Predictable and Composable SDRAM Controller

Benny Åkesson

TU/e
Technische Universiteit Eindhoven
University of Technology

Where innovation starts
Presentation Outline

Embedded system design

- Verification problem
- Predictable SDRAM controller
- Composable SDRAM controller
- Conclusions
Trends in Embedded System Design

► MPSoC design gets **increasingly complex**
  – Moore’s law allows increased component integration
  – Digital convergence creates a market for highly integrated devices

► The resulting embedded systems
  – have a large number of IP components
  – run many different applications

► System **life time is decreasing**
  – Pressure to reduce cost and time to market
Applications

- An application performs a well-defined function for the user
  - E.g. media decoder, game, or telephony

- Applications may have real-time requirements
  - A certain computation must be finished before a deadline

- Applications may have different types of real-time requirements
  - Any combination of hard, soft, or non-real-time applications
Tasks and Use Cases

- An application can be partitioned into tasks to exploit parallelism
  - Tasks typically execute on different processors
  - Tasks communicate through a shared data structure

- Many applications may execute simultaneously
  - We refer to a set of concurrently executing applications as a use case
  - Starting or stopping an application causes a use case transition.
Current trend is towards highly parallel heterogeneous platforms
- Good balance between performance, power consumption, and flexibility

Components in platform:
- Many processing elements (cached CPUs, ASIPs, accelerators) [PE]
- Interconnect (e.g. hierarchical buses, NoC)
- Non-uniform distributed memory architecture (e.g. SRAM, SDRAM) [MEM]
- Peripherals [PERI]
- Connectivity and I/O [IO]
Resource sharing

- IP components communicate by sending requests and responses
  - Use communication protocol, such as AXI, OCP, DTL

- An IP performing resource access is referred to as a requestor
  - An application is perceived as a set of requestors in hardware

- Resources in the platform are shared to reduce cost
Mapping

Binding from functionality to platform

- Tasks are mapped to CPUs and processing elements
- Data structures are mapped on memories
- Communication requirements are derived and mapped
  - Bandwidth and latency requirements on interconnect and memories
  - Communication requirements may be diverse depending on applications

Additionally,

- Buffers are sized for the required throughput
- Arbiter settings are derived to provide required bandwidth in timely manner
Presentation Outline

Embedded system design

Verification problem

Predictable SDRAM controller

Composable SDRAM controller

Conclusions
Verification

- Verification is asserting that a design meets its specification

- Commonly done by simulation of applications executing on platform
  - Slow process with poor coverage - focus on critical use cases

- Resource sharing results in interference between applications
  - All use cases must be verified instead of all applications
    - Verification complexity increases exponentially with number of applications
  - Behavior of an application dependent on other applications in use case
    - Verification must be repeated if application is added, removed, or modified
    - Circular verification process

- Verification is costly and effort is expected to increase in future!
Formal Verification

- Formal verification is alternative to simulation-based approaches
  - Provides analytical bounds on latency or throughput

- Approach requires both predictable hardware and software
  - Hardware and software must be captured in performance models
  - Most industrial hardware not designed with formal analysis in mind
  - Suitable application models exist, but are not widely adopted by industry

  - Research has developed interconnect and SRAM controller with corresponding performance models.
  - Problem remains for complex resources, such as SDRAM controllers.
Problem with SDRAM

► The time to serve a request in an SDRAM is variable
  – Depends on if target row is open, if read or write, if time to refresh etc.
  – Difficult to guarantee that latency requirements are satisfied

► It follows that offered bandwidth is also variable and traffic dependent
  – Problem to guarantee that hard bandwidth requirements are satisfied
  – Worst-case bandwidth very low
    • Less than 40% of peak bandwidth for all DDR2 devices
    • Lower for faster memories, such as DDR3

► SDRAM bandwidth is scarce and must be efficiently utilized
  – Additional interfaces cannot be added due to cost constraints
Problem Statement

- Current trends make it increasingly difficult to design embedded systems that satisfies real-time requirements

- We require an SDRAM controller that can satisfy the requirements of embedded applications

- The controller should reduce verification effort by
  - enabling formal verification of real-time requirements
  - isolate applications to enable independent verification by simulation
Presentation Outline

Embedded system design
Verification problem
Predictable SDRAM controller
Composable SDRAM controller
Conclusions
Enabling formal verification

- Predictable systems enable formal verification of hard real-time requirements
  - Assumes performance models of system (applications + hardware)

- Definition: A predictable system is defined as a system in which there is a **useful** bound temporal behavior

- Bounding the temporal behavior of a memory controller involves bounding net bandwidth and latency

- We proceed by looking at how this is done with current controllers
Some memory controllers are statically scheduled
- Execute static sequence of SDRAM commands
- Static mapping from read and write bursts to applications (TDM)

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, such as changes in requested bandwidth, read/write ratio etc.

- not scalable
  - Combinatorial explosion in number of schedules to create, store and verify
  - One schedule per use-case
Dynamically Scheduled Controllers

- Other controllers are dynamically scheduled
  - Dynamic front-end and back-end scheduling
  - SDRAM commands scheduled dynamically in run-time

- Dynamically scheduled controllers are:
  - **flexible**
    - Adapt to variations in traffic
  - **efficient**
    - Can reorder requests to fit with memory state
  - **unpredictable**
    - Difficult to provide analytical bounds on net bandwidth and latency
    - Typically verified by simulation
Overview of Approach

► We use a hybrid approach
  – Combines properties of statically and dynamically scheduled controllers
  – Best of two worlds?
Predictable SDRAM

- Predictability through precomputed memory access patterns
  - Patterns are precomputed sequences of SDRAM commands

- There are five types of memory access patterns
  - Read, write, read/write switch, write/read switch, and refresh patterns

- Pattern to request mapping:
  - Read request maps to read patterns (potentially first write/read switch)
  - Write request maps to write patterns (potentially first read/write switch)
  - Refresh pattern scheduled by memory controller when required

Mapping from requests to patterns to bursts
Memory Access Patterns

► Patterns result in scheduling at higher level
  – Pattern history replaces command history
  – Less state and fewer constraints, making them easier to analyze

► Read/write patterns composed of interleaved bank accesses
  – Maximum bank parallelism
  – Efficient without relying on locality
  – Requires large requests (64 bytes for 16-bit memory with 4 banks)
    • Smaller requests are supported by masking

► Memory access patterns lets us provide lower bound on bandwidth
  – Bounds all categories of memory efficiency

---

Read pattern for DDR2-400

---

TU/e
Technische Universiteit Eindhoven
University of Technology
## Analysis of Memory Efficiency (BL 8)

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Efficiency</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh eff.</td>
<td>98.1%</td>
<td>Issued every 7.8 µs. Pattern is 32 cycles</td>
</tr>
<tr>
<td>Read/write eff.</td>
<td>84.2%</td>
<td>Assume read/write switch after every pattern</td>
</tr>
<tr>
<td>Bank eff.</td>
<td>100.0%</td>
<td>No bank conflicts for DDR2-400</td>
</tr>
<tr>
<td>Data eff.</td>
<td>100.0%</td>
<td>Assuming 100%. Determined when application is characterized.</td>
</tr>
</tbody>
</table>

- Worst-case efficiency = \(98.1\% \times 84.2\% \times 100\% \times 100\% = 82.6\%\)
  - Corresponds to 660.9 MB/s of net bandwidth
Predictable Front-end Arbitration

- Approach fits with any predictable front-end arbiter
  - Example: Round-Robin, TDM, or our own priority-based arbiter
  - Latency computed in number of interfering requests
  - Latency bound in clock cycles is easily derived since:
    - Request to pattern mapping is known
    - Pattern to cycle mapping is known (length of patterns)

- Provides **hard latency bound on latency and net bandwidth!**
Presentation Outline

Embedded system design
Verification problem
Predictable SDRAM controller
Composable SDRAM controller
Conclusions
Isolating applications

► Composable systems provides isolation between applications

► Definition: A composable system is a system in which applications are independent in the time domain
  – Cannot affect each others behavior by even a single clock cycle
Composability simplifies verification for the following five reasons:

1. **Linear verification complexity**
   - Applications can be verified in isolation

2. **Increases simulation speed**
   - Only need to simulate application and its required resources

3. **Incremental verification process**
   - Verification process can start when first IP is available

4. **Increased IP protection**
   - Verification process no longer requires IP of ISVs

5. **Functional verification is simplified**
   - Bugs caused by, e.g. race conditions, are independent of other applications
Overview of our approach

► We make arbiter composable by delaying responses and flow control
  – Emulates maximum interference from other requestors
  – Creates a temporally independent interface per requestor

► Our approach to composability is based on predictability
  – Cannot emulate worst-case interference unless you know what it is

► Approach does not have any assumptions on the applications
  – Works with all applications that cannot be formally verified

Requestor in composable system is unaffected when higher priority requestor changes behavior.
Composable Resource Front End

- Implemented by adding a Delay Block to the front-end
  - Computes worst-case scheduling time and finishing time at arrival
  - Worst-case scheduling time used to delay flow-control signal
  - Responses held in block until worst-case finishing time

- Composability can be enabled/disabled per application at run-time
  - Allows slack to be used to improve performance for best-effort applications
Presentation Outline

Embedded system design
Verification problem
Predictable SDRAM controller
Composable SDRAM controller

Conclusions
Conclusions

► We presented an SDRAM controller that addresses the use-case verification problem in SoCs.

► **Predictability** enables formal verification of real-time requirements
  – Covers all possible interactions with the system
  – Achieved by memory patterns and predictable arbitration
  – Requires performance model of entire system (applications + hardware)

► **Composability** enables verification by simulation in isolation
  – Only covers simulated traces
  – Achieved by emulating worst-case interference from other requestors
  – Our implementation does not have any assumptions on the application
Questions?

k.b.akesson@tue.nl