SCALE DRAM Subsystem Power Analysis

by

Vimal Bhalodia

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author ....
Department of Electrical Engineering and Computer Science
August 16, 2005

Certified by ....
Krste Asanović
Associate Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

SCALE DRAM Subsystem Power Analysis

by

Vimal Bhalodia

Submitted to the Department of Electrical Engineering and Computer Science
on August 16, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

To address the needs of the next generation of low-power systems, DDR2 SDRAM offers a number of low-power modes with various performance and power consumption tradeoffs. The SCALE DRAM Subsystem is an energy-aware DRAM system with various system policies that make use of these modes. In this thesis, we design and implement a DDR2 DRAM controller and test a version of the SCALE DRAM Subsystem in hardware. Power measurements from the actual DRAM chips are taken and compared to datasheet derived values, and an analysis of the DRAM refresh requirements is performed. Some notable power consumption results include active powerdown being much closer to precharge powerdown and reads taking much less current than the datasheet indicates. In addition, based on the refresh tests, a system that powers down at least 12.3s for each 32MB of traffic can save power using delayed refresh and ECC data encoding.

Thesis Supervisor: Krste Asanović
Title: Associate Professor

Acknowledgments

First and foremost, I'd like to thank my thesis advisor, Krste Asanović, for making DRAM exciting. I learned something new from every meeting I had with him, and whenever I had a problem, he could always suggest a solution without handing me the answer.

A big thank you also goes out to Brian Pharris for his thesis work on the SCALE DRAM Subsystem, Elizabeth Basha and Eric Jonas for their advice on working around FPGA issues, Jared Casper for helping with the tester baseboard, and Gautham Arumilli for documenting and helping debug the DRAM board.

This project was partly funded by the DARPA HPCS/SGI Ultraviolet project, NSF CAREER Award CCR-0093354, the Cambridge-MIT Institute, and equipment donations from Xilinx and Intel.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

SCALE is a programmable processer architecture designed to efficiently handle a wide range of parallel workloads in embedded sytems[1]. The SCALE-0 processor consists of a MIPS control processor, a 4-lane vector-thread unit, and a 32KB unified cache based around 8-word cache lines. This cache directly interfaces with the SCALE DRAM Subsystem, which uses four 256Mbit DDR2 DRAM chips to present a total of 128MB main system memory.

In modern energy-sensitive computing applications, the memory subsystem can account for up to 90% of the non-I/O power consumption[2]. DRAM-based memory modules already implement several different power modes, each with its own performance and energy cost. In order to create a power-efficient memory system, the DRAM controller must implement a mode transition policy which saves as much energy as possible while maintaining an acceptable level of performance.

Delaluz et al[2] found that in systems without cache, a policy which scheduled chip powerdown after several idle cycles provided significant energy savings without sacrificing performance. By delaying powerdown, this policy avoided both the performance and energy penalty of reactivation under memory access patterns with high spatial locality. On the other hand, systems with cache benefitted most from a policy of immediate powerdown, since spatial locality was already handled by the cache.

In his thesis, Pharris[3] designed the SCALE DRAM Subsystem to include several policy modules. These modules independently control address translation, memory request scheduling, and

DRAM power mode transitions. A computer simulation of the DRAM Subsystem performance under various benchmarks agreed that the best static policy was immediate powerdown.

1.1 Overview

In this thesis, we implement a DDR2 DRAM controller and use it to take actual power measurements of Micron 256Mbit DDR2 SDRAM chips under various power modes and transitions. We also profile data corruption in a single DRAM chip when subject to delayed refresh intervals.

Experimental energy consumption data is important for simulation and evaluation of DRAM mode transition policies. Every policy optimizes for a different situation and spends different amounts of time in the various power modes and transitions. In order for a policy evaluation to be correct, the energy cost of each state and transition must be correct.

Power consumption and refresh requirements on datasheets are generally conservative worst-case estimates designed to increase manufacturing yield while maintaining a marketable product. For a given parameter, this project will attempt to provide a more realistic energy estimate based on current consumption under operation. In addition to determining the average value and comparing to a datasheet-derived value, we also examine dependence on external factors such as supply voltage, operating frequency, and temperature.

1.2 DDR2 SDRAM Overview

A standard DRAM cell stores a single bit as charge on a capacitor controlled by a transistor. The simple structure of DRAM cells allows them to be packed tightly, resulting in affordable high capacity, high density modules. The downside is that bits get corrupted or lost due to charge leakage off the capacitor, requiring extra circuitry to refresh the stored data periodically.

An SDRAM module is organized as a set of banks each of which contains an independent DRAM cell array. This array is broken up into rows and columns, with a dedicated active row.

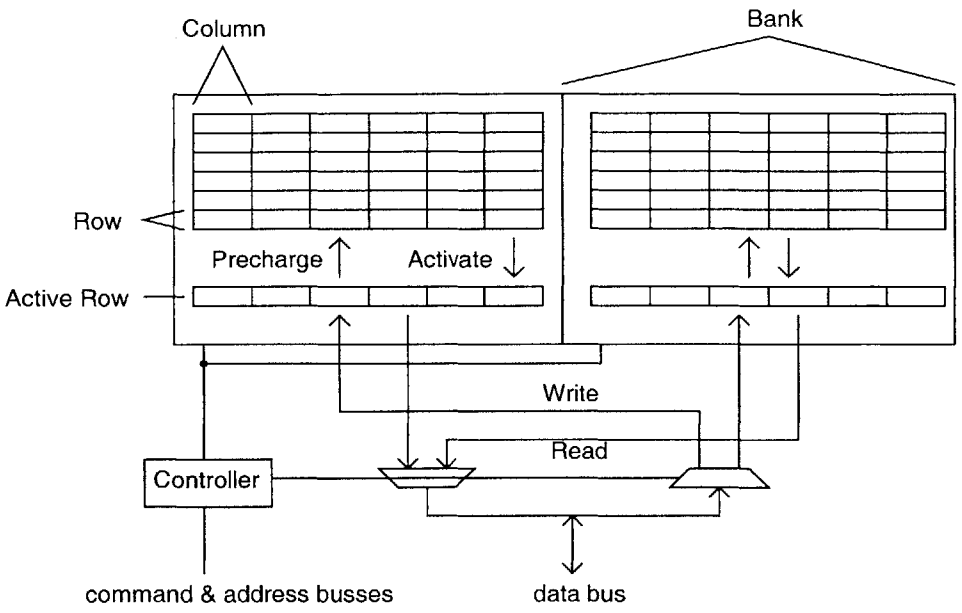

Figure 1-1: SDRAM module organization[3].

1.2.1 Operation

Each SDRAM bank can be in one of two states: precharged and active. When the bank is precharged, a SDRAM memory transaction begins by selecting the desired bank and activating the desired row by loading it onto the sense amplifiers, putting the bank in the active state. After an appropriate number of cycles known as the RAS to CAS latency, the column is selected along with either a read or write command, and after another delay known as the CAS latency, the data is read in or out in fixed-length bursts on both the rising and falling clock edges. Requests from the same row but different column can be handled by changing the column address and waiting another CAS latency before accessing the data. A request to a different row requires precharging the current row and activating the new one.

As mentioned before, DRAM cells will start to lose data unless they are periodically refreshed. In-between memory accesses, the DRAM controller can issue refresh commands to the SDRAM banks. These commands precharge the currently active row and then select a row to be refreshed based on an internal controller. Refresh commands must be issued periodically, generally on the order of once every 10us.[4]

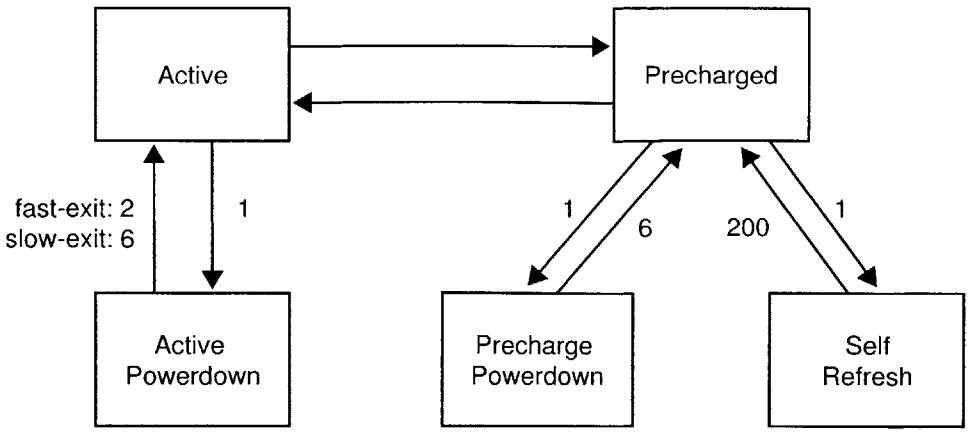

Figure 1-2: DDR-II SDRAM power mode transitions and associated delay.

1.2.2 Power modes

DDR2 SDRAM modules offer a number of low-power modes to conserve energy. Each of these modes has different relative energy consumption, reactivation delay, and transition possibilities.

Active Powerdown - This state is entered from the active state when a powerdown is initiated, and requires a short resynchronization time to return to the active state. Depending on the DRAM configuration, this mode can either be fast-exit or slow-exit. Fast-exit has a lower resynchronization time than slow-exit, but has higher power consumption.

Precharge Powerdown - If no rows are currently active and a powerdown is initiated, this state is entered. It offers lower power consumption than either of the active powerdown states, and has a resynchronization time on par with slow-exit.

Self Refresh - This is the lowest power state, and can be entered from the Precharge state. While minimal energy is consumed and refresh commands do not need to be periodically issued, exiting this state takes on the order of several hundred cycles.

1.2.3 Timing Constraints

DDR2 DRAM commands are subject to two classes of timing constraints. The first class is bank timing constraints which govern how close commands addressed to the same bank can be issued. The second class is bus timing constraints, which govern how close commands from any bank can

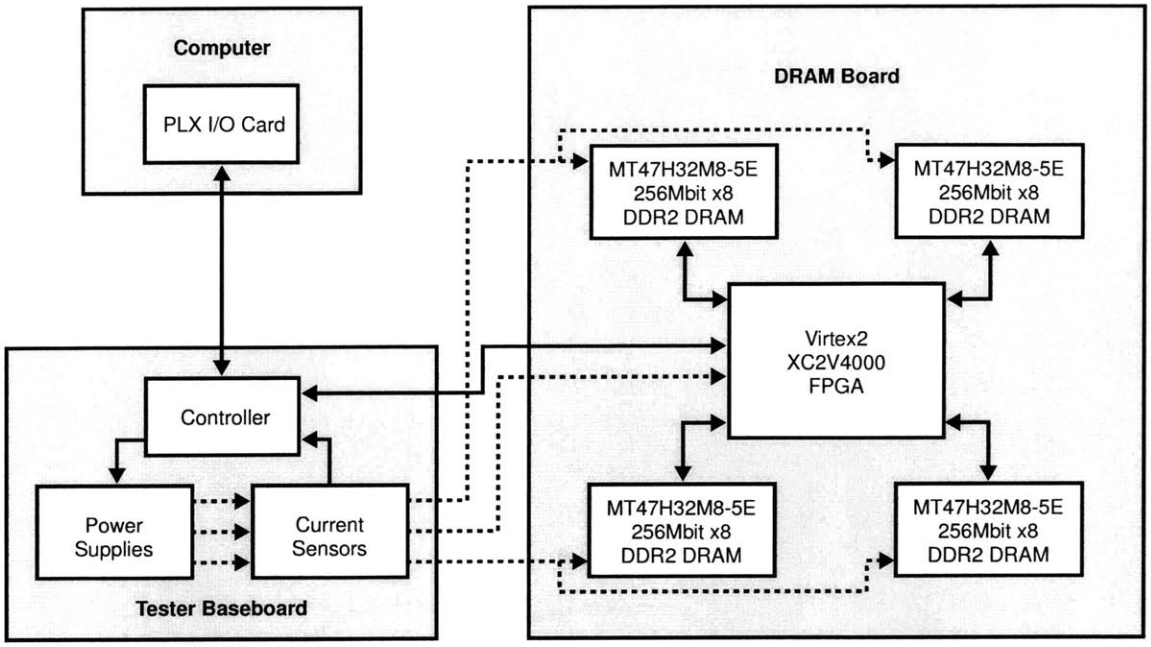

Figure 1-3: Hardware setup.

be to each other. A list of timing constraints can be found in the DDR2 datasheet[4].

1.3 Hardware Setup

1.3.1 SCALE DRAM Board

The SCALE DRAM board is the primary memory testing platform for this project. The board itself consists of a Xilinx Virtex-II FPGA directly connected to several memory modules[5].

Virtex-II FPGA The Xilinx Virtex-II XC2V4000 FPGA contains all the logic required to drive the DDR2 DRAM chips, as well as extra logic to interface with the baseboard and execute test patterns.

256Mbit DDR2 DRAM There are four Micron 256Mbit 8Mx8x4 Bank DDR2 DRAM chips attached to the FPGA with dedicated address and data busses. These four chips are used in the

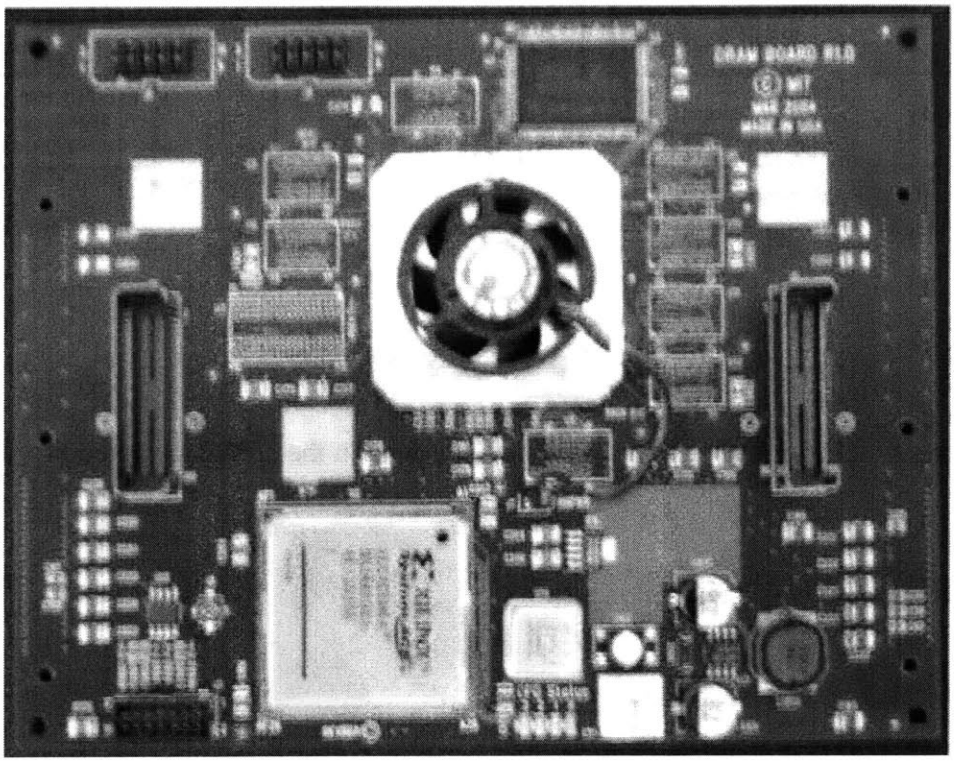

Figure 1-4: SCALE DRAM Board.

power and refresh tests. Two more chips are attached to the FPGA via a shared address and data bus, however these chips are not currently used.

1.3.2 Tester Baseboard

The tester baseboard both supplies power to the DRAM board and allows the DRAM board to interface with a PC. It contains 16 voltage-adjustable current-monitored power supplies which can be used to power sets of chips on the DRAM board. The standard sampling mode reads values from the current sensors and sends them directly to a PC via the PLX card interface. The baseboard controller can also pass requests from the PC to the DRAM board via the AHIP protocol[7].

Chapter 2

Design

Part of this thesis includes designing, implementing, and testing a DDR2 DRAM controller, as well as testing a basic implementation of the SCALE DRAM Subsystem. Both the DRAM controller and the SCALE DRAM Subsystem designs are targeted for the Virtex2 FPGA on the DRAM board, and include platform-specific optimizations and design decisions. The biggest impact of using an FPGA to drive the DRAM is that the system logic becomes the performance bottleneck, not the DRAM itself.

2.1 DDR2 DRAM Controller Design

The DDR2 DRAM controller module (ddr_controller) presents an 32-bit out-of-order pipelined memory interface to a single 256Mbit DDR2 SDRAM chip. It generates all initialization, control, and refresh signals and maintains all state necessary for the proper operation of the DDR2 chip.

2.1.1 Controller Interface

The ddr_controller module interfaces with external logic that runs at half the DDR2 DRAM clock rate. The interface is basically a pipelined out-of-order memory. Requests consist of either a read or a write operation, along with bank, row, and column addresses. Writes also include the data to be written and a mask specifying which bytes within each word should be overwritten. The

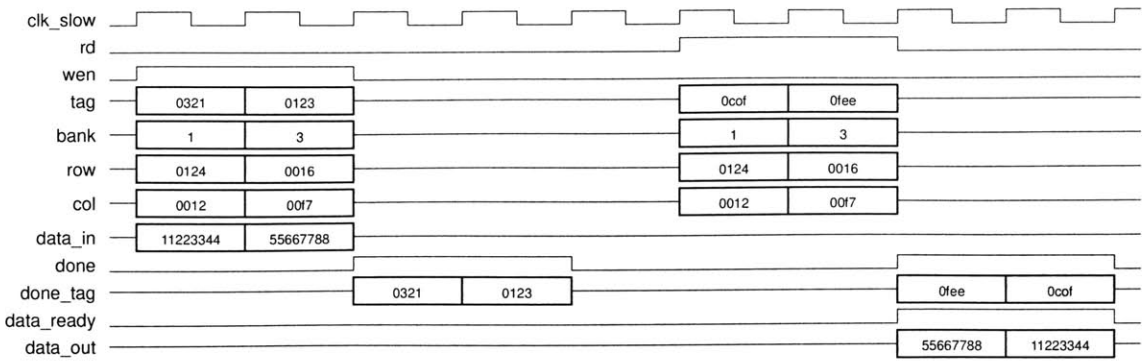

Figure 2-1: Sample ddr_controller write and read requests.

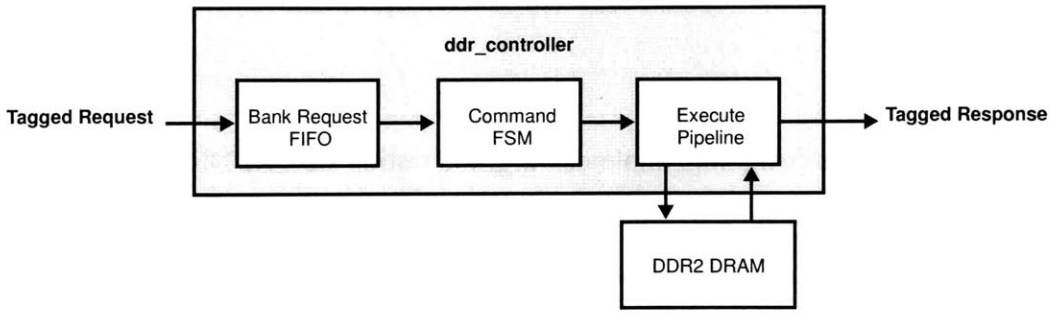

Figure 2-2: ddr_controller block diagram.

controller will indicate when it is ready to start accepting requests to each bank.

2.1.2 Request Stages

The modules which make up ddr_controller are arranged in three main stages. The first stage is the Bank Request FIFO that buffers requests into the controller. Next is the Command FSM stage which is responsible for translating memory access requests into DDR2 DRAM commands. Last, the Execute stage is a pipeline which handles the actual I/O with the DRAM chip. Both the Bank Request FIFO stage and the Command FSM stage run at half the DRAM clock rate, while the Execute stage runs at the full clock rate.

Bank Request FIFO Stage

The Bank Request FIFO accepts and buffers memory access requests until ready to be processed by the corresponding bank in the Command FSM. A new memory request is dequeued from the

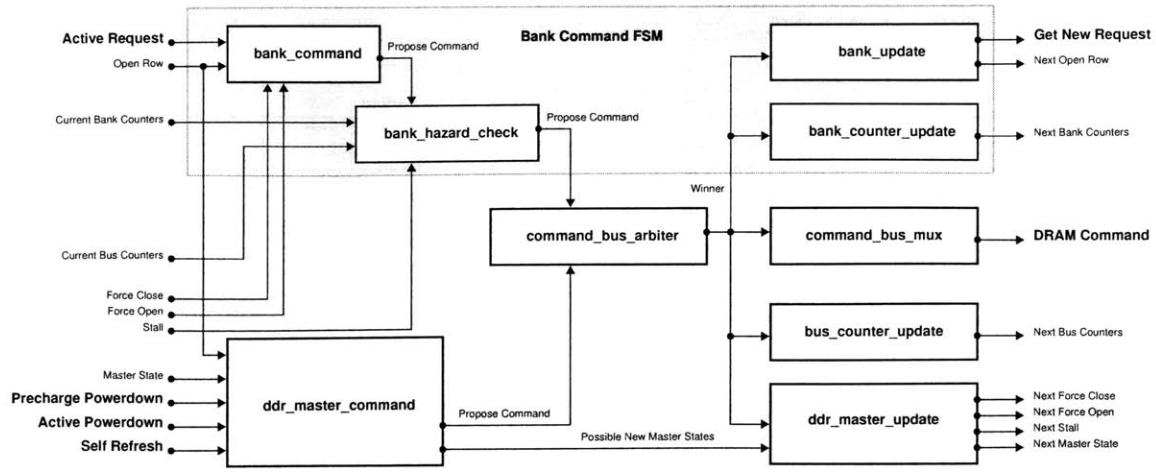

Figure 2-3: Command FSM block diagram.

fifo and presented to the Command FSM stage as the new active request on the cycle after the last command from the current active request wins arbitration, or if there is no current active request.

Command FSM Stage

The Command FSM stage generates possible commands from each of the banks and the chip master, verifies them against timing constraints, selects one to issue, and determines new bank and timing constraint state in a single half-DRAM-speed cycle.

State For each bank, the state consists of the current active request as presented by the Bank Request FIFO, the current open row, and four counters corresponding to timing constraints for the four primary commands (PRECHARGE, ACTIVE, READ, WRITE).

In addition to the bank, the bus maintains a separate set of counters for each of the four primary commands. A fifth requester, known as the chip master, is a more traditional state machine which transitions between named states that represent different physical states the chip is in.

Propose Command At the beginning of each cycle, each bank proposes a command to be issued on the command bus based on its state. The chip master also proposes a command if it needs to, and has the ability to override the commands proposed on the next cycle. If the chip master sets *next_force_close* high for a given bank, then on the next cycle if the bank has a row open, the bank

Request	Active Row	force_close	force_open	stall	Command	New Request
X	X	X	X	1	NOP	0
X	none	1	0	0	NOP	0
X	row_a	1	0	0	PRECHARGE	0
X	none	0	1	0	ACTIVE	0
X	row_a	0	1	0	NOP	0
none	X	0	0	0	NOP	1
op:row_a:col_b	none	0	0	0	ACTIVE	0
op:row_a:col_b	row_c	0	0	0	PRECHARGE	0
op:row_a:col_b	row_a	0	0	0	op	1

Figure 2-4: Proposed command truth table for each bank.

proposes a PRECHARGE, otherwise it proposes a NOP. The *next_force_open* signal has similar behavior with the ACTIVE command.

Hazard Check The command proposed by each bank is verified against the corresponding bank and bus counters in the hazard check phase. If either the bank counter or the bus counter for the proposed command is not 0, the command is changed to a NOP. The master may also pause a bank by setting *next_stall* high, causing the proposed command in the following cycle to be turned into a NOP regardless of counter state.

The commands proposed by the chip master are not subject to any hazard checking, since they are usually exclusive and have more deterministic timing than the four primary bank commands.

Arbitrate In the arbitrate phase, one of the up to five non-NOP proposed commands is selected as the winner. The four banks each have equal priority, and the arbiter cycles through a four different fixed orders based on the last bank that successfully issued a command. The chip master has the lowest priority, and is the default command issued if all the banks propose NOPs. Each command requester is notified whether or not it won arbitration.

Issue and Update Based on the results of arbitration, the correct next state can be determined. If a bank does not propose a command or does not win arbitration, then its state remains the same and all the non-zero bank counters are decremented. If a bank does propose a command and wins arbitration, then the new bank state is determined by which command was just proposed, and the bank counters are updated to include new timing constraints caused by the command about to be

Stage	Read	Write	Other
1	Issue read command	Issue write command Send write data to IOB	Issue other command
2			
3		Drive data strobe	
4		Drive data strobe	
5	Latch incoming data	Drive data strobe	
6			
7	Latch incoming data		
8			
9	Signal done, read data from IOB	Signal done	

Figure 2-5: Execute Stage pipeline diagram.

issued. Each chip master state has two possible next states - one if the chip master wins arbitration, and one if it doesn't.

The arbiter winner is an input to two large muxes. One of them selects the command to be issued on the next cycle and passes it to the Execute stage. The other selects which set of values should be used to update the bus counters to reflect new timing constraints caused by the command that is about to be issued.

Execute Stage

The Execute stage is a simple 9-stage pipeline that issues commands onto the DDR DRAM command bus and reads and writes data when appropriate. This pipeline runs at the full DRAM clock rate, however since its input runs at half the DRAM clock rate, every other command is a NOP.

2.1.3 Request Handling

Initialization

Upon a reset of the ddr_controller, the chip master begins the power-up and initialization sequence for the DRAM chip. During this sequence, the chip master sets the *stall* signal high for each of the banks, preventing them from requesting any operations. Once the sequence is complete, *stall* is brought low, allowing normal DRAM operation.

Read and Write Request

A read or write request issued to the ddr_controller module is immediately enqueued in the appropriate Bank Request Fifo. In the event a Bank Request Fifo is full, the corresponding *bank_ready* signal will be brought low, indicating that no more requests to that bank should be issued.

The Bank Command FSM will request the next command from the corresponding Bank Request FIFO if it either does not have any active requests, or if the last command corresponding to the previous request was acknowledged.

On the next cycle, the new active request is presented to the Bank Command FSM. Based on the state of the bank, a command is proposed. This proposed command is then checked against the Bank Counters and the Bus Counters to see if it violates any timing constraints. If no constraints are violated, the proposed command is sent to the Arbiter, otherwise a NOP is sent.

The Arbiter selects one of the valid commands to be requested, sends an acknowledgement back to the corresponding bank or Master FSM, and passes the command on to the Execute Stage pipeline. If a bank requests a NOP or its requested command is not acknowledged, then its state does not change for the next cycle.

If the bank requests a command and that command is acknowledged, then the bank state is updated for the next cycle, as are the Bank Counters and Bus Counters to reflect new timing constraints caused by issuing this command.

Refresh and Self Refresh

Two events can trigger the refresh cycle. Approximately every 7us, the ddr_controller must issue a REFRESH command. A timer in the Master FSM sends a signal when a refresh is necessary, and resets when a REFRESH command is issued. The refresh cycle also begins if the *self_refresh* signal is brought high.

Upon entering the refresh cycle, the chip master sets *force_close* to high for each of the banks. This causes the banks to immediately request a PRECHARGE if a row is open, otherwise request a NOP. The *force_close* signal effectively acts like *stall* once all banks have precharged.

Once all banks are in the precharged state, a REFRESH_ALL command is requested. If the

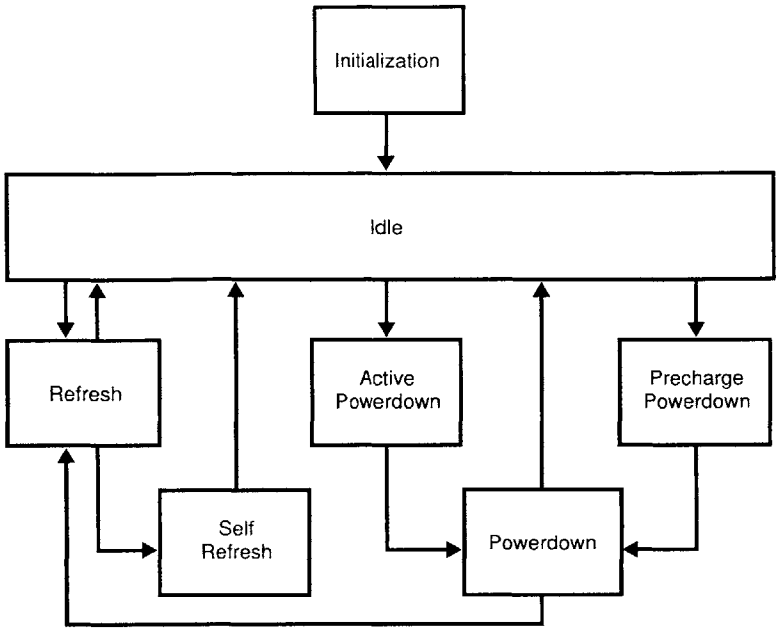

Figure 2-6: Chip master FSM states.

self_refresh signal is high, then CKE is brought low at the same time the REFRESH_ALL command is issued, causing the DRAM to enter the self refresh powerdown mode. If *self_refresh* is low, then *force_close* is brought back low, allowing the banks to resume normal operation.

If the DRAM is in self refresh powerdown mode and *self_refresh* is brought back high, then after the appropriate resynchronization time has elapsed, the *force_close* signals are brought back low, allowing the banks to resume normal operation.

Precharge and Active Powerdown

When the *active_powerdown* signal is brought high, the chip master brings *stall* high for each of the banks, preventing them from issuing more commands. It then brings *CKE* low, causing the DRAM to enter powerdown mode. If all banks were in the precharged state, then the DRAM is in precharge powerdown mode, otherwise it is in active powerdown.

When the *precharge_powerdown* signal is brought high, the Master FSM brings *force_close* high for each of the banks. Once all the banks are precharged, *CKE* is brought low, entering precharge powerdown mode.

There are two powerdown exit conditions. If both *precharge_powerdown* and *active_powerdown* are brought low, then *CKE* is brought high, and after the resynchronization time has elapsed, *stall* is brought low allowing banks to resume normal operation. Powerdown is also exited if a refresh is requested, in which case after the resynchronization time has elapsed, the refresh cycle begins.

2.1.4 Design Extensions

Several extensions to the DDR controller design have been added specifically for debugging and power measurement purposes. These modifications are designed to be modular and easily removed when the ddr_controller is not used for DRAM-specific tests.

Active Powerdown Bank State

DDR2 DRAM does not have distinct commands for precharge powerdown and active powerdown. Rather, the powerdown state entered is based on the current bank state. Under normal operation, the *precharge_powerdown* signal causes all the banks to be precharged and then initiates a powerdown, while the *active_powerdown* signal simply initiates a powerdown without looking at the bank states. If the banks are all precharged when *active_powerdown* goes high, the DRAM will enter the precharge powerdown state rather than the active powerdown state.

For the purposes of power state measurement, the active powerdown state has two hard-coded 4-bit flags. The ACTIVE_BANKS_OPEN flag specifies which banks must be opened before entering active powerdown, while ACTIVE_BANKS_CLOSED specifies which banks must be closed. When combined with the *force_open* and *force_close* signals, the exact state of the banks can be set before entering active powerdown.

Master Command Pattern

Certain specific command patterns are impossible to generate simply by issuing read/write requests to the ddr_controller. To generate these patterns, an extra set of states is added to the chip master FSM. The controller switches to these states at the end of a refresh cycle if the *debug_in[0]* signal is high.

The first state initializes the bank states. The next 8 states cycle continuously issuing commands. By setting the *stall* signal, the banks are prevented from issuing commands, but they also do not update their own states based on the commands issued by the chip master. None of the issued commands are subject to timing constraints, however the number of cycles between each state transition can be set to avoid conflicts. This command cycle is exited whenever a refresh is needed, and resumes at the end of the refresh cycle if the *debug_in[0]* signal is still high.

Refresh Stall

Approximately every 7us, the refresh counter expires and the chip master enters a refresh cycle which precharges all the banks, issues a REFRESH_ALL command, and resets the counter. If the *debug_in[1]* signal is high, then the chip master will issue a NOP instead of a REFRESH_ALL when in a refresh cycle, effectively disabling refresh. If *self_refresh* is brought high while *debug_in[1]* is high, then the chip will enter precharge powerdown instead of self refresh, and will not periodically exit for a refresh cycle.

2.2 SCALE DRAM Subsystem Design

The SCALE DRAM Subsystem is a flexible energy-aware memory system designed to interface with the SCALE cache. The design, analysis, and evaluation of this system is the subject of the Pharris thesis[3], and will not be covered in detail here.

Due to limitations of hardware, many features of the SCALE DRAM Subsystem are not actually implemented. This section gives a quick overview of the actual system which has been implemented and tested in hardware.

2.2.1 SIP Interface

The DRAM subsystem interfaces with the SCALE cache via the SIP interface, and controls four 256Mbit DDR2 DRAM chips. SIP consists of two 40-bit wide unidirectional channels. SIP transactions have 3-bit opcodes, 32-bit addresses, and 5-bit tags, as well as between 1 and 8 data words.

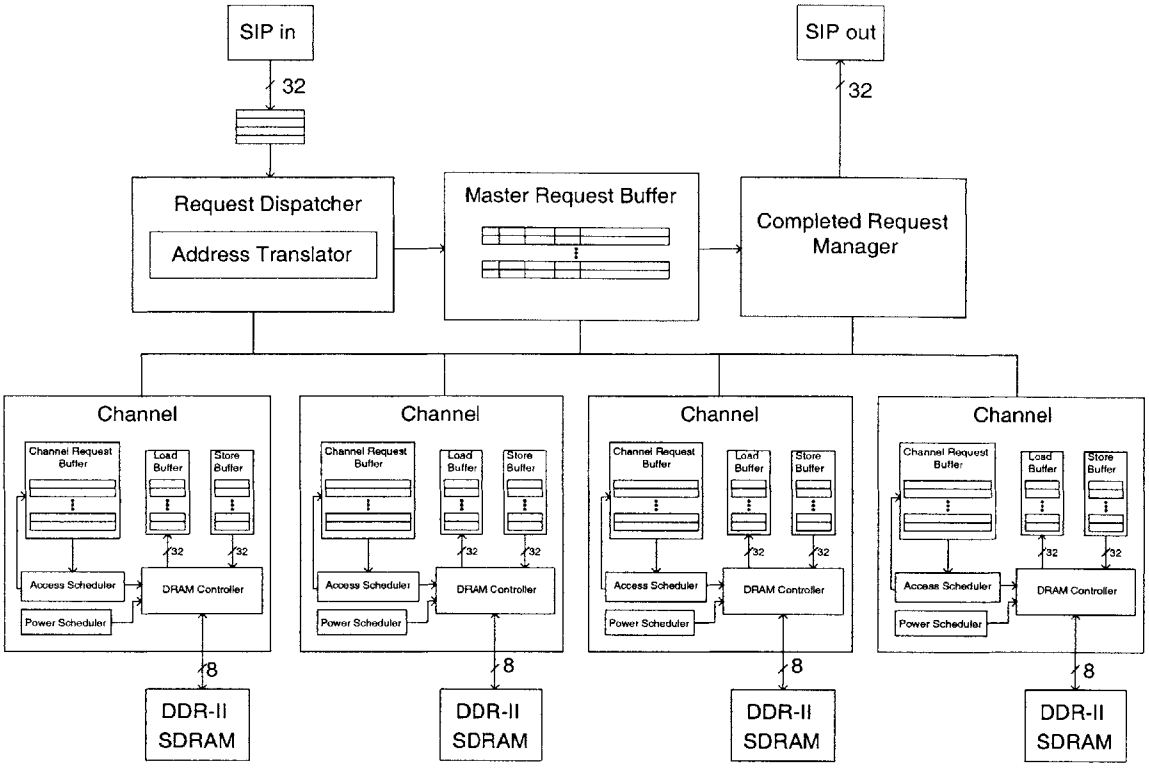

Figure 2-7: SCALE DRAM Subsystem block diagram[3].

The SIP protocol supports byte, half-word, single word, and 8-word cache line read and write requests. Only single word and 8-word line requests have been tested in the physical implementation, and work is in progress to support 4-word half-line requests.

2.2.2 Organization

The DRAM Subsystem is organized into several fully pipelined stages which translate a SIP request into a series of DRAM requests, collect the results, and output the appropriate response over SIP. All of these stages run at half the DRAM clock rate.

Request Dispatcher Accepts requests from SIP and generates the appropriate commands to be passed to each Channel. The Address Translator module within the Request Dispatcher determines how SIP memory addresses map to the physical chip/bank/row/column addresses.

Channel Accepts memory requests from the Request Dispatcher and buffers read and write data to/from the DDR2 DRAM. There is one ddr_controller module per Channel, responsible for actually interfacing with the DRAM chip. The Channel should also be responsible for scheduling powerdown transitions for the ddr_controller, as well as intelligently reordering incoming requests, however both those features are not yet implemented.

Master Request Buffer Keeps track of requests in progress, and notifies the Completed Request Manager once all parts of a request have completed.

Completed Request Manager Assembles responses to completed requests and sends them out over SIP. For store requests, the Completed Request manager just sends an acknowledgement with the corresponding tag. For load requests, the Completed Request Manager reads the data out of the Load Buffer of each Channel and rearranges them according to the Address Translator configuration.

2.3 Implementation Notes

The design presented in the previous two sections represent the final working versions tested in hardware and used for the power analysis in the next chapter. The original designs came from Pharris's thesis[3], however some parts were modified and others completely redone in order to create a working hardware implementation.

2.3.1 DDR2 DRAM Controller Notes

Request Stage

The original ddr_controller module was designed to run at 200MHz and issue commands every cycle. It accomplished this by having four simple bank state machines that converted requests into commands, a fixed-order arbiter, and a pipelined execute stage. After adding a stall signal and

refresh cycle handling, this design worked for isolated memory requests, however it simply could not enforce bus timing constraints, making bank interleaving impossible.

In order to support full bank interleaving with arbitrary timing constraints, the ddr_controller request stage had to be completely redesigned. While the new design correctly issues commands from multiple banks, the logic is too slow to run at the full DRAM clock rate. A quick simulation benchmark of random fully interleaved read/write requests show only a 20% performance penalty when issuing commands every other cycle, mainly because most command sequences already have at least 2 cycle timing constraints spacing them out.

Execute Pipeline

The execute pipeline timing was derived from trial and error. The initial controller design assumed the delay between the FPGA I/O registers and the DRAM I/O buffers was insignificant due to the short traces, however the majority of the delay turned out to be in the FPGA I/O pads themselves, and combined with the wire and DRAM buffer delays, accounted for almost half a clock cycle. A 4ns wire delay was used to make simulation match the actual hardware behavior.

DDR IOB

The hardest part of the system was latching read data in from the DRAM clock domain to the FPGA clock domain. The DDR input registers in each IOB split a DDR input line into two signals that are maintained for a full cycle before getting overwritten. These signals transition based on the data strobe, and need to be latched into the FPGA clock domain before they can be processed. At first, the registers that latch data from the DDR input register were manually placed and routed to meet timing constraints, however with modifications to the v2_ddr4_iob_8 module and carefully selected timing constraints, later iterations of the ddr_controller module did not require any manual place and route modifications.

Interestingly, while the request stage logic is the slowest logic in the controller, performance is actually constrained by the DDR I/O configuration. Specifically, each data strobe input must be routed to eight DDR input registers using local routing resources on the FPGA. The skew in the

data strobe signal across multiple inputs worked out to 1.5ns, along with a delay of 1ns from the input pad in the worst case across all four controllers. Assuming the data strobe and data arrived at the same time on the FPGA pads, the delay and skew added by internal FPGA routing make 135MHz a practical limit for four controllers in the current board without fully manual place and route.

2.3.2 SCALE DRAM Subsystem Notes

Once the ddr_controller module was functioning correctly, the rest of the DRAM subsystem pretty much worked with minimal changes from the original version. Even with the most straightforward policies, the DRAM Subsystem still was a large piece of logic, using up most of the FPGA resources. Since the only off-chip testing interface is AHIP, most of the tests had to be implemented entirely in the FPGA, and occasionally ran into size and speed issues. The DRAM subsystem also used all four chips, adding some extra place and route conflicts that were not present when only one controller was used at a time.

Chapter 3

DDR2 DRAM Properties

This section measures various parameters of the 256Mbit DDR2 DRAM chips installed on the DRAM board. Unless noted otherwise, all measurements are taken with 125MHz DRAM clock, 1.8V supply voltage, 32° C package temperature, 7us refresh cycle, on-die termination enabled, and chip select active on all idle cycles. It is important to note that datasheet values are provided based on a 200MHz clock, 1.8±0.1V supply voltage, up to 85° C package temperature, on-die termination disabled, and chip select disabled on all idle cycles.

3.1 Power Measurements

The power measurements on the following section are taken from the Tester Baseboard. There are two independent 1.8V power supplies with manually calibrated current sensors hooked up to 500kHz ADCs. Each power supply powers two DDR2 DRAM chips.

3.1.1 Methodology

Because the current sensor sampling rate is significantly lower than the DDR2 DRAM operating frequency, each test consists of a tightly looping command pattern which gets averaged into a single current reading. Each test is run three times - once with both chips executing it, and once with each of the two chips idling. For each run, 20 sequential measurements are taken. The reported average

Pattern	Measured Current (mA)	Standard Deviation (mA)	Cycles Per Pattern
idle	179.9	1.5	N/A
w0-w0-w0-w0-w0-w0-w0-w0	266.0	2.1	16
r0-r0-r0-r0-r0-r0-r0-r0	191.3	2.2	16
r0-w0-r0-w0-r0-w0-r0-w0	214.5	1.6	80
w0-w1-w2-w3-w0-w1-w2-w3	286.9	1.9	16
r0-r1-r2-r3-r0-r1-r2-r3	194.7	1.7	16
r0-w1-r2-w3-r1-w0-r3-w2	224.0	1.6	16

Table 3.1: Read/Write current measurements.

Pattern	Measured Current (mA)	Standard Deviation (mA)	Cycles Per Pattern
idle - all banks precharged	179.9	1.5	N/A
idle - all banks active	181.0	1.7	N/A
p0-a0-p0-a0-p0-a0-p0-a0	206.1	1.5	40
p0-p1-p2-p3-a0-a1-a2-a3	234.3	1.6	16
p0-a1-p2-a3-p1-a0-p3-a2	232.4	1.6	16

Table 3.2: Precharge/Active current measurements.

and standard deviation values represent the current averaged across both chips on all runs of a given test assuming one chip is executing the test and the other is idling.

3.1.2 Active Operation Current

Each tested command pattern consists of a fixed ratio of active to idle commands. Assuming that idle current does not change and ignoring the effects of internal DRAM state transitions and refresh cycles, the current draw for a single cycle of a command can be determined.

Because precharges and actives must always occur in a fixed ratio and generally within an interval smaller than the current sampling period, it is not possible to distinguish the two commands. The reported current is the average current of a precharge cycle and an active cycle.

The measured current values for each pattern are listed in Tables 3.1, 3.2, and 3.3. Based on these patterns, estimates for each cycle of a given command is calculated in Table 3.4 along with the corresponding datasheet values.

Of the values calculated, both burst and interleaved read currents are significantly lower than

Pattern	Measured Current (mA)	Standard Deviation (mA)	Cycles Per Pattern
idle	179.9	1.5	N/A
rp0-a0-rp0-a0-rp0-a0-rp0-a0	207.3	1.7	40
wp0-a0-wp0-a0-wp0-a0-wp0-a0	221.6	1.9	48
rp0-rp1-rp2-rp3-a0-a1-a2-a3	226.0	1.5	16
wp0-wp1-wp2-wp3-a0-a1-a2-a3	280.9	2.5	16

Table 3.3: ReadAP/WriteAP/Active current measurements.

Command	Measured Current Delta (mA)	Datasheet Current Delta (mA)
Idle	0	0
Burst Write	86	95
Interleaved Write	106	
Burst Read	11	85
Interleaved Read	15	200
Precharge/Active	107	45
ReadAP/Active	46	55
WriteAP/Active	101	
Refresh		200

Table 3.4: Active command current measurements.

Chip State	Measured Current Delta (mA)	Datasheet Current Delta (mA)
Idle (chip select disabled)	0	0
Active Powerdown (Fast Exit)	-64	-10
Active Powerdown (Slow Exit)	-69	-24
Precharge Powerdown	-72	-25
Self Refresh	-76	-25

Table 3.5: Powerdown current measurements.

the datasheet values, while precharge and active are significantly higher. The low value for read current can be attributed to the point-to-point connection between the FPGA and the DRAM resulting in very little drive current required compared to the datasheet reference 25Ω output load. It is worth noting that there is much less of a difference between interleaved read current and burst read current than the datasheet values.

3.1.3 Powerdown Current

The powerdown tests are similar to the active command tests, except instead of running a command pattern, the chip transitions to a powerdown state. A summary of the difference in current between the idle with chip select disabled state and the various powerdown states can be found in Table 3.5. For each power state, we also measure the sensitivity to clock frequency in Figure 3-1 and supply voltage in Figure 3-2.

From the powerdown measurements, two results stand out. First, the idling current measured is significantly higher than the datasheet spec. The datasheet current deltas are based on a 30 mA per chip idle current with chip select disabled. In our tests, idle current works out to at least 76 mA per chip, with no measurable difference between idling with chip select active and disabled. The second interesting result is that there is much less of a difference between active powerdown fast exit and active powerdown slow exit than the datasheet claims. The relative values of these results

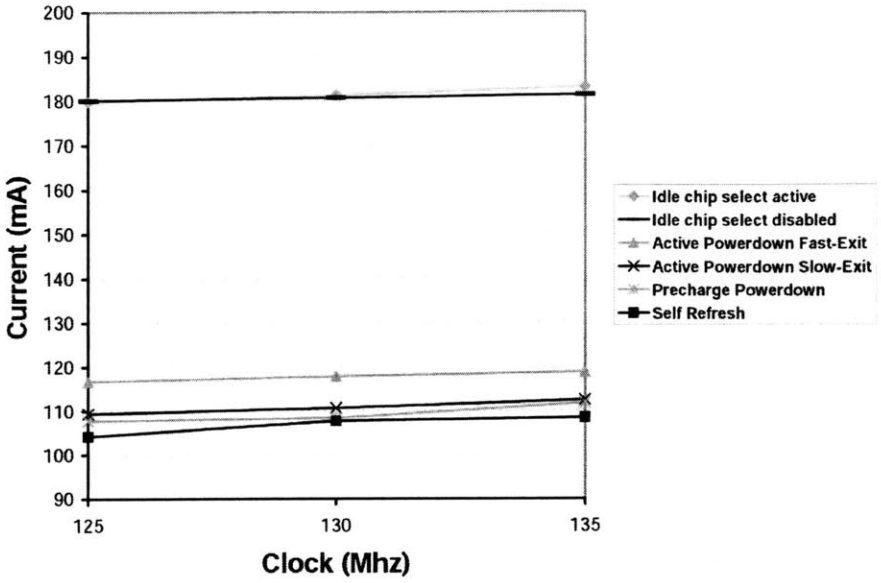

Figure 3-1: Powerdown current for varying clock, 1.8V.

do not appear to depend on either clock frequency or supply voltage.

3.2 Refresh Interval

The refresh interval test profiles the refresh characteristics of a single 256Mbit DDR2 DRAM chip. According to the datasheet, a refresh command must be issued every 7.8125us, and refreshes one row per bank, thus each row requires a refresh every 64ms. This test looks at both the rate and distribution of data corruption when the rows are not refreshed. Since data corruption is a result of charge leaking from the DRAM cells, and leakage is strongly correlated with temperature, tests are performed for two different chip package temperatures.

3.2.1 Methodology

Two different fixed byte patterns are loaded into every word address in banks 0 and 1, for a total of 16MB of data. Refresh stall pin is then brought high for a specific number of seconds. After this delay, refresh stall is brought back low, allowing the controller to resume normal refresh while the

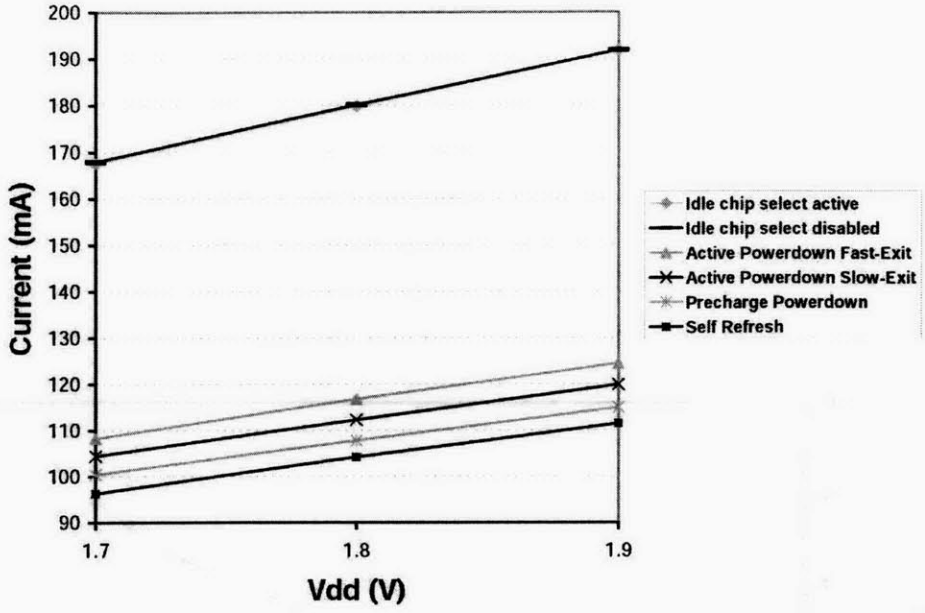

Figure 3-2: Powerdown current for 125MHz, varying Vdd.

Parameter	Number of words
Total tested	4194304
Per run failed average	98117 (2.34%)
Per run failed standard deviation	3099
Failed 1 or more runs	284073 (6.77%)
Failed 5 or more runs	110130 (2.62%)

Table 3.6: Detailed failure statistics for 160s interval.

values are read out sequentially and compared to the pattern.

3.2.2 Refresh Block Results

Figure 3-3 shows the data corruption rate as a function of the refresh delay for both tested temperatures. As expected, there is a crude exponential relationship between refresh delay and bit corruption rate.

The word address of each corrupted word was recorded for 10 runs with a 160s refresh delay at 32°, and the results are summarized in Table 3.6. Of the rows that failed at least 1 run, Figure 3-4 shows the failure consistency.

Each run uses one of two data patterns with alternating bits. Assuming a word fails due to a

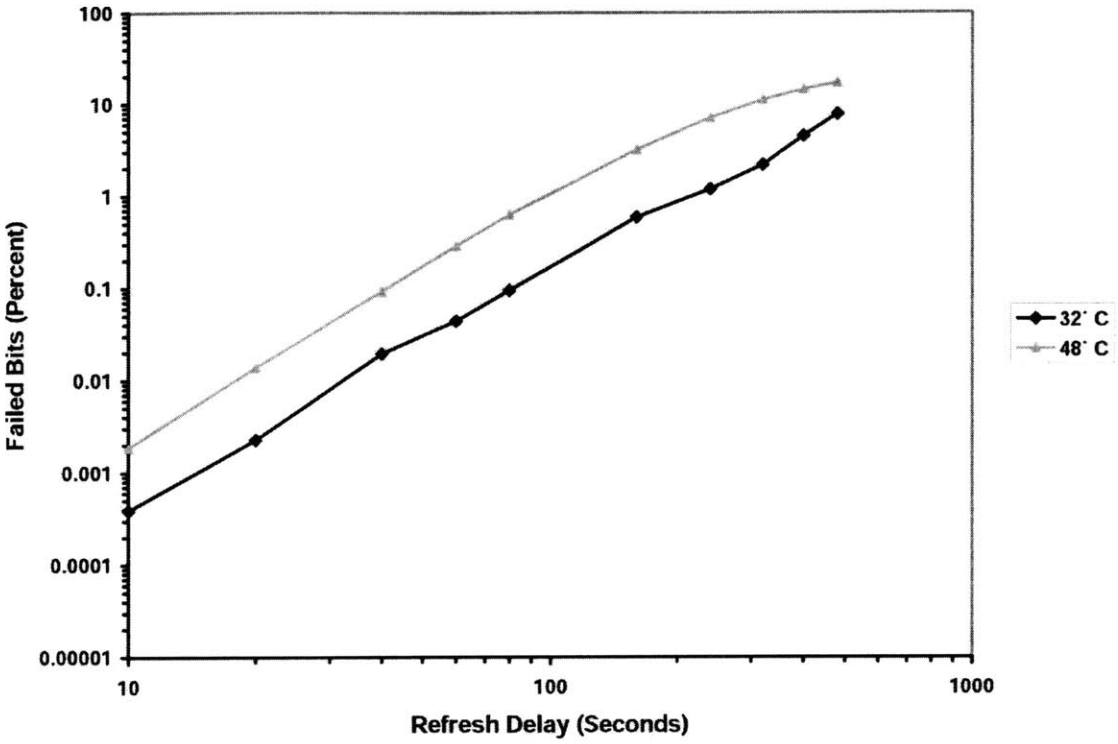

Figure 3-3: Data corruption rate for varying refresh delays.

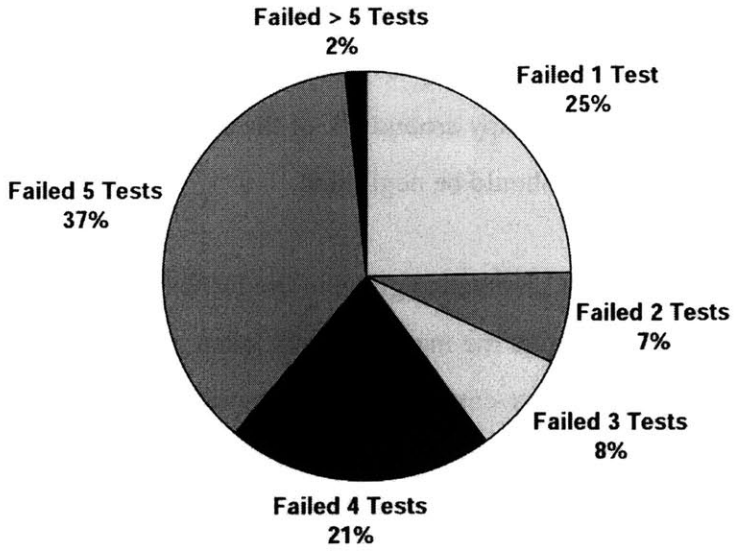

Figure 3-4: Distribution of failed rows across 160s refresh delay runs.

single faulty bit that consistently corrupts one of the two possible values which can be stored on it, each word is expected to fail on half the trials. As shown in Figure 3-4, the bulk of the failed rows failed 4 or 5 of the trials, with more random failure distribution accounting for the large number of only 1 trial failures.

3.3 Error Analysis

There are several possible error sources in the test setup. Most of them would affect only the actual measured current value, not the relative value compared to other tests on the same chip.

Calibration Each current sensor was manually calibrated with the assumption that the raw value returned is linearly proportional to the actual measured current. Raw values from a 36.8Ω resistor, a 22.7Ω resistor, and an open-circuit were used to calculate a best-fit offset and ratio at the 1.8V power setting. Current through a 14.0Ω resistor was then measured and found to be within 2mA of the expected value.

Refresh The command patterns can be interrupted for a refresh cycle once every 7.5us. These refresh cycles, which last around 380ns, will obviously have a different power consumption value from the pattern. Since the cycles occupy around 5% of the command bus time, and less than 25% of a single sample time, the effect should be negligible.

DRAM Chip State The current parameters given in the DDR2 DRAM datasheet are individual samples for a single command, while the measurements taken are for multiple complete patterns which are averaged together. Energy-consuming state transitions that occur in the DRAM chip become part of the measured current, even if they did not occur during the cycle the command was issued or active.

One example of this is the write-recovery process when switching from a burst write to a burst read. Based on the values calculated in Table 3.4, an 80-cycle r0-w0-r0-w0-r0-w0-r0-w0 pattern should take 190 mA, but it is measured as taking 215 mA.

Chapter 4

Policy Evaluation

In his thesis, Pharris[3] explored a number of different operating policies for the SCALE DRAM Subsystem using a software simulator and power consumption values from the Micron DDR2 DRAM datasheet. Since then, significant changes have been made to the underlying DDR2 DRAM controller design as well as parts of the DRAM Subsystem in order to function in hardware. These changes make the software DRAM Subsystem model no longer an accurate representation of the performance of the actual SCALE DRAM Subsystem, however work is in progress to update the model.

In this section, we will look at some of the important policy-relate conclusions drawn from the old software model and discuss how they may be affected by both the new system design as well as updated power consumption estimates.

4.1 Address Policy

The address translator module within the Request Dispatcher is responsible for mapping 32-bit SIP addresses to chip, bank, row, and column addresses. When selecting an address mapping policy, performance and power tradeoffs can vary greatly depending on the actual workload.

	Bank Mapping Enabled	Bank Mapping Disabled
4 Chip Striping	row:bank:col:chip	bank:row:col:chip
2 Chip Striping	chip[1]:row:bank:col:chip[0]	chip[1]:bank:row:col:chip[0]
No Chip Striping	chip:row:bank:col	chip:bank:row:col

Table 4.1: Example address translator policies

4.1.1 Maximizing Performance

For the majority of the benchmarks, an address policy which striped 8-word cache line requests across all four chips and mapped rows across banks resulted in the highest performance.

By striping requests across all four chips, the effective memory bandwidth is a little less than twice the SIP bandwidth, so the memory system is capable of saturating its interface to the processor. With striping across only two chips, the peak bandwidth is equal to SIP bandwidth, but precharge/active and refresh overhead makes it slightly less. The downside to striping is that all four chips must be active for each cache line request, so the only time a chip may powerdown is if there are no requests coming from the processor.

Because each bank may have an open row, striping rows across banks, also known as bank mapping, allows four times as many rows to be open per chip, lowering the number of precharge/actives required for local access patterns. There are very few cases where bank mapping decreases performance, and the measured power values show that both the number of open banks and bank interleaving have little effect on power consumption.

4.1.2 Minimizing Power

The minimum power address policy depends much more on workload and powerdown policy. Every workload exhibits a certain locality over a given period of time. The general chip striping policy should be to stripe cache line requests across the minimum number of chips such that the total address space exceeds the workload locality for a duration on the order of the powerdown threshold. Because powerdown is applied to a whole chip, striping across fewer chips increases the chance that a given chip will be idle long enough to powerdown. If a certain number of chips must remain active, then it is best to maximize performance, since the faster requests are handled, the sooner the next powerdown opportunity will arrive.

Command Type	Precharge/Active Break Even Point
Idle	100
Write	5
Read	26

Table 4.2: Bank interleaving power compared to precharge/active power

Once again, bank mapping is a good idea for minimizing power as well. Though it increases the number of open rows, unless requests are local enough to fit within a single row, more power is saved by avoiding the extra precharge/active cycles than is spent keeping the rows open. The values in Table 4.2 show how many cycles of each request type must be addressed to the same row before more energy is saved by not having bank interleaved accesses than is spent by the extra precharge/active cycle to switch rows.

4.2 Powerdown Policy

Both Delaluz et al[2] and Pharris[3] found that in a system with a shallow powerdown state and a deep powerdown state, the most effective policy was to transition to the shallow state as soon as possible, and then use a threshold value to determine when to transition to the deep powerdown state. These findings were based on the assumption that the deep powerdown state offers significant power savings over the shallow powerdown state while also having significantly higher resynchronization time. Because DDR2 DRAM has more than two powerdown states, a state must be chosen for each role.

4.2.1 Shallow Powerdown State

The main characteristic of a shallow powerdown state is having a low resynchronization time while offering some amount of power savings over the idle state. The three possible shallow power states are active powerdown fast exit, active powerdown slow exit, and precharge powerdown.

For the purposes of power consumption, active powerdown slow exit requires 4 extra cycles of idle power consumption, but gains lower consumption during active powerdown when compared to fast exit. Assuming idle current is 76mA (the minimum possible value observed), any active

powerdown duration longer than 15 cycles will consume less power with slow exit than with fast exit. Of course fast exit will have better benchmark performance than slow exit, however that is much more workload dependent.

Precharge powerdown has lower power consumption than active powerdown slow exit, but occasionally will precharge a row that needs to be activated on the next access. As long as these accidental precharges happen less often than once every 36 cycles of powerdown, precharge powerdown will consume less power, though again with a certain performance penalty.

4.2.2 Deep Powerdown State

A deep powerdown state is supposed to offer significant reduction in power consumption at the expense of a long resynchronization time. The only two states that could qualify for a deep powerdown state are precharge powerdown and self refresh. Thanks to the 200 cycle self refresh resynchronization time, a the chip will have had to be in deep powerdown for longer than 3600 cycles before self refresh consumes less power than precharge powerdown.

While it is not entirely unreasonable for DRAM to be in a deep powerdown state for much longer than 3600 cycles, in most cases the higher level system is knowingly going to sleep, and can explicitly request a transition to self refresh instead of waiting for a powerdown policy to switch. One of the major benefits of self refresh mode is that the DRAM controller does not need to maintain any state about the DRAM chip, other than the fact that it is in the self refresh state. Powerdown policies that include the self refresh state should look at the impact of powering down the controller as well as the DRAM during the deep powerdown state.

4.3 Refresh Policy

For extended powerdown periods, the bulk of DRAM power consumption goes into the periodic refresh cycles needed to maintain data integrity. According to the DDR2 DRAM datasheet, self refresh mode a refresh cycle occurs every 7.8125us and consumes 200 mA of current.

As discussed earlier, refreshes are required because charge gradually leaks out of storage cells,

causing them to lose the stored value. A combination of bit profiling, error correcting codes, and temperature dependent refresh intervals could allow further power savings by decreasing the frequency of refresh commands.

4.3.1 Bit Profiling

Based on the error distribution in Figure 3-4, more than half of the word errors come from words that are consistently failing for a given refresh delay. Mapping the usable address space to work around these spots is a quick way to decrease the data corruption rate when operating on a longer refresh cycle.

While it is possible to design and implement a hardware address mapper that handles bad memory areas, it might be a better idea to leave this task to higher level software. The Linux kernel already has a patch that lets it avoid using known bad memory spaces. An application-specific kernel could partition the memory into long-term storage and short-term scratch areas, and assign the less robust memory spaces to the short-term role where they will be refreshed more frequently.

4.3.2 Error Correcting Codes

Standard ECC SDRAM modules use an extra 8 bits per 64 bits of data to store error correcting code. These configurations can correct single-bit errors and detect some multi-bit errors. If the data corruption rate is sufficiently low within a DRAM chip, organizing the data internally in an ECC configuration could compensate for errors caused by longer refresh intervals.

For every 8 words read, one refresh cycle must be skipped to save power. For every 8 words written, two refresh cycles must be skipped. A balanced load of 16MB data read and 16MB written will require at least one refresh delay longer than 12.3 seconds to save power. Based on the corruption rates measured for the 20 second refresh delay, the standard ECC scheme combined with bit profiling to identify any known bad areas should result in a quite reliable system for long-term low duty cycle DRAM storage.

4.3.3 Temperature Based Refresh

One of the power-saving features of mobile SDRAM is a special mode known as Temperature Compensated Self Refresh. Essentially the refresh interval during self-refresh mode is adjusted based on the ambient temperature. The 7.8125us refresh interval quoted in the DDR2 DRAM datasheet assumes the DRAM is operating at its maximum temperature of 85° C.

As shown in Figure 3-3, there is about an order of magnitude difference in bit corruption rate between operating at 32° C and 48° C. Assuming this relationship holds for the entire temperature range, it should be possible to set the refresh delay to 7.8125ms at 32° C and not experience any data corruption.

Chapter 5

Conclusion

In addition to providing a functional hardware implementation of the SCALE DRAM Subsystem, this thesis demonstrates that worst case specifications are frequently less than ideal for common case optimizations. Though we are unable to use the new DDR2 DRAM power figures to simulate and evaluate policies on actual SCALE DRAM Subsystem workloads, we identify several unusual results which would require closer examination.

As far as powerdown states are concerned, active powerdown fast exit should be reserved for systems where power savings is an afterthought and performance is the main driver. Active powerdown slow exit should be reevaluated against precharge powerdown, since it actually is quite a bit closer in power consumption, and has the additional benefits of avoiding accidental precharges. The self refresh powerdown state is rather unexciting by itself, but coupling it with a memory controller that can also power down could result in significant power savings for deep sleep modes.

On the refresh side, the datasheet refresh requirements are extremely conservative. Of course considering the primary purpose of memory is to store information without error, this is is normal, but significantly more aggressive refresh settings coupled with bit profiling and ECC data storage could result in a lower power memory for low duty cycle applications.

5.1 Future Work

5.1.1 DRAM Controller

The current DRAM controller design is optimal for the hardware it is running on, but should be redesigned if moved to an FPGA that is significantly faster relative to the DRAM operating frequency in order to ship the performance bottleneck back out to the DRAM.

The current DRAM controller can only issue a command every other cycle because the hazard checking, arbitration, and update stages are relatively slow. Certain commands such as precharges have many fewer timing constraints, and might be able to squeeze in opportunistically on the idle cycles.

5.1.2 SCALE DRAM Subsystem

The current implementation of the SCALE DRAM Subsystem has a direct chip striped address map, no powerdown state support, and no ability to reorder requests. While these functions were designed and evaluated in the software simulator, implementing them in hardware could highlight FPGA-specific design issues that were not present in the software model.

Bibliography

[1] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper and K. Asanović. The Vector-Thread Architecture In ISCA 2004

[2] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, M. J. Irwin. DRAM Energy Management Using Hardware and Software Directed Power Mode Control. In *Proceedings of the International Conference on High Performance Computer Architecture (HPCA)*, Monterrey, Mexico, January 19-24, 2001.

[3] B. Pharris. *The SCALE DRAM Sub-System.*

[4] 256Mb: x4,x8,x16 DDR2 SDRAM Datasheet, Micron Corporation.

[5] DRAM Board Online Documentation.
http://www.icsi.berkeley.edu/ beck/membdoc/membd1/Membdoc.htm

[6] Xilinx Virtex-II Patform FPGAs: Introduction and Overview, Xilinx Corporation.

[7] Tester Baseboard Online Documentation.