Design and Analysis of Real-Time Systems
Precision-Timed ARM – An Example of a Predictable Microarchitecture

Stephen A. Edwards  Columbia University
Sungjun Kim  Columbia University
Edward A. Lee  UC Berkeley
Isaac Liu  UC Berkeley
Hiren D. Patel  University of Waterloo
Jan Reineke  Saarland University
Predictability and Temporal Isolation

- Many embedded systems are real-time systems
  → Need for Timing Predictability

- Trend towards integrated architectures:
  → Need for Temporal Isolation

- Side airbag in car, Reaction in <10 mSec
- Crankshaft-synchronous tasks, Reaction in <45 µSec
- Audio + video playback with latency and bandwidth constraints
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
Outline

- Introduction
- **Precision-Timed ARM (PTARM) Pipeline**
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
Pipelining: Hazards

Forwarding helps, but not all the time...

LD R1, 45(r2)
DADD R5, R1, R7
BE R5, R3, R0
ST R5, 48(R2)

Unpipelined: FDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEEMWFDEE
Our Solution: Thread-interleaved Pipelines

Each thread occupies only one stage of the pipeline at a time
→ No hazards; perfect utilization of pipeline
→ Simple hardware implementation (no forwarding, etc.)
→ Latency of instructions independent of micro-architectural state
→ Microarchitectural timing analysis becomes trivial
Our Solution: Thread-interleaved Pipelines

Each thread occupies only one stage of the pipeline at a time
→ No hazards; perfect utilization of pipeline
→ Simple hardware implementation (no forwarding, etc.)
→ Latency of instructions independent of micro-architectural state
→ Microarchitectural timing analysis becomes trivial

Drawback: reduced single-thread performance

Reineke et al., Berkeley 8
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
Second Problem: Memory Hierarchy

- Register file is a temporary memory under program control.
- Cache is a temporary memory under hardware control.

PRET principle: any temporary memory is under program control.

PRET principles implies using a scratchpad rather than a cache.

- Hardware thread registers
- Interleaved pipeline with one set of registers per thread
- Scratchpad
- SRAM scratchpad shared among threads
- DRAM main memory
- I/O devices
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
Memory Hierarchy: Dynamic RAM vs Static RAM

Register file is a temporary memory under program control.
- Why is it so small?

Cache is a temporary memory under hardware control.
- Why is replacement strategy application independent?

PRET principle: any temporary memory is under program control.

**DRAM**
- Slow ➔ High Latency
- High Capacity

**SRAM**
- Fast ➔ Low Latency
- Low Capacity

**Instruction word size.**

Separation of concerns.

Dynamic RAM Organization Overview
Dynamic RAM Organization Overview

DRAM Cell
Leaks charge ➔ Needs to be refreshed (every 64ms for DDR2/DDR3) therefore “dynamic”
Dynamic RAM Organization Overview

**DRAM Cell**
Leaks charge \(\Rightarrow\) Needs to be refreshed (every 64ms for DDR2/DDR3) therefore “dynamic”

**DRAM Bank**
- Array of DRAM Cells
- Sense Amplifiers and Row Buffer
- Sharing of sense amplifiers and row buffer

Reineke et al., Berkeley 16
Dynamic RAM Organization Overview

DRAM Cell
Leaks charge \(\rightarrow\) Needs to be refreshed (every 64ms for DDR2/DDR3) therefore “dynamic”

DRAM Bank
= Array of DRAM Cells
+ Sense Amplifiers and Row Buffer
Sharing of sense amplifiers and row buffer

DRAM Device
Set of DRAM banks +
- Control logic
- I/O gating
Accesses to banks can be pipelined, however I/O + control logic are shared

Reineke et al., Berkeley 17
Dynamic RAM Organization Overview

**DRAM Cell**
Leaks charge $\rightarrow$ Needs to be refreshed (every 64ms for DDR2/DDR3) therefore “dynamic”

**DRAM Bank**
Array of DRAM Cells + Sense Amplifiers and Row Buffer
Sharing of sense amplifiers and row buffer

**DRAM Module**
Collection of DRAM Devices
- Rank = groups of devices that operate in unison
- Ranks share data/address/command bus

**DRAM Device**
Set of DRAM banks +
- Control logic
- I/O gating
Accesses to banks can be pipelined, however I/O + control logic are shared
DRAM Memory Controller

Translates sequences of memory accesses by Clients (CPUs and I/O) into **legal** sequences of DRAM commands

- Needs to obey all timing constraints
- Needs to insert refresh commands sufficiently often
- Needs to translate “physical” memory addresses into row/column/bank tuples
Dynamic RAM Timing Constraints

DRAM Memory Controllers have to conform to different timing constraints that define minimal distances between consecutive DRAM commands. Almost all of these constraints are due to the sharing of resources at different levels of the hierarchy:

- Needs to insert refresh commands sufficiently often
- Rows within a bank share sense amplifiers
- Banks within a DRAM device share I/O gating and control logic
- Different ranks share data/address/command busses

Reineke et al., Berkeley 20
General-Purpose DRAM Controllers

- Schedule DRAM commands dynamically
- Timing hard to predict even for single client:
  - Timing of request depends on past requests:
    - Request to same/different bank?
    - Request to open/closed row within bank?
    - Controller might reorder requests to minimize latency
  - Controllers dynamically schedule refreshes
- Non-composable timing. Timing depends on behavior of other clients:
  - They influence sequence of “past requests”
  - Arbitration may or may not provide guarantees
General-Purpose DRAM Controllers

Memory Controller


RAS B1.R3
CAS B1.C2
RAS B1.R4
CAS B1.C3
RAS B1.R3
CAS B1.C5

RAS B1.R3
CAS B1.C2
CAS B1.C5
RAS B1.R4
CAS B1.C3

Reineke et al., Berkeley 22
General-Purpose DRAM Controllers

Thread 1:
- Store B4.R3.C5

Thread 2:
- Load B3.R5.C3

Arbitration

Memory Controller
General-Purpose DRAM Controllers

Thread 1
- Store B4.R3.C5

Thread 2
- Load B3.R5.C3

Arbitration

Load B3.R3.C2
Load B3.R5.C3
Store B4.R3.C5

Memory Controller

Reineke et al., Berkeley 24
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
Predictable DRAM Controllers: Predator (Eindhoven) and AMC (Barcelona)

Closed-page policy: timing independent of previously accessed row

Spread each request over all banks, pipeline accesses to banks.

Statically precomputed sequences for writes, reads, write->read, read->write, refresh.

Predictable and/or composable arbitration:
- Predator: CCSP
- AMC: TDMA
Predictable DRAM Controllers: Predator (Eindhoven)

Closed-page policy: timing independent of previously accessed row

Spread each request over all banks, pipeline accesses to banks.  
→ increases access granularity

Statically precomputed sequences for writes, reads, write->read, read->write, refresh.
Predictable DRAM Controllers: Predator (Eindhoven) and AMC (Barcelona)

Predictable and/or Composable Arbitration (e.g. time-division multiple access)

Thread 1

... Thread 2
Load B3.R3.C2
Load B3.R5.C3

Memory Controller

Load B3.R3.C2
Load B3.R5.C3

Reineke et al., Berkeley 28
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
  - Integration into Precision-Timed ARM
PRET DRAM Controller: Three Innovations

- Expose internal structure of DRAM devices:
  - Expose individual banks within DRAM device as multiple independent resources

- Defer refreshes to the end of transactions
  - Allows to hide refresh latency

- Perform refreshes “manually”:
  - Replace standard refresh command with multiple reads
PRET DRAM Controller: Exploiting Internal Structure of DRAM Module

- Consists of 4-8 banks in 1-2 ranks
  - Share only command and data bus, otherwise independent
- Partition into four groups of banks in alternating ranks
- Cycle through groups in a time-triggered fashion
PRET DRAM Controller: Exploiting Internal Structure of DRAM Module

- Consists of 4-8 banks in 1-2 ranks
  - Share only command and data bus, otherwise independent
- Partition into four groups of banks in alternating ranks
- Cycle through groups in a time-triggered fashion

- Successive accesses to same group obey timing constraints
- Reads/writes to different groups do not interfere

Diagram:

Rank 0:
- Group 0: Bank 0, Bank 1
- Group 2: Bank 2, Bank 3

Rank 1:
- Group 1: Bank 0, Bank 1
- Group 3: Bank 2, Bank 3
PRET DRAM Controller: Exploiting Internal Structure of DRAM Module

- Consists of 4-8 banks in 1-2 ranks
  - Share only command and data bus, otherwise independent
- Partition into four groups of banks in alternating ranks
- Cycle through groups in a time-triggered fashion

- Successive accesses to same group obey timing constraints
- Reads/writes to different groups do not interfere

Provides four independent and predictable resources
PRET DRAM Controller: Exploiting Internal Structure of DRAM Module


PRET DRAM Controller

- Read Pattern
- Read Pattern
- Write Pattern

...
Pipelined Bank Access Scheme

Reineke et al., Berkeley 35
The sharing of resources within banks and ranks exploits bank parallelism for high bandwidth. We periodically provide access to the four resources every $N$ cycles. In rank 0, we read from resource $R_0$, and in rank 3, we write to resource $R_3$. The controller performs read commands to the independent resources in a periodic and pipelined fashion, exploiting bank parallelism and reducing interference amongst the resources. This is unlike conventional DRAM controllers that view the entire memory device as a single resource.

Table 1: Overview of DDR4 timing parameters at the example of the Qimonda HYS64T64-5EM.4:

- Four-bank activation window: Interval in which maximally four banks may be activated.
- Refresh cycle time: Time interval between a refresh command and a row activation.
- $t_{RAS}$: Time to precharge the DRAM array before the next row activation.
- $t_{WR}$: Write to read time: Time between the end of a write data burst and the start of a column-read command.
- $t_{CL}$: Row-to-column delay: Time from row activation to first read or write to a column within that row.
- $t_{RCD}$: Row activation to command delay: Time from the issuance of a row activation command to the earliest possible command, excluding precharge.
- $t_{FAW}$: Four-bank activation window: Interval in which maximally four banks may be activated.
- $N$: Refresh cycle time: Time interval between a refresh command and a row activation.

Figure 1: A diagram illustrating the periodic and pipelined access scheme employed. The backend views the memory device as four independent resources: each resource consisting of two banks within the same rank. The backend translates each access request into a row access command bus, preserving the pipelined access scheme. In the example, we perform a read from resource $R_0$ in rank 0 and a write to resource $R_3$ in rank 4. The controller inserts a NOP between any two consecutive requests. This avoids a collision on the data bus between reads and writes. The access slots belonging to the same resource from 0 to 0 cycles are NOP after four access slots: to increase the distance between two accesses. The backend does not issue any refresh commands to the memory module, as is the case for resource $R_0$ in rank 0. The accessed row is precharged after a burst access, as is the case for resource $R_3$ in rank 4. The closed-page policy, also known as the autoflash precharge policy, causes two cycles to occur in every cycle. We use a closed-page policy for a write, requiring 0 cycles. Burst transfer of length 4 appears within its rank 1 cycles after the CAS was issued on the command bus. A burst from rank 0 accesses the second resource with a CAS command bus, respecting the RAS command for accessing the second resource. However, if we waited for 2 cycles before issuing the CAS to access the first resource, it would conflict with the RAS command for accessing the second resource. The backend inserts an additional NOP to resolve this conflict. The burst to rank 1 is performed on the command bus. The DRAM chip appears within its rank 1 cycles after the CAS was issued on the command bus. The burst from rank 0 performs a burst access to the first resource in rank 0. By alternating between ranks, no two adjacent accesses go to the same resource, ensuring that the controller inserts a NOP between any two consecutive requests.

Reineke et al., Berkeley
The periodic access scheme employed consists of the following access requests from the backend: In the example, we perform a read from resource 0 to access the row, perform a burst access, and precharge the bank. This satisfies the write-to-read timing constraint by alternating between ranks so no two adjacent accesses go to the same rank. This avoids a collision on the data bus between reads and writes. If we waited for 3 cycles, 2 transfers would occur in every cycle. We use a closed page policy in case of a write, we need 0 cycles. However, we can see from Figure 1 that if we waited for 2 cycles, there are no requests for a resource, the backend does not send any command bus preserving the pipelined access scheme. If a burst transfer of 16 bytes, which will occupy the data bus for 8 cycles, appears within its rank. The accessed row to be immediately precharged after performing the RAS command and the first CAS command need to be 1 cycles apart. In order to meet row to column latency shown in Table 1, or a NOP. We refer to a triple of RAS, CAS and NOP as an access slot. In the diagram, burst from rank 0 and burst to rank 1, with the closed page policy, the posted CAS results in a CAS two cycles later within its rank. With the closed page policy, in case of a write, we need 0 cycles.
Pipelined Bank Access Scheme

- **Resources/Ranks**: Each resource consists of two banks within the same rank.
- **Controller**: Each DRAM controller is specific to our DDR memory module.
- **Bank Parallelism**: Exploiting bank parallelism for high bandwidth.
- **Access Scheme**: Periodic and pipelined access scheme.
- **Command Bus**: Commands issued to independent resources in parallel.
- **Data Bus**: Burst access commands issued from the command bus, preserving the pipelined access scheme.
- **Timing Parameters**:
  - **tRCD**: Row.Column Delay
  - **tCL**: Column Latency
  - **tRAS**: Row Access Time
  - **tRP**: Row Precharge Time
  - **tWR**: Write Recovery Time
  - **tFAW**: Full Activation Window
  - **tWL**: Write to Read Latency

---

Reineke et al., Berkeley 38
PRET DRAM Controller: “Manual” Refreshes

- Every row needs to be refreshed every 64ms
- Dedicated refresh commands refresh one row in each bank at once
- We replace these with “manual” refreshes through reads
  - Improves worst-case latency of short requests

Dedicated refresh commands vs refreshes through reads.

(refresh latencies not to scale)
PRET DRAM Controller: Defer Refreshes
PRET DRAM Controller: Defer Refreshes

- Refreshes do not have to happen periodically
- Refresh every row at least every 64 ms
- Schedule refreshes slightly more often than necessary → Enables to defer refreshes
PRET DRAM Controller: Defer Refreshes

- Refreshes do not have to happen periodically
- Refresh every row \textit{at least} every 64 ms
- Schedule refreshes slightly more often than necessary $\rightarrow$ Enables to defer refreshes
General-Purpose DRAM Controller vs PRET DRAM Controller

**General-Purpose Controller**
- Abstracts DRAM as a single shared resource
- Schedules refreshes dynamically
- Schedules commands dynamically
- “Open page” policy speculates on locality

**PRET DRAM Controller**
- Abstracts DRAM as multiple independent resources
- Refreshes as reads: shorter interruptions
- Defer refreshes: improves perceived latency
- Follows periodic, time-triggered schedule
- “Closed page” policy: access-history independence
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
- Evaluation
- Integration into Precision-Timed ARM
**Conventional DRAM Controller (DRAMSim2) vs PRET DRAM Controller:**

**Latency Evaluation**

---

**Varying Interference:**

- Latency vs. Interference [cycles] for different transfer sizes.

- 4096B transfers, conventional controller.
- 4096B transfers, PRET controller.
- 1024B transfers, conventional controller.
- 1024B transfers, PRET controller.

**Varying Transfer Size:**

- Average latency vs. transfer size [bytes] for conventional and PRET controllers.

---

Reineke et al., Berkeley 45
5.3.1 Derivation of Worst-case DMA Latencies

In order to assess the value of privatization, we also determine the latency for a scenario in which the four resources of the backend can be computed from

\[ \text{latency} = \text{RFP} \cdot n \cdot s + \text{BL} \cdot x \cdot y + \text{DL} \cdot r \cdot s \]

For space reasons, we limit our analysis to the second of the two latencies for a scenario in which the four resources of the backend are shared among four clients in a round-robin fashion. These four latencies include the final refresh.

One could further distinguish between transfers from DRAM to clients could be the four threads of the PT-RM or four cores in a round-robin fashion. These four latencies for a scenario in which the four resources of the backend are shared among four clients.

We denote the latency of a transfer of size \( s \) bytes from initiating the DM transfer until the final refresh as

\[ \text{latency} = \text{RFP} \cdot n \cdot s + \text{BL} \cdot x \cdot y + \text{DL} \cdot r \cdot s \]

Due to the sharing of the resources, only we only consider the former, which incurs higher latencies. We noted before, we associate two latencies with a DM transfer.

For larger transfers, the bandwidth provided by the memory controller becomes more important, and private DRAM resources are less beneficial. This is illustrated in Figure 7. For both burst length 4 and burst length 8, the slight advantage of shared PRET over Predator can mostly be explained by the manual refresh mechanism employed in PRET. For smaller transfers, the bandwidth advantage of private resources is more pronounced.

For comparison, we have also determined access latencies for

\[ \text{latency} = \text{RFP} \cdot n \cdot s + \text{BL} \cdot x \cdot y + \text{DL} \cdot r \cdot s \]

Hiding refreshes saves latency by approximately 66\% compared with burst length 8.

For 64K transfers, burst length 8 reduces the slight flatter slope is due to fewer read-write switches and the use of shared banks.

The slight advantage of shared PRET compared with Predator can mostly be explained by the manual refresh mechanism employed in PRET. For both burst length 4 and burst length 8, the slight advantage of shared PRET over Predator can mostly be explained by the manual refresh mechanism employed in PRET.

\[ \text{latency} = \text{RFP} \cdot n \cdot s + \text{BL} \cdot x \cdot y + \text{DL} \cdot r \cdot s \]

\[ \text{Hiding refreshes} \]

\[ \text{Latencies of transfers using Predator at burst length 8 shared} \]

\[ \text{Latencies of transfers using one of the four resources at burst length 8 shared among four clients using round-robin arbitration} \]

\[ \text{Latencies of transfers using all of the four resources at burst length 8 shared among four clients using round-robin arbitration} \]

\[ \text{Private resources in backend} \]

\[ \text{“Manual” refreshes} \]

\[ \text{Hiding refreshes} \]

\[ \text{Shared Predator BL} = 4 \text{ w/ refreshes} \]

\[ \text{DL}^0_{4,4}(x): \text{Shared PRET BL} = 4 \text{ w/ refreshes} \]

\[ \text{DL}^\pi(x): \text{PRET BL} = 4 \text{ w/ refreshes} \]

\[ \text{DL}^\pi(x): \text{PRET BL} = 4 \text{ w/o refreshes} \]

\[ \text{PRET controller improves worst-case access latency of small transfers} \]
PRET DRAM Controller vs Predator: Analytical Evaluation

- Less of a difference for larger transfers
- Predator provides slightly higher bandwidth due to more efficient refresh mechanism
Outline

- Introduction
- Precision-Timed ARM (PTARM) Pipeline
- PTARM Memory Hierarchy Principles
- PTARM DRAM Controller
  - DRAM Basics
  - Related Work: Predator and AMC
  - PRET DRAM Controller: Main Ideas
  - Evaluation
- Integration into Precision-Timed ARM
Precision-Timed ARM (PTARM) Architecture Overview

- **Thread-Interleaved Pipeline** for predictable timing **without** sacrificing high throughput
- One private DRAM Resource + DMA Unit per Hardware Thread
- Shared **Scratchpad** Instruction and Data Memories for low latency access

http://chess.eecs.berkeley.edu/pret/
Conclusions and Future Work

- PTARM = Thread-interleaved pipeline + Scratchpads + Predictable DRAM:
  - Predictability without sacrificing throughput
  - Temporal isolation between hardware threads
- How to program the inverted memory hierarchy?

Raffaello Sanzio da Urbino – The Athens School
References

Related Work on Memory Controllers:

Work within the PRET project: