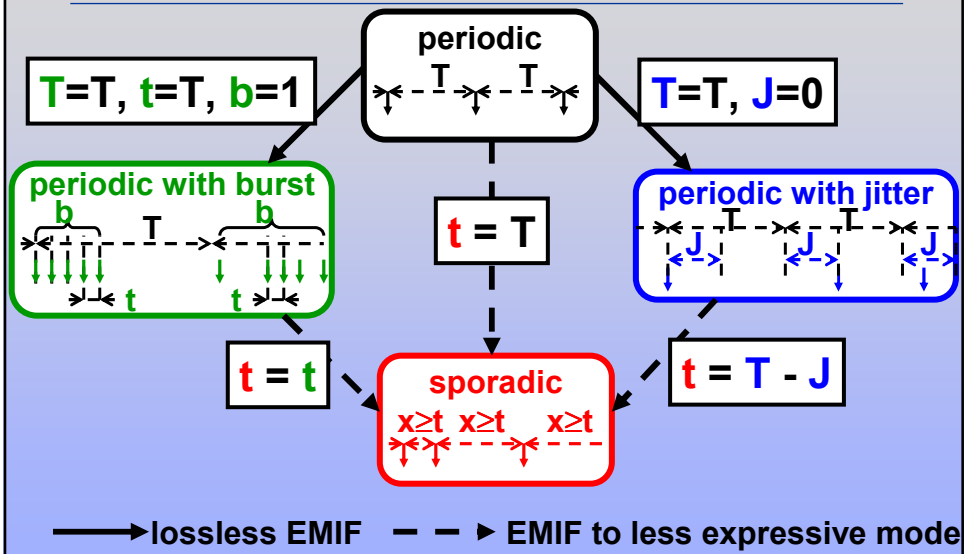
model	params	$n^{+}_{act}(\Delta t)$	$n^{-}_{act}(\Delta t)$
periodic	$< T >$	$\lceil \frac{\Delta t}{T} \rceil$	$\lfloor \frac{\Delta t}{T} \rfloor$
jitter	$< T,J >$	$\lceil \frac{\Delta t+J}{T} \rceil$	$\max\left(0, \lfloor \frac{\Delta t-J}{T} \rfloor\right)$
sporadic	$< t >$	$\lceil \frac{\Delta t}{t} \rceil$	0
burst	$< T,t,b >$	$\lfloor \frac{\Delta t}{T} \rfloor b + \min\left(b, \left\lceil \frac{\Delta t - \lfloor \frac{\Delta t}{T} \rfloor T}{t} \right\rceil\right)$	0

- n_{act} (t) no. of activating events in time t
- burst can be simplified and modeled as jitter greater period with min. event distance

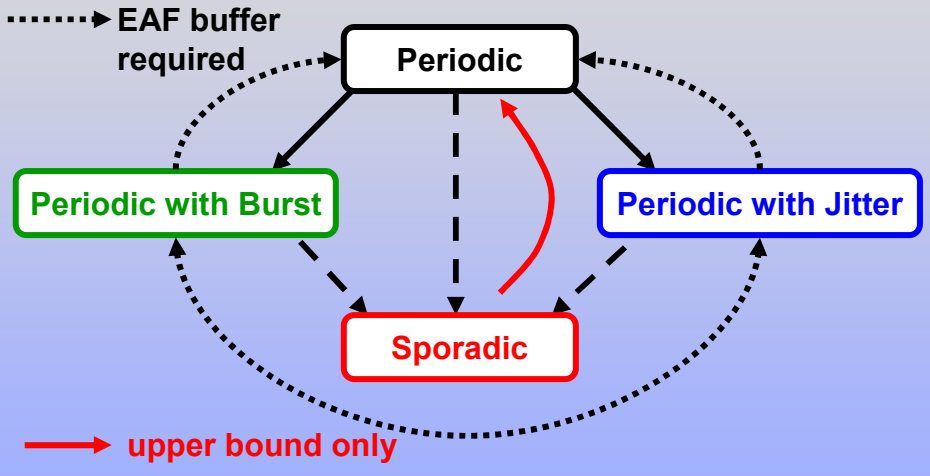
Event stream model adaptation

- composition requires model adaptation
- adaptation for compatible input and output event models
  - simple mathematical transformation to adapt analysis
    Event Model InterFace (EMIF)
  - no added function, no run-time effect
- adaptation for non-compatible input and output event models
  - insert Event Adaptation Function (EAF) to transform models
  - EAF describes interface buffer function
    ⇒ shows that buffer is required to couple subsystems

Event Model Interface Classification

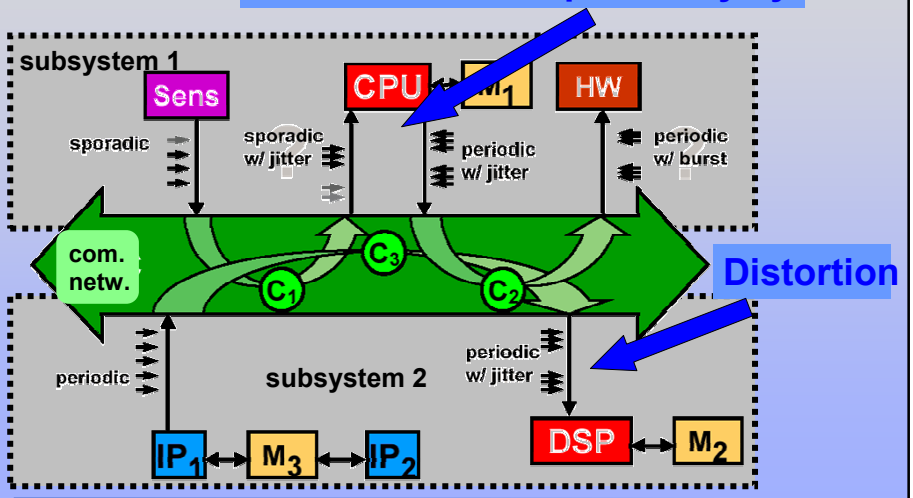


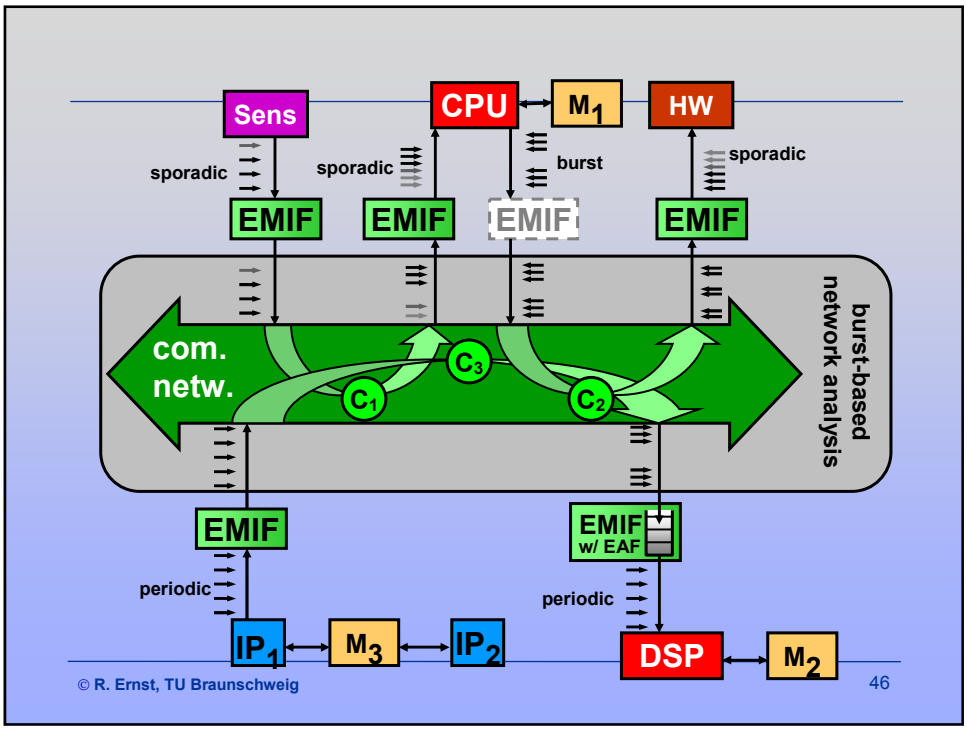
Using EMIFs and EAFs

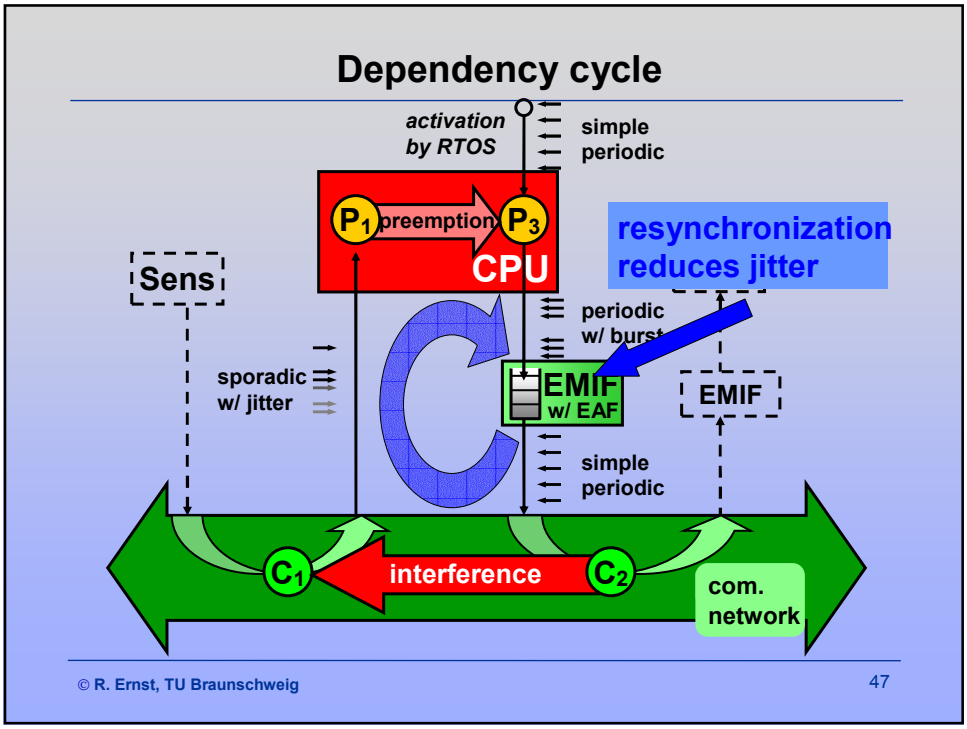


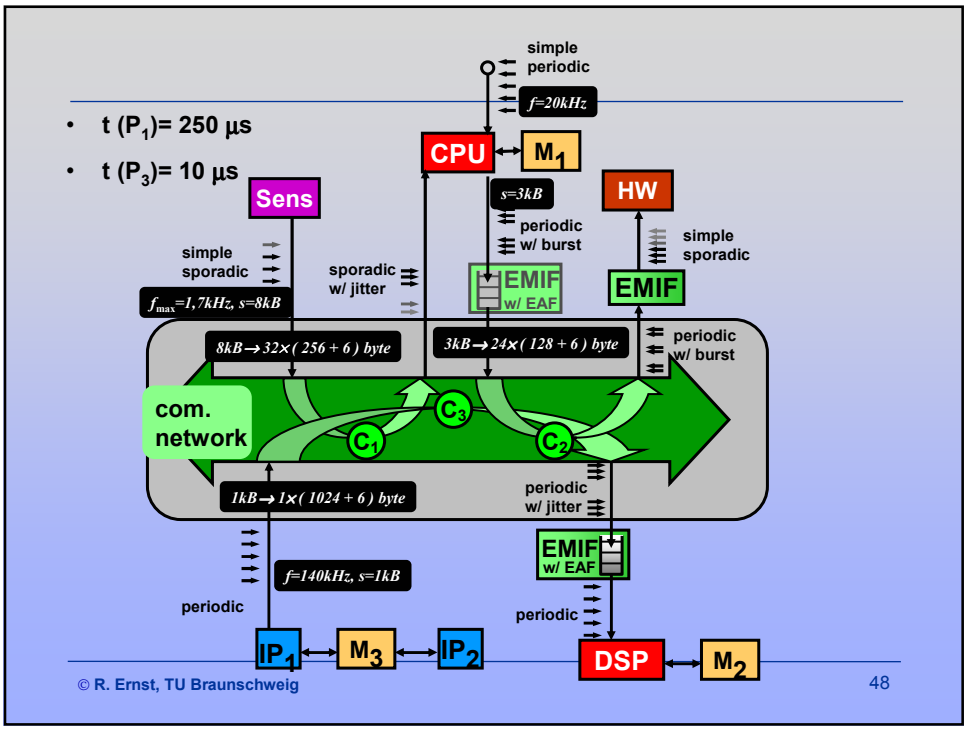
Example revisited

non-functional dependency cycle




Sens
CPU
M1
HW
sporadic
sporadic
burst
sporadic
EMIF
EMIF
EMIF
EMIF
com.
netw.
C3
C1
C2
burst-based
network analysis
EMIF
EMIF
w/ EAF
periodic
periodic
IP1
M3
IP2
DSP
M2
© R. Ernst, TU Braunschweig
46


Dependency cycle
activation
by RTOS
simple
periodic
P1
preemption
P3
CPU
resynchronization
reduces jitter
Sens
periodic
w/ burst
sporadic
w/ jitter
EMIF
w/ EAF
EMIF
simple
periodic
C1
interference
C2
com.
network
© R. Ernst, TU Braunschweig
47

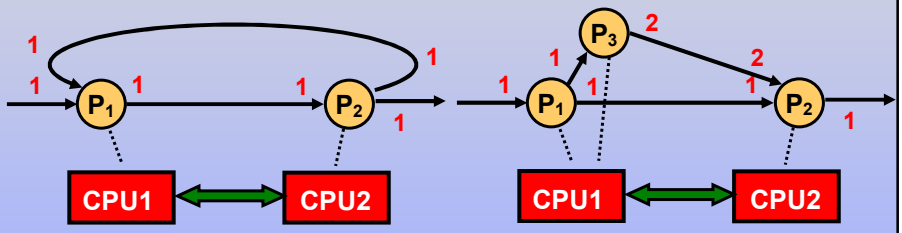


Results

network speed [MByte/s]	network util [%]	buffer size [kB]	C_1 output jitter [μs]	C_2 input jitter [μs]	C_1 worst-case resp. [μs]
480	46,41	36	17,45	—	34,95
300	74,26	36	69,46	—	97,41
240	92,82	39	256,29	—	291,22
480	46,41	—	85,85	265	103,35
300	74,26	—	276,13	275	304,08
250	89,11	—	> 987,82	> 515	> 1ms

Application model compatibility

- stream model represents exact data volume
  - keeps causality for application model activation
  - required for „AND“ activation, e.g. data flow graphs otherwise infinite buffers or starvation

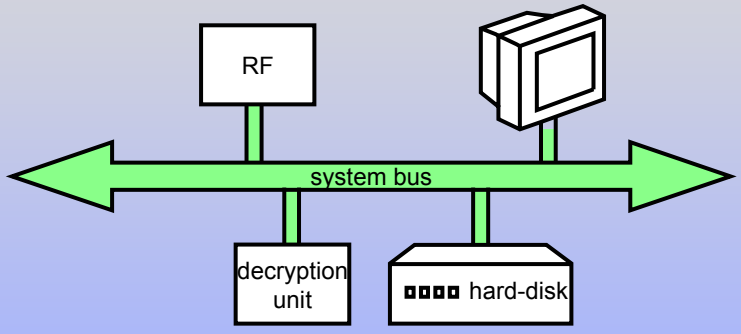


  - not needed for process systems with „OR“ activation or time driven activation

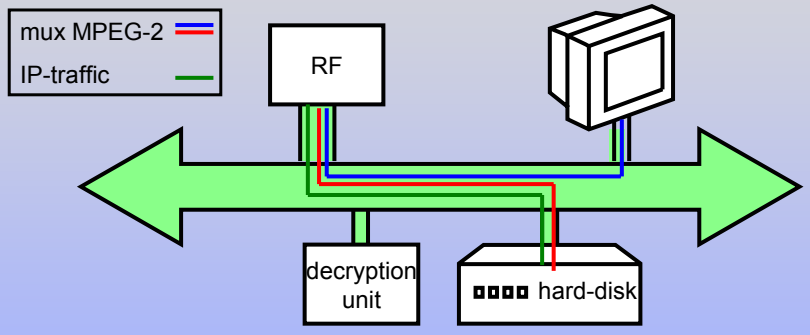
System contexts

- another stream property – partial event order is kept
- application: tagged token (SPI model) to take system contexts into account
- example set top box

Context exploitation: Set top box



Context exploitation: Set top box



- **scenario 1: record video + watch movie + download file via IP**

MPEG stream modeling

- stream properties
  - I, B, P frame types of different data volume
  - encoding with the following constraint on frame types

Min 2 Ⓘ out of 12
Min 2 Ⓟ out of 12
Min 6 Ⓑ out of 12
Max 4 Ⓘ out of 12
Max 4 Ⓟ out of 12
Max 8 Ⓑ out of 12

- stream coding
  - set of constraints on the sequence of tagged tokens

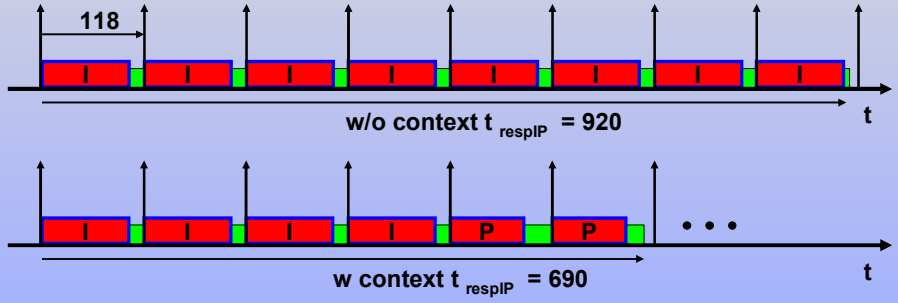
Event stream contexts

- intra event stream context
  - data volume and execution time depend on frame types and frame sequence
- inter event stream context
  - bus load and response times depend on relative phase of streams
- inter + intra event stream contexts
  - merged streams on bus have variable order
- solutions available for cyclo static streams (Chen/Mok) and inter event stream contexts only (Tindell), extended to constrained sequences and combinations

Response time impact

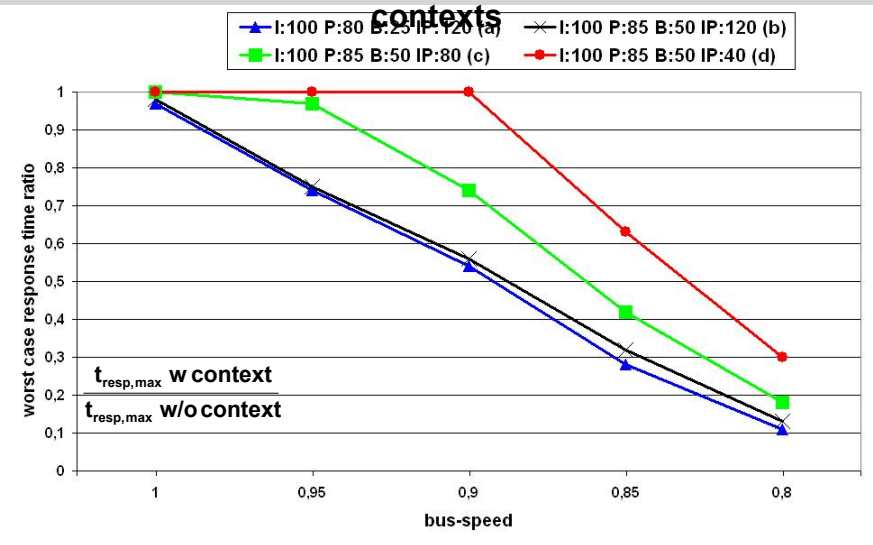
mux
IP

video streams are multiplexed and have higher priority than IP

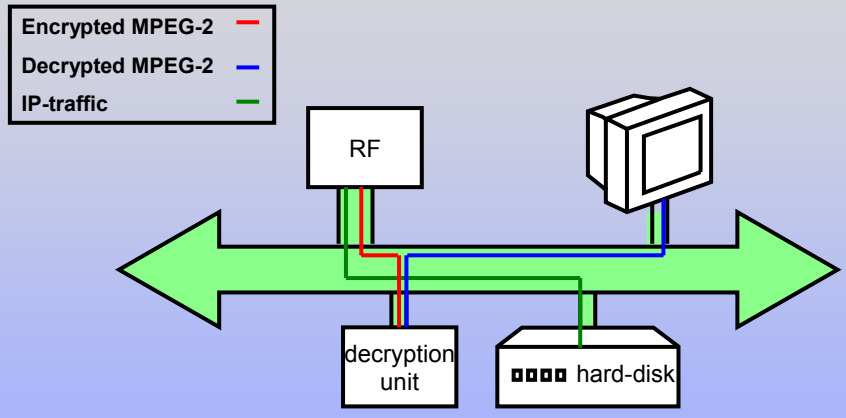


- response times for static priority bus arbitration

Analysis improvement due to intra event stream contexts

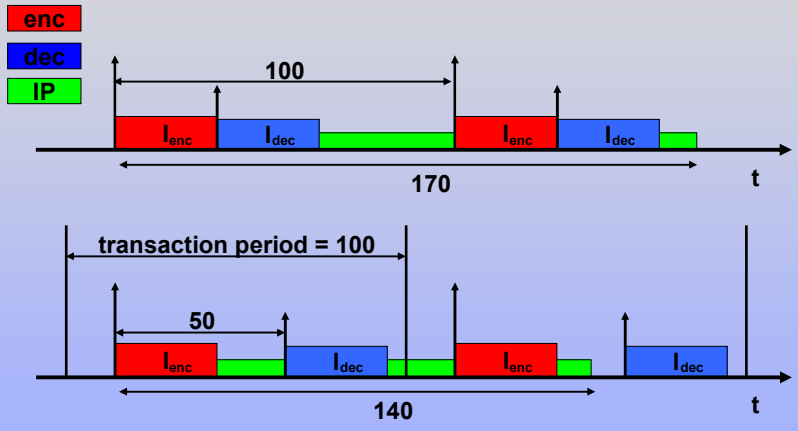


Set top box scenario 2



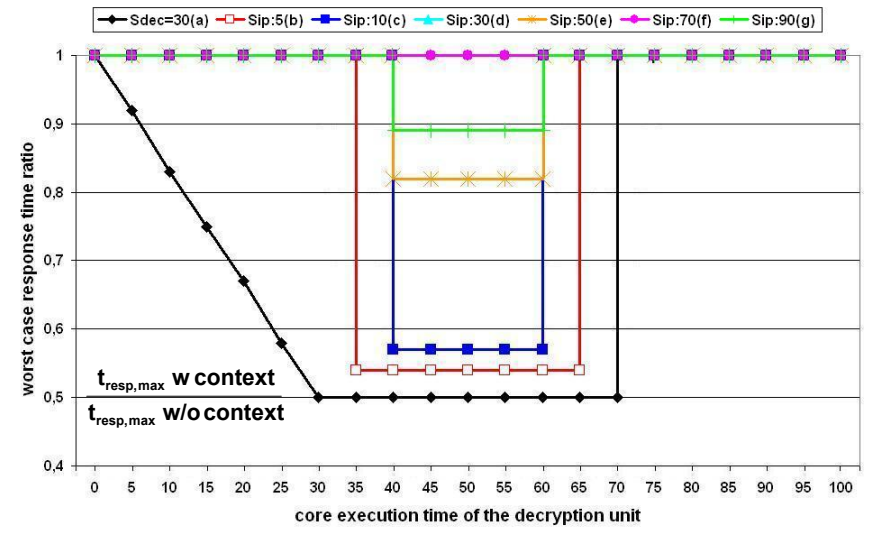
- scenario 2: decript video + download file via IP

Response time impact

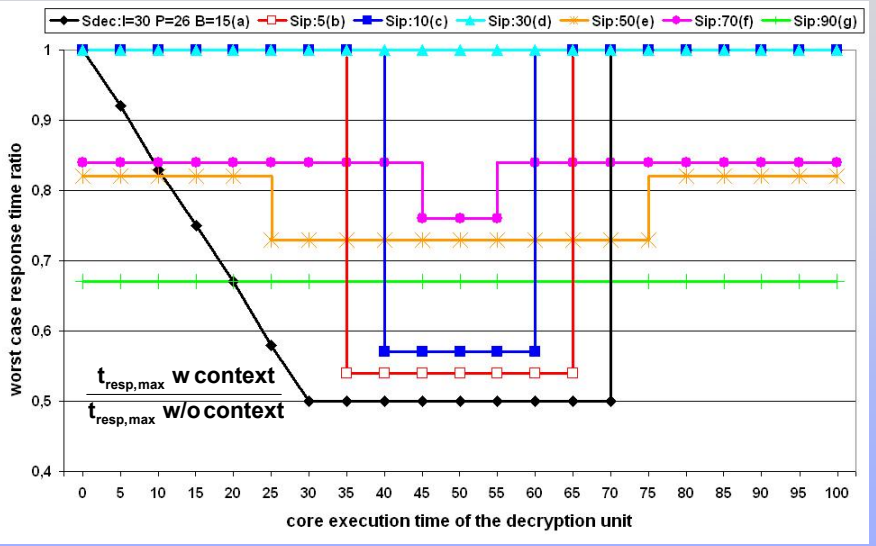


- enc and dec streams are coupled by decryption unit execution time

Analysis improvement due to inter event stream contexts



Analysis improvement due to both intra and inter event stream contexts



Acknowledgement

- The following persons made major contributions to this work
  - Jörn Braam
  - J. Bhavani
  - Rafik Henia
  - Marek Jersak
  - Razvan Racu
  - Kai Richter

Conclusion

- systems integration is key ES design problem
- performance verification is key integration problem
- many hidden performance problems not reflected in system function
- performance verification currently primarily based on simulation – risky and time consuming
- presented compositional analysis based on abstract event flow models
- event flow models enable analysis of complex situations with feedback and contexts
- work applied in several ongoing industry cooperations (SpeAC, FlexFilm, automotive SW, …)

Literature

- see: www.spi-project.org