Final Report

Design and Implementation of an Accurate Memory Subsystem Model in SystemC

Group 11
Nan Li, Yi Wang, Huisheng Zhou and Lei Liang

December 2010

Contents

1 Problem Statement 5

1.1 Background 5
1.1 DRAM Introduction 5
1.2 Ports and Exports in SystemC 6
1.3 Open Core Protocol (OCP) in SystemC 6
1.4 Scheduling Policies 7
1.4.1 FIFO (First-Come First-Served) 8
1.4.2 Priority 8
1.4.3 Round-Robin 8
1.5 Design Requirements 9

2 Roles and Responsibilities 10

3 Design and Implementation 12

3.1 Accurate DRAM Model: Controller and Scheduler 12
3.1.1 General View 12
3.1.2 Accurate DRAM Controller 13
3.1.3 DRAM Scheduler 14
3.2 Accurate DRAM Model: Core Timing Module 15
3.2.1 Requirements 15
3.2.2 System Design 16
3.2.3 System Implementation 21
3.3 Reference DRAM Model 26
3.3.1 General Design Description 26
3.3.2 Design and Implementation 27
3.4 Test Module 29
3.4.1 Basic Design Idea 29
3.4.2 Test Module Implementation 30
3.4.3 Test File Generator 31
3.4.4 Test Generator 31

4 Testing and Results 33

4.1 General Testing Configuration 33
4.2 Test 1 - Refresh 34

4.2.1 Test Generation Constraints 34
4.2.2 Description 34
4.2.3 Testing Results 35
4.2.4 Comparison with Reference Model 36
4.3 Test 2 - Address 36
4.3.1 Test Generation Constraints 36
4.3.2 Description 37
4.3.3 Testing Results 37
4.3.4 Comparison with Reference Model 39
4.4 Test 3 - Read/Write 39
4.4.1 Test Generation Constraints 39
4.4.2 Description 40
4.4.3 Testing Results 40
4.4.4 Comparison with Reference Model 41
4.5 Test 4 - Throughput - Burst Length 42
4.5.1 Test Generation Constraints 42
4.5.2 Description 42
4.5.3 Testing Results 43
4.6 Test 5 - Throughput - Address Range 44
4.6.1 Test Generation Constraints 44
4.6.2 Description 45
4.6.3 Testing Results 45
4.7 Test 6 - Request Generating Rate 46
4.7.1 Test Generation Constraints 46
4.7.2 Description 46
4.7.3 Testing Results 46
4.8 Test 7 - Scheduling Policy 47
4.8.1 Test Generation Constraints 47
4.8.2 Description 48
4.8.3 Testing Results 48
4.8.4 Comparison with Reference Model 50

5 Conclusion 51

5.1 Generalization of Results 51
5.2 Future Work 51

6 References 52

7 Appendix - Project Management Report 53

7.1 General Summary 53
7.2 Follow-up of Objectives 53
7.3 Lessons Learned and Suggestions for Improvement 53
7.4 Final Comment 54
7.5 Final Time Plan 54

1 Problem Statement

1.1 Background

Recently, system-level performance analysis has attracted great attention [1], but most works in this area are focused on evaluation of communication networks, while using a simple ideal model for the memory subsystem. Such models can be dummy slaves that respond immediately, or simply applying a fixed delay. Using a simple model in performance evaluation may generate an over-optimistic result, which may later fail to achieve while implementing the system with a realistic memory module. Moreover, with the complexity of memory controllers increased dramatically, the performance of the memory varies a lot in different situations, making it insufficient to model the memory using a fixed latency.

This brings the necessity to build an accurate memory model for system evaluation. In simulation point of view, the most important thing for a memory model is to accurately model different types of latencies. Nowadays, Double Data Rate (DDR) SDRAMs are very widely used as memory devices. The latency parameters for different types of memory (for example, DDR1, DDR2 and DDR3) may differ. In [2], Srinivasan and Salminen have suggested several types of latencies to consider in a DRAM model that are most significant to the performance.

1.1 DRAM Introduction

Due to its special architecture, DRAM holds many different features from its counterpart SRAM.

Firstly, accessing time is address-dependent. A DRAM is composed of several banks. Each bank is an array of storage units. All units of the same column share one data line. This means that each time only one row in the bank can be accessed; otherwise, data will be corrupted. Now suppose you have just finished an operation on row 0 of bank 0. If you are going to access row 1 of bank 0, you will have to close row 0, open row 1, and

finally perform your operation, resulting a long accessing time. Nevertheless, if you still access row 0, you can skip that close open toggling, resulting a short accessing time.

Secondly, storage gets refreshed periodically. DRAM is a dynamic logic device, which means the information is stored on capacitors instead of registers. Due to charge leakage, all DRAM storage units need to be refreshed regularly to keep the integrity of the data. During refresh, no operation is allowed for the DRAM. After refresh, all open rows are closed. Additional time will be spent on open the row before a read or write access.

Finally, accessing the DRAM is burst oriented. Burst access fits the real application situation where a running process usually request consecutive RAM addresses. Also, since modern DRAM is clocked at a lower frequency than the system, burst access is must.

1.2 Ports and Exports in SystemC

A great feature of SysemC is that it separates computation part and communication part of the system. It has a higher level abstraction of communication modeling, which use function call replacing pin-level signal transferring. This greatly facilitates the modeling work and greatly reduces simulation costs. The most explicitly use of this feature is use “port” (sc_port) and “export” (sc_export).

For a particular communication, the initiator (like a master) can instantiate a sc_port and register a SystemC interface to it. The target (like a slave) can instantiate a sc_export and implement the interface registered at master. When sc_port is bounded to sc_export, the initiator can call the functions defined in target through a sc_port call.

1.3 Open Core Protocol (OCP) in SystemC

Open Core Protocol is an openly-licensed protocol that describes system-level integration requirements of components. Open Core Protocol International Partnership (OCP-IP) provides a convenient tool kit for development of OCP-compatible components in SystemC, called OCP TLM Kit. Several abstraction levels are provided in

this tool kit: TL1, TL2, TL3 and TL4, within which the only cycle accurate level is TL1. Since cycle accuracy is a must in the requirement of our project, TL1 is the level we will be working with.

In TL1, communication between modules is abstracted into transactions, and passed from one module to another through OCP TL1 sockets. Passing of transactions is done with nb_transport_fw() call by master module, and nb_tranport_bw() call by slave module. Master module has to register a call back function for nb_transport_bw() call, and slave module has to register a call back function for nb_transport_fw() call.

A phase is assigned each time a transaction is passed. There are four basic phases in TLM: BEGIN_REQ, END_REQ, BEGIN_RESP and END_RESP, and OCP extended them with additional BEGIN_DATA and END_DATA phases for single-request multiple-data (SRMD) burst transfer. Detailed requirements for allowed phase transformation can be found in [3].

An OCP transaction can have different kinds of extensions. Each OCP socket has a set of configurations showing which extensions it supports. These configurations can be read in from a file during runtime.

Each OCP TL1 transaction is a burst of data words. At most one word (matching the size of bus width) is transferred in a cycle. Each transfer is called a beat. There are two types of bursts: multiple-request multiple-data (MRMD) and single-request multiple-data (SRMD). In MRMD burst, one request is sent for one data word, thus for a burst of length L, L requests are sent. In SRMD burst, only one request is sent for the entire transaction. Specially, in a write SRMD burst, after the request (which contains no data), outstanding data phases (with BEGIN_DATA and END_DATA) are used to transfer the data to be written.

1.4 Scheduling Policies

If a master has multiple threads running, it is possible to put a scheduler between the master and the memory controller, in order to achieve better performance either of

certain threads, or of the memory usage. There are several kinds of scheduling policies. The most commonly used are the following:

1.4.1 FIFO (First-Come First-Served)

All the requests are put in a single queue, and sent to the memory controller in the order that they come in.

Such policy ensures timely fairness, but might not achieve the best memory usage and lowest average delay.

1.4.2 Priority

Priority scheduling policy assigns each thread with a fixed priority. Each thread has its own request queue. When more than one queue is non-empty, the request from the thread of the highest priority is served.

Such policy ensures lowest delay for requests from high priority threads, but those from low priority may starve.

1.4.3 Round-Robin

In round-robin scheduling, each thread take turns to have the highest priority. The length of time a thread keeps the highest priority is the slot size of this thread. In simple round-robin scheduling, slot size is the same for each thread. In weighted round-robin scheduling, the slot size can be different for each thread.

Such policy ensures some timely fairness, and increases overall memory efficiency, because it reduces the number of switching of threads (resulting in fewer row hops).

1.5 Design Requirements

In this project, we need to build an accurate DRAM model in SystemC based on [2]. The basic requirements for the model are:

- Simulation oriented.
  The model is used in system-level evaluation, which means simulation efficiency is a major concern, and pin-level accuracy is not necessary. So it is more desirable to use transaction-level function calls instead of signal assignments in inter-module communication.

- Cycle accurate.
  According to [2], the delay of DRAM access is divided into several parts. So, it can be calculated accurately based on the current state of the DRAM.

- OCP compatible.
  Open Core Protocol (OCP) is used in many system-on-chip applications, and is strongly supported by industry. Our design needs to be compatible with OCP standard, so it can be easily plugged into existing OCP-compatible platforms to perform system-level simulation.

- Configurable.
  The model needs to be highly configurable, so it can be used for various types of DRAMs.

Besides the accurate model, we need to build a simple memory model as reference so we can compare it with the accurate model. This model should be able to give the response immediately, or with a fixed delay.

A test module which can generate requests of certain patterns is also needed. It should send request of different kinds, collect the responses, and compare the performance of the accurate model with the simple reference model.

All modules should be created using OCP-IP SystemC TLM Kit [4], thus compatible with OCP standard.

2 Roles and Responsibilities

The project is partitioned into four parts, as illustrated in Fig. 1.

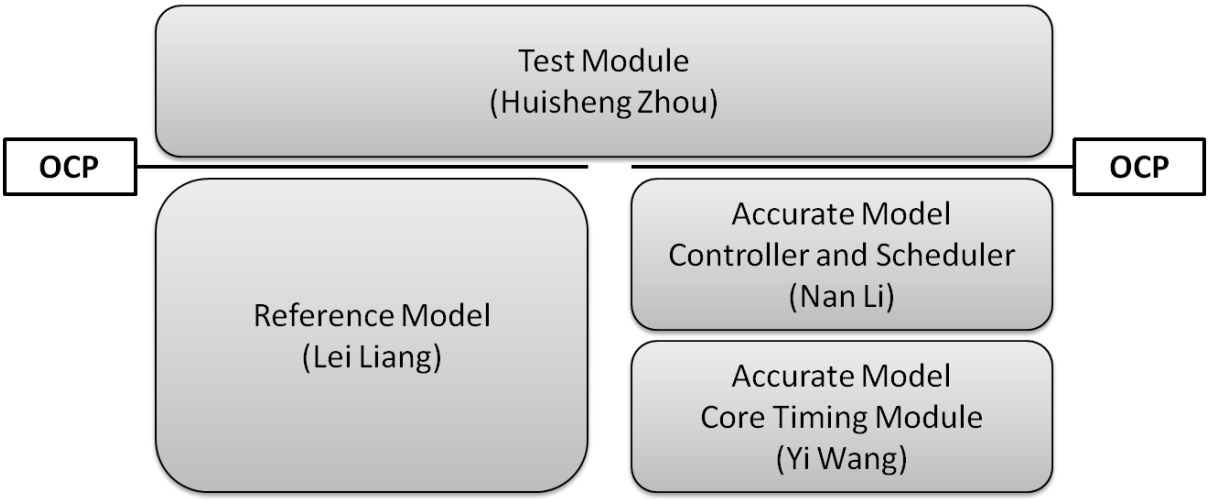


Fig. 1 Project Partition

Considering the work balance of project members, we divide the accurate DRAM model into two parts: (a) a protocol dependent controller and scheduler part, and (b) a protocol independent core timing module. Part (a) deals with all protocol related issues, while part (b) models DRAM timing behavior. A simple internal interface is designed to couple these two parts. The same external clock will be used for both the controller and the core timing module, so the modules are synchronized and reflect the behavior of a real DRAM.

Both the reference model and the accurate model are implemented as OCP TL1 slaves, while the test module is implemented as an OCP TL1 master. Some of the OCP extensions will be implemented, such as burst and thread-id support.

The test module will be capable of automatically generating transactions of certain patterns. The statistical characteristic of these transactions can be controlled.

The responsibilities of group members are assigned as the following:

Nan Li (group leader) is mainly responsible for managing team work, designing and implementing the DRAM controller and scheduler of the accurate DRAM model.

Yi Wang (group member) is mainly responsible for designing and implementing the DRAM core timing module of the accurate DRAM model.

Huisheng Zhou (group member) is mainly responsible for designing and implementing the test module, and recording test results.

Lei Liang (group member) is mainly responsible for designing and implementing the reference model, and comparing the test results of the reference model and the accurate model.

3 Design and Implementation

3.1 Accurate DRAM Model: Controller and Scheduler

3.1.1 General View

The controller module acts as an interpreter that translates OCP requests into internal requests, and also converts internal responses into OCP responses. Some manipulation on requests and responses might be done, such as cutting OCP requests for long data into several internal requests for smaller data.

The scheduler module can collect requests from different masters, and schedule, or even reorder them so the memory can handle them most efficiently.

Since the scheduler is application specific, we decided to build it as a separate module. It will have both a master socket and a slave socket. So the whole accurate model will look like Fig. 2.

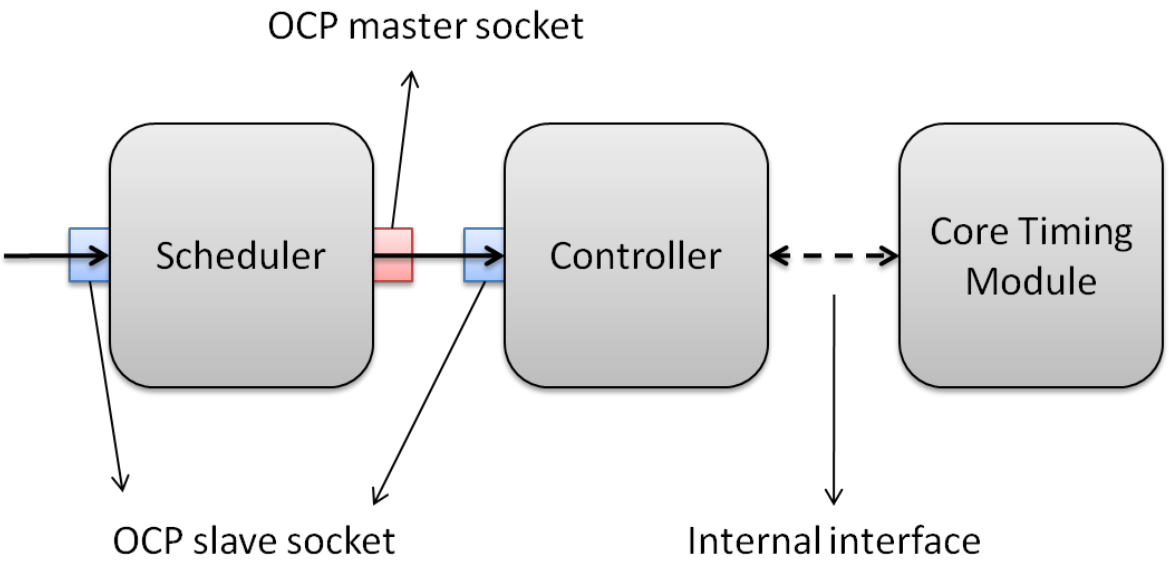


Fig. 2 Accurate DRAM model block diagram

3.1.2 Accurate DRAM Controller

The structure of the controller is shown in Fig. 3.

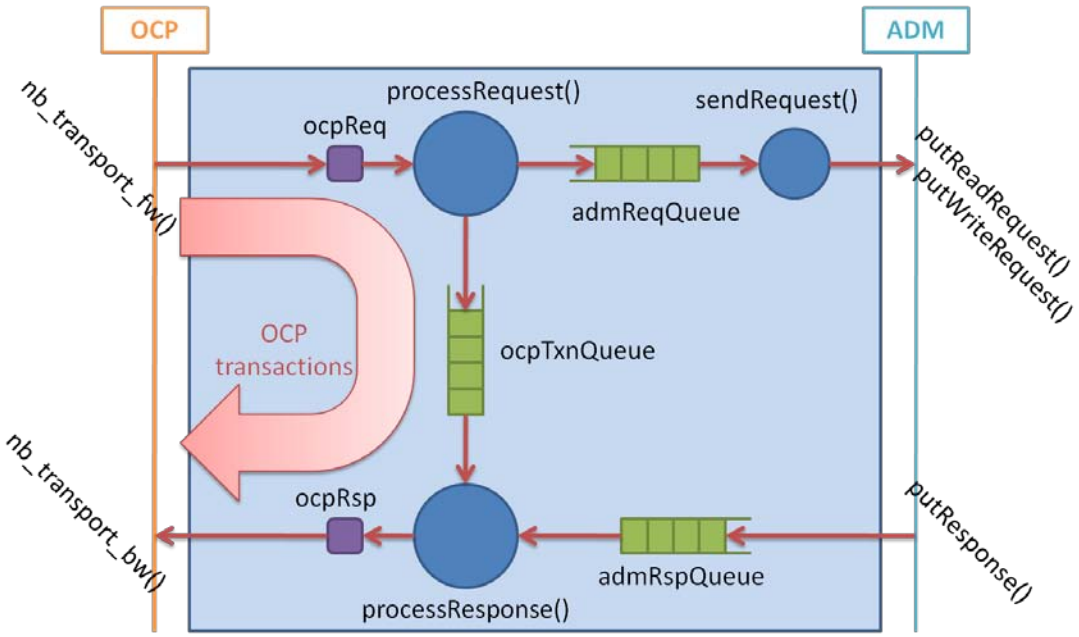


Fig. 3 Accurate DRAM controller module

The incoming OCP request is stored in ocpReq. The processRequest() procedure processes the incoming request into one or more internal requests and put them into admReqQueue, and also put the original request into ocpTxnQueue, in order to keep track of the transactions that are currently being processed. At each clock cycle, the sendRequest() procedure takes one element from admReqQueue, and tries to send the request to the core timing module via the internal interface. If sending is successful, the element is taken away from the queue. Responses from the core timing module are stored in admRspQueue, and the processResponse() procedure takes one or more elements from this queue, and writes them into the corresponding OCP transaction from ocpTxnQueue, and store the transaction in ocpRsp, which will be later sent to the master through the backward transport channel.

According to OCP specification, there are two types of write commands: non-posted write and posted write. A non-posted write command expects a response (or an acknowledgement), while a posted write command does not. Therefore, the controller

treats these two types of writes differently. More specifically, for non-posted write, the controller puts the write request into ocpTxnQueue, but not for posted write.

The three queues in Fig. 3 are simply implemented with C++ STL container std::deque.

Several interface functions are provided by the DRAM controller. The interface function provided by the DRAM controller via the OCP interface:

```
tlm_sync_enum nb_transport_fw(tlm_generic_payload& txn, tlm_phase& ph, sc_time& tim)
```

The interface functions provided by the DRAM controller via the internal interface:

```
bool putResponse(const adm_data &rsp)
```

3.1.3 DRAM Scheduler

Our DRAM scheduler is a simplified model. Requests from the master are put into queues of infinite size. Based on the scheduling policy, the scheduler selects a queue and sends the first request of the queue to the memory controller.

The structure of the scheduler is shown in Fig. 4.

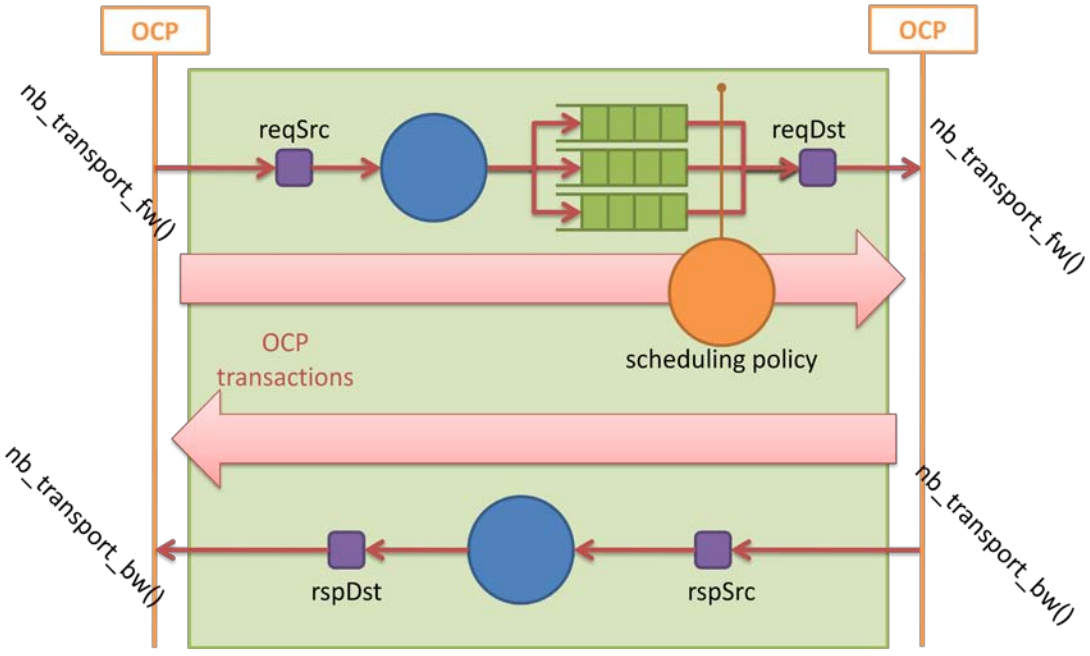


Fig. 4 DRAM scheduler module

There are three scheduling policies implemented:

- FIFO
  The incoming requests are put in the same queue, so they are sent to the controller in the same order as they are received.

- Priority
  The incoming requests are put in separate queues according to their thread_id extension. The request sent to the controller is always taken from the non-empty queue of the highest priority.

- Round-robin
  When round-robin policy is used, the slot size for the threads (in cycles) needs to be specified. When a thread gets the highest priority, a slot size usage counter will be set to the slot size value. Each clock cycle, the counter will decrease by 1. As long as the counter is above zero and the queue is non-empty, the request sent to the controller is taken from this queue. When the counter reaches zero, or the queue becomes empty, the highest priority is given to the next thread with non-empty queue.

The scheduler is both an OCP master and an OCP slave, thus it provides interface function for both:

```
tlm_sync_enum nb_transport_fw(tlm_generic_payload& txn, tlm_phase& ph, sc_time& tim)

tlm_sync_enum nb_transport_bw(tlm_generic_payload& txn, tlm_phase& ph, sc_time& tim)
```

3.2 Accurate DRAM Model: Core Timing Module

3.2.1 Requirements

- Delay behavior oriented
  This requires that the delay model should keep itself on a high level. The focus is on modeling the delay itself, but never on modeling the processes that causes delay.

- Cycle accurate
  This requires that the interval of 2 events of this delay model should be both deterministic and correct.

- Configurable
  This requires the high adaptability and extensive use of the model. All key parameters used in model should be configurable to fit different simulation contexts. This also allows the modeling of some ideal situations, and extreme situations.

3.2.2 System Design

3.2.2.1 Delay Behavior Oriented

As required by the first requirement, this model should be a high level model. So we decide to abstract the DRAM accesses as request-response transactions. For a read access, request is the address, and the response is the fetched data. Since write does not get response within the range of this sub model, it is treated as pure request.

There are 2 beauties for this perspective.

Firstly, this model is a part of an OCP-IP TL1 system, this perspective is on the same level with OCP-IP TL1. Choosing this perspective makes this model more compatible with the other parts of the whole system.

The second one makes more sense. It provides a clear blueprint of the model. In this perspective, DRAM accesses are request-response transactions. Then, modeling DRAM behavior is equivalent to modeling transaction. And modeling transaction is equivalent to modeling both request behavior and response behavior. As this model should be delay behavior oriented, all this model should be about request delay and response delay.

There are large amount of factors affecting the DRAM delays, making it difficult to get a clear picture. But if we see everything from the request-response point of view, it will be easier to pick up those dominant factors by inspecting correlation with and influence on the transactions. The following are 5 dominant factors picked up:

- Preparation delay (tPreparation)
  Prepare delay is the time consumed to get the requesting row open, if it is not at the open state.

- Pre-effect delay (tPre)
  In a read situation, it is the time between the start of a read access and the first burst data pops out. In a write situation, it is the time between the start of a write access and the first data get registered.

- Burst delay (tBurst)
  Burst delay is the time the DRAM needs to finish a burst.

- Post-effective delay (tPost)
  Post-effective delay is the switch time for a write to read switch.

- Refresh delay (tRef)
  Refresh delay is the time it takes the DRAM to complete a refresh. During this period, DRAM is not allowed for any read/write access. Apart from that, after refresh, all the open rows are closed.

3.2.2.2 Cycle Accurate

In this particular design, we need 2 facilities to guarantee the cycle accuracy. The first is that the system should have a deterministic and accurate algorithm to calculate the delays. The other is that the model should at least contain one clocked process that can perform all these delays and tweak the system state accordingly.

A. Delay Calculation Algorithm

a) Request Delay

We define request delay as the minimum time between 2 consecutive request acceptances. Generally we do not want the coming request interfere the serving of the current request, so DRAM will not accept another request until it completes the current one. But thanks to the pipeline technology, in some particular cases, the DRAM can forget about this and accept request earlier.

Pipelined

If the coming request meets the following 2 requirements simultaneously, it gets pipelined, meaning that it gets accepted before DRAM completes serving the current request.

- Requesting row is currently in open state
- Has the same request type (read/write) as the current request

Here since requesting row is open, tPreparation is not necessary. And due to pipeline, tPre get masked. Also since request type not switched, tPost take no effect either. The only affecting factor is tBurst. So we get:

$$\text{Request Delay} = tBurst$$

Pipeline Stalled

If the coming request cannot meet the 2 requirements simultaneously, it has to wait for the current request getting fully served. Typically, it takes a minimum of tPre + tBurst to fully finish serving one request.

Besides, if row opening is involved in current request, the coming request will wait another tPreparation. Also, if current request is a write, the coming read request will have to wait even longer, since it takes a tPost to do a write to read switch. Finally, if the current request burst is lucky enough to cross the refresh period boundary, the coming request will be further delayed by tRef.

In conclusion, we get:

$$\text{Request Delay} = \langle tPreparation \rangle + tPre + tBurst + \langle tPost \rangle + \langle tRef \rangle$$

b) Response Delay

We define response delay is the time between the acceptance of a read request and the first data of the burst pops out. From this definition, it is quite straightforward that tPre is necessarily affecting here. If row opening is involved, additional tPreparation should be added. So we get:

Response Delay = tPre + <tPreparation>

It should be mentioned that the refresh does not affect response delay. Even a burst crosses the refresh period boundary, the refresh will start only after the burst completes.

B. Clocked Units

Here we need 4 units. 2 are used for performing request delay, one for read and the other for write. Another used for performing response delay, and the final one for countdown the refresh period.

The 2 units for request delay can be modeled as countdown timer. When it gets an initial value, it starts countdown to 0, and keeps idle until the next initial value comes.

The unit for response delay can be modeled as a ticker, since it needs to check whether to give the response every cycle.

The unit for refresh countdown is can be modeled as a round timer. The last few cycles are defined as the cycles in refresh.

3.2.2.3 Configurable

Configurability is achieved by parameterize the key factors of the DRAM and some interfaces should be provided for the user to set these parameters according to their own simulation requirements. The criterion for choosing the parameters is that they should have correlation with or influence on the request or response transactions.

The model should be able to get these parameters from a configuration file. This file should be text based. Considering future integration with the controller, it should have the same format with the controller.

The list of provided parameters is shown in Tab. 1.

Name	unit	Remarks
clock period	ns	the system clock period
refresh period	ns	refresh period of the DRAM
refresh duration	ns	duration of each refresh operation
address width	bits	Width of the address bus
bank address width	bits	width of bank region of the address
row address width	bits	width of row region of the address
column address width	bits	width of column region of the address
word length	bytes	width of the data bus
minimum burst length	bytes	length of a minimum burst of the DRAM
hop row delay	ns	time consumed to one hop row operation
open row delay	ns	time consumed to open a row
tCAS	ns	time for DRAM to fetch the first READ data
tDQSS	ns	time for DRAM to register the first WRITE data
tWTR	ns	time for DRAM to perform a WRIT to READ switch

Tab. 1 Provided configuration parameters

3.2.3 System Implementation

3.2.3.1 System overview

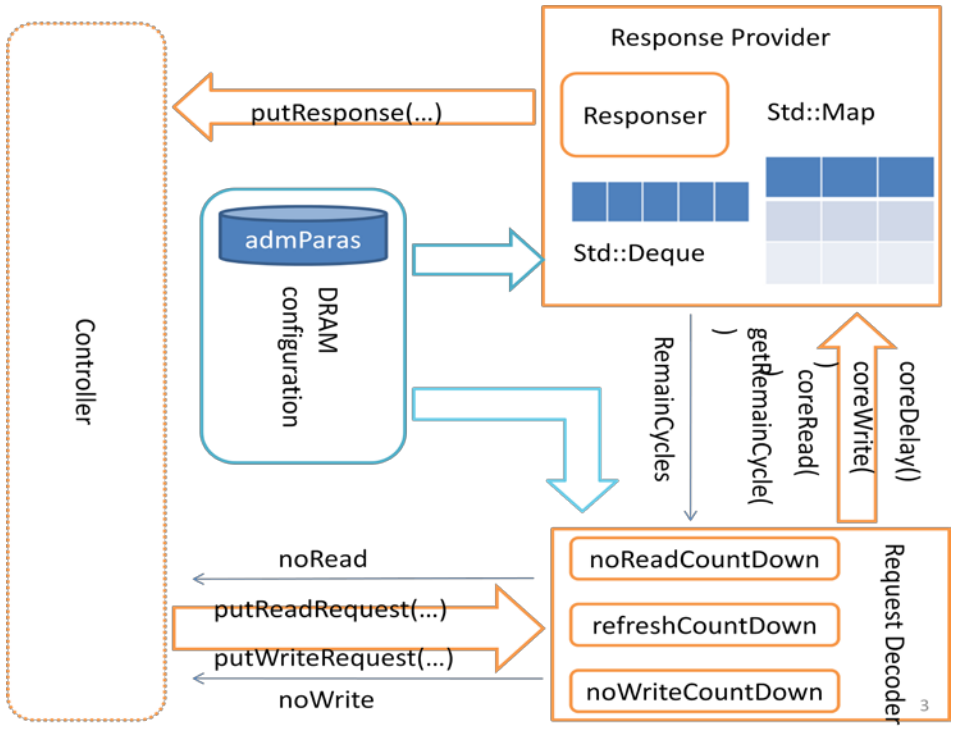


Fig. 5 Architecture of the sub model

Fig. 5 is the overall structure of the sub model. There are several components here:

- The blue rectangular is a basic C++ object
- The orange rectangular is a SystemC module object
- The orange rectangular with round corners is a clocked process
- The blue hallow arrow implies passing the object as a parameter to the constructor of another class
- The blue narrow arrow implies a return value
- The orange hallow arrow implies a sc_port → sc_export connection.

- The cylinder means a data source, which is a file here.

As you can see in Fig. 5, there are 3 major parts of this sub model.

- DRAM Configuration
  As its name implies, is used for configure this delay model

- Request Decoder
  A SystemC module that calculates request delay and response delay, performs request delay and order the Response Provider to perform response delay

- Response Provider
  A SystemC module that performs response delay and where storage space exists.

3.2.3.2 Interaction with Controller

The controller is an OCP-IP TL1 device. Since OCP-IP encapsulate everything in transactions, the delay model should not act like a real DRAM, bursting one data out each cycle. So we decide to also transfer packets between controller and the delay model. The package is defined as a basic C++ object holding the length of the burst and an array holding all burst data.

In order to ensure correct functionality after this mode is integrated with the controller. We determine some synchronization mechanism. The most convenient way to do this is using the delta delay. According to our functionality requirements, the synchronization will be as follows:

Time 0: delay model updates state

Time 0 + delta: controller issues request

Time 0 + 2 delta: delay model provides response

Time 0 + 3 delta: controller processes response

3.2.3.3 DRAM Configuration

DRAM configuration is implemented as a basic C++ object. Its constructor will read all information from the configuration file and convert the unit of the timing parameters from "ns" to "cycles". Then the configuration object is passed to the constructor of Request Decoder and Response Provider.

The configuration file uses OCP parameter file format which is also used by the controller. Each line of the file represents one parameter. It looks like:

```
clockPeriod i:20
```

"clockPeriod" is the name of the parameter. "i" is the data type of the parameter. "20" is the given value.

3.2.3.4 Request Decoder

Request Decoder is a SystemC module consists of a sc_export with implementation of methods putReadRequest and putWriteRequest, a sc_port to initiate a transaction towards the Response Provider, 3 clocked units for read request delay, write request delay and refresh, and several fields retaining DRAM's state, mainly the number of open row in each bank. The overall architecture is shown in Fig. 6.

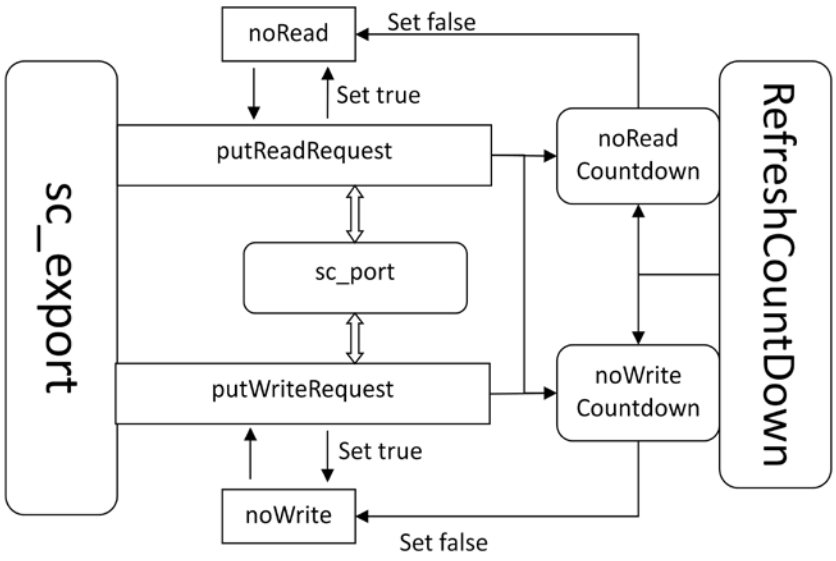


Fig. 6 Architecture of Request Decoder

noRead and noWrite are two flags that indicates whether the DRAM can accept the coming request of that type. If a flag holds a "true" value, it means that the request delay has not elapsed, and the coming request will get rejected.

Flags get "true" value only from two sc_export methods. They are set "false" only when the corresponding countdown counts to zero. Two countdown processes get their initial value from two sc_export methods but the counting value may be changed by the refresh countdown process if refresh occurs. Only RefreshCountdown part is allowed to change the noReadCountdown and noWriteCountdown during counting.

If refresh start point comes, the refresh countdown process will add additional tRef to the current count value of both countdown process, resulting in even longer request delay.

The controller can call putReadRequest and putWriteRequest through sc_export at a clock edge to put request to the Request Decoder. Both methods have a Boolean return value indicating whether the request has been accepted or rejected. Both methods implement the algorithm defined in A of "System Design" section when comes to calculating delays. The work flow is shown in Fig. 7.

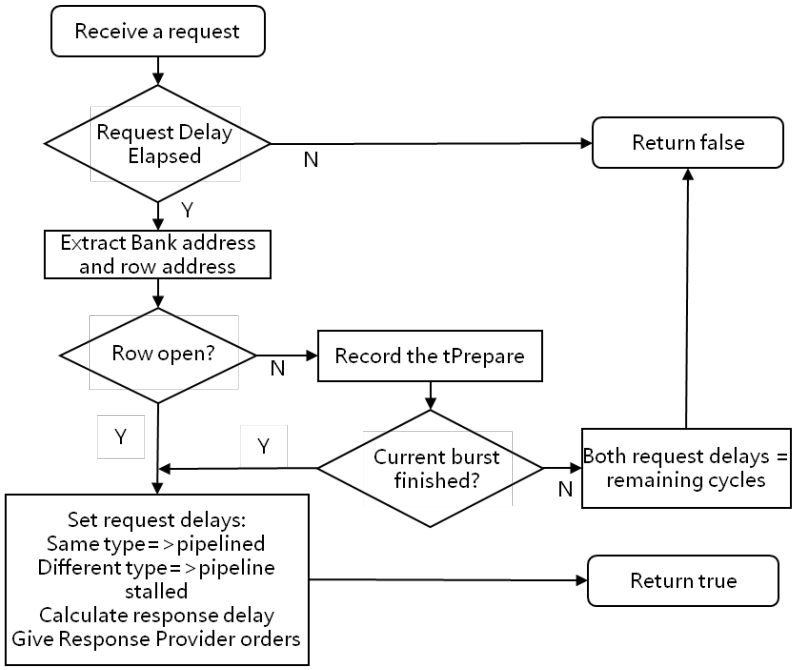


Fig. 7 Work flow of the putReadRequest/putWriteRequest methods

3.2.3.5 Response Provider

The Response Provider is designed to performs response delay and provide storage space. It consists of a sc_export with implementation of 4 methods, one clocked process to put response, one std::deque and one std::map. Its overall structure is shown in Fig. 8.

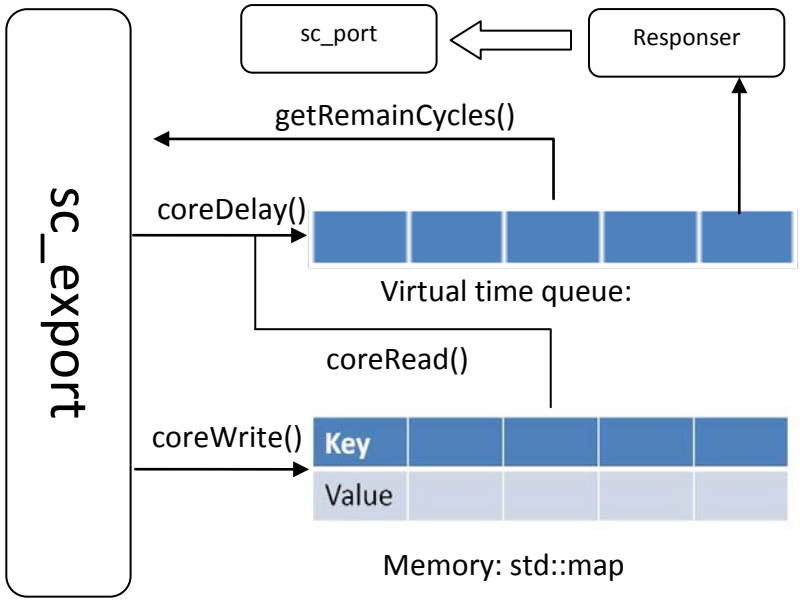


Fig. 8 Architecture of Response Provider

When trying to model the response delay, traditional wait() method is not enough, because, it suspends the module until the serving completes. This makes it impossible to model the pipeline behavior. So we decide to use virtual time instead of real time.

As shown in figure 5, the virtual time is implemented by a queue data structure. Each slot in the queue represents a clock cycle. The response delay in virtual time is achieved by adding dummy elements (packets with length 0) before the response packet. Every cycle the "Responser", which is a clocked process, will pick up an element from the other end of the queue. It calls the putResponse() method only when it gets a valid response packet. Thus, we ensure cycle accuracy. In this way, pipeline is easy to model by reducing the dummy ahead of the response packet.

The memory part is the place to store the write data and the data source for a read request. It is insensible to occupy the computer RAM according to the volume of the DRAM we models. A better way is that the RAM space only gets occupied only after it is used.

In order to make things easy, we decide to use C++ standard library to implement the virtual time queue and the memory. Deque and Map not only provides all features we need, but also are provided right with any C++ compiler which is a must for using SystemC. The user does not need to install additional components in order to run this model.

3.3 Reference DRAM Model

The reference model is a simple DRAM model which is frequently used in NoC nowadays. It is an ideal model whose behavior is greatly different from the real DRAM. It has the same reaction delay to the different sequence of Read and Write operations. This reference model is designed to make comparison with the accurate model. It will be connected to the test module and certain tests will be carried on it. Though analysis the different behavior between the reference model and accurate model, we will see how the accurate acts like the real DRAM much more.

3.3.1 General Design Description

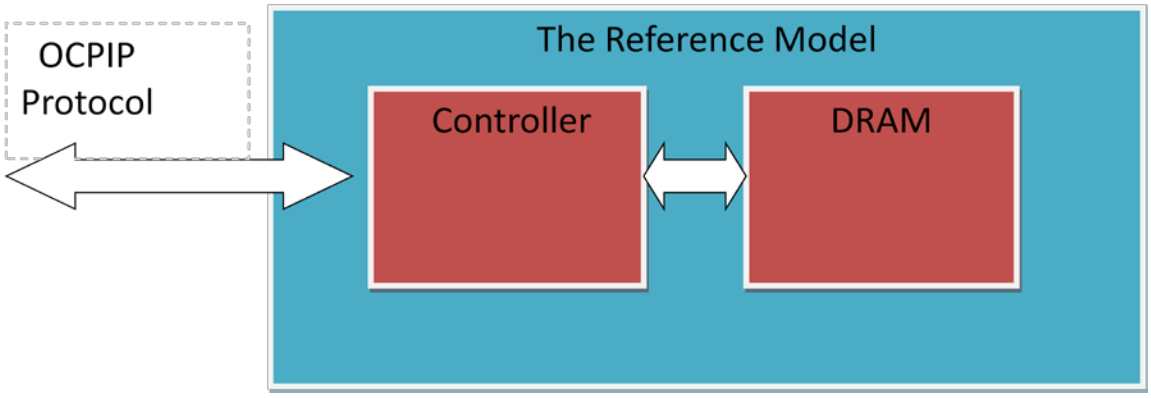


Fig. 9 Reference model

As shown in Fig. 9, the reference model is designed into two parts. The controller is used to communicate with outside though OCP-IP protocol. It receives Read and Write operation and make certain response to both DRAM and outside world. The DRAM is a model to store data. It also has some simple feature as the real DRAM, such as certain delay to read operation.

3.3.2 Design and Implementation

The components in the reference model are shown in Fig. 10.

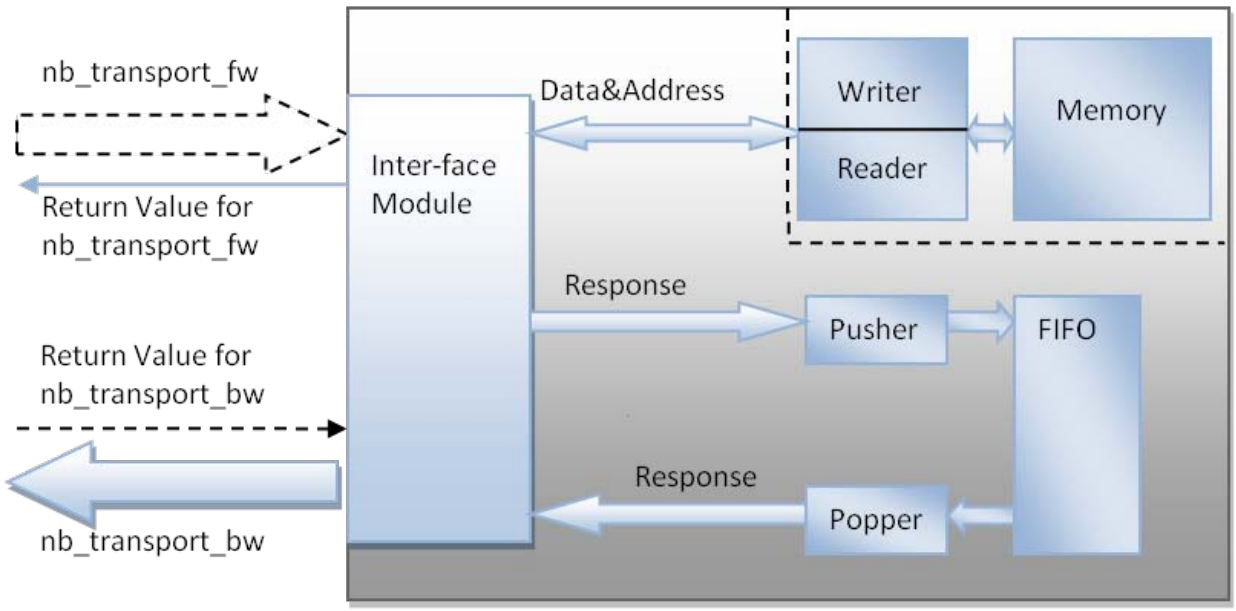


Fig. 10 Implementation of reference model

The interface module translates the information from OCP-IP transaction into several operations. According to the operations, data will be written into or read from the memory. All the response commands are put into the FIFO first. After certain cycles, these commands will be popped out to the OCP-IP transaction.

3.3.2.1 Controller

The controller is implemented in two parts which are interface module and delay module.

The interface module is used to communicate with outside through OCP-IP protocol and execute different operations. Every clock cycle, according to the commend received, the interface module need to decide whether to write date from the transaction into DRAM or put the read data from DRAM into transaction, whether to send Response commands to Pusher or not. It also need to block the input port if the operation has not yet finished or the Response command has not been accepted.

Since the Response commands need to be sent out after certain delay, delay module is needed here. The delay module is implemented in three parts: FIFO, Pusher and Popper.

FIFO is used to implement the certain delay. As shown in Fig. 11, each member in FIFO contains two data fields. One is a pointer which points to one OCP-IP transaction. One is a Boolean Flag to verify whether the pointer is valid or not. At the start of the program, some members with a invalid pointer are pushed into the FIFO. The number of members pushed into the FIFO is the cycle of delay. After that, one member will be popped out from the FIFO and one member will be pushed into the FIFO at each cycle.

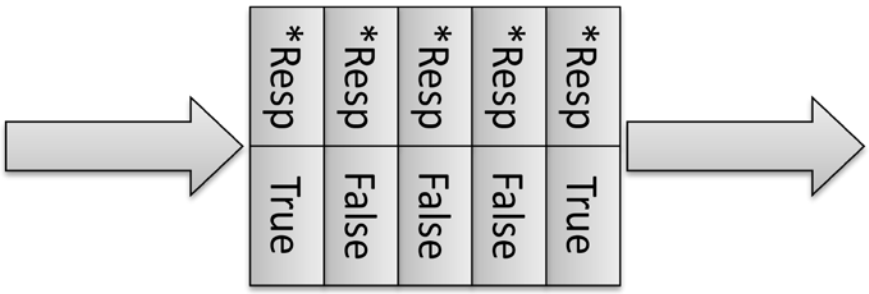


Fig. 11 Members in FIFO

Pusher is used to generate the member and put it into the FIFO. If no operation at the operation cycle, a Pointer with the false flag will be put into the FIFO. However, if there are write or read operation, the pointer which points to the ongoing OCP-IP transaction and a true flag will be put into the FIFO.

Popper is used to send Response to the OCP-IP transaction. In each cycle, if the member popped out from FIFO contains a valid transaction pointer, a Response is sent out. No operation will be carried out if the member contains an invalid pointer.

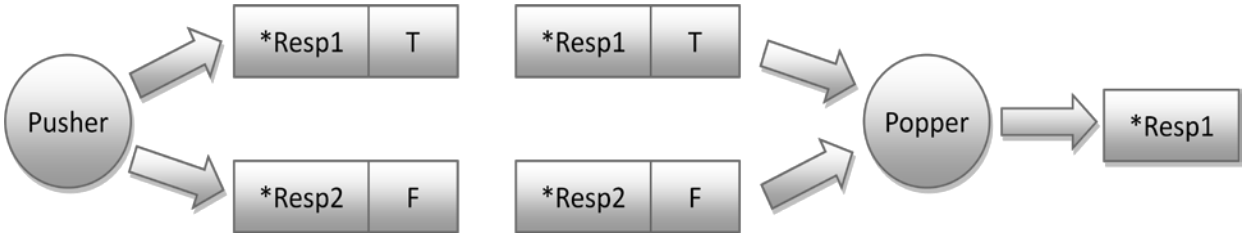


Fig. 12 Input and output data from Pusher and Popper

3.3.2.2 DRAM

The DRAM contains three parts which are writer, reader and memory.

In the OCP-IP transaction, every data is 32-bit. But in the real DRAM, every address corresponds to an 8-bit data. So writer need to split the data into four parts and then write into the memory. And the reader needs to combine the data from four addresses and send out.

Since modeling one large memory through creating exactly the same space is not efficient, mapped data structure is used to solve this problem. With this method, the space can be created according the use of the memory.

3.4 Test Module

Test module design is part of the accurate model design work. It is a master module which generates and send stimulus to the dram module and test its performance.

3.4.1 Basic Design Idea

The test module is designed at TLM level, the interface between test module and memory subsystem follows the OCP-IP standard. The test module is first designed as a simple test module with a write request and read request in order to get a clear idea on how to use OCP-IP standard. After that simple module, a lot of other stimulus is added in order to make the test module more accurate and efficient. The statistics which had been tested are recorded in a log file so that it would give a clear view on dram

performance. At last, a test file generator is designed to make the test module more efficient.

3.4.2 Test Module Implementation

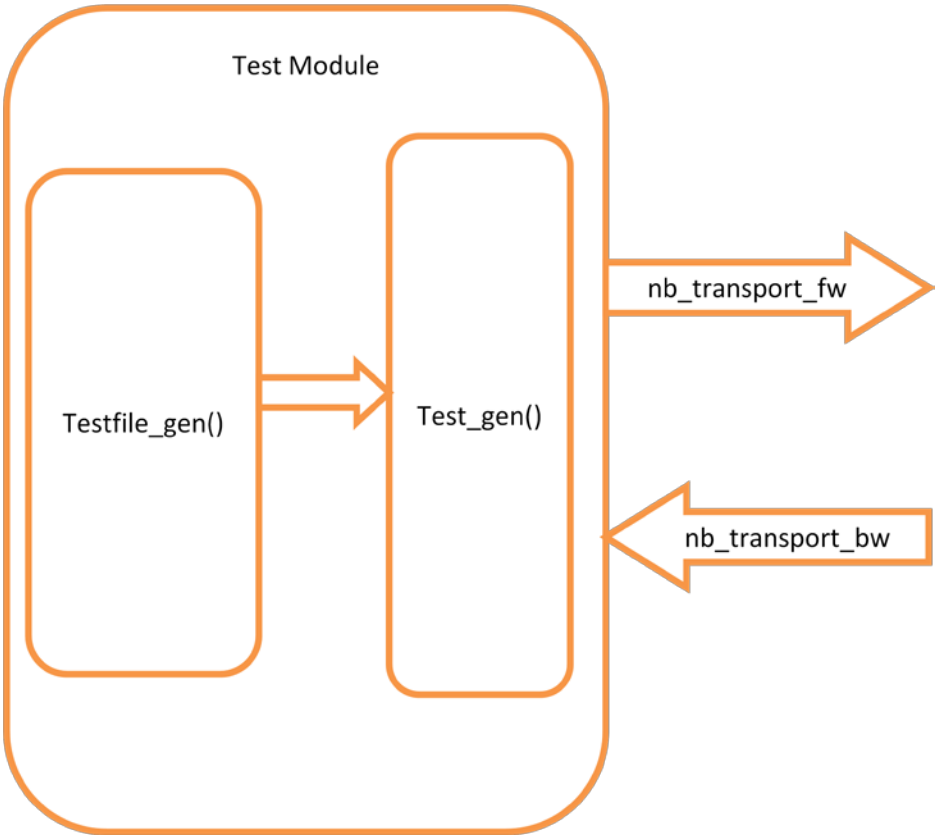


Fig. 13 Test module

Fig. 13 shows the structure of the test module. It consists of two parts: Testfile_gen() and Test_gen(). Testfile_gen() is a test file generator which generate test file for different tests. Test_gen() is the main part of the test module. It will send request and receive response from the memory and calculate latency and record them into a log file. Test module calls nb_transport_fw() to send request and get return value and phase from memory.

3.4.3 Test File Generator

The test file generator is implemented to generate certain kind of stimulus or random test stimulus. The test file has five factors: mode, time, address, thread id, burst length. Mode is represented as ".w" (means write mode) or ".r" (means read mode). Time means when the request is sent out, it is represented as cycle. Address is represented in hexadecimal. Thread id represent which thread send the request and burst length represent the how long the burst is. ".e" indicates end of test file. The following is an example of a test file.

```
.r 0 0x25fc 0 4
.w 11 0x242a 0 4
.w 13 0x17c 0 4
.r 20 0x2b78 0 4
.e
```

With the test file generator, you can generate request with random time, address, thread id or consecutive time, address or fixed address, time and burst length. You can also generate test file with multiple threads.

3.4.4 Test Generator

The test generator reads the commands from test file and then sends requests according to the commands and then collects the statistics and calculates delay. When all requests are sent and all responses are received, the test generator will make a log file to record all the necessary information.

The test generator has 3 threads: test(), handleResponse() and countCycle() and 2 functions: burstWrite(), burstRead(). The test() thread will read the commands from test file and then call the functions according to the commands. The burstWrite() and burstRead() functions are the main part, they will generate the transactions and send request to the memory. The handleResponse() thread will handle the response returned from memory and release the transaction. The countCycle() is used to record the time cycle and calculate delay. The structure of test generator is shown below in Fig. 14.

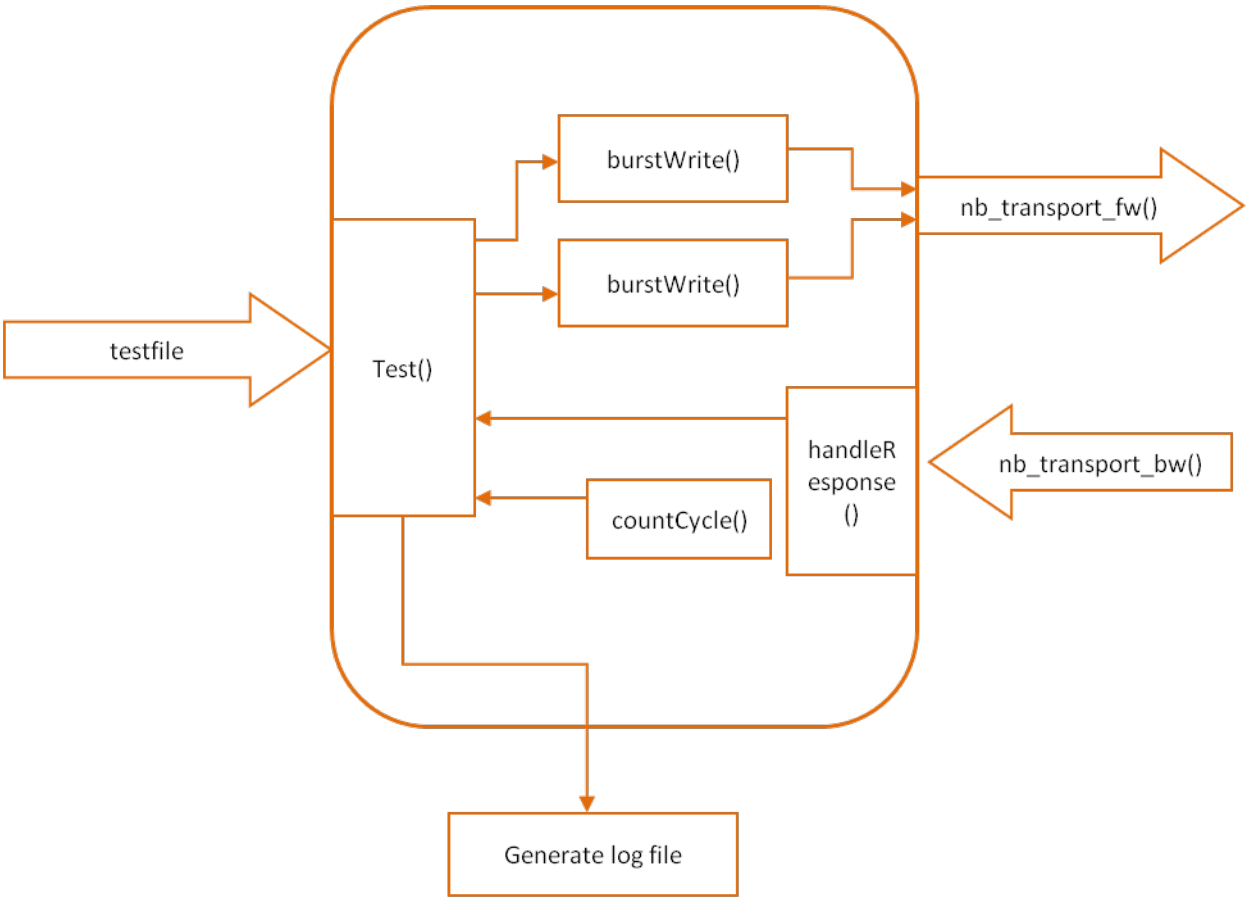


Fig. 14 Test generator

The output is a comma-separated values (CSV) file, which can be easily read into software tools like Microsoft Excel, as shown in Fig. 15.

	A	B	C	D	E	F	G	H
1	ID	Type	Address	Length	Start	End	Latency	ThreadID
2	0	read	0x25fc	4	0	10	10	0
3	1	write	0x242a	4	11	17	6	0
4	2	write	0x17c	4	16	24	8	0
5	3	read	0x2b78	4	23	45	22	0
6								

Fig. 15 An example of log file

4 Testing and Results

Using the test module that we designed, we carried out a series of tests to verify that our accurate model reflects the major characteristics of a real DRAM. Each test aims at one of the DRAM characteristics, and some are done in comparison with the reference model, which is a dummy slave with a constant response delay.

4.1 General Testing Configuration

The parameters of the DRAM are set to the values shown in Tab. 2.

Name	Unit	Value
clock period	ns	5
refresh period	ns	7600
refresh duration	ns	120
address width	bits	25
bank address width	bits	2
row address width	bits	13
column address width	bits	10
word length	bytes	4
minimum burst length	bytes	4
hop row delay	ns	55
open row delay	ns	15
tCAS	ns	10
tDQSS	ns	10
tWTR	ns	10

Tab. 2 Configuration parameters in testing

The values are chosen according to [2], and reflect the actual setting of a common DDR2 memory.

For the address width, 25 is the number of effective bits, which does not include the word-length related bits (addr[1] and addr[0]). So, the actual width of the address is 27, which means the size of the DRAM is 128 Mbytes. The address is composed in the following way:

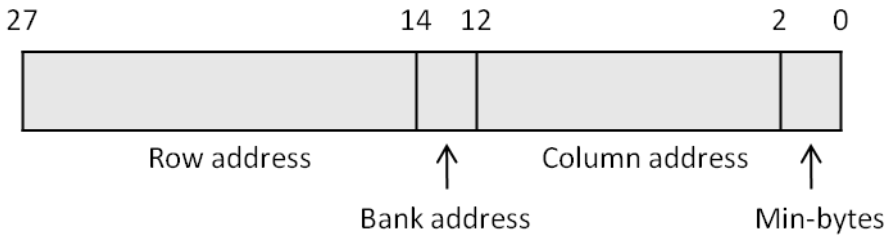


Fig. 16 Address components

The comparison with reference model is done by running the simulation using the same test file for both the accurate model and the reference model. The reference model has a fixed delay of 10 cycles.

4.2 Test 1 - Refresh

4.2.1 Test Generation Constraints

- Continuous write access (write requests are sent one after another)
- Burst length = 128 (words)
- Address range: 0x0 - 0x4000 (access to random address in range)

4.2.2 Description

This test is designed to verify that the accurate DRAM model has the refresh behavior as described in 1.2. The constraints are selected so that refresh has high probability to occur in the middle of a transaction, so we can identify refreshes from the abnormally

high delay of certain transactions. Address range is selected so all generated addresses are in the same row of all banks, so there will be no row hops during the test, because row hops might also cause high delay, which will make the refresh hard to identify.

4.2.3 Testing Results

The latencies of the transactions are shown in Fig. 17. It is obvious that periodically there is a transaction having a distinguishable high latency. The period is about 1500 cycles, which agrees with the calculated refresh period: 7600 ns / 5 ns = 1520 (cycles).

The worst case is that when a transaction's addresses cross the bank boundary, the row is not activated when receiving the request, and there happens to be a refresh in the middle of the transaction. The worst case latency can be calculated as follows:

Latency(worst case)
= Delay(pipeline) + Delay(refresh) + 3 * Delay(open row) + Delay(burst)
= 128 + 120 / 5 + 3 * (15 / 5) + 4 = 165 (cycles)

The worst case in the testing result agrees with the calculation.

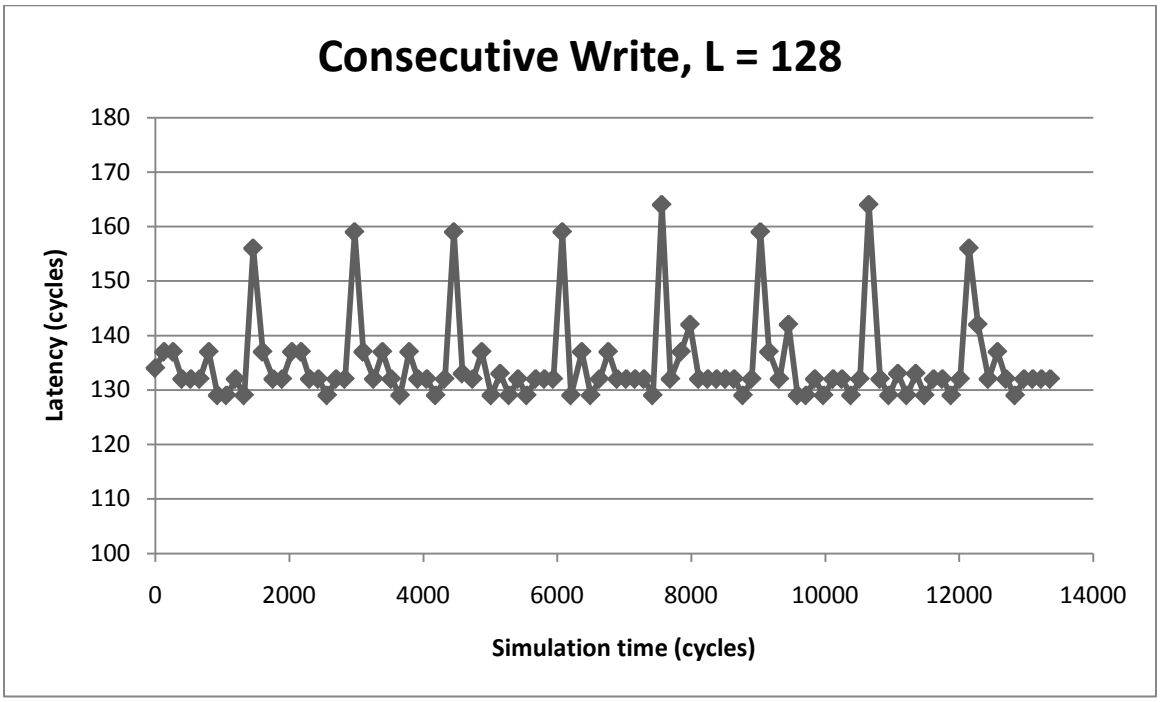


Fig. 17 Result of Test 1

4.2.4 Comparison with Reference Model

The testing result with the reference model is shown in Fig. 18. Latencies of all transactions are the same value, 138 cycles, which can be calculated as:

Latency(ref. model)
= Delay(pipeline) + Delay(fixed)
= 128 + 10 = 138 (cycles)

The result with the reference model does not show the refresh behavior of a DRAM.

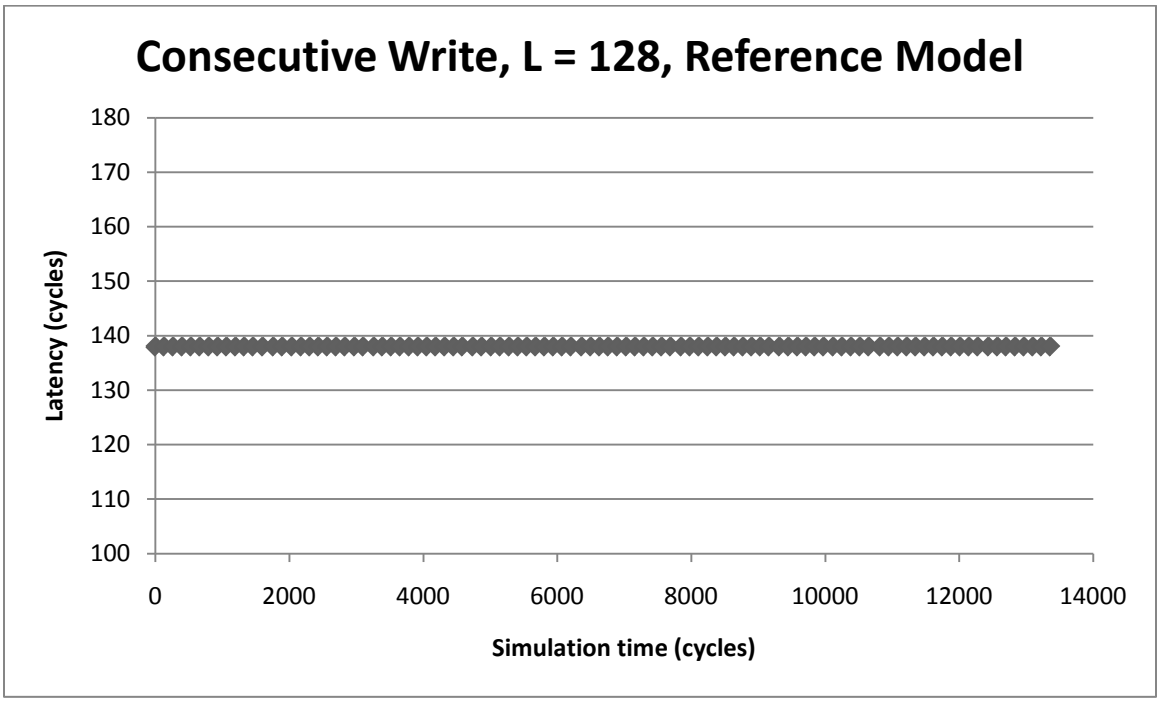


Fig. 18 Result of Test 1, reference model

4.3 Test 2 - Address

4.3.1 Test Generation Constraints

- Read access
- Consecutive address, random address

- Burst length = 8 (words)
- Average request generating rate = 25%

4.3.2 Description

The test is designed to verify the row-hopping penalty delay of the DRAM model. For consecutive addresses, most accesses do not cause a hopping of row (closing current row plus opening another row), while random addresses are more likely to cause a row hop.

Average request generating rate is 25%, which means in each cycle the probability of generating a request is 1/32 (25% * 1/8). In other words, in average there will be a request in every 32 cycles.

4.3.3 Testing Results

The testing result is shown in Fig. 19. The latencies of accesses to random addresses are obviously higher than accesses to consecutive addresses. The difference can also be observed from the distribution diagram shown in Fig. 20.

The difference is caused by row hopping penalty delay of the accurate DRAM model. Plus, random addresses also have probability to cross bank boundary or even row boundary, which makes the latency higher.

The result proves that our accurate model can correctly represent the DRAM behavior for different address patterns.

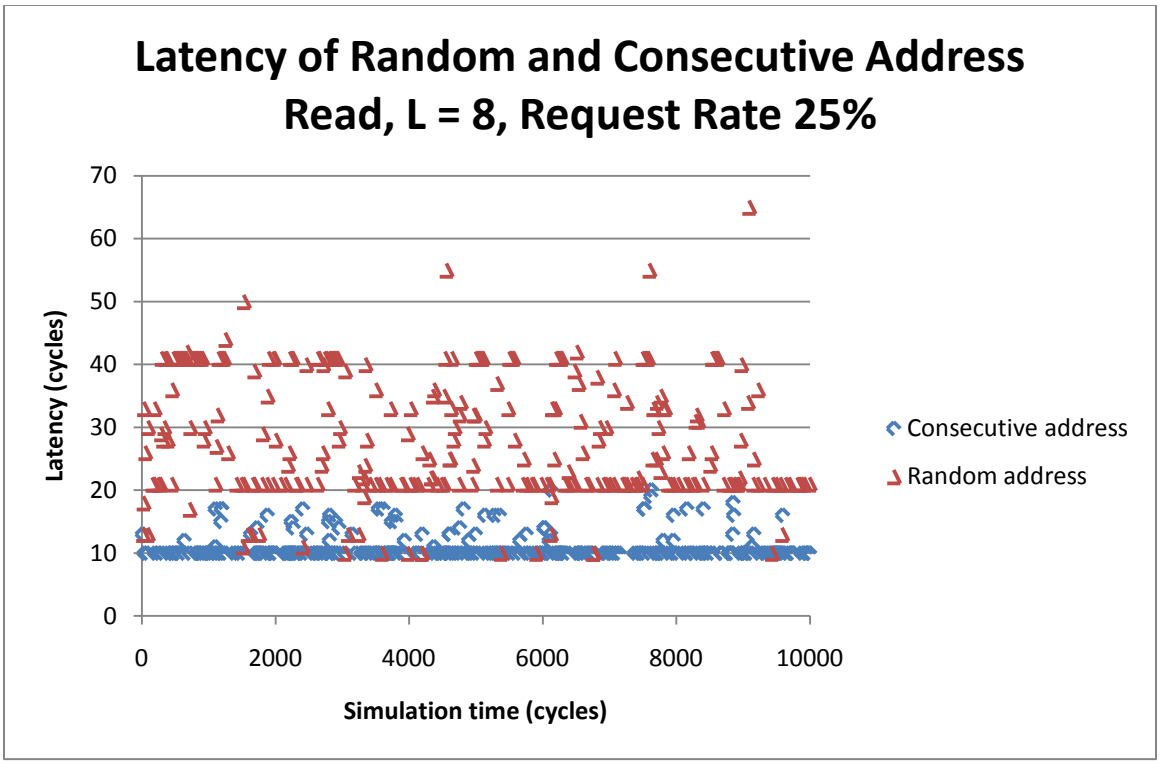


Fig. 19 Result of Test 2

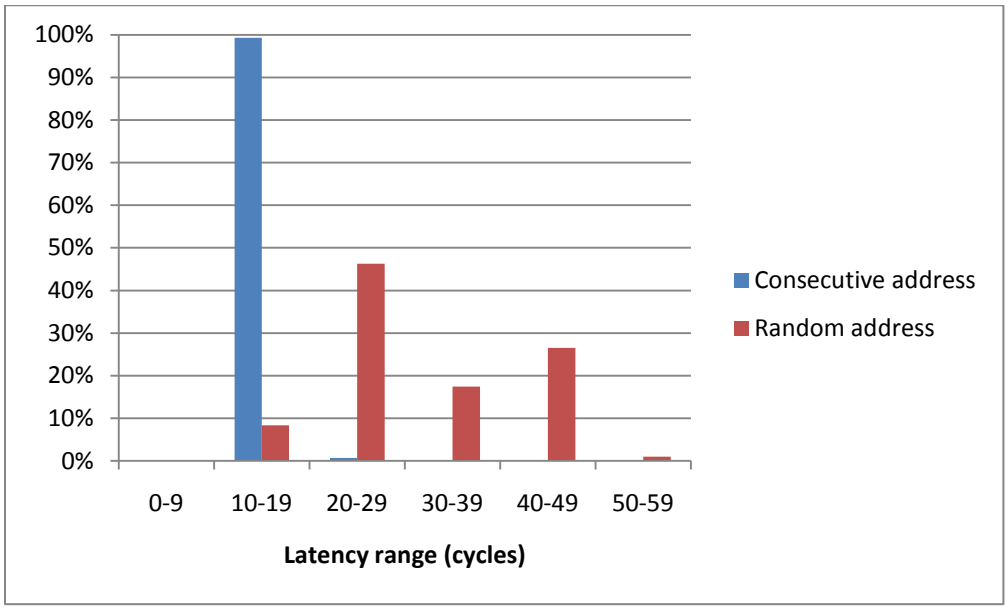


Fig. 20 Result of Test 2, latency distribution

4.3.4 Comparison with Reference Model

The test result of reference model with the same test file is shown in Fig. 21. In this case, there is not much difference between consecutive address and random address. The reference model is not capable of modeling row hopping behavior of DRAM.

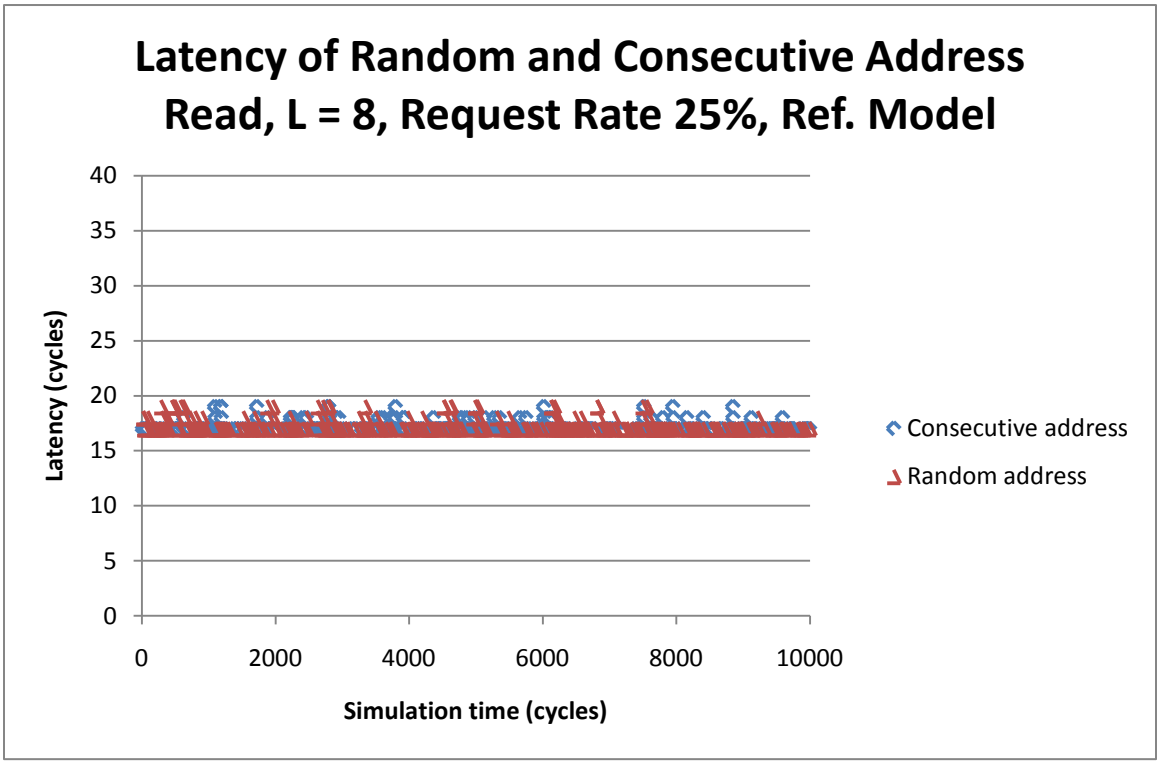


Fig. 21 Result of Test 2, reference model

4.4 Test 3 - Read/Write

4.4.1 Test Generation Constraints

- Read access, write access, read/write randomly mixed access
- Burst length = 4 (words)
- Average request generating rate = 25%

- Address range: 0x0 - 0x4000 (access to random address in range)

4.4.2 Description

The test is designed to verify the read-write switch penalty delay of accurate DRAM model. To be specific, the DRAM model has a write-to-read penalty, so the latency of read/write mixed transactions are expected to increase.

The address range is chosen to be in one row, in order to avoid row hopping penalty, because it is greater than write-to-read penalty, thus might conceal the latter.

4.4.3 Testing Results

The result of the test is shown in Fig. 22. As expected, the read/write mixed access has higher latencies than either of pure read or write access, which can be observed more clearly from comparison of average latency in Fig. 23.

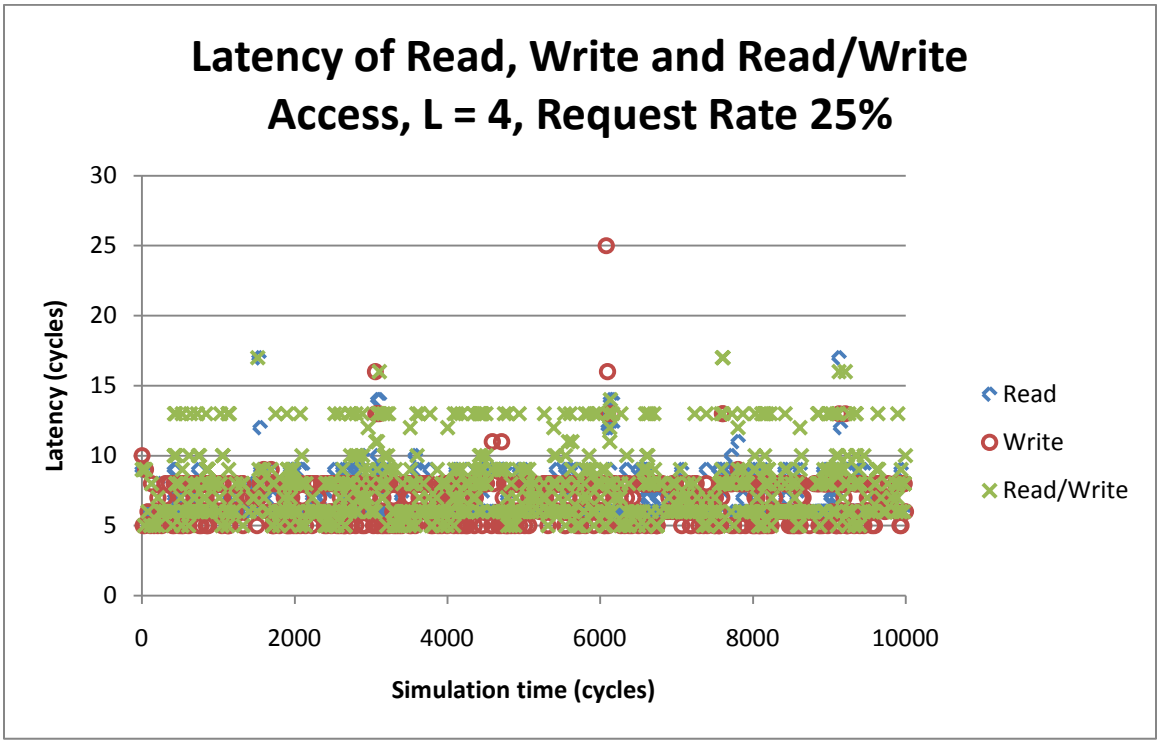


Fig. 22 Result of Test 3

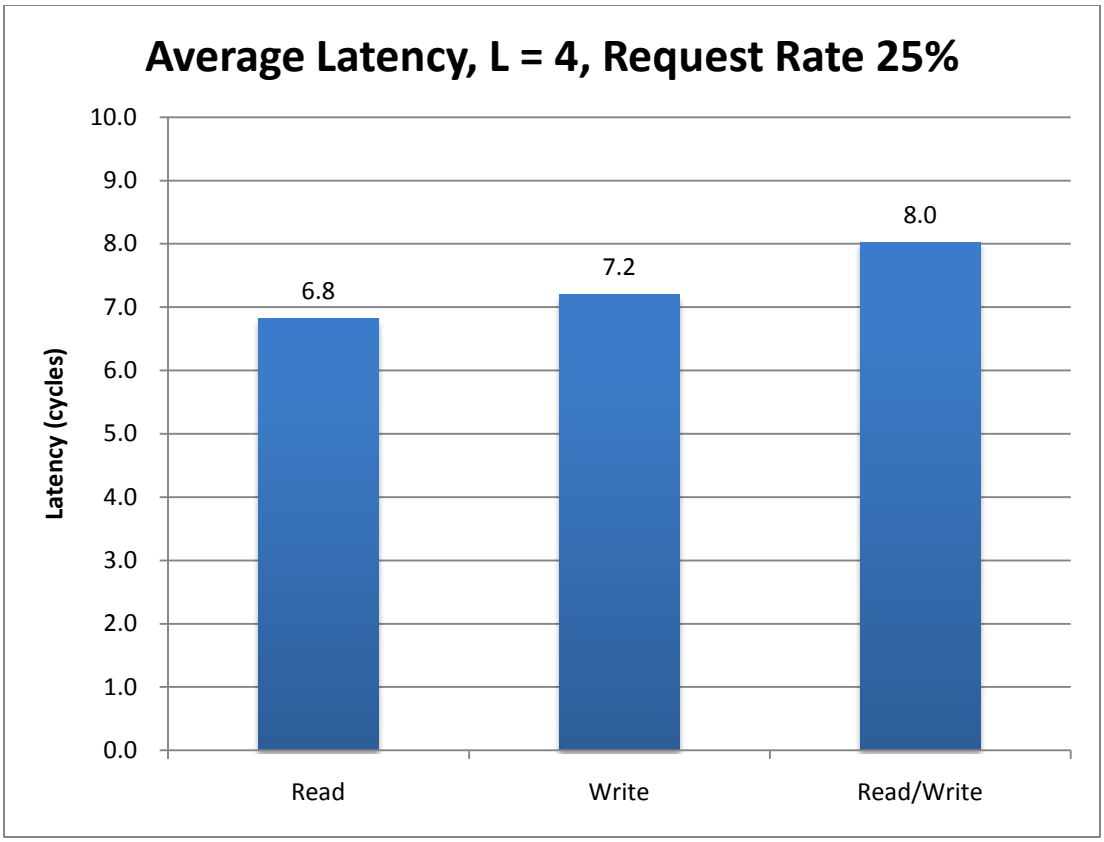


Fig. 23 Result of Test 3, average latency

The result shows that our accurate model can correctly model read-write switching penalty.

4.4.4 Comparison with Reference Model

The average latency of the reference model with the same test file is shown in Fig. 24. Using the reference model, the average latency of read/write mixed access is in between of pure read and pure write access, not showing the delay penalty caused by read-write switching. Therefore, accurate DRAM behavior cannot be simulated by such a simple model.

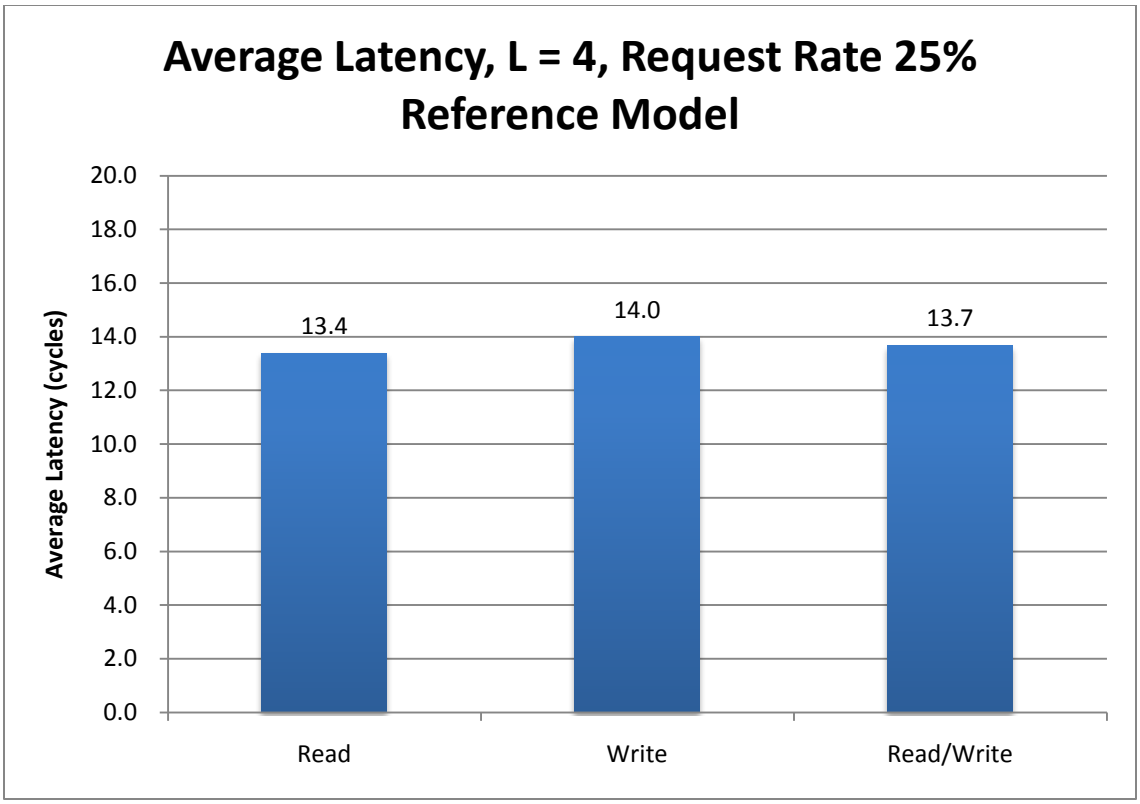


Fig. 24 Result of Test 3, average latency, reference model

4.5 Test 4 - Throughput - Burst Length

4.5.1 Test Generation Constraints

- Continuous read access
- Burst length = 1, 2, 4, 8
- Address range: 0x0 - 0x4000 (access to random address in range)

4.5.2 Description

The test is designed to measure the throughput of the accurate DRAM model with different burst lengths. In DRAM, there is a minimum burst length (4 in our case). A request with a burst length less than the minimum burst length will be treated as a

request with the minimum burst length, and will take the same amount of time. This will decrease the efficiency, or the throughput, of the DRAM.

4.5.3 Testing Results

The throughputs of the DRAM for different burst lengths are shown in Fig. 25. With burst length less than 4, the throughput is decreased significantly, because the DRAM still needs 4 cycles to handle such a request.

The upper bound of the throughput can be calculated as:

$$\Theta_{\text{upper}} = \frac{\text{L}}{\left\lceil \frac{\text{L}}{\text{BL}_{\text{MIN}}} \right\rceil \cdot \text{BL}_{\text{MIN}}}$$

In our case, $\text{BL}_{\text{MIN}} = 4$, so the upper bound of throughputs for L = 1, 2, 4, and 8 are:

$$\Theta_{\text{upper}}(1) = \frac{1}{\left\lceil \frac{1}{4} \right\rceil \cdot 4} = \frac{1}{4} = 25\%$$

$$\Theta_{\text{upper}}(2) = \frac{2}{\left\lceil \frac{2}{4} \right\rceil \cdot 4} = \frac{2}{4} = 50\%$$

$$\Theta_{\text{upper}}(4) = \frac{4}{\left\lceil \frac{4}{4} \right\rceil \cdot 4} = \frac{4}{4} = 100\%$$

$$\Theta_{\text{upper}}(8) = \frac{8}{\left\lceil \frac{8}{4} \right\rceil \cdot 4} = \frac{8}{8} = 100\%$$

The result agrees with the calculation.

The testing result shows that our DRAM model's throughput can be quite different for different burst lengths, which is actually the case of a real DRAM. The limited throughput can limit the overall performance in system evaluation, so our model can be used to get a more accurate result.

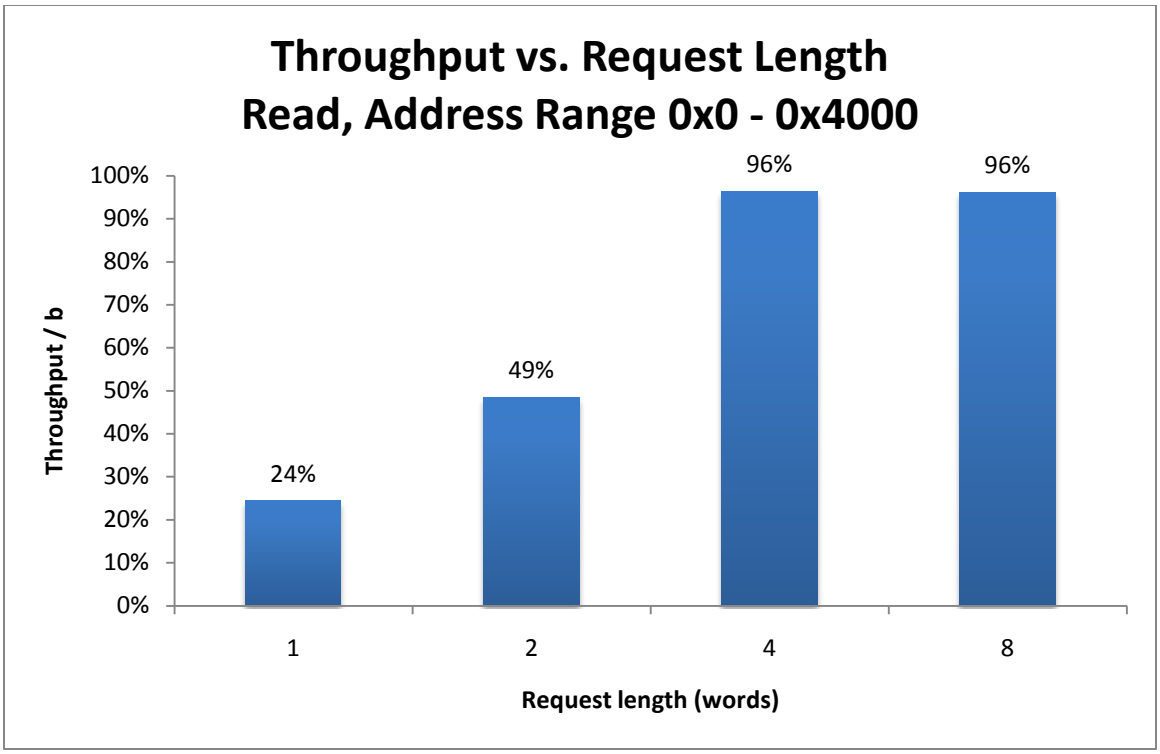


Fig. 25 Result of Test 4

4.6 Test 5 - Throughput - Address Range

4.6.1 Test Generation Constraints

- Continuous read access
- Burst length = 8
- Address range:
  0x0 - 0x4000 (in the same row),
  0x0 - 0x8000 (in 2 rows),
  0x0 - 0x10000 (in 4 rows),
  0x0 - 0x20000 (in 8 rows)
- Access to random address in range

4.6.2 Description

The test is designed to measure the throughput of the DRAM model with requests to different address ranges.

Address range is related to the probability of row hopping. The bigger the address range is, the more probable to hop the row on each access. As discussed before, hopping row can cause increased latency, and it can also cause decreased throughput, because no data are transferred during hopping the row.

4.6.3 Testing Results

The result is shown in Fig. 26.

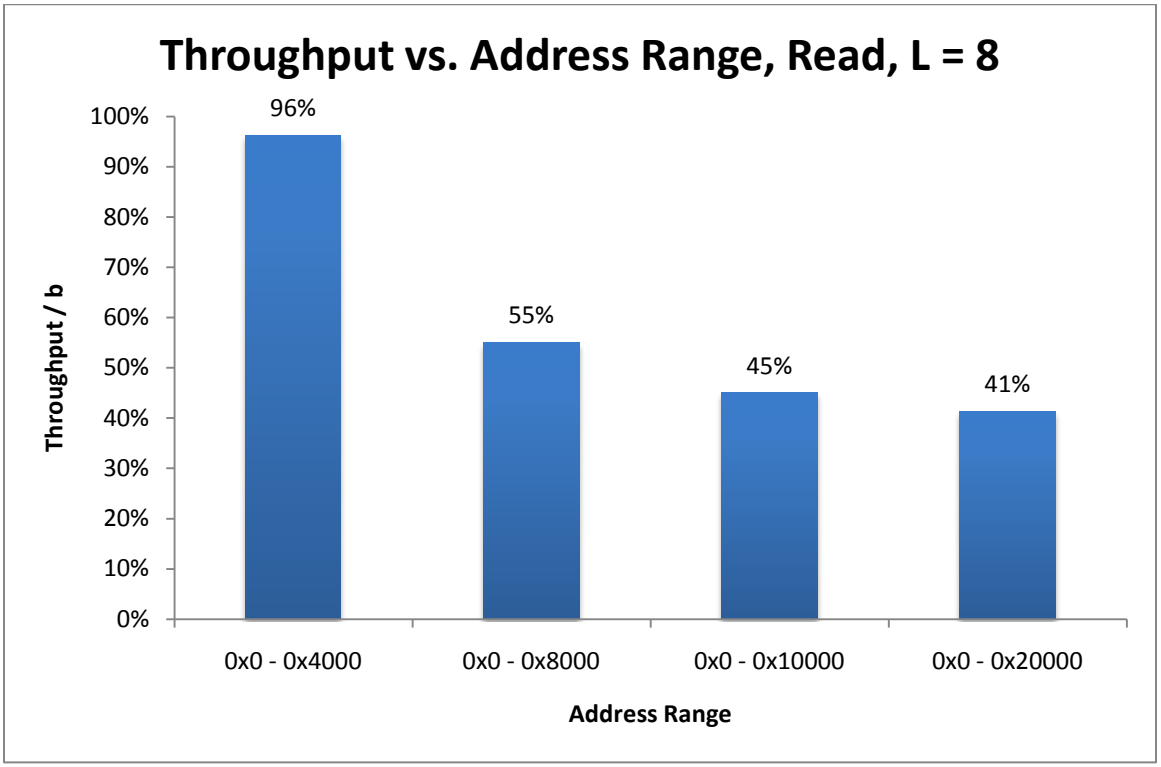


Fig. 26 Result of Test 5

The result shows the capability of our accurate model to limit the throughput accordingly for different address range.

4.7 Test 6 - Request Generating Rate

4.7.1 Test Generation Constraints

- Read access
- Burst length = 8
- Address range:
  0x0 - 0x4000 (in the same row),
  0x0 - 0x8000 (in 2 rows),
  0x0 - 0x10000 (in 4 rows),
  0x0 - 0x20000 (in 8 rows)
- Access to random address in range
- Average request generating rate = 5% to 100% (step 5%)

4.7.2 Description

This test is designed to see how the request generating rate will affect the average latency under different address ranges.

4.7.3 Testing Results

The result is shown in Fig. 27.

First of all, we see that the average latency increases as the request generating rate grows. This is caused by the contention brought by higher request rate.

Secondly, larger address range will result in higher latency, which is similar to the consecutive address compared to random address case that we discussed in 4.3.

Finally, all the lines tend to remain constant after a certain point. This does not mean the latencies remains the same, but means the request generating rate cannot increase further after this point. In fact, the saturation points of each line are exactly the values of

the throughputs for these address ranges in Test 5. Here the maximum request generating rate is limited by the throughput.

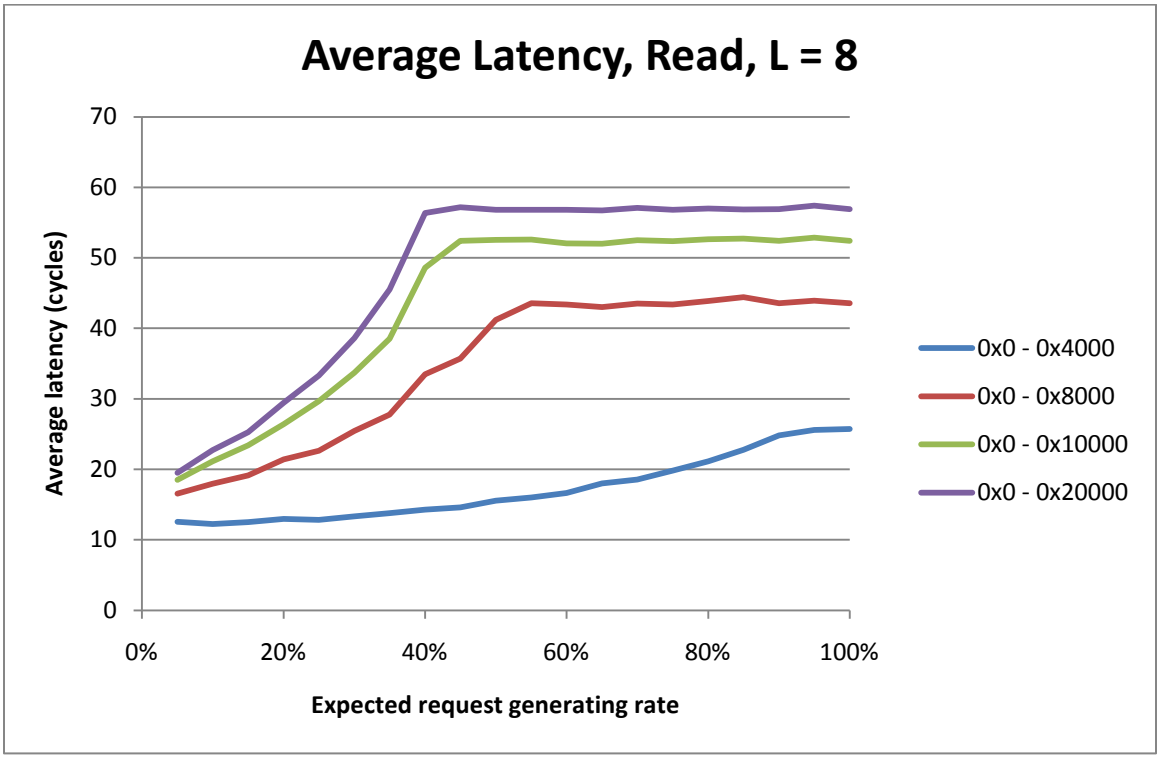


Fig. 27 Result of Test 6

4.8 Test 7 - Scheduling Policy

4.8.1 Test Generation Constraints

The master sends requests from four threads, and the DRAM scheduler is used.

- Thread 0: read, L = 16, rate = 12%, address range: 0x0 - 0x4000
- Thread 1: write, L = 16, rate = 12%, address range: 0x4000 - 0x8000
- Thread 2: read, L = 8, rate = 12%, address range: 0x8000 - 0xc000
- Thread 3: write, L = 8, rate = 12%, address range: 0xc000 - 0x10000

4.8.2 Description

This test is designed to demonstrate that the memory performance of the accurate model can be optimized by using a self-designed scheduler described in 3.1.3, therefore proves that this model can help in designing an actual DRAM scheduler.

The address ranges are selected so that each thread only accesses the addresses in the same row, because in reality, a thread is more likely to access nearby memory locations.

4.8.3 Testing Results

The result is shown in Fig. 28 and Fig. 29. It can be observed that, the average latency can be reduced by choosing a proper scheduling policy.

Priority scheduling policy does not help decreasing the latency in average, but reduces the latency of high priority threads significantly. This is because using this policy does not guarantee less number of thread switching.

By contrast, round-robin policy reduces the average latency by various amounts. The reduced amount is related to the slot size of the round-robin. The reduction is because of the fact that round-robin scheduling policy decreases the number of thread switching, and since each thread is accessing one row of the DRAM, it decreases the number of row hopping, thus shortening the average latency.

The result tells that our accurate DRAM model is possible to help in designing a DRAM scheduler.

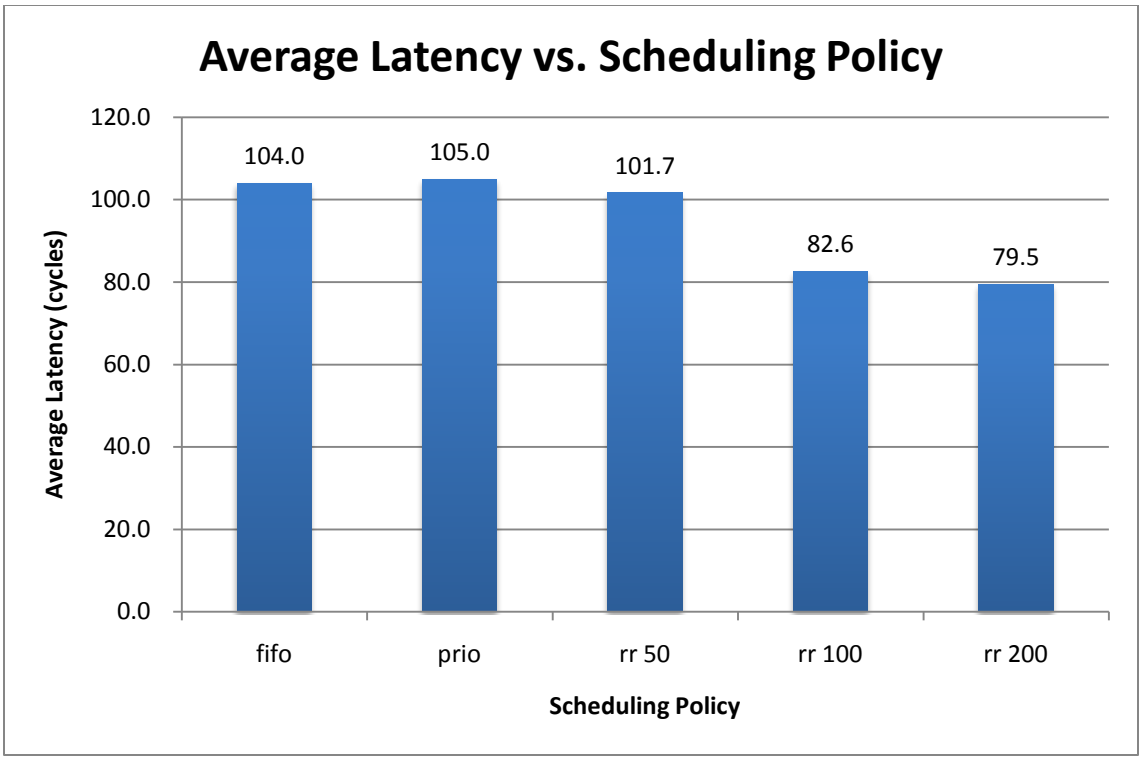


Fig. 28 Result of Test 7, overall

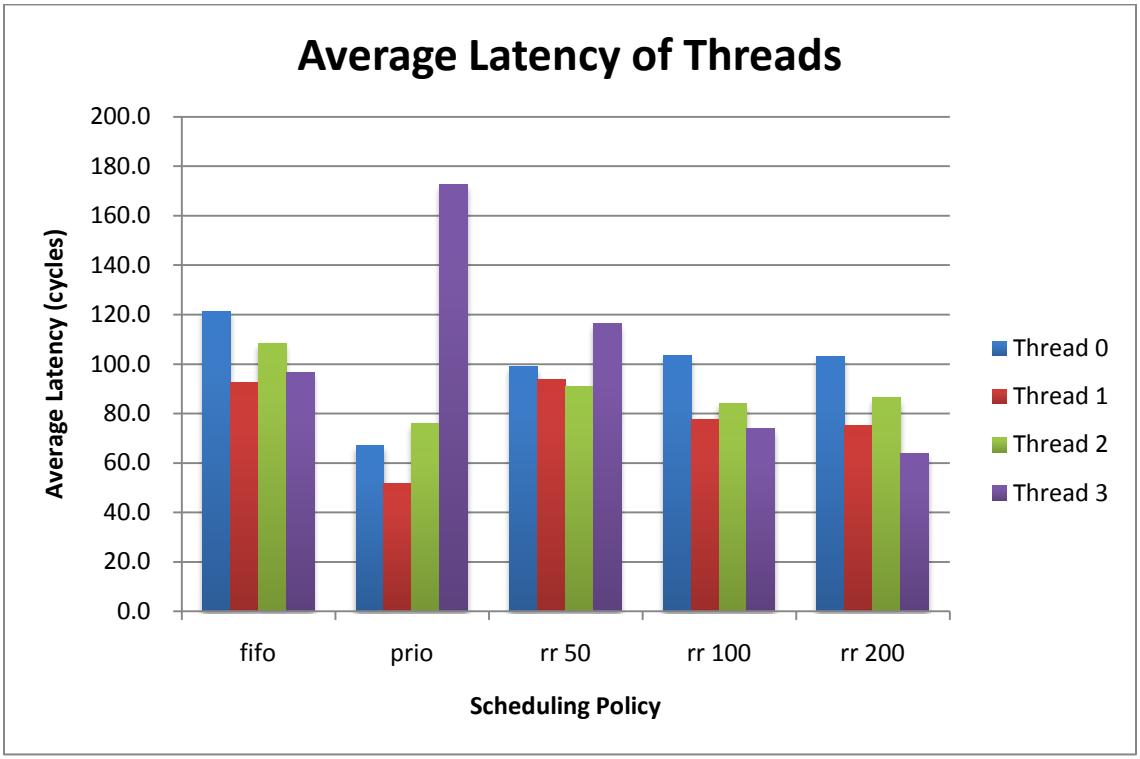


Fig. 29 Result of Test 7, average latency of threads

4.8.4 Comparison with Reference Model

The result of the same test using the reference model is shown in Fig. 30. There is no obvious reduction of latency using different scheduling policy on the reference model, which means simple models like the reference model will not help in designing a DRAM scheduler.

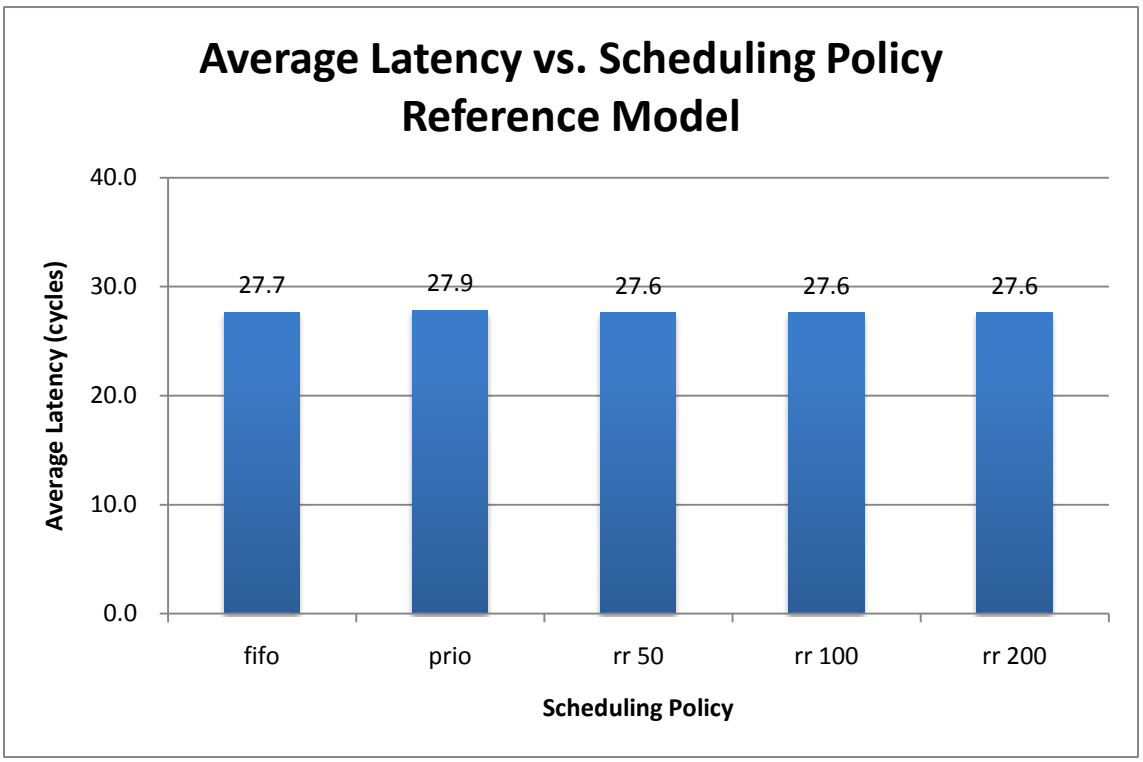


Fig. 30 Result of Test 7, overall, reference model

5 Conclusion

5.1 Generalization of Results

Our accurate DRAM model can closely reflect the behavior of an actual DRAM. The major delay-related characteristics of DRAM, such as refresh, row hopping penalty, and write-to-read penalty, are all presented in our model.

Our model, if used in system evaluation, can give more realistic results than a dummy slave. As memory being a more and more performance-affecting component in a system, this becomes especially important.

With the help of our model, it is possible to design a scheduler to optimize the memory access performance in multi-threaded applications. The selection of scheduling policy depends greatly on how the memory model reacts in simulation. Our accurate DRAM model can help this process by providing accurate delay information in simulation.

5.2 Future Work

The next phase of our work is to find a more realistic testing environment so we can see how our model performs in a real system evaluation. We will need to improve our model for code efficiency so it can be used in fast simulation.

Another direction is to design a better scheduler based on our accurate DRAM model. The scheduler will focus on the DRAM itself, trying to optimize the performance of memory access.

6 References

[1] T. Bjerrengaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Computing Surveys, vol. 38, 2006.

[2] K. Srinavasan and E. Salminen, "A Memory Subsystem Model for Evaluating Network-on-Chip Performance," unpublished.

[3] R. Günzel and H. Alexanian, "OCP Modelling Kit User Manual", Jan. 20, 2010.

[4] OCP-IP Association, "SystemC OCP Models." [Online]. Available: http://www.ocpip.org/systemc_ocp_models.php

7 Appendix - Project Management Report

7.1 General Summary

The project is completed successfully. We have reached all the goals as planned, and the results look good. Although at first we underestimated the amount of work in some part of the project, namely the testing part, we managed to finished it in time by reassigning the responsibilities of team members to focus on testing.

7.2 Follow-up of Objectives

The following goals defined in the project plan are obtained:

- Create an accurate DRAM model
- Create a reference model with fixed latency
- Create a test module
- All modules are created using OCP-IP TLM Kit, and are compatible with OCP standard
- The work is delivered before the deadline (December 16, 2010)

In the test module, we did not use the Transaction Generator as planned, but generated random transactions directly from the test module, because the Transaction Generator is not OCP-compatible, and its output is hard to handle. This change did not cause the decreasing of project quality.

7.3 Lessons Learned and Suggestions for Improvement

- There are always a lot to learn
  Learning is always an important part in a project. Actually, in our project, most of the time is spent on reading manuals and specifications. And unlike taking lectures, we are all on our own.

- Project partitioning is very important
  A good partitioning must have very clear partition boundaries, so each member in the team can have an accurate specification of the job.

- It is wise to leave some time margin before the deadline
  Nobody knows what situation might come. In our case, there is a change of focus in the middle of the project. We suddenly need more people to work on testing, and the time is limited. This kind of situation is not very predictable, so putting some time margin here and there in the time plan is a good idea.

- Automation is preferred
  Although it may seem not necessary and cost a lot of time at first, it is still nice to have fully automated process, especially in testing. Because then it is possible to re-run the tests very conveniently after changing some testing configurations, which is very likely to happen in testing, even if everything seems to be so sure in the beginning.

7.4 Final Comment

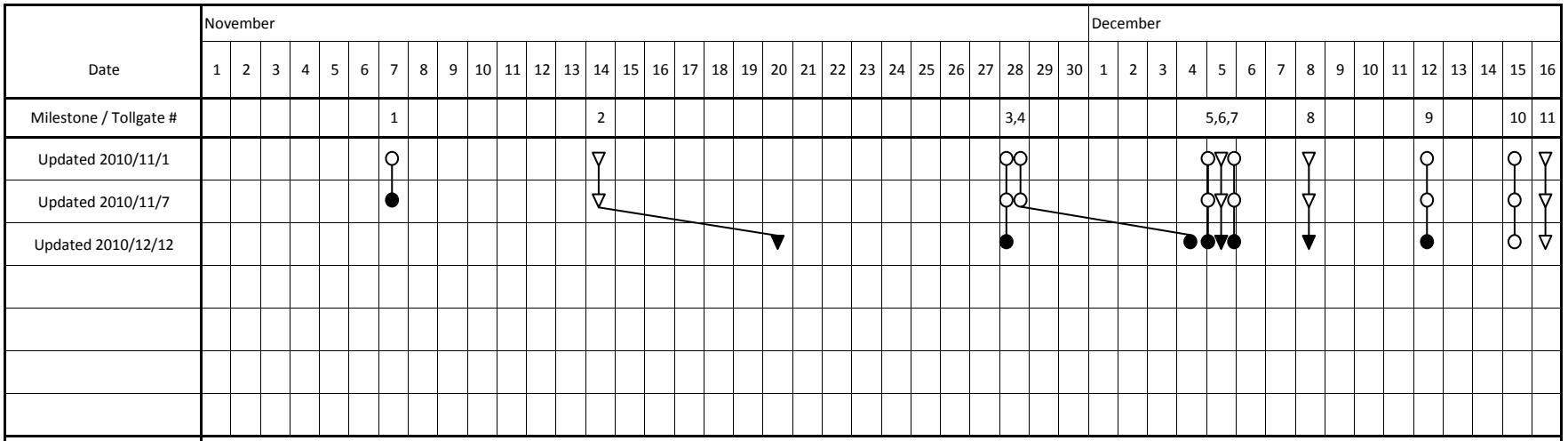
For this project, the learning curve is steep, since the OCP protocol is very complicated and needs time to be fully understood. All the team members tried hard to make their works done. As the group leader, I am very proud of the members of our group.

7.5 Final Time Plan

See next page.

Time Plan

Project Group 11: Design and Implementation of an Accurate Memory Subsystem Model in SystemC



Project manager

Nan Li

Milestone ○ / Tollgate ▽

1	○ Project plan ready	5	○ Test module ready	9	○ Documentation ready
2	▽ Pre-study finished	6	▽ Implementation finished	10	○ Presentation ready
3	○ Reference model ready	7	○ Integration successful	11	▽ Presentation done
4	○ Accurate model ready	8	▽ Testing finished	12	