

# Timing analysis and timing predictability

## Architectural Dependences

Reinhard Wilhelm

Saarland University, Saarbrücken, Germany

ArtistDesign Summer School in China 2010



# What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

# What does the execution time depends on?

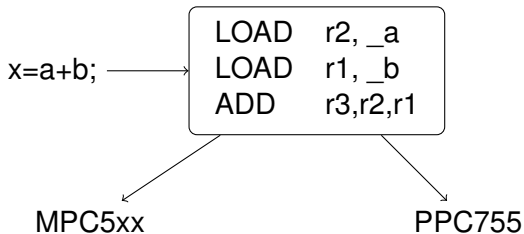
- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - ▶ Caused by caches, pipelines, speculation, etc.
  
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

# What does the execution time depends on?

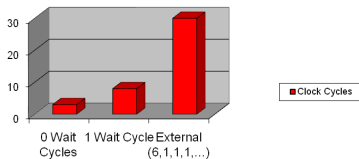
- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - ▶ Caused by caches, pipelines, speculation, etc.
  - ▶ Explosion of the space of inputs **and** initial states  
⇒ all exhaustive approaches infeasible
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

# What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - ▶ Caused by caches, pipelines, speculation, etc.
  - ▶ Explosion of the space of inputs **and** initial states  
⇒ all exhaustive approaches infeasible
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)
  - ▶ External interferences as seen from analyzed task

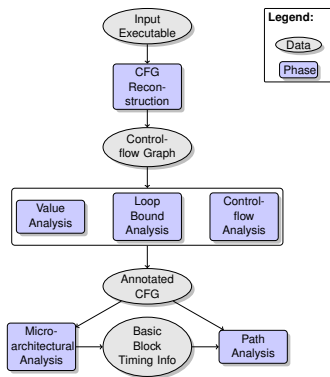


Execution Time depending on Flash Memory  
(Clock Cycles)



Execution Time (Clock Cycles)





## ■ Value Analysis:

- ▶ determines enclosing intervals for the values in registers and local variables

## ■ Loop Bound analysis:

- ▶ determines loop bounds

## ■ Control Flow Analysis:

- ▶ determines infeasible paths

## ■ Micro-architectural Analysis:

- ▶ combined cache and pipeline analysis
- ▶ derives invariants about architectural execution states, computes bounds on execution times of basic blocks

## ■ Global Bound Analysis:

- ▶ determines a worst-case path and an upper bound

- Predictability: not a boolean property
- Some performance-enhancing features, like certain
  - ▶ Caches
  - ▶ Pipelinesare analyzable, others are not...
- Explore trade-offs between (worst-case )predictability, (average-case) performance, and costs
- Goal:  
Design architectures with high worst-case performance that can be precisely and efficiently determined

- Predictable cores are a prerequisite for predictable multi-cores
- The main culprit: *Sharing* of resources
  - ▶ Main memory, caches
  - ▶ Busses
  - ▶ I/O
  - ▶ Flash memory
- introduces variability of execution times and, thus, imprecision,
- increases the state space to analyze and, thus, the complexity.

- 1 Pipeline Analysis
- 2 Predictability of architectures
  - Timing anomalies and domino effects
  - Classification of architectures
- 3 Extension to multi-core
  - Why multi-core?
  - Predictable multi-core
  - Analysis of current multi-core

## 1 Pipeline Analysis

## 2 Predictability of architectures

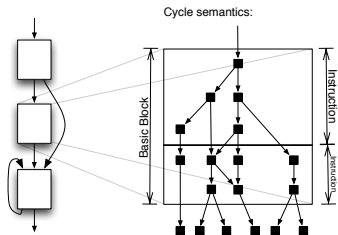
- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- Analysis of current multi-core

# Pipeline analysis

## Cyclewise evolution of processor model



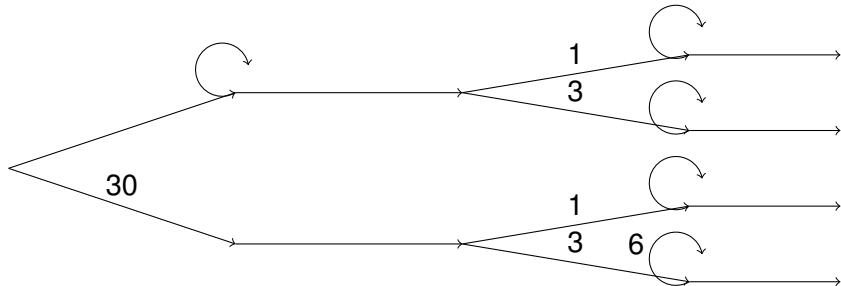
### Pipeline analysis:

- simulates the concrete pipeline on abstract states
- counts the number of steps until an instruction retires
- non-determinism results from abstraction – more non-determinism from "stronger" abstractions
- timing anomalies require exhaustive exploration of paths.

# (Concrete) Instruction Execution

instruction MUL

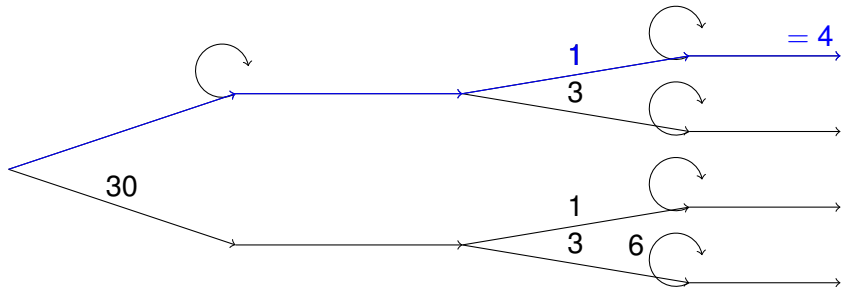
Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



# (Concrete) Instruction Execution

instruction MUL

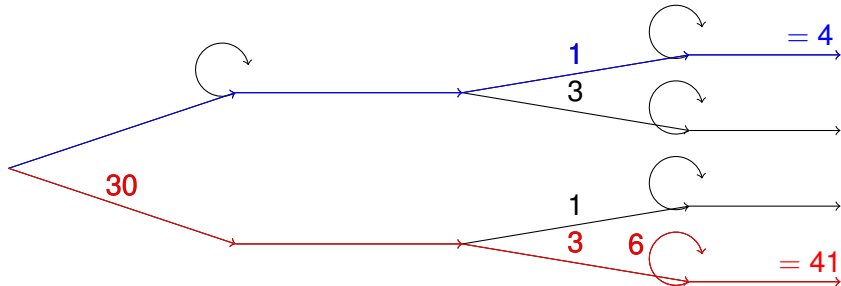
Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



# (Concrete) Instruction Execution

instruction MUL

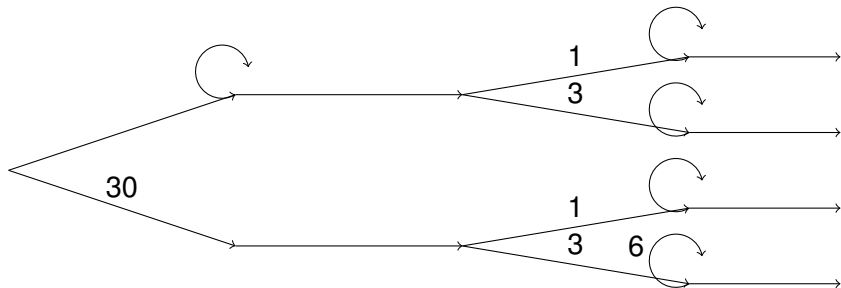
Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



# (Abstract) Instruction Execution

instruction MUL

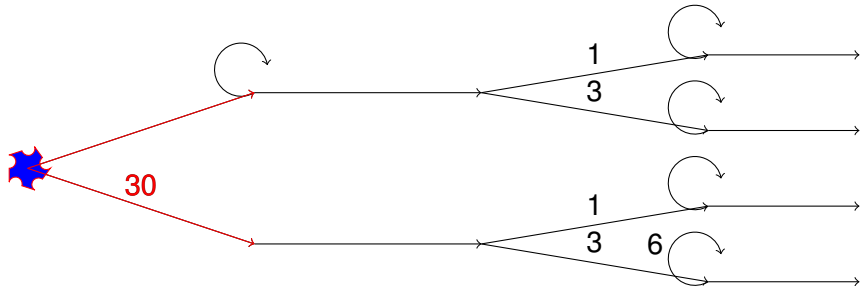
Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



# (Abstract) Instruction Execution

instruction MUL

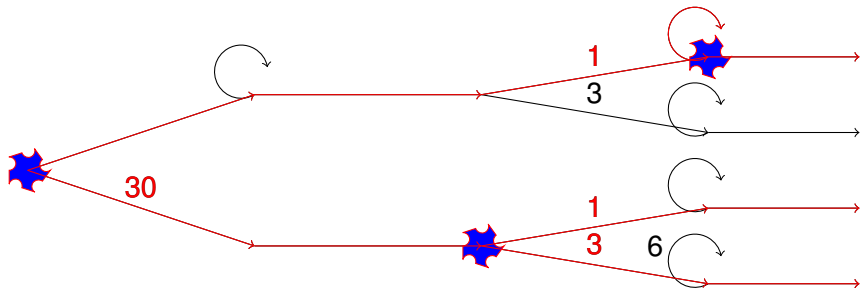
Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



# (Abstract) Instruction Execution

instruction MUL

Fetch	Issue	Execute	Retire
I-cache miss?	Unit occupied?	Multicycle?	Pending instructions?



*function exec (b: basic block, s: concrete pipeline state) t: trace*

- Interprets instruction stream of  $b$  starting in state  $s$  producing trace  $t$
- Successor basic block is interpreted starting in initial state  $last(t)$
- $length(t)$ : gives number of cycles

# An Abstract Pipeline Executing a Basic Block

*function exec (b: basic block, s: abstract pipeline state) t: trace*

- Interprets instruction stream of b  
(annotated with cache information)  
starting in state s producing trace t
- *length*(t): gives number of cycles

*function exec (b: basic block, s: abstract pipeline state) t: trace*

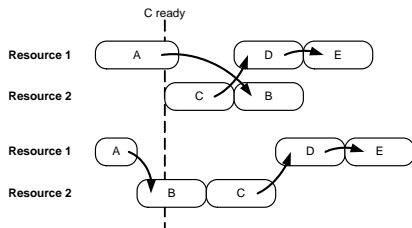
- Interprets instruction stream of  $b$   
(annotated with cache information)  
starting in state  $s$  producing trace  $t$
- $length(t)$ : gives number of cycles
- What are the differences?
  - ▶ Abstract states may lack information, e.g. about cache contents
  - ▶ Traces may be longer (but never shorter)

*function exec (b: basic block, s: abstract pipeline state) t: trace*

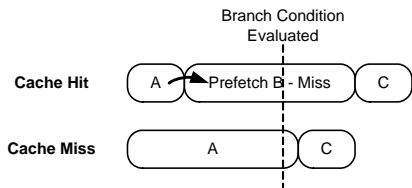
- Interprets instruction stream of  $b$   
(annotated with cache information)  
starting in state  $s$  producing trace  $t$
- $length(t)$ : gives number of cycles
- What are the differences?
  - ▶ Abstract states may lack information, e.g. about cache contents
  - ▶ Traces may be longer (but never shorter)
  - ▶ Starting state for successor basic block?
    - ★ sets of states
    - ★ combine by least upper bound (join)

## Timing anomalies

- When local worst-case does not lead to the global worst-case



Scheduling anomaly.



Speculation anomaly.

- Assuming local best case leads to higher overall execution time
- Assuming local worst case leads to shorter overall execution time
  - ▶ Cache miss in the context of branch prediction
- Treating components in isolation may be unsafe
- Implicit assumptions are not always correct:
  - ▶ Cache miss is not always the worst case!
  - ▶ The empty cache is not always the worst-case start!

# An Abstract Pipeline Executing a Basic Block

Processor with timing anomalies

*function analyze ( $b$ : basic block,  $\underline{S}$ : analysis state)  $\underline{T}$ : trace*

*Analysis states =  $2^{PS \times CS}$*

*PS = set of abstract pipeline states*

*CS = set of abstract cache states*

- Interprets instruction stream of  $b$   
(annotated with cache information)  
starting in state  $\underline{S}$  producing set of traces  $\underline{T}$
- $\max(\text{length}(\underline{T}))$ : upper bound for execution time
- $\text{last}(\underline{T})$ : set of initial states for successor block
- Union for blocks with several predecessors

## ■ Abstract Domain of Pipeline Analysis

- ▶ Power set domain
  - ★ Elements: sets of states of a state machine
- ▶ Join: set union

## ■ Pipeline Analysis

- ▶ Manipulate sets of states of a state machine
- ▶ Store sets of states to detect fixpoint
- ▶ Forward state traversal
- ▶ Exhaustively explore non-deterministic choices due to timing anomalies

# An Example Sophisticated Processor

MPC7448

- Out-of-order execution but in-order completion
- 3 levels of speculation
- 2 levels of caches
- Dynamic branch prediction
- Up to 16 instructions running in parallel
- Completion of up to 3 instructions + 1 branch per clock cycle
- Functional units:
  - ▶ Fetch and Branch Prediction Unit
  - ▶ Dispatch Unit
  - ▶ Bus Unit (BU)
  - ▶ Memory Controller Unit
  - ▶ Load/Store Unit (LSU)
  - ▶ Complex Fixed Point Unit (CFX)
  - ▶ 3 Simple Fixed Point Unit
  - ▶ Floating Point Unit

- LSU is a major source of splits, because of the queues:
  - ▶ every new memory instruction:  
check if conflict with any of the queued instructions
- Lsuq1: queues in LSU reduced to one slot
- Mälardalen benchmarks

# Splits	Total	LSU	CFX	BU
Standard	27496.33	61.35%	3.15%	35.05%
Lsuq1	6334.50	39.43%	24.38%	34.26%

## 1 Pipeline Analysis

## 2 Predictability of architectures

- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- Analysis of current multi-core

## 1 Pipeline Analysis

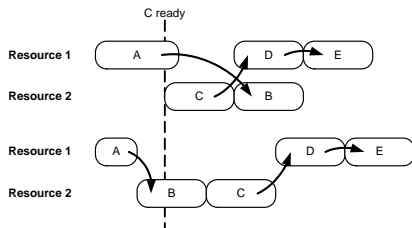
## 2 Predictability of architectures

- Timing anomalies and domino effects
- Classification of architectures

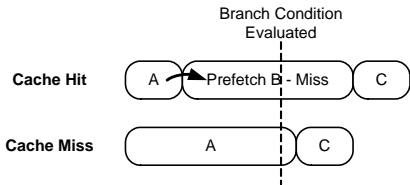
## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- Analysis of current multi-core

- When local worst-case does not lead to the global worst-case



Scheduling anomaly.



Speculation anomaly.

- One event triggers another one which triggers another one...
- Unbounded effect of a timing accident
- ⇒ Always analyse all cases
- ⇒ Particularly, there doesn't exist an upper-bound on the difference to the worst-case:
  - All possible delays to access a shared resource

## 1 Pipeline Analysis

## 2 Predictability of architectures

- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- Analysis of current multi-core

- *Timing compositional*
  - ▶ No timing anomalies
  - ▶ e. g., ARM7
- *Compositional with bounded effects*
  - ▶ Timing anomalies but no domino effects
  - ▶ e. g., TriCore (probably)
- *Non-compositional architectures*
  - ▶ Timing anomalies, domino effects
  - ▶ e. g., PPC 755

*from Wilhelm et al.: Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems, IEEE TCAD, July 2009*

- 1 Pipeline Analysis
- 2 Predictability of architectures
  - Timing anomalies and domino effects
  - Classification of architectures
- 3 Extension to multi-core
  - Why multi-core?
  - Predictable multi-core
  - Analysis of current multi-core

## 1 Pipeline Analysis

## 2 Predictability of architectures

- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- Analysis of current multi-core

- Applications
  - ▶ AUTOSAR
  - ▶ IMA
- Behavior of the system
  - ▶ Compositionality: Behavior of composed system derived from behavior of components
  - ▶ Composability: Behavior of components not changed by the composition
- composability of the resource behavior threatened by the interference on shared resources

## 1 Pipeline Analysis

## 2 Predictability of architectures

- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- **Predictable multi-core**
- Analysis of current multi-core

Minimise sharing in multi-processor architectures:

- Interferences might be huge (bus contention, cache pollution)
- Huge overestimation if analysis is possible at all
  - ▶ Set of tasks that might be executed in parallel
  - ▶ Cache contents

Minimise sharing in multi-processor architectures:

- Interferences might be huge (bus contention, cache pollution)
- Huge overestimation if analysis is possible at all
  - ▶ Set of tasks that might be executed in parallel
  - ▶ Cache contents

PROMPT (PRedictability Of Multi-Processor Timing)

- Start with a generic, parameterisable architecture with predictable (fully timing compositional) cores
- Instantiate architecture for given set of applications, based on their resource requirements

- Simplification of individual components
- Elimination of interferences on shared resources:
  - ▶ Wherever it is not absolutely needed
  - ▶ Private resources for private uses
  - ▶ Shared resource for global state

# Predictability of Multi-Core Architectures

Delays for accesses to the shared global state

- Bounding delay of a single access to shared resources
  - ▶ Deterministic bus access protocol
  - ▶ e.g. TDMA , Round-Robin [Paolieri et al., 2009]

## Delays for accesses to the shared global state

- Bounding delay of a single access to shared resources
  - ▶ Deterministic bus access protocol
  - ▶ e.g. TDMA , Round-Robin [Paolieri et al., 2009]
- Cumulative delay bound functions
  - ▶ Bound for a sequence or pattern of accesses
  - ▶ How?
    - ★ Lower- and upper-bound functions [Schranzhofer et al., 2009]
    - ★ Traces of cache accesses [Pellizzoni and Caccamo, 2007]

## Delays for accesses to the shared global state

- Bounding delay of a single access to shared resources
  - ▶ Deterministic bus access protocol
  - ▶ e.g. TDMA , Round-Robin [Paolieri et al., 2009]
- Cumulative delay bound functions
  - ▶ Bound for a sequence or pattern of accesses
  - ▶ How?
    - ★ Lower- and upper-bound functions [Schranzhofer et al., 2009]
    - ★ Traces of cache accesses [Pellizzoni and Caccamo, 2007]
- Optimization of Bus access protocol [Rosen et al., 2007]
  - ▶ Determine a TDMA access protocol
  - ▶ Fix-point iteration:
    - ★ Size of the TDMA slots
    - ★ Upper-bound on the delay – from the size of slots
    - ★ Scheduling analysis – from upper-bounds on the WCET of tasks

- single Fully timing compositional architectures
  - ▶ delay bounded by a constant:  
access to shared resources, preemptions
- single Disjoint instruction and data caches
- single Caches with LRU
- multi A shared bus protocol with bounded access delay
- multi Private caches
- multi Private memories, or, only share the lonely resources

## 1 Pipeline Analysis

## 2 Predictability of architectures

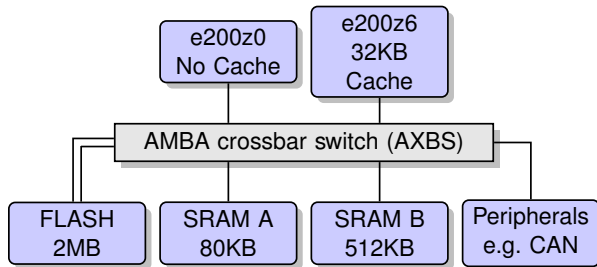
- Timing anomalies and domino effects
- Classification of architectures

## 3 Extension to multi-core

- Why multi-core?
- Predictable multi-core
- **Analysis of current multi-core**

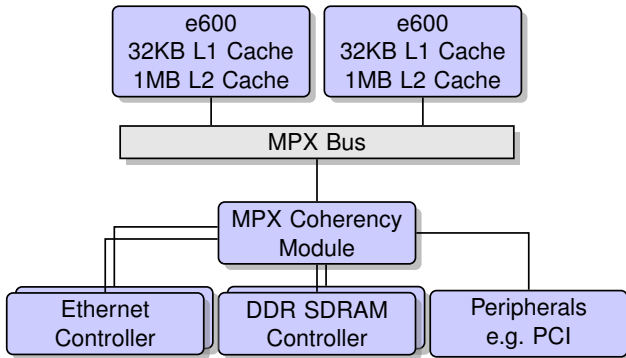
# Smart configuration of existing multi-core

MPC5668G - An automotive processor



# Smart configuration of existing multi-core


MPC8641D - An avionics processor




- Static timing analysis
  - ▶ Efficiency and precision
  - ▶ Strongly depends on the architecture
- Caches
  - ▶ predictability and sensitivity metrics
  - ▶ LRU is the most predictable policy
- Timing analysis of multi-core
  - ▶ Hard but possible
  - ▶ Predictable multi-core: less complexity and more precise results
- Recommendations for the design of multi-core
  - ▶ Predictable single-core
  - ▶ Sharing only if needed

# References


- C. Ferdinand et al.: Cache Behavior Prediction by Abstract Interpretation. *Science of Computer Programming* 35(2): 163-189 (1999)
- C. Ferdinand et al.: Reliable and Precise WCET Determination of a Real-Life Processor, *EMSOFT* 2001
- R. Heckmann et al.: The Influence of Processor Architecture on the Design and the Results of WCET Tools, *IEEE Proc. on Real-Time Systems*, July 2003
- St. Thesing et al.: An Abstract Interpretation-based Timing Validation of Hard Real-Time Avionics Software, *IPDS* 2003
- L. Thiele, R. Wilhelm: Design for Timing Predictability, *Real-Time Systems*, Dec. 2004
- R. Wilhelm: Determination of Execution Time Bounds, *Embedded Systems Handbook*, CRC Press, 2005
- St. Thesing: Modeling a System Controller for Timing Analysis, *EMSOFT* 2006
- J. Reineke et al.: Timing Predictability of Cache Replacement Policies, *Real-Time Systems*, Springer, 2007
- R. Wilhelm et al.: The Determination of Worst-Case Execution Times - Overview of the Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)* 7(3), 2008.
- R. Wilhelm et al.: Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems, *IEEE TCAD*, July 2009
- R. Wilhelm et al.: Designing Predictable Multicore Architectures for Avionics and Automotive Systems, *RePP Workshop*, Grenoble, Oct. 2009


 Paolieri, M., Quiñones, E., Cazorla, F. J., Bernat, G., and Valero, M. (2009).  
 Hardware support for wcet analysis of hard real-time multicore systems.


In *ISCA*, pages 57–68.


 Pellizzoni, R. and Caccamo, M. (2007).  
 Toward the predictable integration of real-time COTS based systems.

In *RTSS*, pages 73–82.


 Rosen, J., Andrei, A., Eles, P., and Peng, Z. (2007).  
 Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip.

In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, pages 49–60.


 Schranzhofer, A., Chen, J.-J., and Thiele, L. (2009).  
 Timing predictability on multi-processor systems with shared resources.