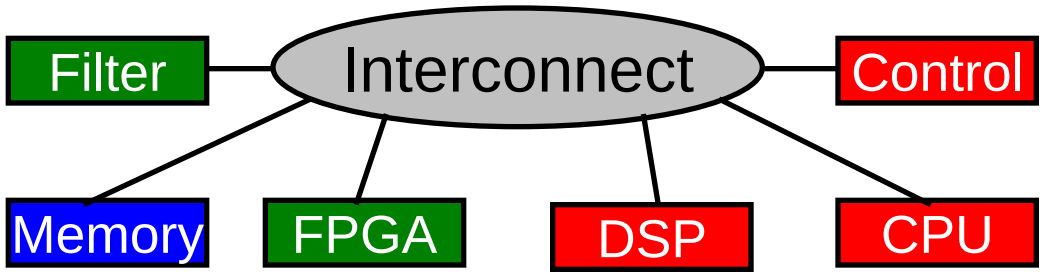
An analytical model for a memory controller offering hard real-time performance

Benny Åkesson

Introduction

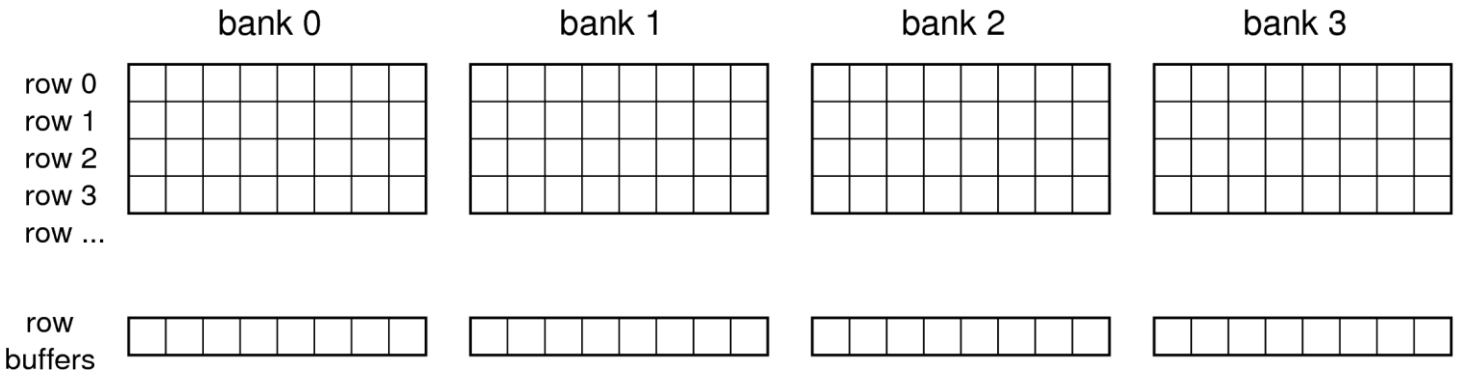
- The number of memory clients, *requestors*, grows in embedded systems

- Diversity in memory requirements with regard to bandwidth and latency
  - CPU, DSP (low average latency)
  - Filter (minimal guaranteed bandwidth)
  - Control system (low worst-case latency)



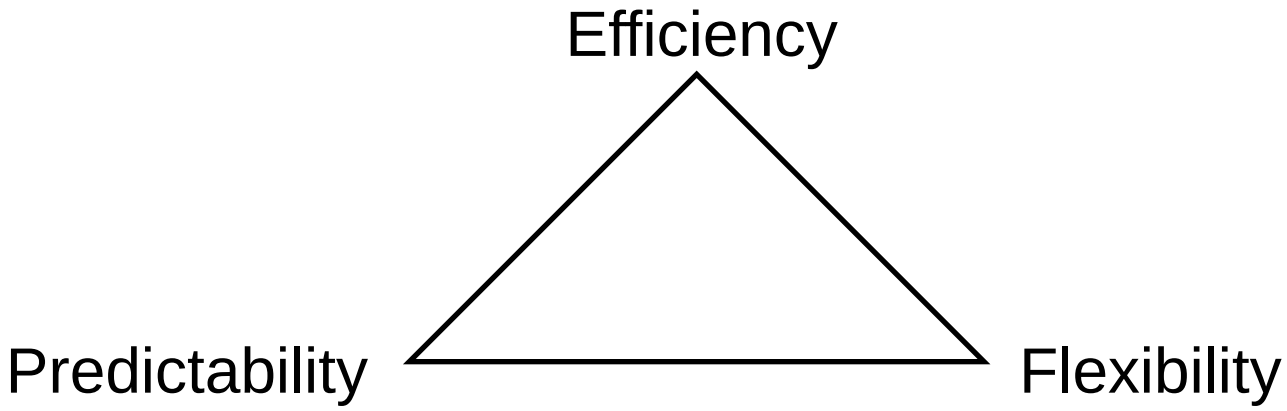
SDRAM layout

- SDRAMs have a multi-bank architecture and is organized in banks, rows and columns.
- Memory efficiency measures percentage of useful cycles.
- Memory cycles are wasted as:
  - Rows are opened and closed
  - Read/write switches
  - Memory is refreshed



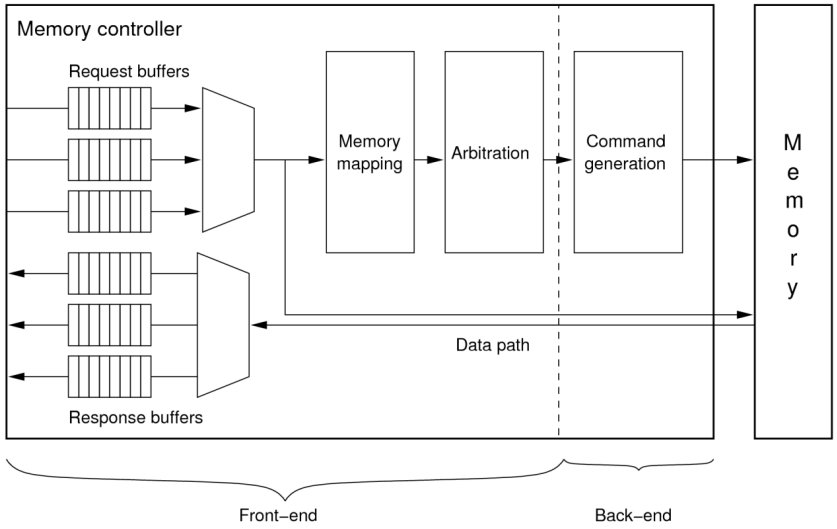
Problem statement

- Embedded systems require a memory service that offers:
  - Flexibility
  - High memory efficiency
  - Real-time guarantees on (net) bandwidth and latency (predictability)

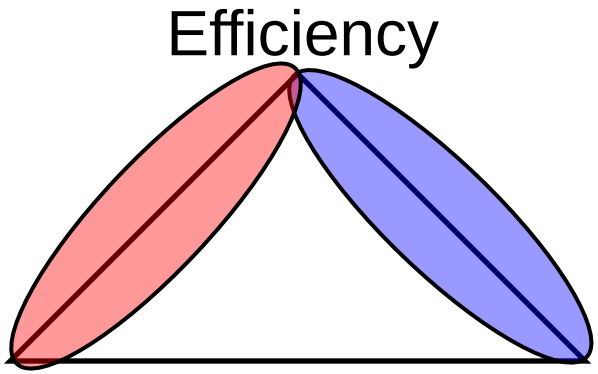


Memory controller overview

- Four functional blocks
  - Memory mapping
  - Arbitration
  - Command generator
  - Data path

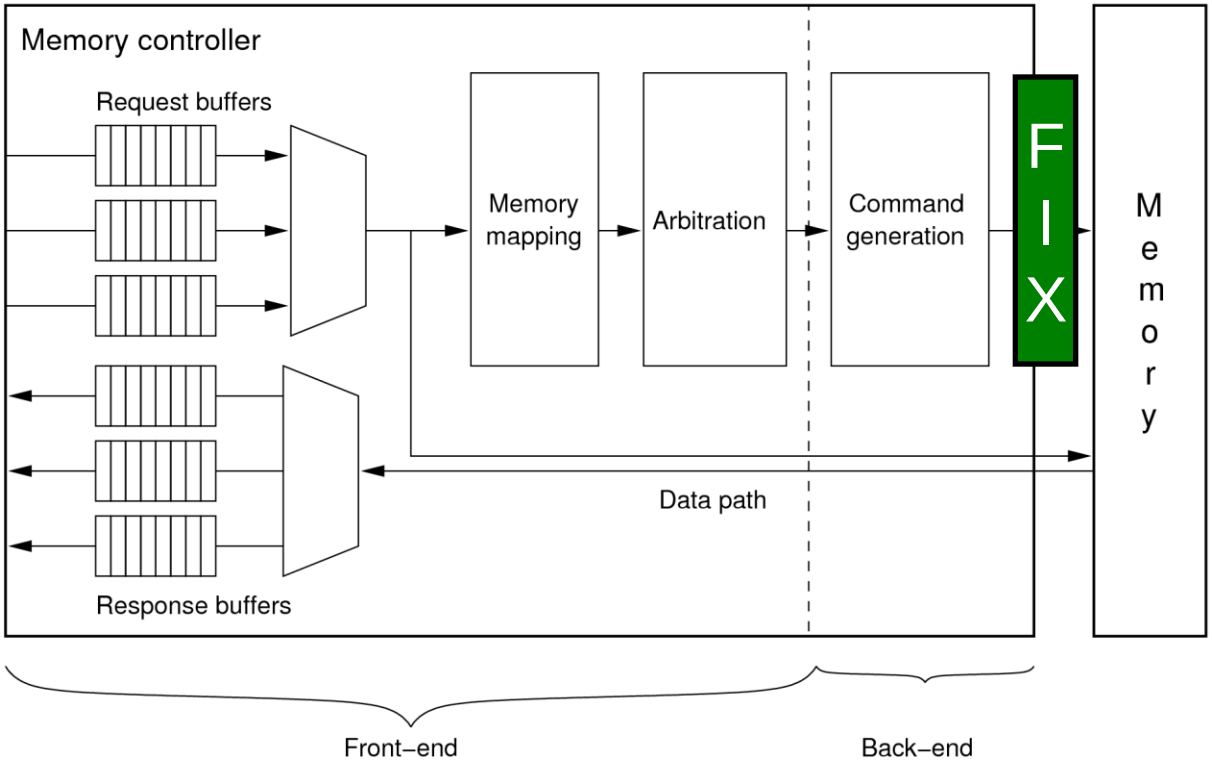


Static controller design



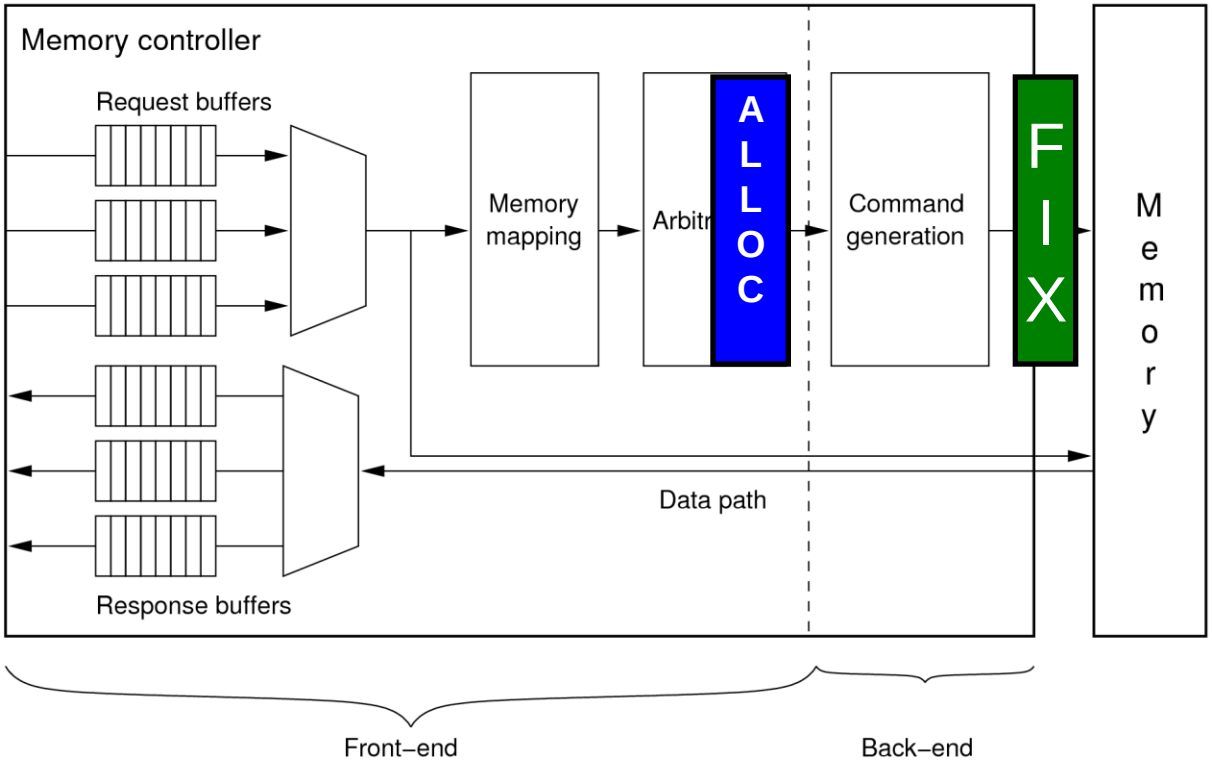
Dynamic controller design

Proposed solution [1 / 3]



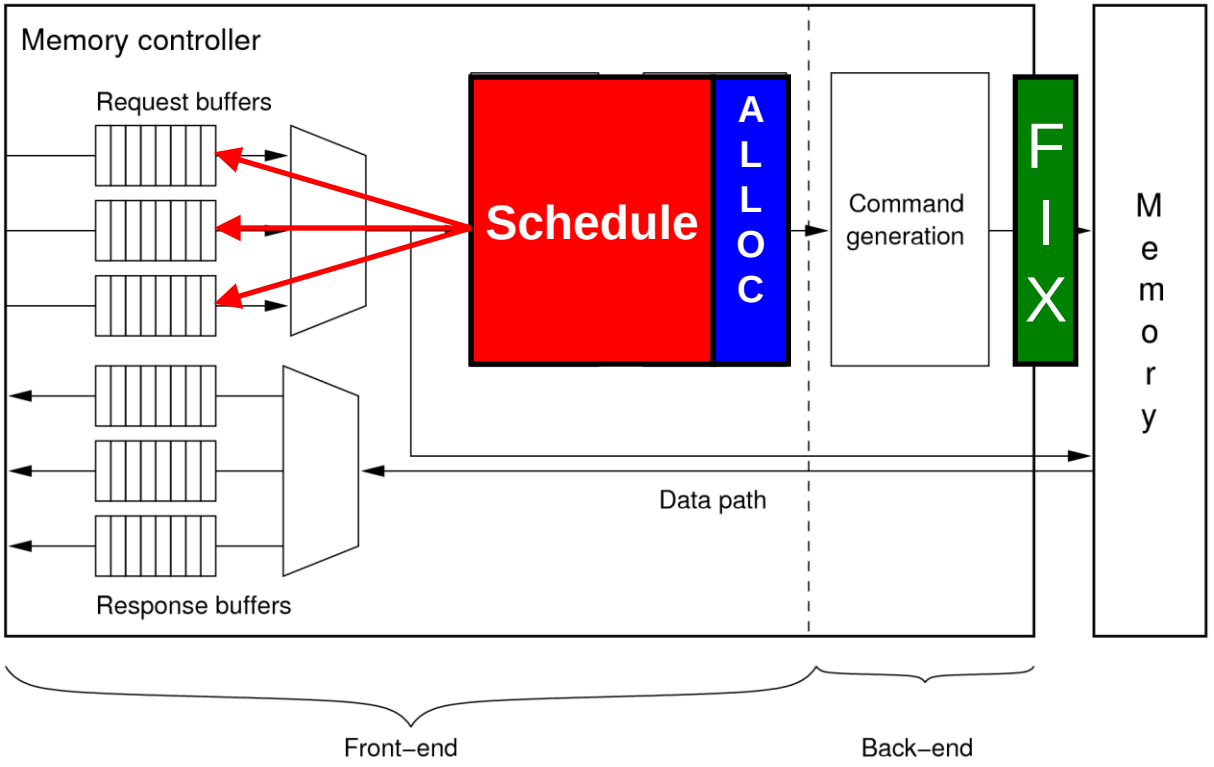
- Gross to net bandwidth translation by fixing back-end schedule

Proposed solution [2 / 3]



- Allocate net bandwidth to requestors

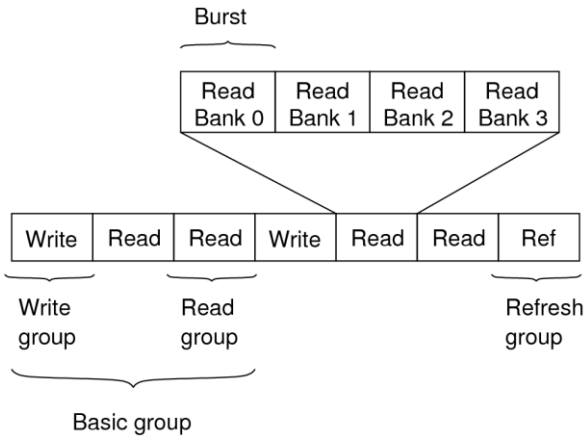
Proposed solution [3 / 3]

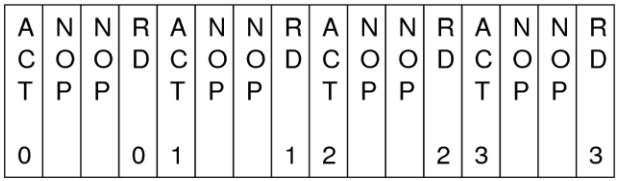


- Dynamically schedule requests for increased flexibility.

Back-end schedule

- Composed of read, write and refresh groups
  - Groups contain low-level SDRAM commands
  - One burst for every bank
- Fixed back-end schedule
  - Translates gross to net bandwidth
  - High predictable efficiency

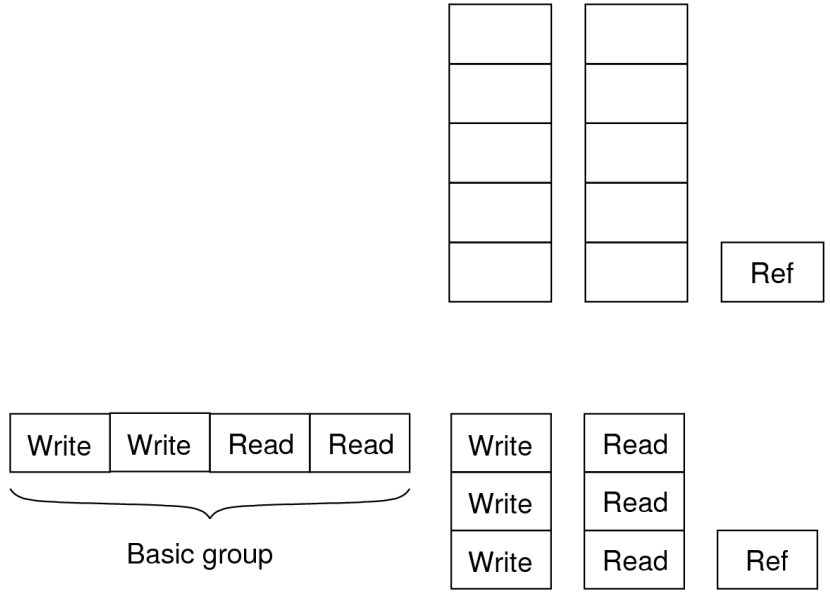




Read group

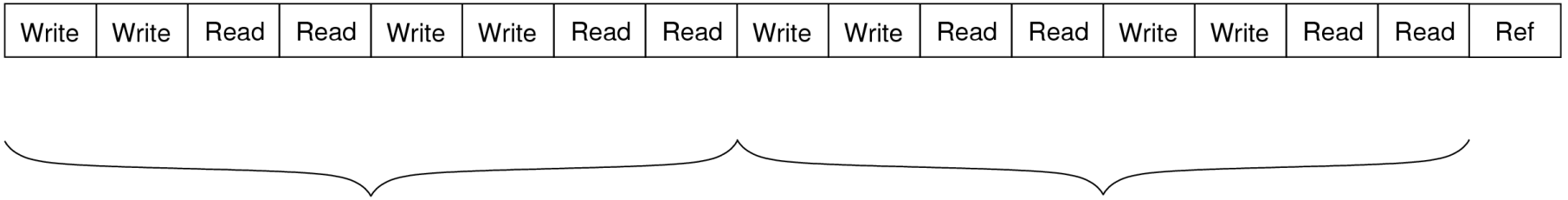
Creating a back-end schedule

- Include refresh group
  - Determines length of schedule
- Determine basic group layout
  - Read/write mix
  - Affects latency and efficiency
- Repeat basic group
- Algorithm uses exhaustive search



Allocation scheme

- Back-end schedule divided into service periods



- Bursts are allocated to requestors
  - Corresponds to net bandwidth
  - According to bandwidth requirements
  - Introduces discretization errors

Bandwidth guarantee

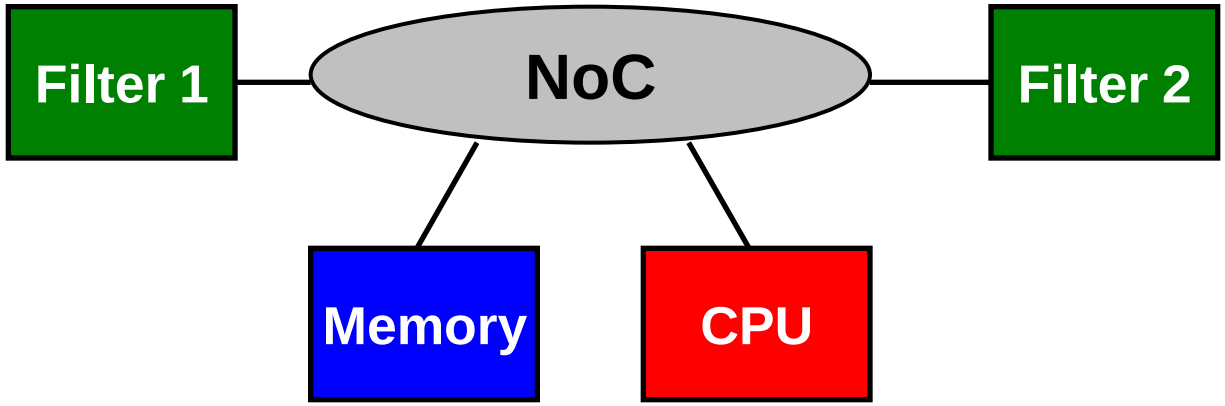
- There is no such thing as an unconditional guarantee!
- Constraints for bandwidth guarantees
  - Requestors must be backlogged
  - Requestors can only read or write
  - Requestors must use specific access patterns
- Guarantee provides analytical base for worst-case latency

Dynamic front-end scheduler

- Bridge between allocation scheme and back-end schedule
- Dynamically chooses a requestor that fits with the current burst
- Our implementation is a QoS-aware FCFS scheduler
  - Low latency and high bandwidth traffic classes
  - Low latency is preferred while within budget

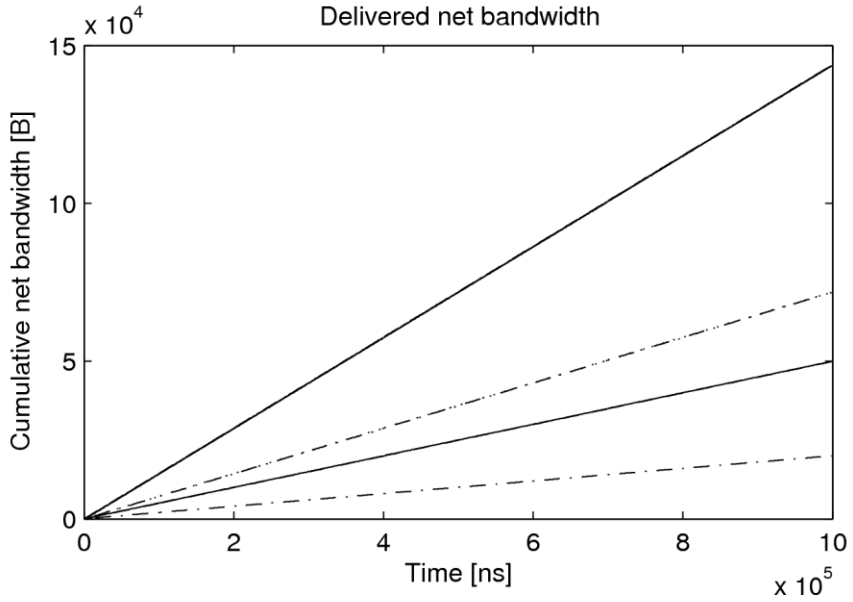
Example system

- Based on a Philips video processing SoC
- Connected through Philips Æthereal NoC
- Two filters provide eight high bandwidth requestors
- A CPU with three low latency requestors has been added

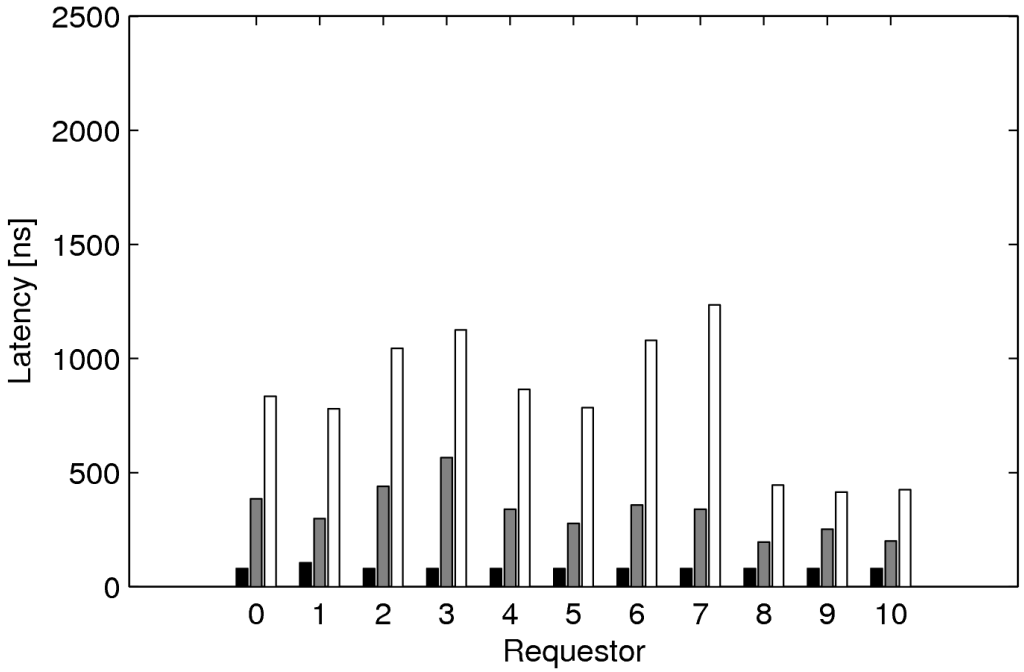


Bandwidth results

- Net bandwidth is delivered in real-time
- Maximum discrepancy of 0.22% from requested bandwidth
- Loads up to 89.3% have been successfully simulated



Latency results



- Solution is flexible. Lower latency to low latency requestors:
  - 75.8% lower worst-case latency
  - 42.5% lower average latency

Conclusions

- Our solution provides hard real-time guarantees on:
  - Minimal net bandwidth
  - Maximum worst-case latency
- Guarantees are provided through constraints
- Based on analytical model
  - Guarantees are provided without simulation