Predictable and composable multiprocessor systems for car-entertainment: breaking resource dependencies

***Marco Bekooij*, Rene van de Berg, and Kees Goossens**

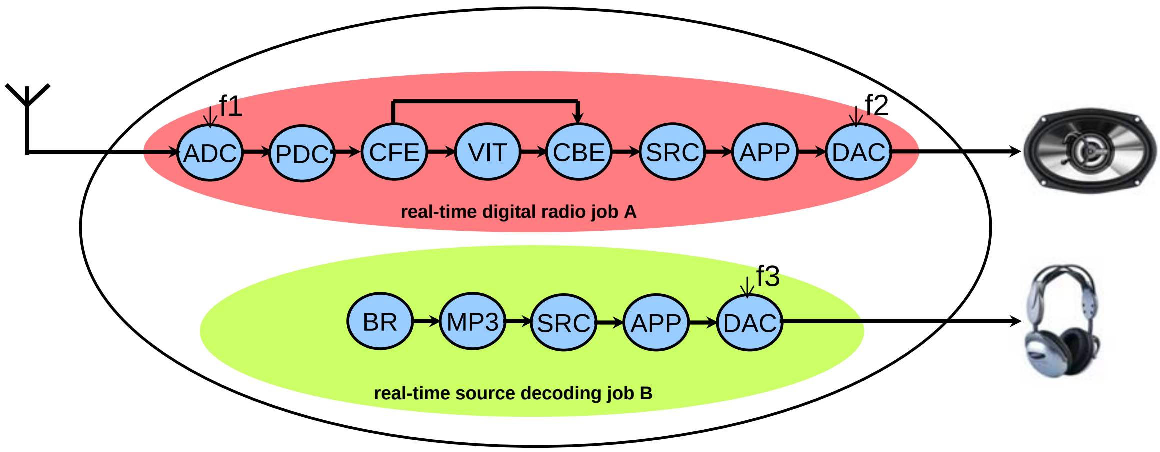
Outline

- Objectives
- Car entertainment
  - Application characteristics & system requirements
- Breaking resource dependencies
  - What resource dependencies are
  - Why we want to break them
  - How we break them
- Predictable and composable FPGA demonstrator

Our objectives:

1. Enable independent development of components
2. Maintain robustness despite increasing resource sharing
3. Reduce design and verification effort

Strategy: Divide and conquer

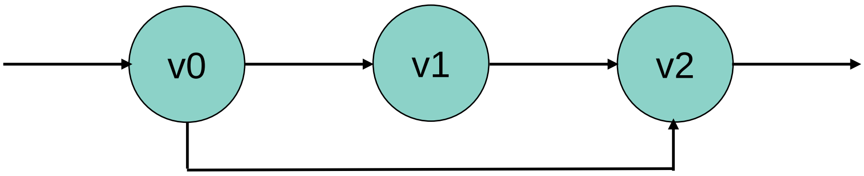


- Temporal isolation of applications (guaranteed by measures in hardware)
  - Each job can be designed and characterized independently of other jobs
  - Erroneous behavior of a job can not affect behavior other jobs

Objective 1&2

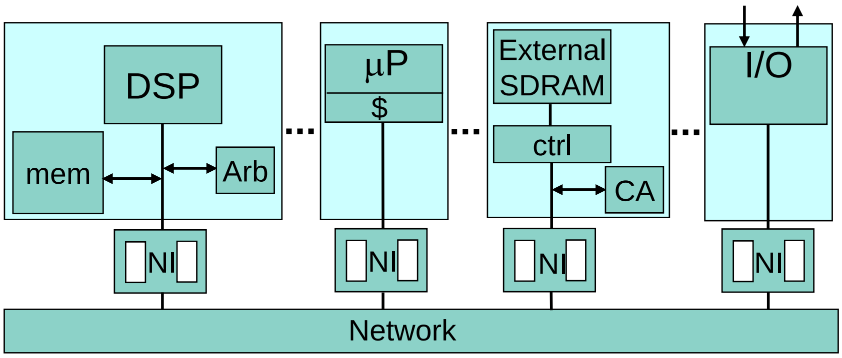
Strategy: Abstraction with guarantees (conservative arrival time data)

Fast (>100Mcc/s) simulation of communicating processes with time (CP-T)

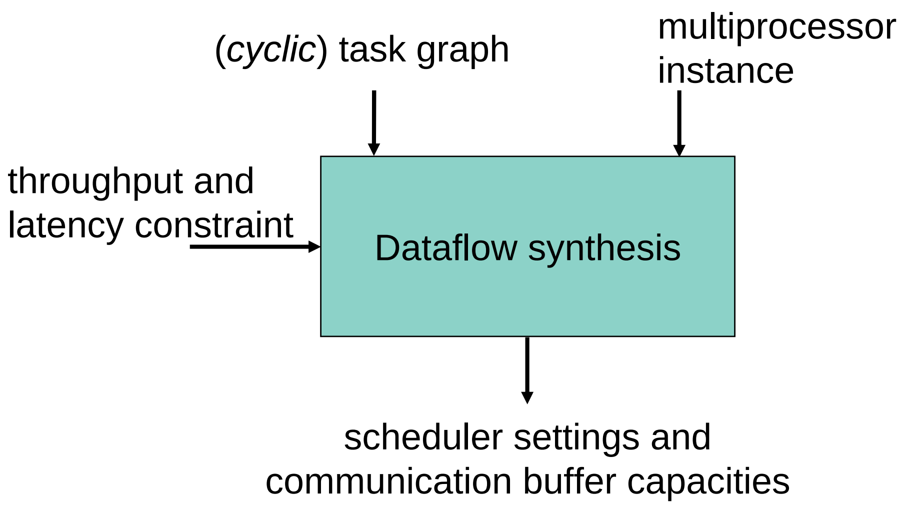


Instead of:

Low-level simulation (e.g. PV-T)



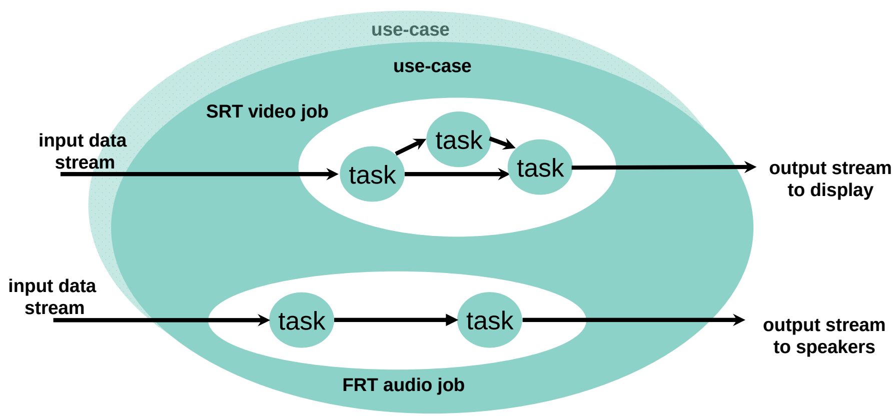
Strategy: Synthesize settings (prevent iterative design space exploration)



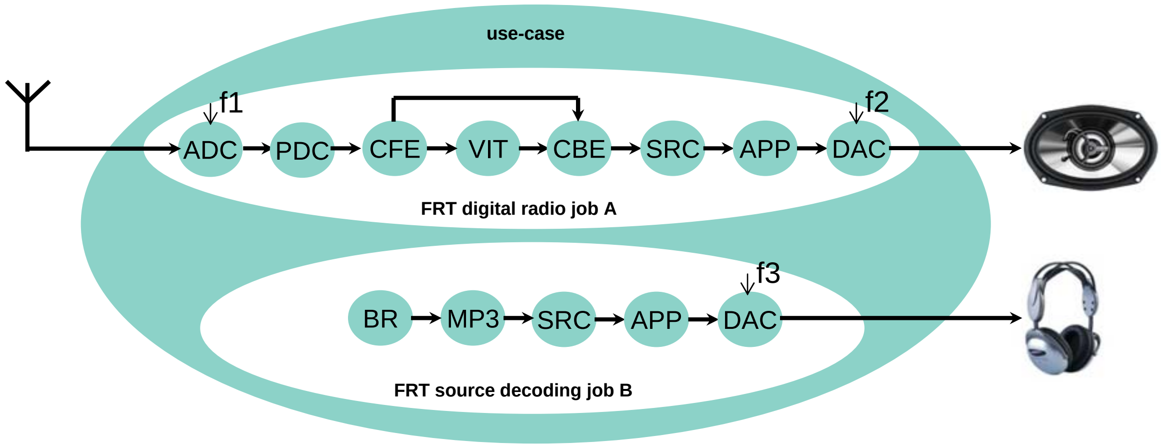
Car entertainment application domain

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*



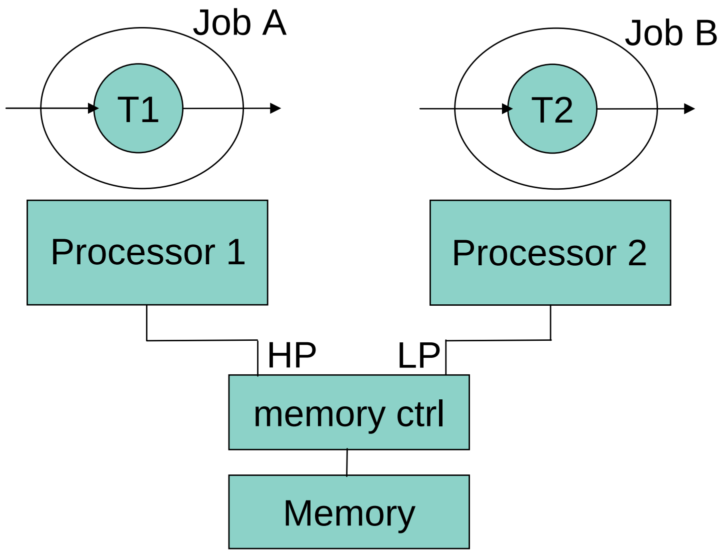
Car entertainment use-case



- 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

Resource dependencies

Example of a resource dependency



- Execution time of a task T2 is dependent on access pattern of processor 1.
  - Access pattern (traffic) is not known at design time!
- Execution time task T2 is unknown but determines throughput job B
- This type of resource dependency is independent of the priority assignment

Why do we want to break resource dependencies?

- Predictability = bounds on arrival time data
  - Analysis of minimum throughput and maximum latency of a job
  - Compute scheduler settings and buffer capacities given throughput and latency constraints of a job

- Composability = temporal isolation of jobs
  - Robustness (fault containment):
    - Prevent that a bug in a job can cause a complete system failure
  - Safely measure average performance:
    - Measure average throughput of a soft real-time job independently of other jobs
  - Security:
    - Prevent eavesdropping and withstand denial of service attacks

Common questions

- Composability ⇔ predictability?
  - No: predictable system = real-time system
    - Slack of other jobs results typically in higher throughput ⇒ not composable
  - No: composable system = virtual system
    - Temporal isolation alone does not guarantee arrival time data

- Is temporal isolation not a too strict requirement?
  - Earlier arrival data ⇒ higher quality?
  - What is an acceptable amount of interference?
    - how do we verify that there is always less interference?
  - How do we guarantee that race-conditions are not triggered by other jobs?

Breaking of resource dependencies

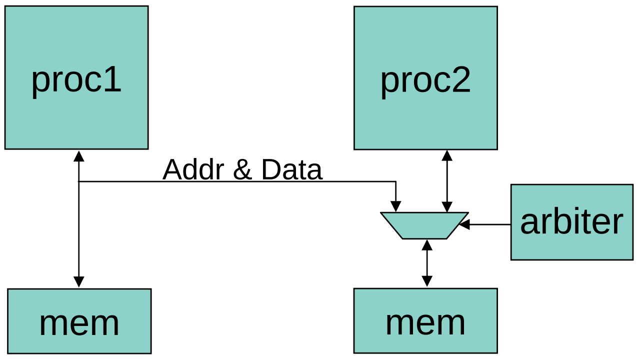
- Use only budget schedulers
  - Guaranteed cycle budget in a predefined interval of time (e.g. TDM, CBS)

- Budget is guaranteed, therefore it is independent of:
  - execution times of tasks
  - traffic in system
  - task model, e.g. data dependent input output behavior and execution rates

Flavors of budget schedulers

1. Work-conserving (slack is available for other jobs) :
   – Predictability
2. Non work-conserving (slack not available for other jobs):
   – Predictability & composability

Predictable memory port arbitration



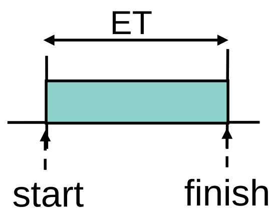
Credit based memory port arbiter (= a budget scheduler)
• Proc2 has 9 clock cycles out of 10 clock cycles priority

Maximum interference is bounded by construction

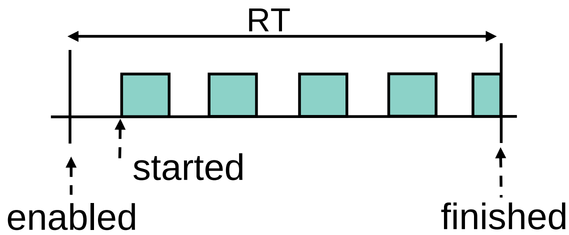
Dataflow analysis

Response time

Execution Time (ET)



Response Time (RT)

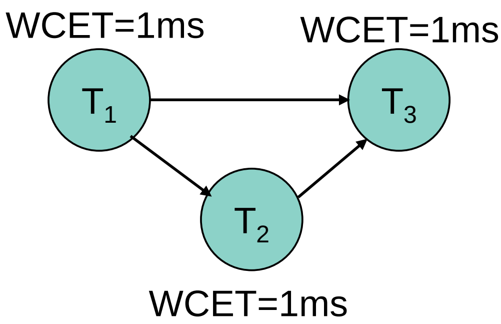


TDM: $RT = ET + (P - S)\lceil ET/S \rceil$

period time-slice

Simple throughput analysis example

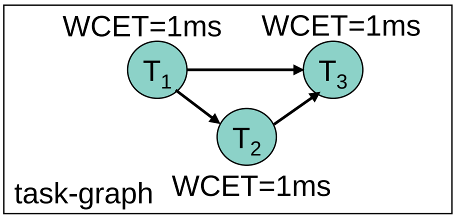
Task-graph



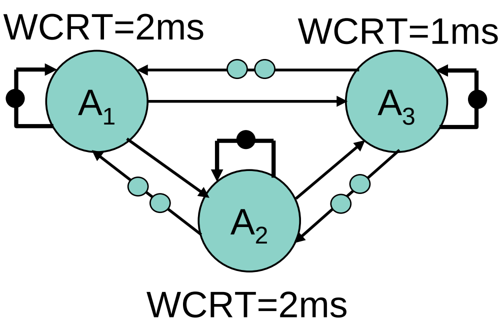
Assume:
- T_1 and T_2 share one processor, each task get a TDM-slice of 1 ms every 2 ms
- Each task produce and consume one token per execution
- Capacity of each buffer is 2 tokens

What is the minimum throughput?

Throughput analysis

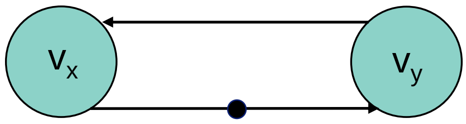


Dataflow graph



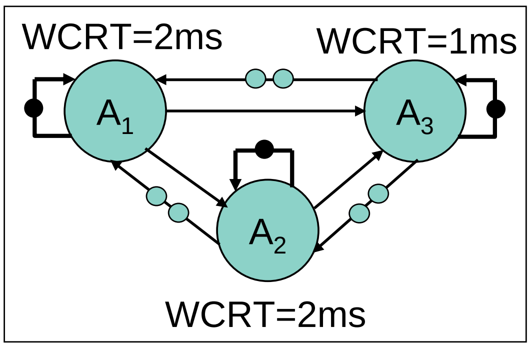
Monotonicity

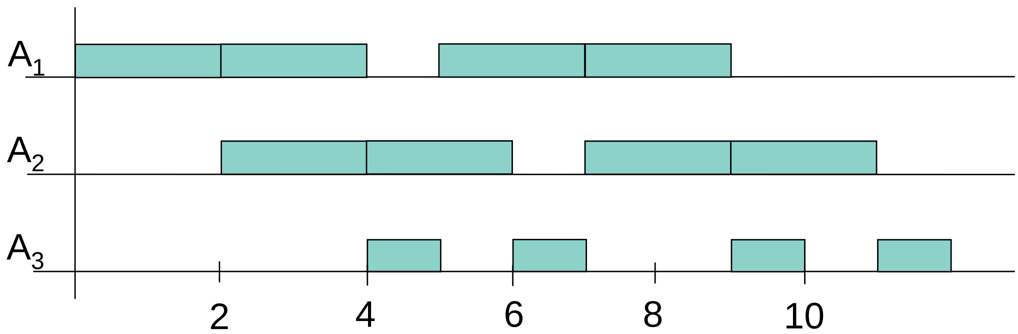
- Monotonic temporal behavior:
  - An earlier production of a token cannot result in a later start of an actor during self-timed execution
- Consequence:
  - Sufficient to show that a schedule exist that satisfies the throughput and latency constraints given worst-case response times
  - Smaller response 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_x and v_y

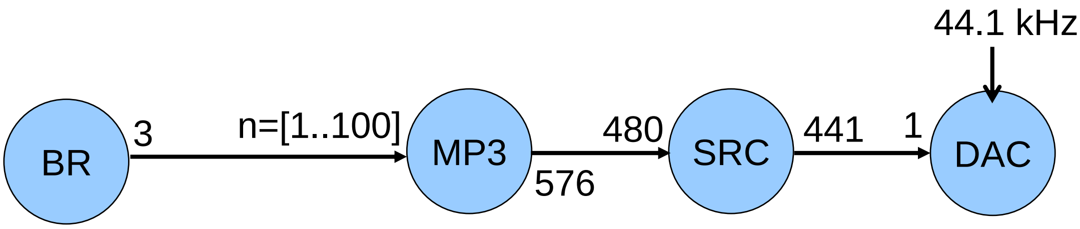
Valid schedule





1/Throughput = 2.5 ms/token

Cyclic data dependencies

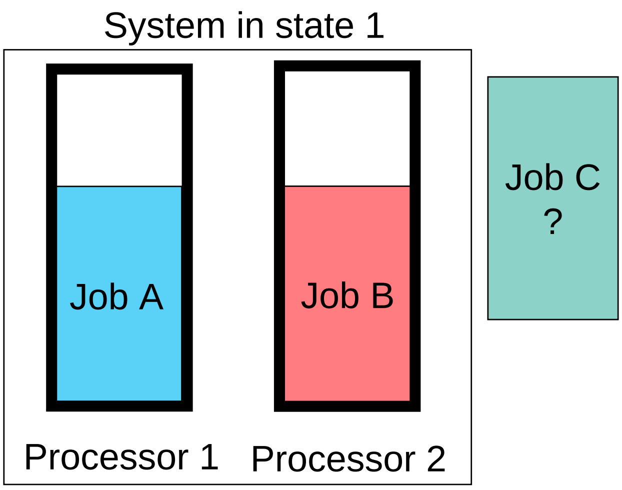


- Digital to analog converter (DAC) determines throughput constraint
- MP3 decoder task consumes each execution a different amount of data
  - No periodic schedule exist for the BR task!
- Block-reader (BR) task must “know” consumption speed MP3 task
  - Implies cyclic data dependency that affects the temporal behavior!

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

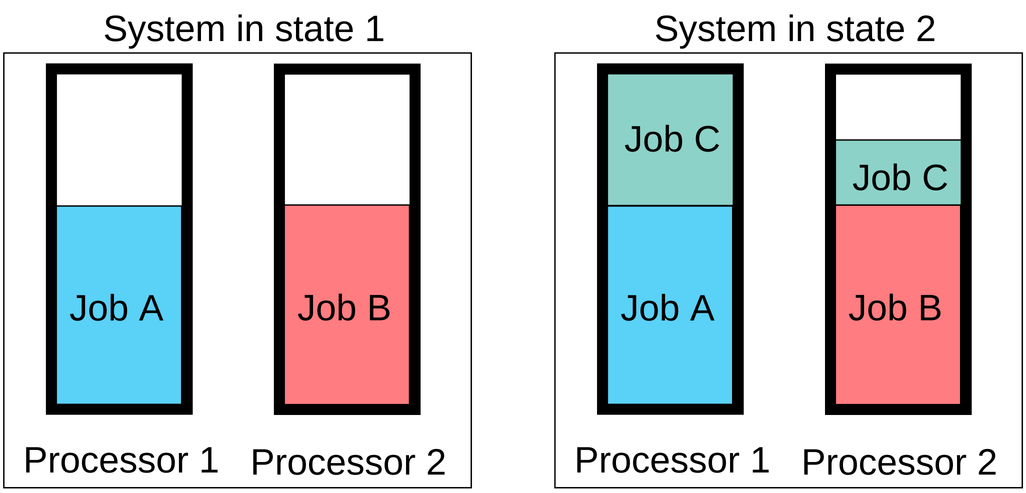
Temporal isolation

Temporal isolation of jobs (no processor sharing)



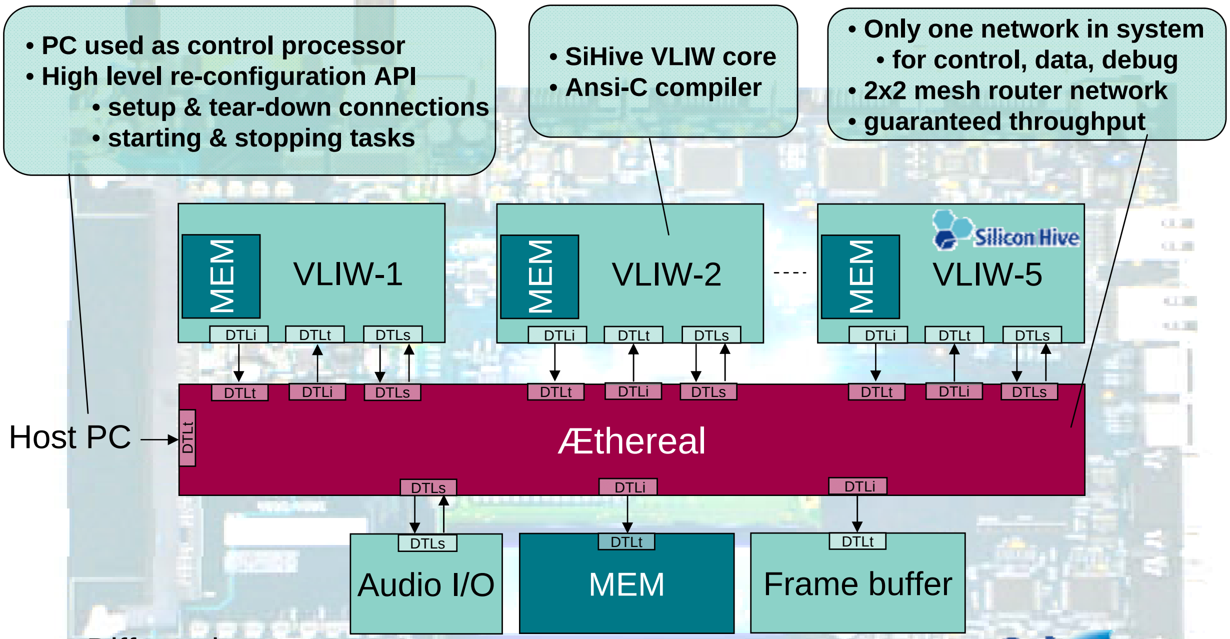
- Insufficient resources available to start job C
  - Check this with admission control

Temporal isolation with processor sharing



- Sufficient resources available to start job C
- Resources allotted to job A and job B remain unchanged
  - Undisruptive reconfiguration

FPGA demonstrator



Differentiators:

• Temporal isolation = behavior of a job cannot be affected by another jobs

• Predictable = upper-bound arrival data can be computed at design time

Summary

- We break resource dependencies because:
  1. Predictability:
     – compute settings given throughput and latency constraints of firm real-time jobs with cyclic dependencies
  2. Temporal isolation:
     – independent characterization of the temporal behavior of software jobs
     – robustness
- Budget schedulers break resource dependencies
  – Examples of budget schedulers are time division multiplex, and constant bandwidth server
- Cost of breaking dependencies:
  – Work conserving: different way of designing your system
  – Non-workconserving: waste slack created by tasks of other jobs
    • but lower cost than private hardware for each job

Questions?

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.
- M. Wiggers, M. Bekooij, and G. Smit. Computation of buffer capacities for thoughput constrained and data dependent inter-task communication. In *Proc. In Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, April 2008