TU/e

technische universiteit eindhoven

Predator: A Predictable SDRAM Controller

Benny Åkesson
Technische Universiteit Eindhoven
The Netherlands

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

Markus Ringhofer
Graz University of Technology
Austria

NXP

Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

Introduction

- MPSoC design is getting increasingly complex
  - Large number of IP components
    - Accelerators and processing elements (with caches)
    - Some have hard real-time requirements on bandwidth and latency
  - Applications get more dynamic
    - Increased input dependence
    - Memory traffic is not fully known at design time
- Communication through shared memory
  - Large storage and bandwidth requirements
  - Sharing cause interference between components *(requestors)*

Sharing SDRAM / Problem Statement

- External SDRAM memories
  - are large and cost-effective.
  - are performance bottle-necks: must be efficiently utilized.

- Access times depend on previous requests, causing
  - additional interference between requestors.
  - variable bandwidth.

Problem to analytically verify that hard real-time requirements on bandwidth and latency are satisfied at design time!

Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

SDRAM Architecture

- SDRAMs are organized in banks, rows and columns.
- A row buffer stores the currently active row.

Example memory:
16-bit DDR2-400B 64 MB:

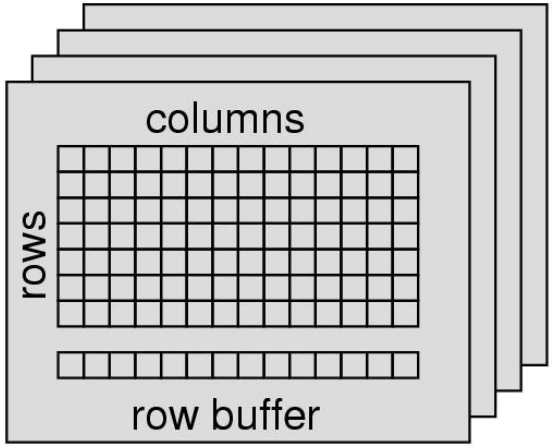
4 banks

8K rows / bank

1024 columns / row

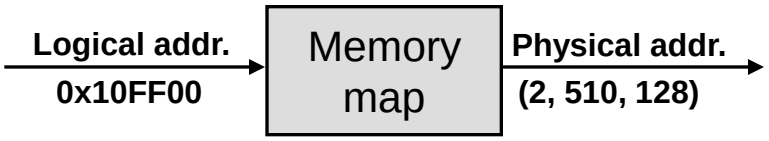
16 bits / column

800 MB/s peak
(gross) bandwidth

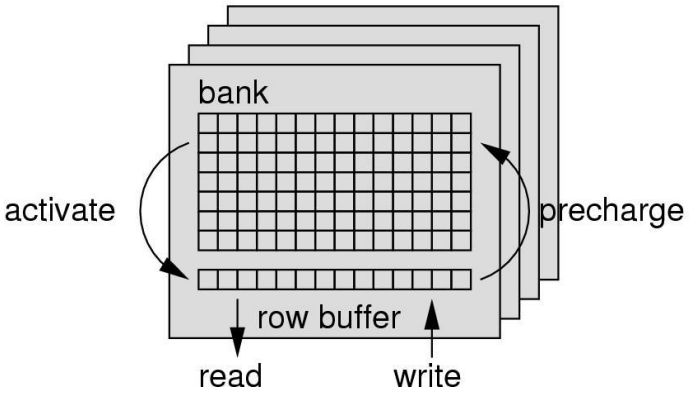


Basic SDRAM Operation

- Memory map decodes logical address to physical address (bank, row, column).



- Requested row is *activated* and copied into the row buffer of the bank.



- *Read* and/or *write* bursts are issued to the active row.
  - Programmed burst size of 4 or 8 words
- Row is *precharged* and stored back into the memory array.

Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

Memory Efficiency

- Memory efficiency
  - The number of clock cycles when requested data is transferred divided by the total amount of clock cycles.
  - Defines the exchange rate between gross and net bandwidth.

- Four categories of memory efficiency for SDRAM:
  - Refresh efficiency
  - Read/write efficiency
  - Bank efficiency
  - Data efficiency

Refresh Efficiency

- SDRAM need to be refreshed to retain data.
  - DRAM cell contains leaking capacitor.
  - Refresh command must be issued every 7.8 µs for DDR2/DDR3 SDRAM.
  - Data cannot be transfered during refresh.

- Refresh efficiency is **independent of traffic.**
  - Depends on storage capacity of the memory device (generally 95 – 99%).

Read / Write Efficiency

- Cycles are lost when switching direction of the data bus.

- Read/write efficiency **depends on traffic.**
  - Determined by frequency of read/write switches

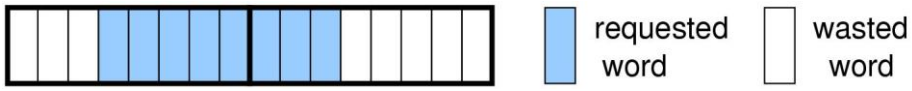
Bank Efficiency

- Bank conflict occurs when a read or write targets an inactive row.
  - Requires precharge followed by activate

- Bank efficiency **depends on traffic**.
  - Determined by address of request and memory map

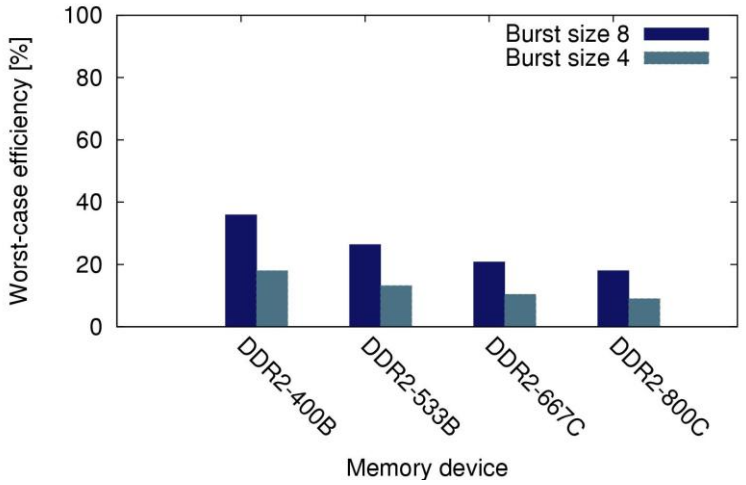
Data Efficiency

- A memory burst can access segments of the programmed burst size.
  - Minimum access granularity
- If data is poorly aligned an extra segment have to be transferred.
  - Cycles are lost when transferring unrequested data.
- Data efficiency **depends on traffic**.
  - Smaller requests and bigger burst size reduce data efficiency.
  - Can be determined at design time if traffic is characterized.



Conclusions on Memory Efficiency

- Memory efficiency is difficult to determine at design time.
  - Highly dependent on traffic
- Require worst-case efficiency to satisfy hard real-time requirements.
  - Every burst targets different rows in the same bank
  - Read/write switch after every burst
- Results in
  - Less than 40% efficiency for all DDR2 memories
  - Efficiency drops as memories become faster (DDR3)



Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

Statically Scheduled Controllers

- Some memory controllers are statically scheduled.
  - Execute static sequence of SDRAM commands
  - Static mapping from read and write bursts to requestors (TDMA)

- Statically scheduled controllers are
  - **predictable**
    - Latency of requests and available net bandwidth can be computed
    - Analytical verification at design time
  - **inefficient**
    - Cannot adapt to variations in traffic
  - **not scalable**
    - Combinatorial explosion in number of schedules to create, store and verify

Dynamically Scheduled Controllers

- Other controllers are dynamically scheduled
  - Dynamic memory access scheduler.
  - SDRAM commands generated dynamically in run-time.

- Dynamically scheduled controllers are
  - **flexible**
    - Adapt to variations in traffic.
  - **efficient**
    - Can reorder requests to fit with memory state.
    - Schedule refresh when it does not interfere.
  - **unpredictable**
    - Difficult to provide analytical bounds on net bandwidth and latency.
    - Typically verified by simulation.

Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

Overview of Approach

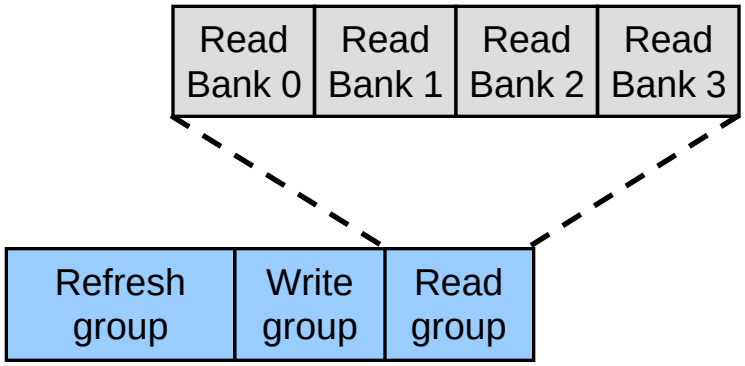
- We use a two-step hybrid approach.
  - Combines properties of statically and dynamically scheduled controllers.

1. Memory access groups
   - Precomputed sequences of SDRAM commands
   - Read, write, and refresh groups
   - Predictability of statically scheduled controllers

2. Predictable run-time arbitration
   - Read and write groups are dynamically scheduled
   - Scalability and flexibility of dynamically scheduled controllers

Memory Access Groups

- Read/write groups composed of one burst per bank.
  - Maximum pipelining
  - Reduces bank conflicts
- Minimum access granularity of 64 B (burst length 8).
  - Suitable for L2 caches and many accelerators.
  - Smaller accesses supported by masking



Analysis of Memory Efficiency (BL 8)

- Worst-case analysis for 16-bit DDR2-400B 64 MB with burst length 8:

Category	Efficiency	Comment
Refresh eff.	98.1%	Issued every 7.8 µs. Group is 31 cycles.
Read/write eff.	84.2%	Assume read/write switch after every group.
Bank eff.	100.0%	No bank conflicts for DDR2-400 due to access pattern.
Data eff.	100.0%	Assuming 100%. Determined when application is characterized.

- Worst-case efficiency = 98.1% x 84.2% x 100% x 100% = 82.6%
  - Corresponds to 660.9 MB/s of net bandwidth

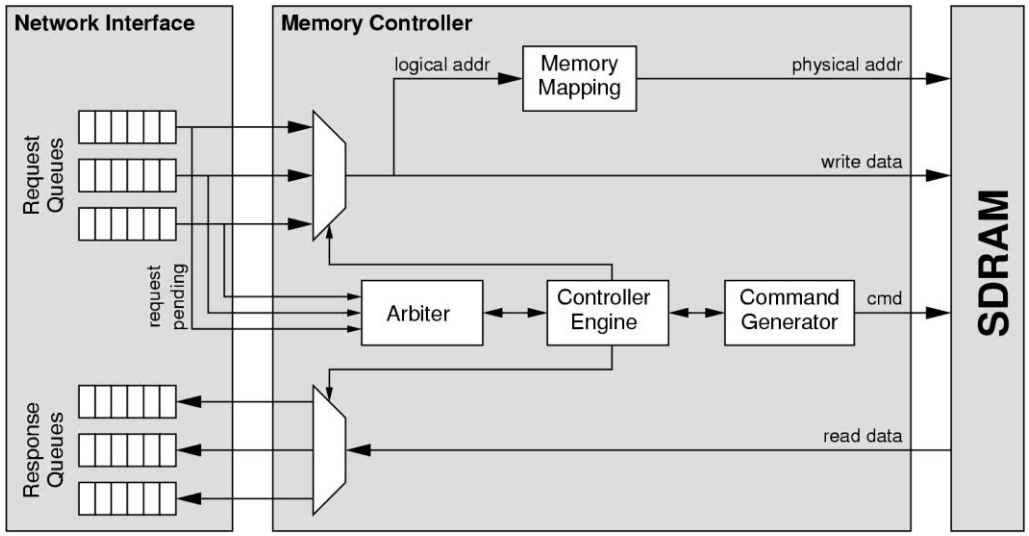
Predictable Arbitration

- We require a predictable arbiter
  - Provides an upper bound on latency of a request
  - Example: Weighted Round-Robin, Fair Queuing
- Arbiter unaware of memory controller design
  - Latency computed in number of groups
  - Time bound is derived
    - Group compositions are known
    - Possible group combinations are known
- Provides latency bound on net bandwidth!

Architecture

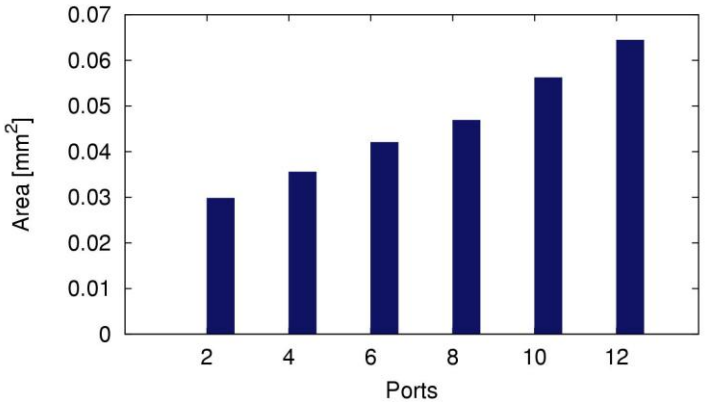
- Memory controller integrated with Æthereal network-on-chip.

- Four functional units
  1. Controller Engine
  2. Arbiter
  3. Memory Mapping
  4. Command Generator



Synthesis Results

- Controller synthesized in 0.13µm CMOS technology.
  - Six ports and speed target of 200 MHz requires 0.042 mm^2
  - Scales linearly with number of ports



- Controller is small for two reasons:
  1. Queues in the network interface are reused.
  2. Command generator does not require registers to track memory state.

Experimental Setup

- Simulated controller in SystemC model
  - Four requestors asking for 165 MB/s each
  - Total load = 99.9% of net bandwidth
  - Requests are 64 B

- Simulation uses Credit-Controlled Static-Priority (CCSP) arbiter.
  - Consists of rate regulator and static-priority scheduler
  - Isolates requestors
  - Negligible over-allocation
  - Efficient hardware implementation

Experimental Results

- Results after 10^8 ns
  - All requestors receive their allocated bandwidth
  - No latency bound is exceeded
  - Bound less tight for low priority requestors
    - Worst-case is very unlikely with static-priority scheduler.

Requestor	Bandwidth [B]	Max [ns]	Bound [ns]
r_0	16499968	204	340
r_1	16500032	304	615
r_3	16499968	463	1185
r_4	16499968	732	2810

Presentation Outline

Introduction

SDRAM Basics

Memory Efficiency

Related Work

Predator

Conclusions

Conclusions

- Predator is a predictable SDRAM memory controller using
  - memory access groups (read, write and refresh groups).
  - predictable arbitration.
- Our solution provides
  - lower bound on memory efficiency.
  - upper bound on latency.
- Implementation
  - is light weight.
  - scales linearly with the number of ports.

Predator allows us to verify hard real-time requirements on net bandwidth and latency at design time.

Questions?

k.b.akesson@tue.nl