SDRAM Controllers for Mixed Time-Criticality Systems

Benny Akesson and Kees Goossens
Eindhoven University of Technology
Embedded systems get increasingly complex
- Increasingly complex applications (more functionality)
- Growing number of applications integrated in a device
- More applications execute concurrently
- Requires increased system performance without increasing power

The resulting complex contemporary platforms
- are heterogeneous multi-processor systems with distributed memory hierarchy to improve performance/power ratio
- use a shared single off-chip SDRAM to reduce cost
Applications have different time-criticality

**Firm real-time requirements (FRT)**
- E.g. software-defined radio application
- Failure to satisfy requirement may violate correctness
- No deadline misses tolerable

**Soft real-time requirements (SRT)**
- E.g. media decoder application
- Failure to satisfy requirement reduces quality of output
- Occassional deadline misses tolerable

**No real-time requirements (NRT)**
- E.g. graphical user interface
- No actual requirements, but must be perceived as responsive
Complex systems have **mixed time-criticality**
- Firm, soft, and no real-time requirements in one system
- We refer to this as **mixed real-time (MRT)** requirements

There are suitable memory controllers for either FRT and SRT/NRT
- No good solutions for mixes between these types

The contributions of this presentation are
- a **survey of FRT and SRT/NRT memory controllers**, respectively
- an **overview of MRT requirements** and why existing controllers fail to satisfy them
- a **trajectory to evolve current controllers** to fit with MRT requirements
Introduction

SDRAM overview

- Firm real-time controllers
- Soft/no real-time controllers
- Mixed real-time controllers

Conclusions
An SDRAM is organized in **banks**, **rows** and **columns**
- A **row buffer** in each bank stores a currently active (open) row

SDRAM cells suffer from leakage
- Needs to be **refreshed** regularly to retain data
Memory map decodes address to bank, row, and column

Row is **activated** and copied into the row buffer of the bank

Read bursts and/or write **bursts are issued to the active row**
- Programmed **burst length** (BL) of 4 or 8 words

Row is **precharged** and stored back into the memory array
Execution times of requests are variable and traffic dependent
- Can vary by an order of magnitude
- Three reasons for overhead cycles:
  - Activating and precharging (opening and closing) rows
  - Switching direction of data bus from read to write
  - Refreshing the memory

Memory efficiency
- The fraction of clock cycles when requested data is transferred
- Determines the provided net bandwidth
- High efficiency is required since bandwidth is a scarce resource
Presentation Outline

- Introduction
- SDRAM overview
- Firm real-time controllers
- Soft/no real-time controllers
- Mixed real-time controllers
- Conclusions
FRT requirements must be satisfied even in worst-case scenario

Typical goals of firm real-time controllers:
- Maximize the **worst-case** net bandwidth
- Minimize the **worst-case** response time
- A **trade-off** between the two, since they are contradictory
SDRAM performance is highly dependent on **locality**
- Request served quickly if it targets an open row
- No overhead of opening and closing rows

FRT controllers are typically **unable to exploit locality**
- Locality has to be guaranteed also in worst case
- Difficult for a single executing application
  - Requires intimate knowledge of memory accesses
- More or less impossible for multiple concurrent applications
  - Memory accesses mixed by memory arbiter
- Makes average and worst-case performance very different
  - One reason why it is expensive to provide firm performance guarantees
As a result, FRT controllers use close-page policies [Akesson, Paolieri, Reineke]
- Precharge banks immediately after each request
- Assumes that every request targets closed rows

Benefits of policy
- Reduces worst-case overhead of opening/closing rows
- Increases guaranteed net bandwidth

Drawbacks of policy
- Sacrifices best and average-case performance and power
- Limits max efficiency of 16-bit DDR3-800 with 64B requests to 80%
  - Results from the Predator SDRAM controller [Akesson]
Controllers are classified as **statically** or **dynamically** scheduled
- Depends on SDRAM command scheduling mechanism

**Statically scheduled controllers**
- Pre-compute SDRAM schedule at design time
- Bandwidth and execution time bounded by inspecting schedule
  - Suitable for FRT requirements
- Restricted to applications with well-specified memory behavior

- Suitable for single applications without input dependence [Bayliss]
  - Application-specific memory controller
  - Possible to derive optimal page policy if full memory trace is known
Dynamically scheduled FRT controllers

- Schedule commands at run-time based on incoming requests
- Challenge is to analyze command scheduler
  - Required to bound net bandwidth and execution times

- Analysis often assumes large fixed-size requests [Akesson, Paolieri]
  - Large enough to exploit maximum bank-level parallelism by interleaving
  - Requires 64-256 B requests depending on memory device
A hybrid controller combines static and dynamic scheduling

Approach based on pre-computed memory patterns [Akesson]
- Patterns are statically scheduled sequences of SDRAM commands
- Dynamically scheduled at run time

There are five types of memory patterns
- Read, write, r/w switch, w/r switch, and refresh patterns

Read pattern for DDR2-400
Request to pattern mapping:
- Read request → read pattern (potentially first w/r switch)
- Write request → write pattern (potentially first r/w switch)
- Refresh pattern issued when required

Patterns result in scheduling at higher level
- Less state and fewer constraints, making them easier to analyze

Memory patterns let us provide lower bound on bandwidth
- E.g. 1008 MB/s (63%) from a 16-bit DDR3-800 with 64 B requests
All presented types of controllers have **bounded execution time**
- Bounding response times requires **predictable arbitration**
- Bounds number of interfering requests from other memory clients

Different controllers uses **different arbiters**
- Statically scheduled controllers uses a static schedule
- [Paolieri] employs Round-Robin arbitration
  - Targeting homogeneous chip multi-processors
- [Akesson] supports a variety of predictable arbiters
  - E.g. (Weighted) Round-Robin, Credit-Controlled Static-Priority, and Frame-Based Static-Priority
  - Targets heterogeneous MPSoCs
Introduction
SDRAM overview
Firm real-time controllers
Soft/no real-time controllers
Mixed real-time controllers
Conclusions
Same controllers normally used for SRT/NRT requirements

- Dynamically scheduled high-performance controllers

SRT applications are verified by simulation rather than formally

- Firm transaction-level guarantees are not necessary
- Sufficient to satisfy application-level deadlines with high probability
  - May correspond to thousands of memory requests

Typical goals of soft/no real-time controllers:

- Maximize the average net bandwidth
- Minimize the average response time
- A trade-off between the two, since they are contradictory
SRT controllers do not have to guarantee locality
- Requires locality to **offset miss penalties** with **high probability**

**Open-page policies** are common in SRT controllers
- Rows are speculatively kept open to exploit locality
- Average efficiency is hence typically higher than for FRT controllers
- Best-case memory efficiency is hence around 98%
  - All requests are either reads or writes to the same row
  - Efficiency losses only due to mandatory refresh activities
SRT controllers are **flexible** and supports most memory traffic
- SRT Controllers are dynamically scheduled
- Does not require formal analysis of supported memory traffic
- Enables supports of e.g. variable request sizes

Fine-grained scheduling at level of **single SDRAM bursts**
- Reduces wasted data of memory patterns (data efficiency)
- Reduces response times of sensitive clients
- Low worst-case memory efficiency
  - Cannot guarantee locality or bank-level parallelism
  - Worst-case efficiency about 16% for DDR3-800 with BL=8 words
  - Bound determined by activate-to-activate delay within a bank
  - Bound derived from memory spec. and applies to most controllers
Memory efficiency is optimized using **sophisticated mechanisms**

- Preference for requests that target open rows [Several]
  - Reduces overhead of opening and closing rows
  - Increases response times for clients targeting closed rows

- Read/write grouping [Several]
  - Reduces read/write switching overhead
  - Increases response times for requests in wrong direction
Reduction of Response Times

Preference for reads over writes [Shao]
- Reads are often blocking while writes are posted
- Reduces stall cycles on processor
- No problem unless other application waits for data

Preemption of low-priority requests in service [Lee]
- Reduces response times of high-priority clients
- Increases response times of low-priority clients
- Reduces memory efficiency due to preemption overhead

Interactions between mechanisms are complex
- Difficult to derive useful bounds on bandwidth and response times
- May even be difficult to guarantee the default 16% net bandwidth
Presentation Outline

Introduction
SDRAM overview
Firm real-time controllers
Soft/no real-time controllers
Mixed real-time controllers
Conclusions
MRT controllers must efficiently support FRT, SRT and NRT

Current FRT controllers treat SRT/NRT clients like FRT clients
- Expensive both in terms of bandwidth and power

Current SRT/NRT controllers treat FRT like SRT/NRT clients
- Guarantees are either not formally proven or very pessimistic
- Worst-case may be maximum observed case plus a safety margin
- Deadlines may be missed in corner cases

MRT controllers are likely to evolve from current controllers
- Either from FRT controllers or SRT/NRT controllers
Evolving FRT controllers to MRT requires **five issues** to be solved

1. **Trade-offs** between worst/average performance
   - Only guarantee **sufficient** bandwidth and response times for FRT
   - Then maximize average-case performance for SRT/NRT
   - Can be done by moving to **predictable open-page policies**
     • Sacrifices worst-case guarantees to exploit (limited) locality

- Increases best-case efficiency from 80% to 98%
- Reduces worst-case efficiency from 63% to around 40%
- Preliminary results with the Predator controller [Akesson]
- 16-bit DDR3-800 with BL=8 and 64B requests
2. Providing **robust** FRT guarantees in presence of SRT/NRT
   - FRT behavior is well-specified, but SRT/NRT may not be
   - Guarantees must be independent of behaviors of other clients

3. Increasing flexibility to **support more dynamic traffic**
   - FRT controllers have assumptions or restrictions on traffic
   - Cannot support dynamism present in SRT/NRT traffic
     • E.g. variable request sizes
   - May involve generalizing both controllers and analysis
4. Support for **multiple use-cases**
   - Applications in MRT systems may start and stop at run time
   - Requires **reconfigurable** FRT memory controllers
   - Challenge is to provide FRT guarantees during reconfiguration

5. Predictable **power-down** strategies
   - Reducing power is grand challenge for coming decade
   - Existing power management limited to SRT/NRT controllers
Evolution of SRT/NRT controllers requires **two issues** to be solved

1. **Restrict or simplify** use of sophisticated dynamic features
   - E.g. reordering, read/write grouping, preemption
   - Helps analyzing their impact on FRT clients
   - Required for tighter bounds on FRT performance
2. **Increase access granularity** beyond a single burst
   -restricts traffic is efficiently supported
   -enables more than 16% of net bandwidth to be guaranteed
Presentation Outline

Introduction
SDRAM overview
Firm real-time controllers
Soft/no real-time controllers
Mixed real-time controllers
Conclusions
Complex SoCs have **mixed real-time** (MRT) requirements

- Mix of firm (FRT), soft (SRT), and no real-time (NRT) requirements
- There are suitable controllers for FRT and SRT/NRT, but not MRT

**Firm real-time controllers**

- Maximize bandwidth bound and minimize response time bound
- Static, dynamic, or hybrid SDRAM command scheduling
- Close-page policies to reduce miss penalty
- Predictable arbitration

**Soft/no real-time controllers**

- Maximize average bandwidth and minimize average response time
- Dynamically scheduled with sophisticated mechanisms
- Open-page policies to exploit locality
Evolution of existing FRT controllers
1. Enable **trade-offs** between worst/average performance
   - Predictable open-page policies
2. Providing **robust** FRT guarantees in presence of SRT/NRT
3. Increasing flexibility to **support more dynamic traffic**
   - Generalize analysis
4. Support for **multiple use-cases**
5. Predictable **power-down** strategies

Evolution of SRT controllers
1. **Restrict or simplify** use of sophisticated dynamic features
2. **Increase access granularity** beyond a single burst


[Several] Several different works, listed in paper.

Thank you for your attention!

Any questions?

Our book “Memory Controllers for Real-Time Embedded Systems” from Springer is launched here at ESWEEK. Have a look!