Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration

Benny Åkesson
Technische Universiteit Eindhoven
The Netherlands

Liesbeth Steffens
NXP Semiconductors Research
The Netherlands

Eelke Strooisma
Delft University of Technology
The Netherlands

Kees Goossens
NXP Semiconductors Research &
Delft University of Technology
The Netherlands

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

NXP

Presentation Outline

Introduction

Service Models

CCSP Arbitration

Hardware Implementation

Experimental Results

Conclusions

Trends in MPSoC Design

- MPSoC design gets increasingly complex.
  - Moore's law allows increased component integration.
  - Digital convergence creates a market for highly integrated devices.

- The resulting MPSoCs
  - have a large number of IP components.
  - run many applications with both soft and hard real-time requirements.

MPSoC Constraints

- Resource sharing
  - is required to reduce cost,
  - but introduces interference between applications,
  - which makes it difficult to satisfy real-time requirements.

- Resource arbiter requires an implementation that
  - is small, for multiple instances to be used in the system.
  - reserves service without over allocating.
  - runs at high clock frequency to schedule on fine granularity.
    - reduces latency and buffers.

Application Requirements

- Hard real-time requestors
  - Example: Audio post processing IP
  - Request patterns are typically regular and predictable
  - Deadlines for individual requests are loose, but must always be satisfied
  - Require guaranteed minimum service rate and bounded maximum latency

- Soft real-time requestors
  - Example: Video decoding on cache-based processor
  - Often very bursty request patterns
  - Tight task-level deadlines (may span thousands of requests)
  - Occasional deadline misses acceptable
  - Require guaranteed minimum service rate and low average latency

Related Work

- Existing arbiters fail to satisfy requirements for three reasons:
  - Allocation granularity coupled to latency
    - All frame-based arbiters
  - Latency coupled to rate
    - Fair queuing family, weighted and deficit round-robin
  - Cannot run at high clock speed with small implementation
    - Sporadic server (complex accounting)
    - Constant bandwidth server (EDF scheduler needs complex priority queue)

Main Contributions

- We present a Credit-Controlled Static-Priority Arbiter
  - Comprised of a rate regulator and a static-priority scheduler
  - Resembles a (σ,ρ) regulator with static-priority scheduler

- Contributions
  - Regulator decouples allocation granularity from latency
  - Static-priority scheduler decouples latency from rate
  - Small implementation that runs at high speed
  - Regulates provided service as opposed to requested service

Presentation Outline

Introduction

Service Models

CCSP Arbitration

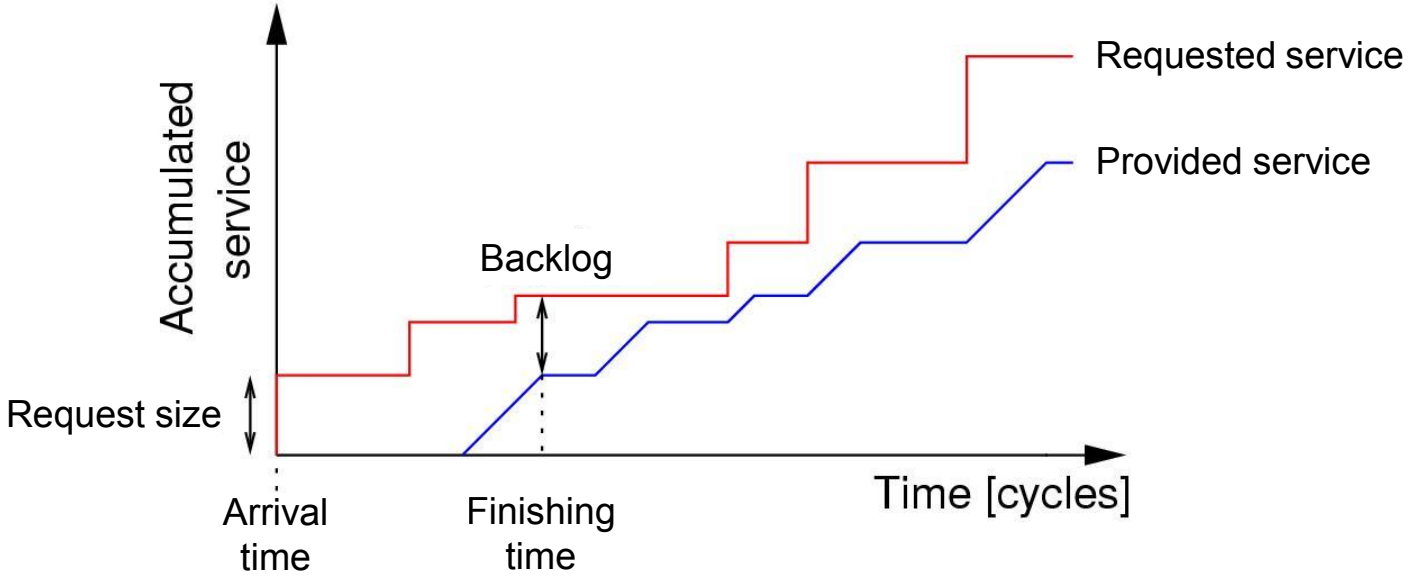
Hardware Implementation

Experimental Results

Conclusions

Service Curves

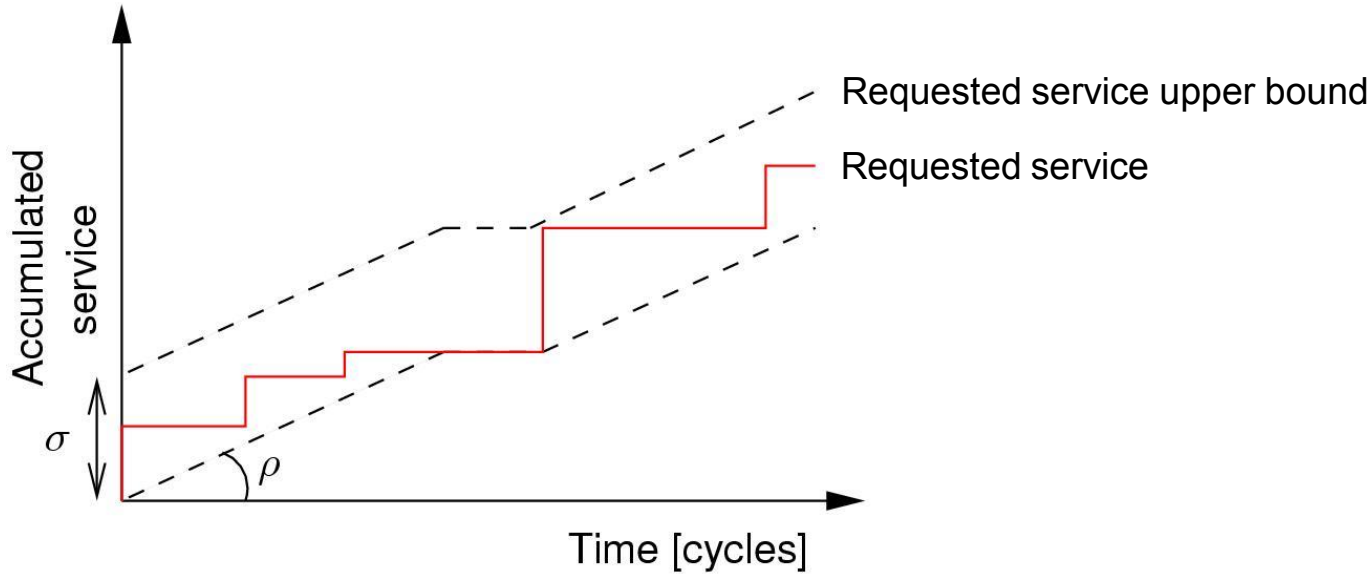
- Service curves model interaction between requestors and resource.
  - Service measured in service units, taking one service cycle to serve.



- We need bounds on service curves to work analytically.

Requested Service Model

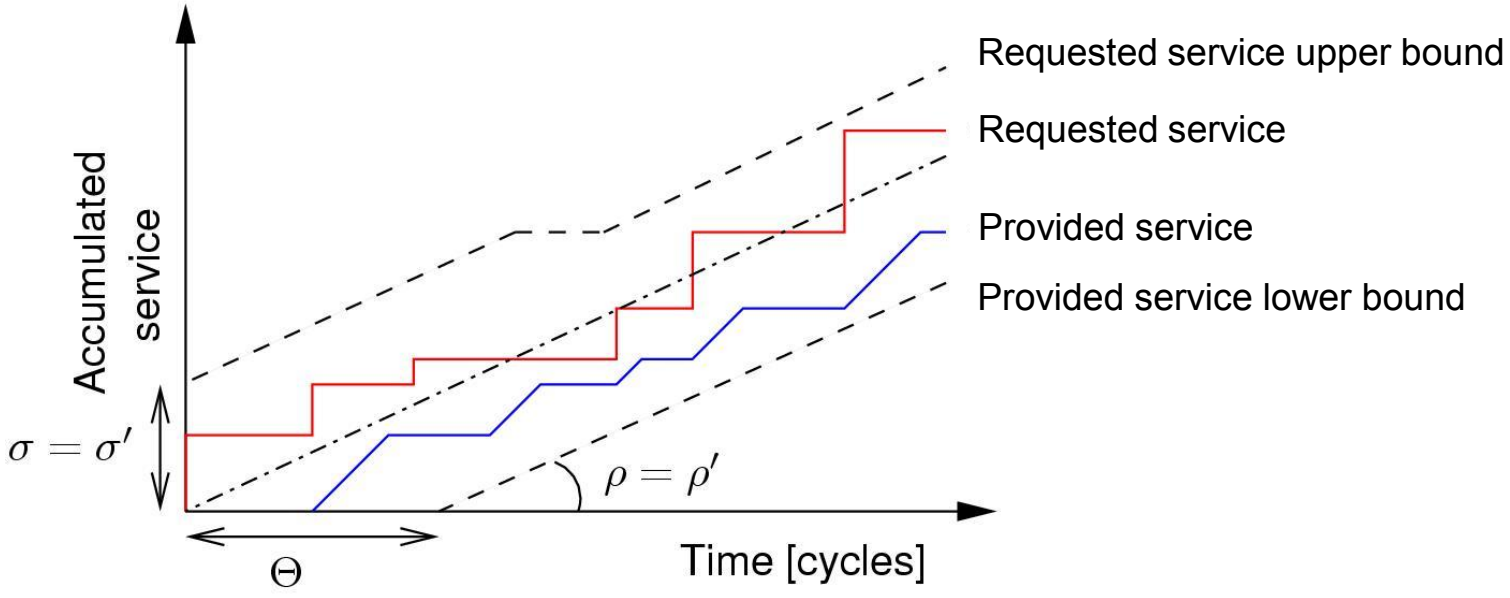
- We use the (σ,ρ) model [Cruz91] to upper bound requested service
- Requestors are assumed to be accurately characterized



Provided Service Model

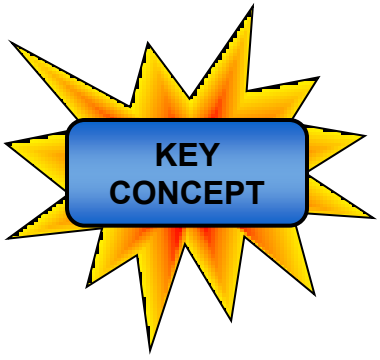
- Service is allocated to a requestor according to an allocated burstiness, σ', and an allocated service rate, ρ'.

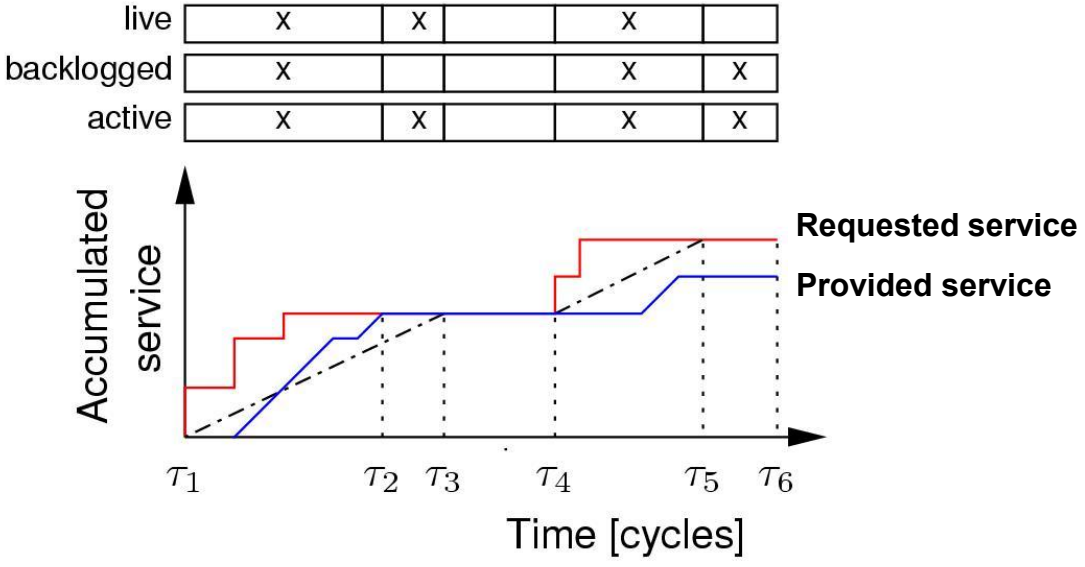
- Allocated service rate guaranteed to active requestor after service latency Θ.
  - Provides a lower bound on provided service.



Active Periods

- An active period of a requestor is the maximum interval in which it is backlogged or live.
- A requestor is live if it requested more service than allocated on average since start of active period.





Presentation Outline

Introduction

Service Models

CCSP Arbitration

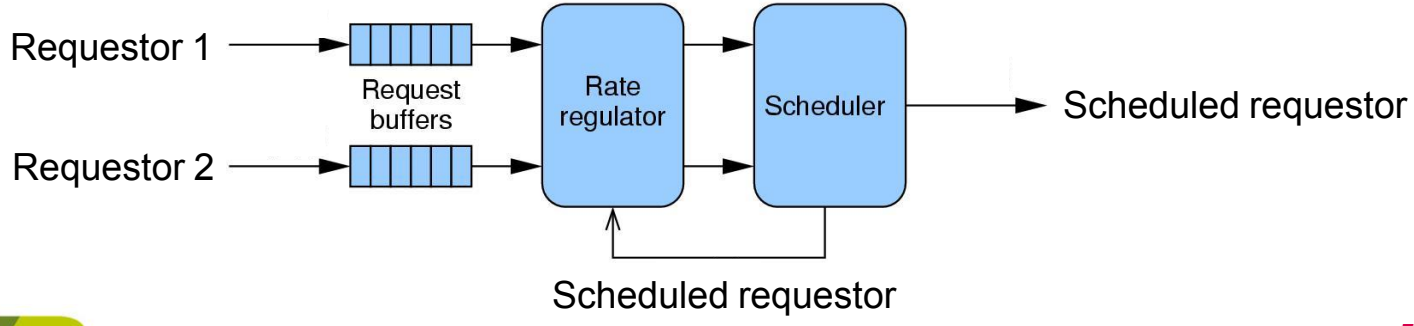
Hardware Implementation

Experimental Results

Conclusions

Credit-Controlled Static-Priority Arbitration

- Arbiter consists of a rate regulator and a static-priority scheduler
- Regulator enforces an upper bound on provided service
  - Enforcement required to provide latency bound
- Static-priority scheduler schedules highest priority requestor
- We consider a preemptive and non-work-conserving instance.



Benefits of Provided Service Regulation

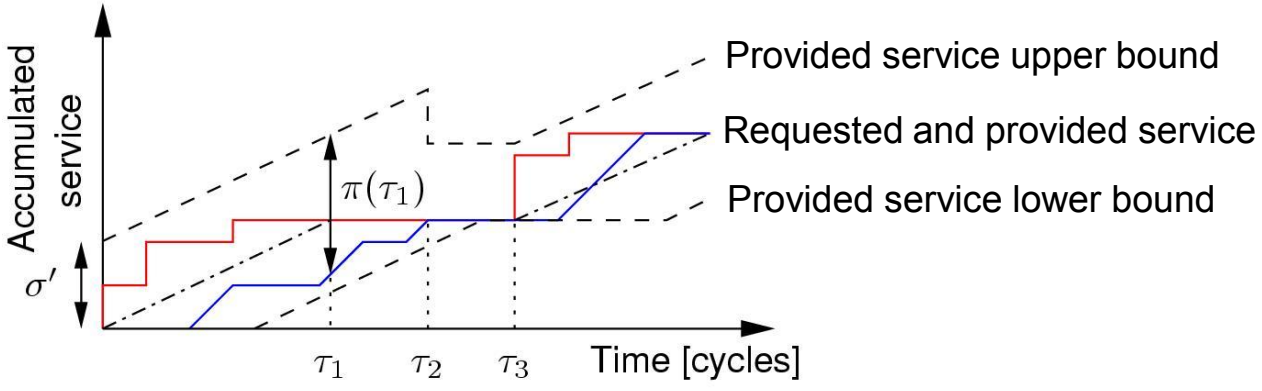
- Benefits of regulating provided service instead of requested service:

1. Implementation is less complex
   - Only aware of request at head of buffer (smaller state)

2. Size of request does not have to be known up front
   - Example: decoding time of a video frame / SDRAM access time
   - Requested service regulation needs worst-case assumptions on size
   - We charge one unit per cycle and preempt when budget is depleted

Accounting

- Accounting based on active period
  - Upper bound on provided service increased with ρ' for active requestor
  - Inactive requestor reset to current provided service + σ'
- Service curves go to infinity!
  - Represented as finite potential, π, in hardware
  - Potential = current provided service bound – current provided service
  - Requestor eligible if it has potential for at least a service unit, π(t) ≥ 1 – ρ'



Key Analytical Results

- Critical instance for a requestor happens when all higher priority requestors start active periods simultaneously
- Active requestor gets allocated rate, ρ', after service latency assuming σ' ≥ σ.
  - Same bound as for (σ,ρ) regulator with static-priority scheduler

$$\Theta = \frac{\sum_{i=0}^{p-1} \sigma'_i}{1 - \sum_{i=0}^{p-1} \rho'_i}$$

- CCSP belongs to the class of latency-rate servers.
  - Useful for both network calculus and data-flow analysis
- The finishing time of a request is derived.

Presentation Outline

Introduction

Service Models

CCSP Arbitration

Hardware Implementation

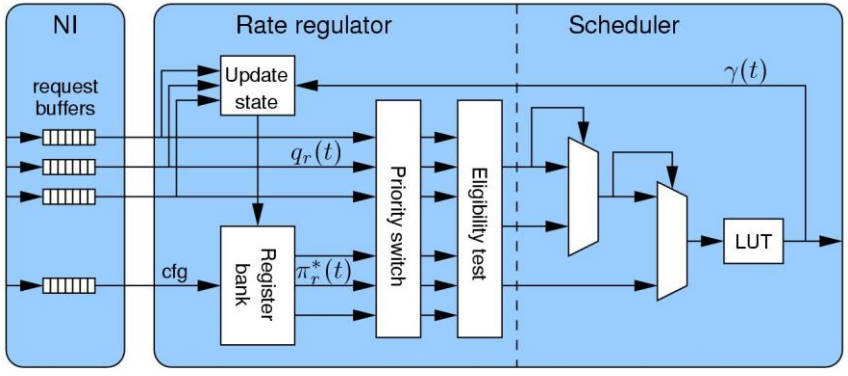
Experimental Results

Conclusions

Hardware Implementation

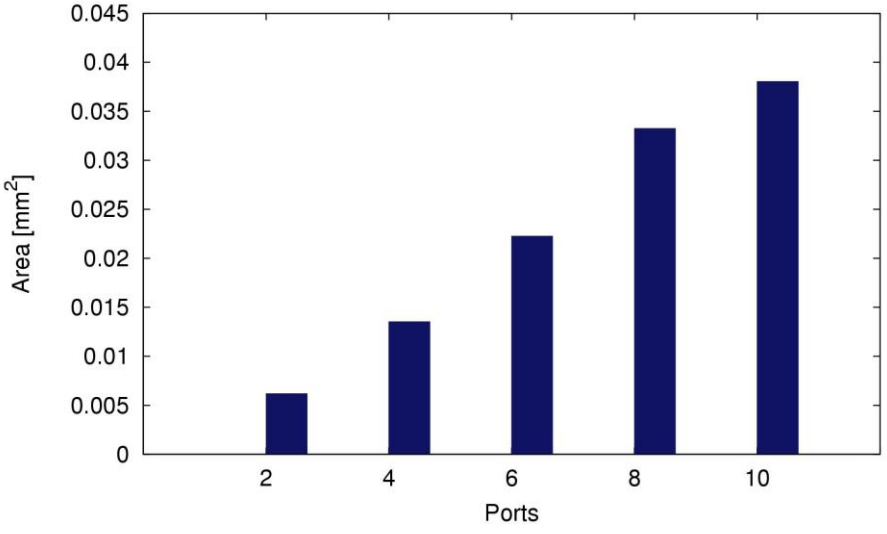
- Arbiter integrated into Predator SDRAM controller
  - Used in context of predictable MPSoC interconnected with Æthereal NoC

- Functional units:
  - Request buffers
  - Priority switch and look-up table (LUT) for configurable priorities
  - Logic performing eligibility test
  - Multiplexer tree implementing static-priority scheduler
  - Register bank storing potential and state machine that updates it



Synthesis Results

- Synthesis results
  - 90 nm CMOS process
  - Speed target of 200 MHz to serve as arbiter for a DDR2-400 memory
  - Instance with 6 ports requires 0.0223 mm2
  - Speed target met up to 10 ports – area scales linearly
  - Largest contributors to area are state registers



Presentation Outline

Introduction

Service Models

CCSP Arbitration

Hardware Implementation

Experimental Results

Conclusions

Use case – H.264 decoder

- Simulated SystemC models of memory controller and arbiter with H.264 use case executing on TriMedia 3270 processor.

- Soft real-time application consisting of
  - Read and write channels for TriMedia (TM_rd, TM_wr)
  - Display controller (DC)
  - File reader (FR)

- Two hard real-time periodic traffic generators (HRT_1, HRT_2)
  - Modeling e.g. pixel processing engines

Configuration

- Memory controller service unit is 64 B, taking about 80 ns to serve.
  - Total load is 90.7% of offered bandwidth (high load!)
- Priority assignment:
  - High priorities to soft real-time application for low average service latencies
  - Low priorities to hard real-time requestors
- Use case was simulated for 200 ms

Experimental results (1)

- Measured max cases lower than analytical bounds
  - Worst-case gets increasingly unlikely with lower priority
  - Worst-case characterizations cannot necessarily happen simultaneously

Requestor	σ’	ρ’	priority	avg. Θ	max Θ	Θ
TM_rd	8.0	0.106	0	3.19	9	N/A
TM_wr	4.0	0.061	1	8.60	18	N/A
DC	2.0	0.047	2	0.10	2	N/A
FR	4.4	0.017	3	55.67	63	N/A
HRT_1	4.4	0.340	4	0.17	**10**	**20**
HRT_2	3.4	0.340	5	2.23	**23**	**47**

Experimental results (2)

- Inverting all priorities to test tightness of analytical bound
  - Traffic generators create critical instance in beginning
  - Maximum measured values closer to bounds

Requestor	σ’	ρ’	priority	max Θ	Θ
HRT_2	3.4	0.340	0	**0**	**0**
HRT_1	4.4	0.340	1	**4**	**5**

- All simulation results are identical to (σ,ρ) regulator with static-priority scheduler, although CCSP has benefits of regulating provided service.

Presentation Outline

Introduction

Service Models

CCSP Arbiter

Hardware Implementation

Experimental Results

Conclusions

Conclusions

- We presented a Credit-Controlled Static-Priority Arbiter
  - consists of rate regulator and static-priority scheduler
- Regulator decouples allocation granularity from latency
  - No dependence on frame sizes etc.
- Static-priority scheduler decouples latency and rate using priorities
- Small implementation that runs at 200 MHz with up to 10 requestors
- Same results as a (σ,ρ) regulator with static-priority scheduler
  - Both analytically and during simulation.
- Regulates provided service as opposed to requested service
  - Implementation less complex
  - Size of request does not have to be known up front

Questions?

k.b.akesson@tue.nl