International Journal for Modern Trends in Science and Technology
Volume: 02, Issue No: 10, October 2016
http://www.ijmtst.com
ISSN: 2455-3778

Time and Low Power Operation Using Embedded Dram to Gain Cell Data Retention

D. Singa Reddy[1] | B. Govardhana[2]

[1]PG Scholar, Department of ECE, Geethanjali Engineering College, Nannur-V, Kurnool-Dist.
[2]Assistant Professor, Department of ECE, Geethanjali Engineering College, Nannur-V, Kurnool-Dist.

To Cite this Article
D. Singa Reddy and B. Govardhana, Time and Low Power Operation Using Embedded Dram to Gain Cell Data Retention, International Journal for Modern Trends in Science and Technology, Vol. 02, Issue 10, 2016, pp. 106-111.

ABSTRACT

Logic compatible gain cell (GC)-embedded DRAM (eDRAM) arrays are considered an alternative to SRAM because of their small size, non rationed operation, low static leakage, and two port functionality. But traditional GC-eDRAM implementations require boosted control signals in order to write full voltage levels to the cell to reduce the refresh rate and shorten access times. The boosted levels require an extra power supply or on-chip charge pumps, as well as nontrivial level shifting and toleration of high voltage levels. In this paper, we present a novel, logic compatible, 3T GC-eDRAM bit cell that operates with a single-supply voltage and provides superior write capability to the conventional GC structures. The proposed circuit is demonstrated in 0.25µm CMOS process targeted at low power, energy efficient application.

KEYWORDS: *Embedded DRAM, gain cell, data retention time, and low power operation.*

Copyright © 2016 International Journal for Modern Trends in Science and Technology
All rights reserved.

I. INTRODUCTION

Digital system designers, faced with a large and rapidly growing gap between available chip I/O bandwidth and demand for bandwidth, attempt to find clever system partitioning schemes to reduce demand. In a design environment in which millions of devices can be economically integrated onto a chip, but only a few hundred I/O pins can be supported, efficient system partitioning requires ever more on-chip storage. ASIC designs now commonly require large amounts of on-chip memory, usually implemented as static random-access memory (SRAM). SRAM with PFET loads can be supported on any CMOS process but requires about 1,000 λ2 of area per bit. Some vendors offer special processes, adapted from commercial SRAM manufacture, that include polysilicon resistor loads; resistor-load SRAM cells can be made as small as a few hundred λ2. Dynamic memory (DRAM) circuits, while potentially much more compact than SRAMs, have fallen out of favor with ASIC designers. We speculate that this is partly because of the perceived complexity of DRAM circuit design and partly because of the unfavorable scaling of FET leakage currents in sub-micron CMOS that makes dynamic circuitry more problematic. Few computer systems requires as much memory bandwidth per bit as hardware accelerators for interactive 3D graphics. Current high-end graphics hardware must support bandwidth of order 10 Gbytes/second into relatively small amounts of total storage, perhaps a few 10s of MBytes. Graphics systems built with commercial RAM chips therefore feature many-way partitioning and considerable replication of data across partitions. Our research for the past 15 years has focussed on ways to remove the memory bandwidth bottleneck by combining graphics processors with on-chip

RAM. Large improvements can be realized simply by removing the column decoder of conventional RAM designs, thereby liberating the huge bandwidth from many-way parallel access inherent in rectangular memory organizations. We have built and fielded three generations of experimental systems to examine the validity of this idea (references [1-3]). Within the past couple of years, this idea has become main stream, particularly in cost-sensitive applications such as graphics accelerators for PCs.

An efficient implementation of our latest system, Pixel Flow, required on-chip memory with much higher density than could be achieved with conventional SRAM. To satisfy this requirement we developed the 1-transistor embedded DRAM that is the subject of this paper. Designed in scalable geometric rules, the DRAM is about four times denser than P-load SRAM in the same rules, dense enough to permit about 1Mb of RAM on a 10mm square chip in 0.5µ CMOS. It is reasonably well optimized for low power consumption, runs at the processors' speed (100MHz), and is fairly simple and portable to other applications. The embedded DRAM serves as a register file for an array of 256 8-bit processors in a graphics 'enhanced memory chip' (EMC) [3]; each processor 'owns' 384 bytes of memory. The memory layout is bit-sliced, so the organization is 2,048 bits (columns) by 384 words (rows). In the word dimension, the memory is composed of 'pages', each a block of 32 words. Each page is a self-contained memory system with bit cells arrayed along a pair of bit lines, a local sense amp and precharger, and an interface to a (differential) data bus that delivers data between the (12) pages and the processor. The column dimension has no decoder; on every cycle of chip operation, data is read from or written to 2,048 bits of memory.

The organization into many small pages is the central design feature of this DRAM, inspired by Don Speck's observation in the design of the MOSAIC DRAM [4] that short, low-capacitance bit lines allow large voltage differences that can be sensed with simple sense amps. Simple sense amps, in turn, allow compact realizations of small memory modules. This advantageous design 'spiral' also leads to considerable power savings. If data is fetched from one of an array of small modules, the modules that are not accessed remain quiescent and burn no power.

II. Related Work

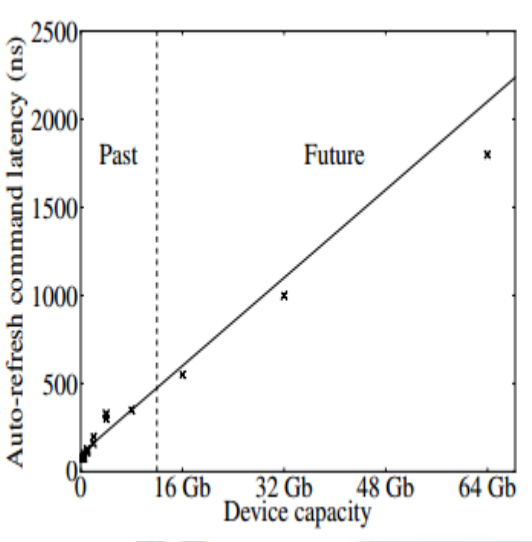
2.1. DRAM Organization and Operation

We present a brief outline of the organization and operation of a modern DRAM main memory system. Physical structures such as the DIMM, chip, and sub-array are abstracted by the logical structures of rank and bank for clarity where possible. More details can be found in [5]. A modern DRAM main memory system is organized hierarchically as shown in Figure (a). The highest level of the hierarchy is the channel. Each channel has command, address, and data buses that are independent from those of other channels, allowing for fully concurrent access between channels. A channel contains one or more ranks. Each rank corresponds to an independent set of DRAM devices. Hence, all ranks in a channel can operate in parallel, although this rank-level parallelism is constrained by the shared channel bandwidth. Within each rank is one or more banks. Each bank corresponds to a distinct DRAM cell array. As such, all banks in a rank can operate in parallel, although this bank-level parallelism is constrained both by the shared channel bandwidth as well as by resources that are shared between banks on each DRAM device, such as device power.

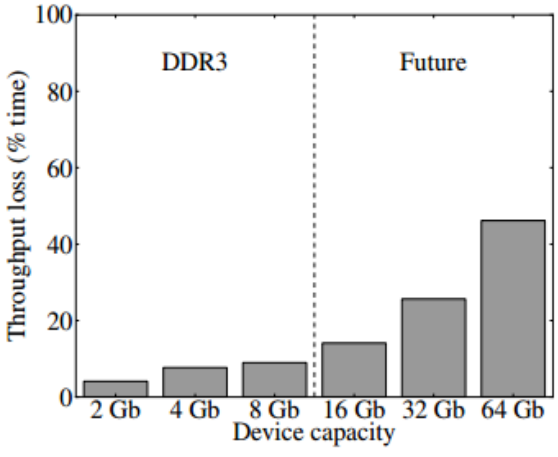
Each DRAM bank consists of a two-dimensional array of DRAM cells, as shown in Figure (b). A DRAM cell consists of a capacitor and an access transistor. Each access transistor connects a capacitor to a wire called a bitline and is controlled by a wire called a word line. Cells sharing a word line form a row. Each bank also contains a row of sense amplifiers, where each sense amplifier is connected to a single bitline. This row of sense amplifiers is called the bank's row buffer. Data is represented by charge on a DRAM cell capacitor. In order to access data in DRAM, the row containing the data must first be opened (or activated) to place the data on the bitlines. To open a row, all bitlines must previously be precharged to VDD/2. The row's wordline is enabled, connecting all capacitors in that row to their respective bitlines. This causes charge to flow from the capacitor to the bitline (if the capacitor is charged to VDD) or vice versa (if the capacitor is at 0 V). In either case, the sense amplifier connected to that bitline detects the voltage change and amplifies it, driving the bitline fully to either VDD or 0 V. Data in the open row can then be read or written by sensing or driving the voltage on the appropriate bitlines.

Successive accesses to the same row, called row hits, can be serviced without opening a new row.

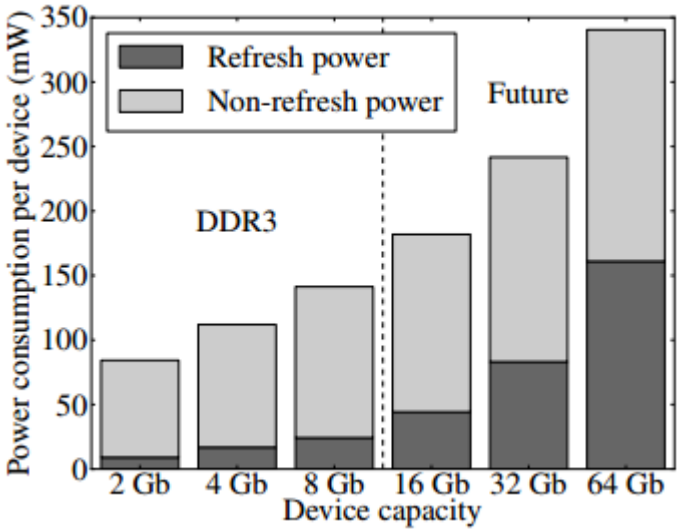
Accesses to different rows in the same bank, called row misses, require a different row to be opened. Since all rows in the bank share the same bitlines, only one row can be open at a time. To close a row, the row's word line is disabled, disconnecting the capacitors from the bitlines, and the bitlines are precharged to VDD/2 so that another row can be opened. Opening a row requires driving the row's wordline as well as all of the bitlines; due to the high parasitic capacitance of each wire, opening a row is expensive both in latency and in power. Therefore, row hits are serviced with both lower latency and lower energy consumption than row misses. The capacity of a DRAM device is the number of rows in the device times the number of bits per row. Increasing the number of bits per row increases the latency and power consumption of opening a row due to longer wordlines and the increased number of bitlines driven per activation [6]. Hence, the size of each row has remained limited to between 1 KB and 2 KB for several DRAM generations, while the number of rows per device has scaled linearly with DRAM device capacity [7, 8, 9].



(a) Refresh latency



(b) Throughput loss



(c) Power consumption

2.2. DRAM Retention Time Distribution

The time before a DRAM cell loses data depends on the leakage current for that cell's capacitor, which varies between cells within a device. This gives each DRAM cell a characteristic retention time. Previous studies have shown that DRAM cell retention time can be modeled by categorizing cells as either normal or leaky. Retention time within each category follows a log-normal distribution [8, 10, 11]. The overall retention time distribution. The DRAM refresh interval is set by the DRAM cell with the lowest retention time. However, the vast majority of cells can tolerate a much longer refresh interval. In a 32 GB DRAM system, on average only ≈ 30 cells cannot tolerate a refresh interval that is twice as long, and only ≈ 103 cells cannot tolerate a refresh interval four times longer. For the vast majority of the 1011 cells in the system, the refresh interval of 64 ms represents a significant waste of energy and time.

III. Implementation

3.1. Overview

A conceptual overview of our mechanism is shown in Figure (d). We define a row's retention time as the minimum retention time across all cells in that row. A set of bins is added to the memory controller, each associated with a range of retention times. Each bin contains all of the rows whose retention time falls into that bin's range. The shortest retention time covered by a given bin is the bin's refresh interval. The shortest retention time that is not covered by any bins is the new default refresh interval. In the example shown in Figure 4, there are 2 bins. One bin contains all rows with retention time between 64 and 128 ms; its bin

refresh interval is 64 ms. The other bin contains all rows with retention time between 128 and 256 ms; its bin refresh interval is 128 ms. The new default refresh interval is set to 256 ms.

A retention time profiling step determines each row's retention time (1 in Figure (d)). For each row, if the row's retention time is less than the new default refresh interval, the memory controller inserts it into the appropriate bin 2 . During system operation 3 , the memory controller ensures that each row is chosen as a refresh candidate every 64 ms. Whenever a row is chosen as a refresh candidate, the memory controller checks each bin to determine the row's retention time. If the row appears in a bin, the memory controller issues a refresh operation for the row if the bin's refresh interval has elapsed since the row was last refreshed. Otherwise, the memory controller issues a refresh operation for the row if the default refresh interval has elapsed since the row was last refreshed. Since each row is refreshed at an interval that is equal to or shorter than its measured retention time, data integrity is guaranteed. Our idea consists of three key components: (1) retention time profiling, (2) storing rows into retention time bins, and (3) issuing refreshes to rows when necessary. We discuss how to implement each of these components in turn in order to design an efficient implementation of our mechanism.

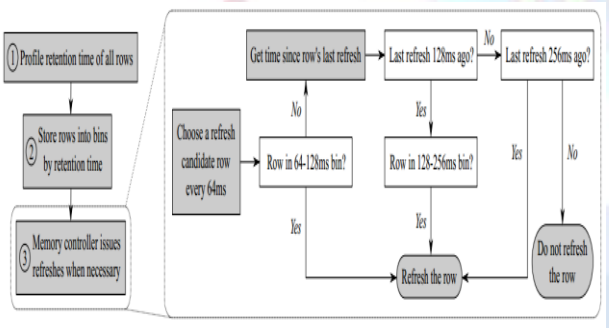


Figure (d): RAIDR operation

3.2. Retention Time Profiling

Measuring row retention times requires measuring the retention time of each cell in the row. The straightforward method of conducting these measurements is to write a small number of static patterns (such as "all 1s" or "all 0s"), turning off refreshes, and observing when the first bit changes . Before the row retention times for a system are collected, the memory controller performs refreshes using the baseline auto refresh mechanism. After the row retention times for a system have been measured, the results can be saved in a file by the operating system. During future boot-ups, the results can be restored into the memory controller without requiring further profiling, since retention time does not change significantly over a DRAM cell's lifetime [8].

3.3. Storing Retention Time Bins

The memory controller must store the set of rows in each bin. A naive approach to storing retention time bins would use a table of rows for each bin. However, the exact number of rows in each bin will vary depending on the amount of DRAM in the system, as well as due to retention time variation between DRAM chips (especially between chips from different manufacturing processes). If a table's capacity is inadequate to store all of the rows that fall into a bin, this implementation no longer provides correctness (because a row not in the table could be refreshed less frequently than needed) and the memory controller must fall back to refreshing all rows at the maximum refresh rate. Therefore, tables must be sized conservatively (i.e. assuming a large number of rows with short retention times), leading to large hardware cost for table storage. To overcome these difficulties, we propose the use of Bloom filters [2] to implement retention time bins. A Bloom filter is a structure that provides a compact way of representing set membership and can be implemented efficiently in hardware [4, 12].

A Bloom filter consists of a bit array of length m and k distinct hash functions that map each element to positions in the array. Figure 5a shows an example Bloom filter with a bit array of length m = 16 and k = 3 hash functions. All bits in the bit array are initially set to 0. To insert an element into the Bloom filter, the element is hashed by all k hash functions, and all of the bits in the corresponding positions are set to 1 ('1' in Figure (e)). To test if an element is in the Bloom filter, the element is hashed by all k hash functions. If all of the bits at the corresponding bit positions are 1, the element is declared to be present in the set' 2'. If any of the corresponding bits are 0, the element is declared to be not present in the set'3'. An element can never be removed from a Bloom filter. Many different elements may map to the same bit, so inserting other elements '4' may lead to a false positive, where an element is incorrectly declared to be present in the set even though it was never inserted into the Bloom filter '5'. However, because bits are never reset to 0, an element can never be incorrectly declared to be not present in the set; that is, a false negative can never occur. A Bloom filter is therefore a highly storage-efficient set

representation in situations where the possibility of false positives and the inability to remove elements are acceptable. We observe that the problem of storing retention time bins is such a situation. Furthermore, unlike the previously discussed table implementation, a Bloom filter can contain any number of elements; the probability of a false positive gradually increases with the number of elements inserted into the Bloom filter, but false negatives will never occur. In the context of our mechanism, this means that rows may be refreshed more frequently than necessary, but a row is never refreshed less frequently than necessary, so data integrity is guaranteed.

The Bloom filter parameters m and k can be optimally chosen based on expected capacity and desired false positive probability [13]. The particular hash functions used to index the Bloom filter are an implementation choice. However, the effectiveness of our mechanism is largely insensitive to the choice of hash function, since weak cells are already distributed randomly throughout DRAM [8]. The results presented by use a hash function based on the xorshift pseudo-random number generator [14], which in our evaluation is comparable in effectiveness to H3 hash functions that can be easily implemented in hardware [3, 15].

IV. Experimental Results

The DRAM has been fabricated on several wafer lots over the past year, and full functionality has been demonstrated. Test performed on wafers with near-nominal fabrication parameters were found to operate at 150MHz, even in packages, where temperatures are generally higher than on a thermally massive wafer test chuck. A schmoo plot for a nominal-fabrication packaged chip is shown in Figure (f), showing its limits of operation over power supply voltage and clock speed.

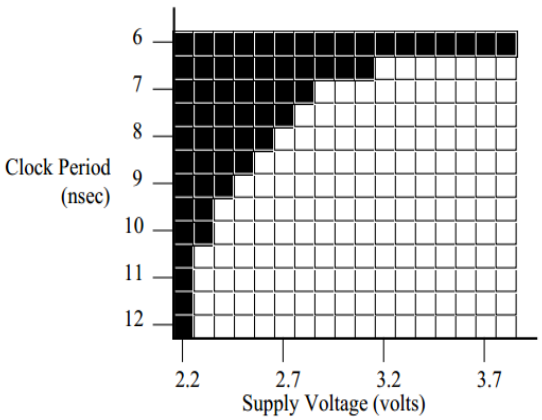


Figure (e). Schmoo plot for a nominal chip.

We also have available a skewed wafer lot, in which a few wafers in the lot have various parameters purposely biased toward either worst-case (slow) or best-case (fast) corners. These wafers also demonstrated full functionality; the DRAM on the slow skewed wafers operates correctly at over 125MHz (design speed was 100MHz). To evaluate data retention time, we devised tests that leave one or more pages of memory quiescent (and unrefreshed) for long, variable periods of time. During these time periods, the tests run memory-intensive code on other pages of memory, in order to maximize noise. These tests were run on wafers mounted on a 'hot chuck', a device that allows the wafer temperature to be controlled fairly accurately. All memory data failures were of the form of initially-high storage nodes going low, confirming the hypothesis that junction leakage is the main data-loss mechanism at work in these DRAMs.

V. Conclusion

We presented Retention-Aware Intelligent DRAM Refresh (RAIDR), a low-cost modification to the memory controller that reduces the energy and performance impact of DRAM refresh. RAIDR groups rows into bins depending on their required refresh rate, and applies a different refresh rate to each bin, decreasing the refresh rate for most rows while ensuring that rows with low retention times do not lose data. To our knowledge, RAIDR is the first work to propose a low-cost memory controller modification that reduces DRAM refresh operations by exploiting variability in DRAM cell retention times.

References

[1] Poulton, J., Fuchs, H., Austin, J., Eyles, J., Heinecke, J., Hsieh, C-H, Goldfeather, J., Hultquist, J., and Spach, S., "PIXEL-PLANES: Building a VLSI-Based Graphic System," Proceedings of Conference on Advanced Research in VLSI, 1985, pp 35-60.

[2] Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., and Israel, L., "A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," Computer Graphics (Proc. of SIGGRAPH '89), Vol. 23, No. 3, pp 79-88.

[3] Molnar, S., J. Eyles, and J. Poulton, "PixelFlow: High-Speed Rendering Using Image Composition," Computer Graphics (Proc. of SIGGRAPH '92), Vol. 26, No. 2, pp. 231-240.

[4] Speck, D., "The Mosaic Fast 512K Scalable CMOS dRAM," Proceedings of Conference on Advanced Research in VLSI, 1991, pp 229-244.
[5] B. Keeth et al., DRAM Circuit Design: Fundamental and High-Speed Topics. Wiley-Interscience, 2008.
[6] Hybrid Memory Cube Consortium, "Hybrid Memory Cube," 2011. Available: http://www.hybridmemorycube.org/
[7] JEDEC, "DDR SDRAM Specification," 2008.
[8] JEDEC, "DDR2 SDRAM Specification," 2009.
[9] JEDEC, "DDR3 SDRAM Specification," 2010.
[10] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled DRAMs," IEEE Electron Device Letters, 2009.
[11] Y. Li et al., "DRAM yield analysis and optimization by a statistical design approach," IEEE Transactions on Circuits and Systems, 2011.
[12] M. J. Lyons and D. Brooks, "The design of a Bloom filter hardware accelerator for ultra low power systems," in ISLPED-14, 2009.
[13] D. E. Knuth, The Art of Computer Programming, 2nd ed. AddisonWesley, 1998, vol. 3.
[14] G. Marsaglia, "Xorshift RNGs," Journal of Statistical Software, 2003.
[15] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," IEEE Transactions on Computers, 1997.