A Reconfigurable Real-Time SDRAM Controller for Mixed Time-Criticality Systems

CODES+ISSS
30-09-2013

Sven Goossens, Jasper Kuijsten, Benny Akesson, Kees Goossens
Problem Statement

- Without special measures:
  - Resource sharing makes functional and timing behavior interdependent
  - Verification effort grows exponentially with the number of applications
    - Can only be done after integration (and may need to be repeated!)

Verification scenarios:

Time: 1 2 3 4 5 6 7

- Non-real-time applications: App 3, App 1, App 4
- Real-time applications: App 2, App 5

= Use-case switch
The CompSOC approach

- **Virtual execution platforms**
- Isolation to reduce verification scenarios:
  - **Predictable virtual platforms**
    - performance isolation (resource budgets)
    - For analyzable firm real-time applications
  - **Composable virtual platforms**
    - Complete cycle-level temporal isolation:
      For verification by simulation

- Applications run in their own virtual platform
- The physical SoC resources are designed to eliminate interference
- Allows independent application development and verification
- **We focus on the SDRAM resource**
Contributions

This work has 3 main contributions:

1. Run-time reconfigurable SDRAM controller architecture
   - (vs. static, single configuration in existing work)
   - SystemC and VHDL (FPGA) prototype

2. Predictable and composable service through *composable memory patterns*

3. Shared through a run-time reconfigurable TDM arbiter, allowing reallocation of TDM slots in a predictable and composable way
SDRAM

- SDRAM consists of multiple **banks**, that each have **rows** and **columns**
- To read/write, a row in a bank first has to be **activated**
- Each bank can have only one active row
- After reading/writing, a row has to be **precharged** before another row can be activated
For an LPDDR3-1600 (800 MHz):

Efficiency (excluding refresh): 
\[
\frac{\text{#cc\_data}}{\text{#cc\_cmds}} \approx 17\%
\]

Naïve command scheduling \(\rightarrow\) low worst-case efficiency
Predictable SDRAM Patterns

- Basic idea: generate valid command series or patterns at design time, schedule them at run time.
- (Note: Switching patterns consist only of NOPs)

- Decrease access irregularity, increase granularity

Worst-case efficiency:
\[ \frac{\text{#cc_data}}{\text{#cc_cmds}} = 82\% \]

Basic idea: generate valid command series or patterns at design time, schedule them at run time.

(Note: Switching patterns consist only of NOPs)

Decrease access irregularity, increase granularity

Worst-case efficiency:
\[ \frac{\text{#cc_data}}{\text{#cc_cmds}} = 82\% \]
Outline

Introduction

Background: Predictable SDRAM

Reconfigurable Controller Architecture

Composable Memory Patterns

Reconfigurable TDM Arbiter

Experiments

Conclusions
Run-time reconfiguration infrastructure (memory mapped)

Reconfigurable TDM arbiter (predictable and composable during reconfiguration)

Reconfigurable back-end, using composable patterns.
• Patterns are reprogrammable at run time.
• Different pattern → different worst-case bandwidth, latency and power trade-off.
• Allows different trade-off per use case.

Details of the back-end, and FPGA synthesis results → In paper
Introduction

Background: Predictable SDRAM

Reconfigurable Controller Architecture

Composable Memory Patterns

Reconfigurable TDM Arbiter

Experiments

Conclusions
Composable Memory Patterns

- Goal: make SDRAM accesses composable $\rightarrow$ complete isolation of clients $\rightarrow$ slots always start at the same time

Predictable patterns:
- Read pattern
- RtW
- Write pattern
- WtR
- Read pattern
- RtW
- Write pattern
- Idle

Predictable patterns have non-constant slot sizes $\rightarrow$ not composable

Composable patterns:
- Read pattern
- Write pattern
- Read pattern
- Idle

Eliminate switching patterns, make remaining pattern lengths equal
Composable Patterns Generation

Naïve solution:

WtR

Read pattern

RtW

Write pattern

merge, max:

Proposed method:

slice:

Added NOPs

Read pattern

Write pattern

• (Note: we only slice within the switching patterns, which contain only NOPs)
• Minimizes impact on worst-case efficiency to 1 cycle (in case the total length is odd)
• (In paper) For a range of memories: **average efficiency loss of 0.22%** (2.6% max)
Reconfiguring a TDM Arbiter

TDM table, 5 slots, 5 applications (A-E)

| A | B | C | D | E | A | B | C | D | E | A | B | C | D | E |

Time

Reconfiguration event:
de-allocate E, move A, add F

Naive reconfiguration flow:

1. De-allocate finished app.
3. Allocate new app.
Reconfiguring a TDM Arbiter

TDM table, 5 slots, 5 applications (A-E)

A’s request arrives (just too late for the start of the slot)

Response time: 6 slots

Reconfiguration event: move A’s slot

Response time: 10 slots > 6 slots

A’s request arrives

Can this effect violate the performance guarantees given to A?
TDM Latency-rate Server

- Guarantee based on two parameters:
  - Client gets a minimum allocated rate ($\rho$),
  - After a maximum service latency ($\Theta$)
- (As long as the client produces enough requests to stay busy)
We model the reconfiguration as a hand-over between two independent latency-rate servers.

1. Deallocate finished app.
2a. Move: allocate new slots
2b. Move: de-allocate old slots
3. Allocate new app.

- The distance between step 2a and 2b matters
Switch off orange server

Switch on blue server

Orange guarantee

Received service (orange + blue)
If the distance between the “switch on” and “switch off” event is at least $\Theta$, then the original guarantees remain valid during reconfiguration.

The paper contains a mathematical proof for this property and a description of the hardware implementation.
Outline

- Introduction
- Background: Predictable SDRAM
- Reconfigurable Controller Architecture
- Composable Memory Patterns
- Reconfigurable TDM Arbiter
- **Experiments**
- Conclusions
Composability Experiment (FPGA)

- Two MicroBlaze cores (MB1, MB2) connected to a DMA
  - synthetic application generates traffic at 90 MB/s
  - record timestamps in request/response buffers
- Six experiments:
  - Using 1) Predictable patterns, 2) Composable patterns:
    A) Reference run:
    B) Interference run:
    C) Reconfiguration run:

\[32 \mu s\]
MB2’s behavior varies wildly across runs, as a result of the interference from MB1
→ Not composable (verification for MB2 has to take MB1 in to account)
MB2’s behavior is constant across runs, MB1 has no influence

→ Composable (can be verified independently)
Introduction
Background: Predictable SDRAM
Reconfigurable Controller Architecture
Composable Memory Patterns
Reconfigurable TDM Arbiter
Experiments
Conclusions
Conclusions

- Run-time reconfigurable SDRAM controller architecture.
  - Memory-mapped configuration ports to various components.
  - FPGA & SystemC implementation.

- Predictable and composable service through *composable memory patterns*
  - Each access has the same length, no explicit switching patterns.
  - *Max. 2.6% overhead*

- TDM reallocation in a predictable and composable way.
  - by enforcing a minimal distance between allocation and de-allocation of slots.
  - Demonstrated on FPGA
For further information:

www.compsoc.eu

Sven Goossens <s.l.m.goossens@tue.nl>
Jasper Kuijsten <jasperkuijsten@gmail.com>
Benny Akesson <kessoben@fel.cvut.cz>
Kees Goossens <k.g.w.goossens@tue.nl>