Optimizing the DRAM Refresh Count for Merged DRAM/Logic LSIs

Taku Ohsawa† Koji Kai‡ Kazuaki Murakami†
† Department of Computer Science and Communication Engineering Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580 JAPAN
‡ Institute of Systems & Information Technologies/KYUSHU
2-1-22-707 Momochihama, Sawara-ku, Fukuoka 814-0001 JAPAN
E-mail: †ppram@c.csce.kyushu-u.ac.jp, ‡kai@k-isit.or.jp

Abstract

In merged DRAM/logic LSIs, the DRAM portion could suffer from shorter data retention time because of heat and noise caused by the logic portion. Frequent refreshes increase power consumption. Also, they disturb normal DRAM accesses leading to performance degradation. In order to overcome this problem, we propose several DRAM refresh architectures. The basic idea is to eliminate unnecessary DRAM refreshes. We have estimated the DRAM refresh count in executing benchmark programs under several architecture models. As a result, in the most effective combination of the architectures, we have obtained more than 80% reduction against a conventional DRAM refresh architecture for most benchmark programs. In addition to it, even when we have taken normal DRAM access into account, we have obtained more than 50% reduction for several benchmarks.

1 Introduction

Merged DRAM/logic LSIs are expected to play an important role in the "system-on-silicon" era [6] [8].

However, information stored in each DRAM cell could be lost unless each bit is refreshed periodically. The period which each DRAM cell retains the information is heavily affected by the ambient temperature, because the characteristic of MOS transistors are getting worse as the ambient temperature rises. For example, the increase of the temperature from 25°C to 70°C would result in reducing the data retention time to 1/30 [2].

Moreover, the DRAM portion of such merged DRAM/logic LSIs could suffer from lower *data retention time* because of higher heat and noise [11] caused by the logic portion on the same chip. It is known that DRAM refreshes have some negative effects in the following manners [4].

- **Power Consumption:** As is generally known, DRAM refresh is major factor to increase the energy consumption of DRAM.
- **Performance:** Since a DRAM refresh is performed by reading a row, a normal memory access may conflict with the refresh and stall (called *refresh-penalty*). This problem will deteriorate as the processor becomes faster [3].

ISLPED98, Monterey, CA, USA

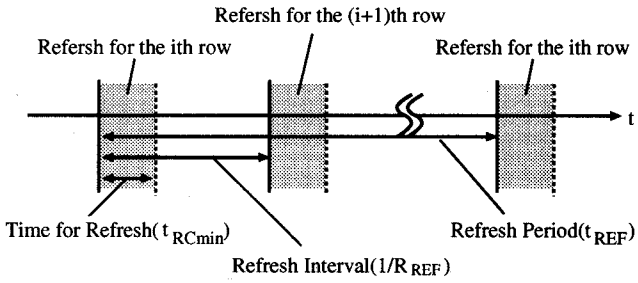

Figure 1: Definitions

It might be necessary to reconsider refresh architectures for DRAM of merged DRAM/logic LSIs. In this paper, we propose several DRAM refresh architectures tailored to them. The basic idea behind them is to eliminate unnecessary DRAM refreshes. In conventional DRAMs, DRAM refreshes are performed to every row of its cell array with a fixed time interval. On the other hand, our proposed architectures can select when and which row to refresh.

The rest of this paper is organized as follows: Section 2 discusses issues on DRAM refresh. Section 3 proposes DRAM refresh architectures and the method for reducing the number of DRAM refreshes. Section 4 presents some simulation models, simulation environments and the simulation results. Section 5 concludes this paper and discusses future work.

2 Backgrounds

2.1 Definitions

Data retention time (t_{RET}) is the time while each cell can keep the information without being refreshed. All the bits in a row of a DRAM matrix can be refreshed simultaneously just by reading that row. Hence every row of a DRAM matrix should be refreshed, or accessed, within a certain time window. The time window is called *refresh period* (t_{REF}), and the refresh period must be shorter than the data retention time of the worst cell of all. *Refresh rate* (R_{REF}) is a tuple of the number of the rows and the refresh period, such as "4096/64msec". *Refresh interval* is the inverse of the refresh rate.

Generally, if an access to a DRAM conflicts with a refresh, the access is not served while the refresh has been performed. The rate of such conflicts between refreshes and normal DRAM accesses is called *refresh-busy rate* (γ). It is defined as follows:

$$\gamma = R_{REF} \cdot t_{RCmin} \tag{1a}$$

$$= \frac{N_{row}}{t_{REF}} \cdot t_{RCmin}, \tag{1b}$$

where N_{row} is the number of rows and t_{RCmin} is the minimum cycle of DRAM access. t_{RCmin} is limited physically.

Here, the size of DRAM cell matrix (M) can be expressed as follows:

$$M = N_{row} \cdot N_{column}, \tag{2}$$

where N_{column} is the number of columns. Since N_{column} data-lines are driven simultaneously by one DRAM refresh, the amount of data-line capacitance (C_{DT}) can be expressed as follows:

$$C_{DT} = N_{column} \cdot C_D, \tag{3}$$

where C_D is the capacitance of a data-line.

Here was a serious problem whenever we would like to increase the memory size for conventional DRAM.If we increase N_{column}, both of C_{DT} and the number of sense amplifiers increase. It is not preferable, since it causes the increase of power consumption. On the other hand, if we increase N_{row}, both of γ and C_D increase. The former causes the increase of refresh-penalty and the latter also increases the power consumption. To solve these problems, both of N_{row} and N_{column} have been doubled equally, and t_{REF} have been extended so far. Furthermore, to reduce C_{DT} charged in one access, DRAM cell array has been divided into some sub-arrays.

However, t_{REF} has the following limitation:

$$t_{REF} \leq \min_{i=1}^{M} \{t_{DR}(i)\}, \tag{4}$$

where $t_{DR}(i)$ is the data retention time of the ith cell. In merged DRAM/Logic LSIs, it seems difficult to continue the conventional approach, since $t_{DR}(i)$ might be limited physically or shortened by both heat and noise.

In order to solve this problem, we propose decreasing the refresh rate , R_{REF}, at some architecture level and redefine R_{REF} specific for merged DRAM/logic LSIs as follows:

$$R_{REF} = \frac{n_{REF}}{t} \left(\neq \frac{N_{row}}{t_{REF}} \right), \tag{5}$$

where n_{REF} is the number of DRAM refreshes and t is a certain time period. Our strategy is to decrease n_{REF}.

2.2 Impact of DRAM Refresh on Energy Consumption

Energy(E) consumed by a DRAM is expressed as follows:

$$E \cong \left(\int \Delta Q_A dt + \int \Delta Q_P dt \right) V_{DD}, \tag{6}$$

where the following notation are used;

ΔQ_A: The amount of charge supplied to the memory array in unit time.

Table 1: Parameters

Clock frequency	100MHz
Refresh rate (R_{REF})	$4096/t_{REF}$
Cache miss rate (CMR)	5%
Memory references per instruction ($MRPI$)	1.4
Clock cycles per instruction (CPI)	0.5

ΔQ_P: The amount of charge supplied to peripheral circuits in unit time.

V_{DD}: The power supply voltage.

It is known that decreasing V_{DD}, $\int \Delta Q_A dt$ and $\int \Delta Q_P dt$ are essential for reducing the energy dissipation [2]. $\int \Delta Q_A dt$ can be expressed as follows:

$$\int \Delta Q_A dt \cong N_{column} \cdot C_D \int \Delta V_D dt, \tag{7}$$

where ΔV_D is the swing of the internal supply voltage.

Here, we assume the power consumption for a normal read/write operation and a refresh operation are the same. If every DRAM access is performed in t_{RCmin}, $\int \Delta Q_A dt$ can be expressed as follows:

$$\int \Delta Q_A dt \cong N_{column} \cdot C_D \cdot n_A \cdot \int_0^{t_{RC}} \Delta V_D dt, \tag{8}$$

where n_A is the total number of DRAM accesses and defined as follows:

$$n_A = n_{RW} + n_{REF}, \tag{9}$$

where n_{RW} and n_{REF} are the total number of read/write operations and that of refresh operations, respectively.

Figure 2 shows n_{RW}, n_{REF} and n_A when a merged DRAM/logic LSI with a processor and a cache has the characteristic shown in Table 1. From Figure 2, we can see that the refresh count increases drastically, if the refresh period (t_{REF}) is less than 20 msec and that decreasing the refresh count helps reduce the power consumption.

Figure 3 shows the relative increase of n_A, or n_{REF}/n_{RW}, varying both N_{row} from 4 Kbit to 32 Kbit and t_{REF}. We can see that the DRAM refresh count increases as the number of rows(N_{row}) increases and that shorter data retention time(t_{RET}) or refresh period(t_{REF}) affects the power consumption and the memory performance.

There are some proposals for reducing the power consumption in static, or standby, mode [12] [1] [5]. In the next section, we propose a couple of architectures for reducing the number of refreshes in both static and normal modes.

3 Architectures for Reducing DRAM Refresh

In this section, we propose some new architectures for reducing the number of DRAM refreshes.

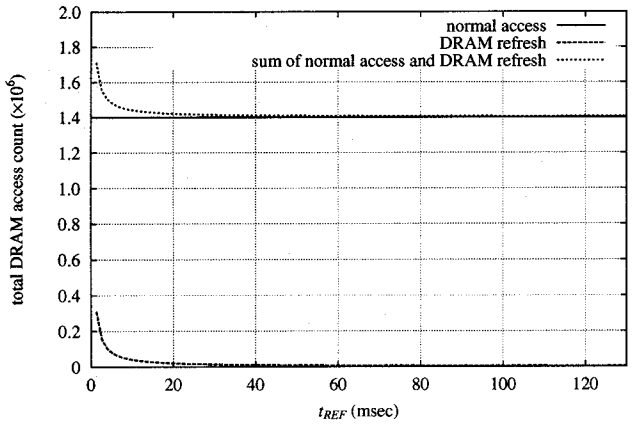

Figure 2: The number of normal access and DRAM refreshes

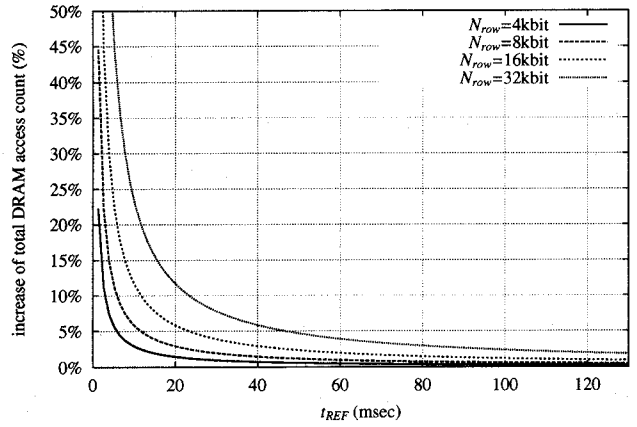

Figure 3: Increase of total DRAM access for various t_{REF} and N_{row}

3.1 Selective Refresh Architecture

Selective Refresh Architecture (SRA) allows DRAM refresh to perform selectively on regions(e.g., rows).

In general, each data (data in the wide sense, i.e., variables, program texts, and so on) in the memory has a period in which they are available. The period is referred to as a *lifetime*. The data must be retained while it is in its lifetime. But, the data which is not read hereafter need not be refreshed. We show an example of the implementation as follows.

The DRAM controller provides a *refresh flag* per row. Each row is refreshed when the corresponding refresh flag is 1 (see Fig. 4). The processor provides a "`store`" instruction that executes a normal store operation and sets the refresh flag to 1 at the same time. On the other hand, the processor provides two types of "`load`" instructions. One is the same as traditional "`load`" instruction. The other, called "**`load and reset RFLAG`**", executes a normal load operation and sets the refresh flag to 0 at the same time. The data can be prepared by a "`store`" instruction, and if it will not be required any longer, it can be abandoned by a "**`load and reset RFLAG`**" instruction.

Figure 5 shows the SRA architecture in detail. If the signal `SRE` (means *set refresh flag*) is 1, the refresh flag selected

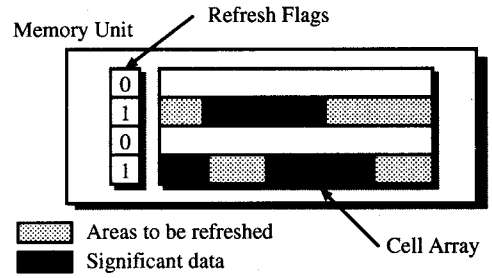

Figure 4: An implementation of SRA

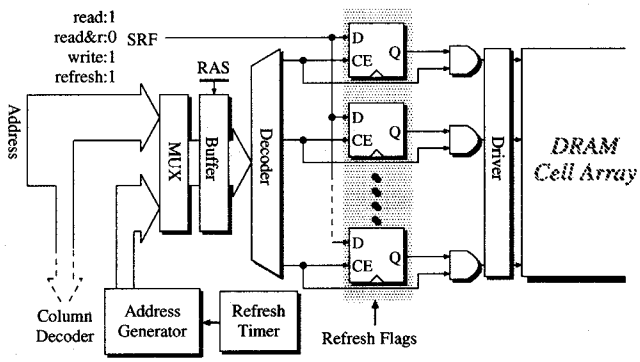

Figure 5: SRA circuit

by the row address becomes 1. Thus, that row is started refreshing.

In SRA, it is easy to know when and which row to start refreshing, because refresh operations to a row is started after the row has been accessed. On the other hand, it is difficult to determine when and which row to stop refreshing. We have the following two ideas to stop refreshing.

(1) Operating system or memory management unit: An operating system or a memory management unit stop refreshing. This is probably the most simple method, but brings some overheads.

(2) Static detection by a compiler: A compiler detects statically when and which data to abandon and embeds the instructions which stop refreshing. This method can control DRAM refresh more fine-grain than the above methods, however it has the following two difficulties.

- Difficulty to detect data no longer needed: Since most data is alive across procedures, a compiler which can perform inter-procedural lifetime analysis is required.
- Difficulty to detect rows no longer required refreshing: Since a row may contain different data, it is difficult to detect rows no longer needing refreshing. Suppose that a row R contains two data: one is in location α and the other is in locations β. Even if the data at location α can be abandoned, the row R still need to be refreshed, since the data at location β may be alive. Present compiler-techniques cannot analyze data location in memory so as to reduce the number of refreshes.

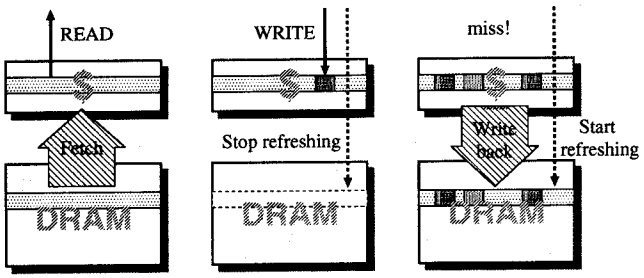

Figure 6: SRA for a write-back cache system

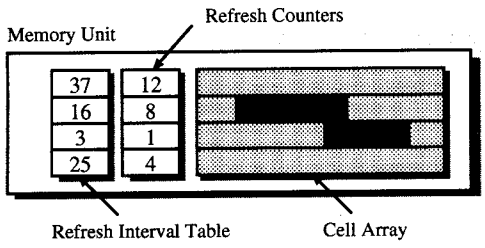

Figure 7: An implementation of VRA

3.2 SRA scheme for Cache Memory System

The SRA scheme ,which refreshes only used rows, can reduce the number of refreshes on DRAM of which the data is copied in external memory system. For example, the system which provides a write-back cache can stop refreshing to a row if all cache lines of the row would became dirty (see Figure 6). This is because each of these cache lines will be written back inevitably. This technique seems to be suitably for the systems which provides caches whose line size is the same as the row size of a DRAM cell array [6] [10].

3.3 Variable Refresh Period Architecture

Variable Refresh Period Architecture (VRA) has multiple refresh period and applies the most appropriate period to each region (e.g., per row).

The data retention time of each cell in DRAM is different [12]. Furthermore, *bad cells*, which have the very short data retention time, is fairly little. On the other hand, the real data retention time of most cells (called *normal cells*) is longer than that of bad cells. In conventional DRAMs, refresh periods of all rows are fixed to less than the worst data retention time. This architecture can make use of the real data retention time corresponding to the data retention time of the good rows. For example, the following implementation is possible.

The DRAM controller provides *refresh counter* and *refresh interval table* per row (see Fig. 7). When the refresh counter becomes 0, the corresponding row will be refreshed. At the same time, the refresh counter will return to the corresponding value of the refresh interval table. The DRAM refresh count can be reduced by setting the refresh interval table to an appropriate value.

However, this implementation lacks reality because (i) more than two refreshing operations for different rows may be required at a same time, (ii) this implementation may suffer from high hardware cost because it requires the plural refresh counters. In order to overcome the above problems, we have the following idea: Divide the refresh periods of each row into N time regions. The row in ith region is performed refreshes with a refresh period (T_i). Prepare the single refresh counter which can count the least common multiple of N periods. The row in ith region is refreshed when the refresh counter becomes multiples of T_i. In particular, it will keep the lower hardware cost by setting all refresh periods to the multiple values of the shortest period [3].

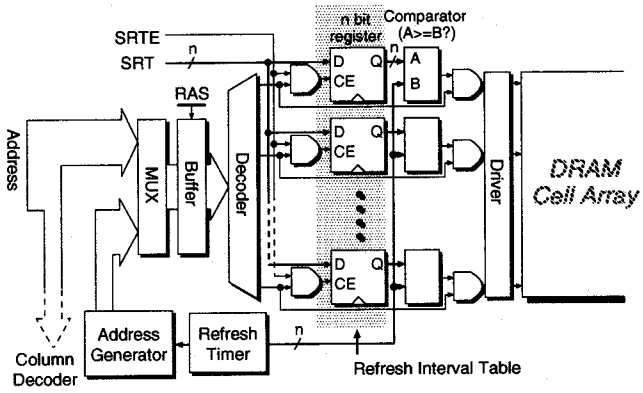

Figure 8: VRA circuit

Table 2: Hardware cost

	Area(mm^2)
16M-bit Conventional DRAM	103.7
16M-bit SRA DRAM	103.7 + 5.2
16M-bit VRA DRAM (2-level)	103.7 + 7.5
16M-bit VRA DRAM (4-level)	103.7 + 14.5
16M-bit VRA DRAM (8-level)	103.7 + 21.2

Figure 8 shows the VRA architecture in detail. If the signal **SRTE** (means *set refresh interval table*) is 1, the refresh interval table selected by the row address is set to the value of **SRT**. It is not necessary to change the refresh interval table continually, but occasionally or only startup time. Though the refresh interval table in this figure is designed as registers, the refresh interval table can be made of PROM or EPROM [9].

3.4 Hardware Cost

We estimate the area of supplemental hardware in figure 5 and figure 8. We design the supplemental hardware with 0.5μm/poly-1/metal-2 process and estimate their area. Then, we compare them to a 16M-bit DRAM designed with 0.5μ m/poly-3/metal-2 process [12]. Table 2 shows the results. Since their processes are different each other, the result is a rough comparison.

4 Experiments

We simulate proposed architectures, SRA and VRA. We execute some benchmark programs and estimate the DRAM refresh count under the some simulation models.

4.1 Experimental Environment and Assumptions

We execute several programs in *SPEC92* and *SPEC95* benchmark suites on *Sparc Sun Solaris2.5*. These programs are compiled by *GNU C Compiler*.

We assume that the system satisfies the following conditions.

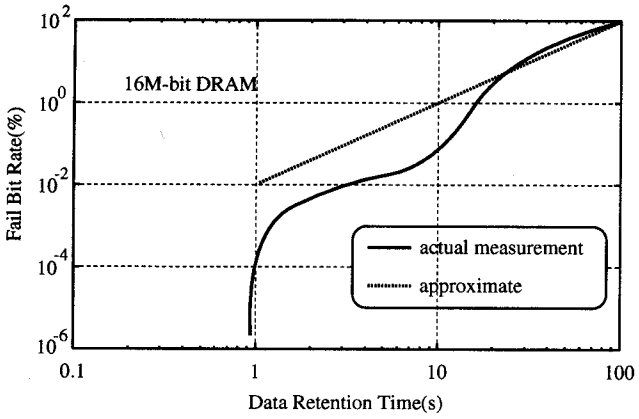

Figure 9: Data retention time of DRAM cells

- The DRAM has a capacity which is only required by the executed benchmark program.
- The program text is stored in the ROM.
- Each DRAM refresh is performed for just one row of the DRAM matrix.
- The number of cells in each row is 4096 and the refresh rate is 4096/64 msec (We assume it is reasonable.).
- The DRAM module consists of one bank.

4.2 Simulation models

Here, we introduce the following five simulation models and an original model as the standard of this simulation.

i. **Original:** This model performs the DRAM refreshes to all rows with a single period which treats the worst cell of all.

ii. **SRA(1):** This model simulates SRA. The lifetime of each data is from the time when the data is accessed, or written, for the first time through the time when the execute of the program finishes. Thus, this model simulates method (1) in the latter of Section3.1.

iii. **SRA(2):** This model is similar to the above model, SRA(1). However, the lifetime of each data is from the time when the data is accessed, or written, for the first time through the time when the data is accessed, or read, at last. Thus, this model simulates method (2) in the latter of Section3.1.

iv. **VRA:** This model simulates VRA. The refresh period is different among rows. In order to decide the refresh period of each row, we approximate the data retention time of every cell with according to the actual measurement value [12] (see Fig. 9). We assume the distribution of the data retention time is uniform and the refresh interval table has ideal value. Thus, each row is refreshed with the most appropriate refresh period.

v. **SRA(1)+VRA:** This model simulates the architecture which has the functions of both SRA and VRA. Consequently, this model is able to perform refreshing with the most appropriate periods. The lifetime of each data is the same as SRA(1).

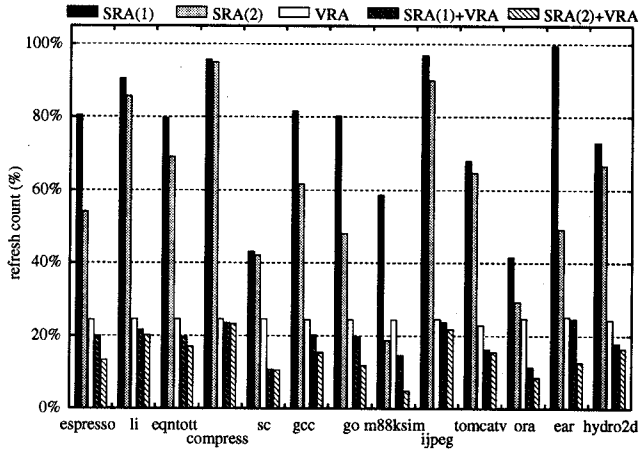

Figure 10: Reduction rate of the DRAM refresh count relative to the original model.

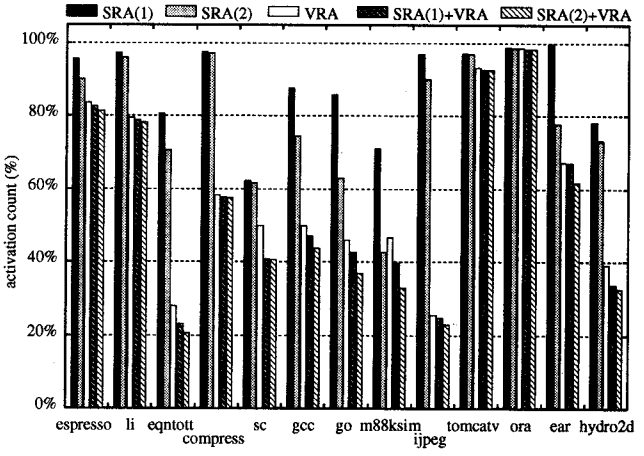

Figure 11: Reduction rate of the DRAM activation count relative to the original model.

vi. **SRA(2)+VRA:** This model similar to the above model, SRA(1)+VRA. However, the lifetime of each data is the same as SRA(2). Thus, this model should bring the ideal reduction in these proposed architectures.

4.3 Results

Figure 10 shows the reduction rate of the DRAM refresh count. First, both SRA(1) and SRA(2) reduce the number of DRAM refreshes by 50% to 10% of the original model. This means the effect of SRA depends on the behavior of applications. Moreover, SRA(2) reduces the number of DRAM refreshes by 60% to 5% of SRA(1). This is because SRA(2) takes careful account of lifetime analysis than SRA(1). For example, SRA(1) brings less effect for applications which allocate large data in the early time.

Secondly, the results of VRA report an average of about 75% reduction, because the approximation of data retention time reflects this result directly.

Thirdly, the results of SRA+VRA show about the values multiplied those of SRA and those of VRA together.

Moreover, we estimate the DRAM activation count,

which is the sum of DRAM refresh count and normal DRAM access count. In this estimation, we assume the following conditions in addition to the assumptions of Section4.1.

- The system has one level cache, and its miss rate is 5% (this number can be considered as a modest miss rate).
- The block size of the cache is below the row size, or 4096 bits (512 bytes).
- The clock frequency of the processor is 100MHz.

Figure 11 shows the reduction rate of the DRAM activation count. The reduction rate of DRAM activation count is more application-dependent than that of DRAM refresh count. It is due to the ratio of execution time to the normal access count.

Here, note that this simulation assumes the DRAM is only in the *normal mode*, in which normal DRAM accesses are occurred continuously. In real systems, however, the DRAM is frequently put into *static mode*, in which it is only keeping memory by performing refreshes, as a consequence of I/O, interruptions and so on. If it is taken into account that the DRAM is sometimes put in the standby mode, the proportion of the DRAM refresh count to the DRAM activation count will increase. Hence the reduction rate of the DRAM activation count will certainly overcome these results.

5 Conclusions

In this paper, we have discussed the issues on DRAM refreshes in merged DRAM/logic LSIs. In order to overcome this problem, we have proposed a couple of DRAM refresh architectures, SRA and VRA. SRA can reduce the number of DRAM refreshes by selecting the row to refresh. VRA can reduce it by varying the refresh period per region.

We have estimated the DRAM refresh count by executing some benchmark programs under several architecture models. As a result, in the most effective combination of the architectures, or SRA(2)+VRA, we have obtained more than 80% reduction against a conventional DRAM refresh architecture for most benchmark programs. Furthermore, even when we have taken normal DRAM access into account, we have obtained more than 50% reduction for several benchmarks.

By the way, we have another approach which reduces the number of DRAM refreshes. Each data whose lifetime overlaps each other is allocated to a common row [7].

For future work, we will establish the method to realize proposed architectures, how to determine values of the refresh interval table, how to analyze the lifetime and so on. Moreover, we will estimate the hardware cost of each implementation.

Acknowledgments

They would like to thank Prof. Hiroto Yasuura, Prof. Mizuho Iwaihara and the members of Yasuura Laboratory for fruitful discussions.

References

[1] Y. Idei, K. Shimohigashi, M. Aoki, H. Noda, H. Iwai, K. Sato, and T. Tachibana. "Dual-Period Self-Refresh Scheme for Low-Power DRAM's with On-Chip PROM Mode Register". *IEEE Journal of Solid-State Circuits*, 33(1):253–259, Feb 1998.

[2] K. Itoh. *"VLSI Memory Design (in Japanese)"*. Baifukan, 1994.

[3] K. Kai, A.Inoue, T. Ohsawa, and K. Murakami. "Analyzing and Reducing the Impact of Shorter Data Retention Time on the Performance of Merged DRAM/Logic LSIs". *To appear in IEICE Transactions on Electronics*, E81-C(9), September 1998.

[4] K. Kai, T. Ohsawa, and K. Murakami. "A DRAM Refresh Architecture for Merged DRAM/Logic LSIs". Technical Report ICD97–77, IEICE, July 1997.

[5] Y. Miyamoto, M. Ihara, T. Mimoto, and K. Sano. "Study of new refresh method for low data retention current (In Japanese)". *Proc. of the 1993 IEICE General Conf*, C–638, 1993.

[6] K. Murakami, S. Shirakawa, and H. Miyajima. "Parallel Processing RAM Chip with 256Mb DRAM and Quad Processors". In *1997 ISSCC Digest of Technical Papers*, pages 228–229, February 1997.

[7] T. Ohsawa, K. Kai, and K. Murakami. "Evaluating DRAM Refresh Architectures for Merged DRAM/Logic LSIs". *To appear in IEICE Transactions on Electronics*, E81-C(9), September 1998.

[8] D. Patterson et al. "Intelligent RAM (IRAM): Chips that Remember and Compute". In *1997 ISSCC Digest of Technical Papers*, pages 224–225, February 1997.

[9] K. Reza and E. Boaz. "A Single Poly EPROM for Custom CMOS Logic Applications". In *Proc. of the Custom Integrated Circuits Conference*, pages 59–62, 1986.

[10] A. Sausbury, F. Pong, and A. Nowatzyk. "Missing the Memory Wall: The Case for Processor/Memory Integration". In *Proc. ISCA '96*, pages 90–101, May 1996.

[11] T. Tsuruda, M. Kobayashi, M. Tsukude, T. Yamagata, and K. Arimoto. "High-speed / High-band Width Design Methodologies for on chip DRAM Core Multimedia System LSIs". In *Proc. of the Custom Integrated Circuits Conference*, pages 265–268, 1996.

[12] H. Yamauchi, T. Iwata, A. Uno, M. Fukumoto, and T. Fujita. "A Circuit Technology for a Self-Refresh 16Mb DRAM with Less than 0.5μA/MB Data Retension DRAM". *IEEE Journal of Solid-State Circuits*, 30(11):1174–1182, November 1995.