High-Performance and Low-Power Memory-Interface Architecture for Video Processing Applications

Hansoo Kim and In-Cheol Park

***Abstract*—To improve memory bandwidth and power consumption in video applications, a new memory-interface architecture is proposed. The architecture adopts an array address-translation technique to utilize the fact that video-processing algorithms have regular memory-access patterns. Since the translation can minimize the number of overhead cycles needed for row-activations in synchronous DRAM (SDRAM), we can improve memory bandwidth and energy consumption significantly. The features of SDRAM and memory-access patterns of video-processing applications are considered to find a suitable address translation. Compared to the conventional linear translation, experimental results show that the proposed architecture reduces about 89% of row-activations and increases the memory bandwidth by 50%. In addition, the proposed architecture reduces the energy consumption by 30% on the average.**

***Index Terms*—Advanced memory interface, array address translation, low-power, MPEG, regular memory-access pattern, synchronous DRAM, video-processing application.**

I. Introduction

As THE resolution of video-processing applications becomes high, video signal processors should deal with a large amount of data within a tightly bounded time. Due to the large quantities, video data are stored in off-chip memories that are usually slow, and thus the system performance strongly depends on the memory bandwidth between processors and external memories. For example, in decoding MP@HL video streams, an MPEG-2 video decoder must feature 10-MB memory as frame buffers and should access about 370 million data/s. To meet the requirement, we must exploit the characteristics of video-processing algorithms that are usually programmed using multi-dimensional arrays and nested loops. Once an algorithm description is completed, the characteristics of arrays such as dimension, size, loop bound, and stride are determined. From the deterministic parameters, most memory-access patterns can be known at compile time [1]. The regularity of memory-access patterns can be effectively used to reduce the number of clock cycles required in array accesses.

In order to improve memory bandwidth, newer DRAM families such as synchronous DRAM (SDRAM) and Rambus DRAM (RDRAM) are now widely used in high-performance video systems as they are equipped with high-performance features. For example, SDRAM has two key features: burst access mode and multiple bank architecture [2]. The burst access mode makes it possible to access a number of data by changing only column addresses, and the multiple bank architecture can hide memory cycles needed for row-activations and precharges by accessing different banks alternatively. Since the number of additional cycles needed for row changes is considerable, we have to reduce the number of row changes or hide these cycles by using the high-performance features of SDRAM. This observation motivated us to research an array address-translation technique to minimize the number of overhead cycles needed for row changes.

Besides the high memory bandwidth, low-power consumption becomes an important factor to be considered in system design. As the power related to memory accesses dominates the whole system power in data-dominated systems, it is essential to reduce the memory power consumption [3]. Among many memory operations, row-activation and column data access are dominant in memory power consumption. Since the switching of the address bus between processors and memories is also a major source of power consumption, we can save power significantly by using the address translation that minimizes both the number of row-activations and the number of commands on the address bus.

In this paper, we propose a memory-interface architecture to improve memory bandwidth and power consumption in video-processing applications, extending our previous work that presented an array address-translation technique [4]. The rest of this paper is organized as follows. Section II briefly explains previous researches on the improvement of memory bandwidth and on the power reduction. Section III describes a simplified architecture and features of SDRAM and Section IV explains the problem we deal with. We propose the array address-translation technique and the memory-interface architecture in Sections V and VI, respectively. Section VII describes the effect of the proposed architecture on energy consumption. Finally, experimental results are shown in Section VIII.

II. Related Works

Memory hierarchy has been actively researched to increase the performance of computer systems [5], [6]. In the hierarchy, as a lower level memory provides a faster access time at the cost of limited amount of entries, memory bandwidth can be increased by reducing the number of accesses to slow higher level memories. Using the locality of a given application, an analytical strategy has been presented for exploring memory configurations such as cache size and line size [6]. Moreover, several memory data-layout techniques have been presented in [7], [8]

Manuscript received April 24, 2000; revised September 6, 2001. This paper was recommended by Associate Editor L.-G. Chen.

H. Kim is with the Digital Media Research Laboratory, LG Electronics, Seoul 137-724, Korea (e-mail: akihiver@lge.com).

I.-C. Park is with the Division of Electrical Engineering, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Taejon 305-701, Korea (e-mail: icpark@ee.kaist.ac.kr).

Publisher Item Identifier S 1051-8215(01)10159-X.

to assign scalar or array variables to physical addresses so as to reduce conflict cache misses by using deterministic memory-access patterns in applications. However, as the memory-access patterns of video applications have little degree of locality, the conventional cache memory systems are not appropriate for the video-processing applications. For example, in the motion-compensation routine of MPEG-2 video decoding, reference macroblocks cannot be determined until the corresponding motion vectors are decoded. This uncertainty makes it difficult to reduce the number of page replacements in the previous techniques. Instead of cache memory systems, on-chip local buffer schemes are used in the video-processing applications in order to exploit burst access modes and synchronize memory-accesses with data operations.

To utilize the fast interface schemes of newer DRAM families, several pre-synthesis optimizations have been proposed, such as variable clustering, access reordering, bank assignment and loop transformations [9]. Though these techniques are very effective in increasing memory bandwidth, a lot of row-activations may happen if logical arrays are linearly mapped to physical addresses, and thus may make the system performance worse in video-processing applications. Although a tile-based memory organization was presented for MPEG-2 video decoding to avoid unnecessary page-breaks in SDRAM [10], [11], it is limited to a specific application and memory type, as a systematic approach to find an optimal organization is not presented.

In [1], it is shown that the memory hierarchy can be optimized in terms of power consumption, as well as system performance, by doing a tradeoff between cache miss rate and cache size. The authors in [12] proposed a low-power memory mapping technique to reduce address bus activity. Considering the regularity and locality of behavioral array accesses in application specifications, the technique reduces the number of transitions on the memory address bus to save the power consumption of off-chip drivers and memory decoding circuitry. However, it did not consider the number of row-activations which is a crucial source of memory power.

This paper presents a memory-interface architecture to increase the memory bandwidth and to reduce power dissipation in SDRAM. The architecture adopts an array address-translation technique that utilizes the regular memory-access patterns, as well as advanced features of SDRAM in order to reduce the number of row changes in SDRAM. Since the address translation does not affect data locality, it can be associated with the memory hierarchy system. To reduce conflict misses in the cache memory, the memory data-layout technique assigns an appropriate physical space to each array by considering interleaving accesses of arrays [7]. An array is partitioned into a set of sub-arrays to further reduce the conflict cache misses [8]. Although the memory data-layout techniques determine the physical address spaces of arrays, they assume an array is linearly mapped to the corresponding physical space. As the proposed architecture deals with how to translate the physical address of a block within an array in order to minimize the number of row changes required to access the block data, the proposed translation can be integrated with the memory data layout techniques. By applying both techniques simultaneously, we can reduce the number of cache line replacements as well as the time to fill cache lines. In addition, the pre-synthesis optimizations such as variable clustering, access reordering and loop transformations can be applied complementarily for scalar variables that have irregular access patterns usually.

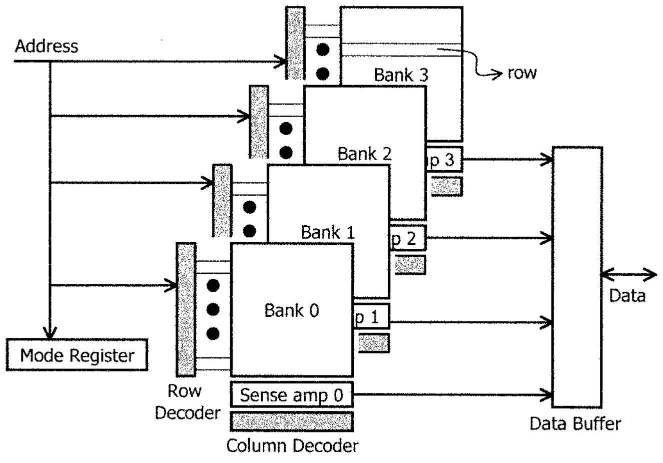

Fig. 1. Architecture of typical SDRAM.

III. Features of SDRAM

In this section, we explain the basic architecture, access operations and an energy consumption model of SDRAM. A simplified architecture of typical SDRAM is shown in Fig. 1. Four banks share the address and data buses, while each bank has its own row decoder, column decoder, and sense amplifier. A Mode Register indicates one of several memory operation modes, including Burst-Length, CAS Latency, or Interleaving. The register is updated according to the command issued on the address bus. A memory-access operation consists of three steps. First, a row-activation command is activated to copy the row data of a designated bank into the sense amplifier. Second, a burst access command is issued to fetch data as many as Burst-Length from the sense amplifier and send them to the data bus. Changing only the column address, we can access other column data on the same row without invoking any additional row-activation. Finally, the sense amplifier is precharged for different row data. Since these steps are pipelined with an external clock, SDRAM can operate at high clock frequency and commands to different banks can be overlapped, e.g., a row-activation command can be overlapped with burst access operations of another bank to reduce the number of cycles for the row-activation. As a result, the memory cycles depend on the physical locations where the successively accessed data are stored.

Let us consider, for example, two burst accesses of which burst length is 4. In Fig. 2(a), RAS0 and CAS0 are commands for accessing data stored in Bank0, while RAS1 and CAS1 are for Bank1. As soon as the preceding burst data are fetched, the data of Bank1 can be accessed without any additional cycles, since Bank1 can be activated during the burst access of Bank0. However, if both burst accesses read two different rows of the same bank, some overhead cycles are inevitable. The second burst data can be fetched, only after the precharge of the sense amplifier, a row-activation and a column access operation are serially finished, as shown in Fig. 2(b). If we assume that the row-activation delay, CAS latency and precharge delay are all

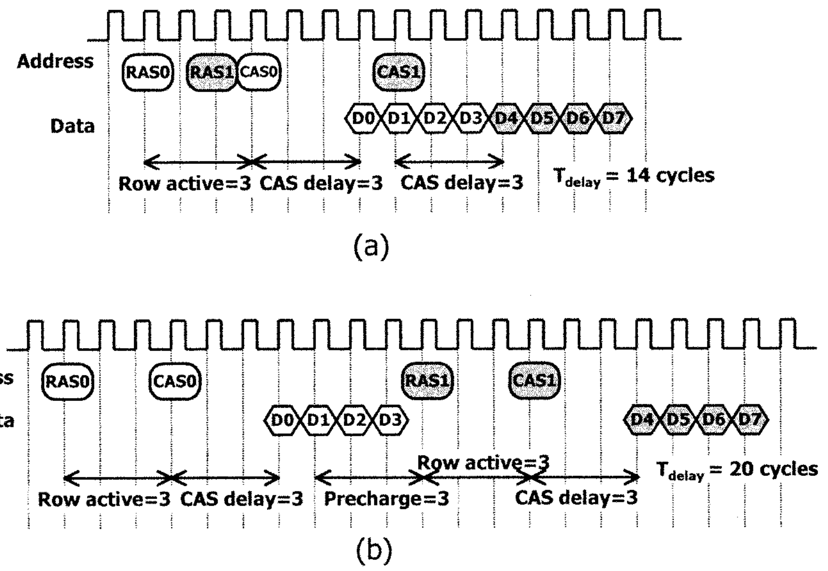

Fig. 2. Timing diagram of two consecutive read cycles in SDRAM. (a) Accessing data stored in two different banks takes 14 cycles. (b) Accessing data in two different rows of the same bank takes 20 cycles.

three cycles, accessing eight data takes six more cycles. We call these additional cycles overhead cycles.

Power consumption related to memory accesses can be divided into two components—core power and I/O power—as follows:

$$P_{\text{total}} = P_{\text{core}} + P_{\text{I/O}} = I_{\text{core}}V_{\text{dd}} + \alpha C V_{\text{dd}}^2 f \qquad (1)$$

where I_{core}, α, C, and f are the average current of SDRAM to be used, average switching activity in I/O buses, I/O capacitance, and operating frequency, respectively.

The power consumption of the SDRAM core, the first term of (1), is calculated by using dc currents in a memory specification [13], which are measured under various test conditions, e.g., normal access (I_{CC1}), active status (I_{CC2}), burst access mode (I_{CC3}), and refresh (I_{CC4}). Thus,the dc current can be estimated by averaging these dc current parameters weighted by the execution time of each operation. The execution time depends on the numbers of row-activations, refreshes and data transfers. Therefore, the dc current of SDRAM can be calculated by (2), where $t_{RAS(\text{MIN})}$, $t_{RP(\text{MIN})}$, and $t_{CK(\text{MIN})}$ denote minimum memory timing parameters for row-activation, precharge and clock period, respectively [14], and CN, DN, and RN denote the numbers of row-activations, data transfers, and refresh commands within a total execution time T, respectively. While the first and the second terms denote the induced currents during data accesses with a row-activation and without any row-activation, respectively, the third and fourth terms denote the induced current of refresh operation and that of active status, respectively [14]

$$I = \frac{1}{T}\begin{bmatrix} I_{CC1} \times CN \times \left(t_{RAS(\text{MIN})} + t_{RP(\text{MIN})}\right) + I_{CC3} \\ \times (DN - CN) \times t_{CK(\text{MIN})} + I_{CC4} \\ \times RN \times \left(t_{RAS(\text{MIN})} + t_{RP(\text{MIN})}\right) + I_{CC2} \\ \times \left(t_{RAS} - (CN + RN) \times t_{RAS(\text{MIN})}\right. \\ \left. - (DN - CN) \times t_{CK(\text{MIN})}\right) \times \dfrac{t_{CK(\text{MIN})}}{t_{CK}} \end{bmatrix}. \qquad (2)$$

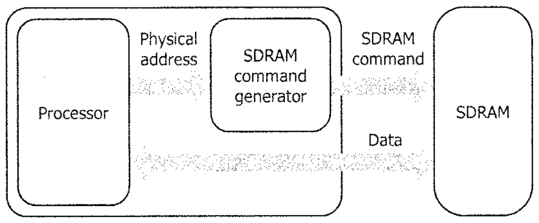

Fig. 3. Block diagram of a conventional memory-interface architecture.

To calculate the column data access time, the term (DN−CN) is used instead of DN, since I_{CC1} has been measured under a test condition that a row-activation and a data access are repeatedly issued. In that case, I_{CC1} already includes the induced current for a data access. Similarly, the last term of (2) is to compensate the active standby current that is already included in I_{CC1}, I_{CC3}, and I_{CC4}.

As shown in (1), the I/O power consumption of SDRAM is proportional to bus switching activity, once the operating frequency and I/O capacitance are fixed. In the command-based memory such as SDRAM and RDRAM, the switching activity is closely proportional to the total number of commands. Hence, reducing the number of commands is important to save power consumption.

IV. Problem Definition

Fig. 3 shows a conventional memory-interface architecture in which a memory controller takes physical addresses as its input and generates proper SDRAM commands. Due to the separate SDRAM command generator, memory operations and data processing operations can be executed in parallel. However, if a sequence of physical addresses requires many row-activations, the overhead cycles become considerable, degrading the system performance.

In the video-processing applications, video data are usually represented by multi-dimensional arrays that are processed in nested loops. A loop statement is expressed with the initial

```
    Simple_MPEG_Video_Decoding_Routine()
    {
       /* A : current frame    */
       /* B : previous frame */
       /* C : IDCT result       */
1:     for (J = 0; J < VSIZE; J += 16 ) {
2:        for ( I = 0; I < HSIZE; I += 16 ) {
3:           Motion_Vector_Decode(&MVY, &MVX);
4:           C = IDCT( );
5:           for (Y = 0; Y < 16; Y ++ ) {
6:              for (X = 0; X < 16; X ++ ) {
7:                 A[J+Y][I+X] = B[MVY+J+Y][MVX+I+X] + C[Y][X];
8:              } /* end of X-for */
9:           } /* end of Y-for */
10:       } /* end of I-for */
11:    } /* end of J-for*/
12: } /* end of routine */
```

Fig. 4. A fragment of the MPEG-2 video-decoding algorithm.

value, bound and stride of a loop index variable. The loop index variable changes from the initial value to the bound representing the loop limit. The stride denotes an incremental step for the loop index variable. The bound and stride of the loop as well as the dimension and size of the array are statically determined. The number of data accessed sequentially can be determined by the loop bound of the innermost loop. As this number is usually fixed in video-processing applications, a set of memory accesses of the innermost loop is called a "block-type access." The indices of the outer loops determine the start address of a block-type access, but do not change the number of data accessed sequentially.

Fig. 4 shows a fragment of a simplified MPEG-2 video decoding algorithm. There are two nested iteration loop pairs in the routine. In the outer loop pair (lines 1–11), the routine performs macroblock processing such as motion vector decoding, inverse discrete cosine transform (IDCT) decoding, and motion compensation (MC). The inner loop pair (lines 5– 9) generates motion compensated 16 $\times$ 16 pixels to be stored into the frame memory. There are many memory operations in the inner loop pair, and the size of a block-type access is fixed to 16 $\times$ 16, which is the size of a macroblock. The motion vector (MVY, MVX) is determined at run time.

If we choose a proper address translation such that the block-type data accessed in the innermost loop are stored in a row or in different banks, the block-type access does not induce any overhead cycles for row-activations. The problem to be solved is to find such an address-translation scheme that converts logical addresses (array indices) into physical addresses. We will show several conditions for optimal address translation and present an efficient searching algorithm in Section V.

Let us explain the motivation of address translation by using a simple example. Consider an algorithm that processes a 2-D array of 8 $\times$ 8 data, as shown in Fig. 5. If logical addresses are linearly mapped to physical addresses, accessing a block of 8 $\times$ 8 data causes seven row changes, as shown in Fig. 5(a). However, if logical addresses are translated as shown in Fig. 5(b), accessing the block does not induce any row change. To minimize the number of row changes that degrade system performance, we propose a translation of logical addresses of a multi-dimensional array into physical addresses. For the address translation, a logical array is partitioned into a set of rectangles called windows and each window is stored in a row of SDRAM. The windows of Fig. 5(a) and (b) consist of 1 line $\times$ 128 data and 8 lines $\times$ 16 data, respectively.

V. Array Address-Translation Technique

The most important problem in array address translation is to find a suitable window size that minimizes the number of memory cycles. The memory cycles consist of pure column data accesses and overhead cycles for row changes. The overhead cycles are categorized into two groups, intra-overhead cycle and inter-overhead cycle. The intra-overhead cycle denotes the row-change overhead that cannot be hidden during a block-type access. We assume that the first desired row of every bank visited in the block-type access is already activated. On the other hand, the inter-overhead cycle denotes the row-change overhead cycles between block-type accesses incurred to activate the first desired row of every bank that is visited in the latter block-type access.

To minimize the overhead cycles, it is necessarily required that any block-type access should take zero or minimal intra-overhead cycles. In this section, we explain several conditions that guarantee the zero or minimal intra-overhead cycles. The conditions depend on the characteristics of inner loops (bounds and strides of loops) and SDRAM parameters, such as the size of a row and the overhead cycles of a row change. Since the number of banks in SDRAM diversifies the conditions, we describe the conditions for 2-D arrays assuming 2-bank, 4-bank, and 2^m-bank ($m > 2$) memories. We focus on 2-D arrays, since the video data are usually represented with 2-D arrays. Even if video-processing algorithms can use multi-dimensional arrays to include a time dimension, they usually access the 2-D video data of a time frame in the innermost loop of the algorithms. Thus considering 2-D arrays is usually sufficient to improve the memory bandwidth of the video-processing algorithms.

Besides the regular block-type accesses, the variation of outer loop also plays an important role in determining memory-access patterns at run time. As the number of inter-overhead cycles depends on the dynamic memory-access patterns, we use a simple memory-access simulation to take into account the dynamic memory-access patterns. We also explain in this section an efficient algorithm that searches for an optimal window.

A. Conditions for Minimal Overhead Cycles

1) Number of Banks = 2:

Condition 1: *All the data of a block-type access are contained in a window.*

It is apparent that the first condition induces no row change, since all the data to be accessed are stored in a row. The window in Fig. 5(b) satisfies this condition and thus the access of 8 $\times$ 8 data needs no row change.

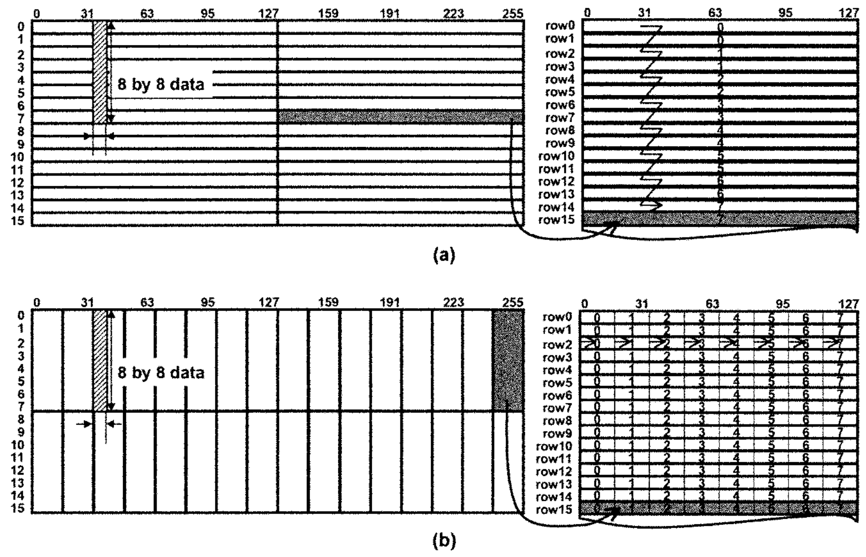

Fig. 5. Row changes for two physical memory maps. (a) Logical array with (1, 128)-windows and its physical memory map. (b) Logical array with (8, 16)-windows and its physical memory map.

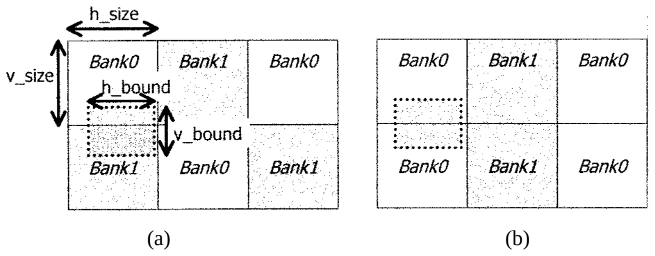

Fig. 6. Two possible mapping cases of a block data when the block data are stored in vertically contiguous windows and the number of banks is two. (a) Mapping A. (b) Mapping B.

Condition 2: *If all the data of a block-type access are contained in windows that are contiguous horizontally, the horizontal size of windows must be greater than the loop bound and the horizontally contiguous windows must be stored in different banks.*

When block data are stored in horizontally contiguous windows, the block-type access must visit different windows. The condition that the horizontal size of windows is greater than the loop bound guarantees that the number of windows to be visited horizontally in any block-type access is at most 2. As the desired rows of two different banks visited in the block-type access are already activated, the number of intra-overhead cycles is zero.

When a block-type access visits vertically contiguous windows, it is impossible to avoid the intra-overhead cycles completely in 2-bank memory. Therefore, it is important to select an appropriate mapping between two possible cases shown in Fig. 6, where h_size (v_size) and h_bound (v_bound) denote the horizontal (vertical) size of windows and the horizontal (vertical) loop bounds, respectively. A block-type access contained only in vertically contiguous windows does not induce intra-overhead cycles in mapping A, while it induces intra-overhead cycles in mapping B. Hence, the probability that a block-type access induces intra-overhead cycles in each mapping can be calculated as equations (3) and (4), where p(vertical break) means the probability that a block-type access visits two vertically contiguous windows. The equations show that if the horizontal size of windows is larger than two times of the horizontal loop bound, the mapping A is superior to the mapping B. For accurate calculation, the number of overhead cycles should be considered

$$P(\text{overhead occurrence in mapping A}) = \left(\frac{h\_bound - 1}{h\_size}\right) \times p(\text{vertical break}) \qquad (3)$$

$$P(\text{overhead occurrence in mapping B}) = \left(1 - \frac{h\_bound - 1}{h\_size}\right) \times p(\text{vertical break}). \qquad (4)$$

2) Number of Banks = 4:

Condition 1, 2: These conditions are the same as the cases of 2 banks.

Condition 3: *If all the data of a block-type access are contained in windows that are contiguous horizontally and vertically, four windows that are contiguous horizontally and vertically should be stored in different banks, the horizontal size of the windows must be greater than the innermost loop bound, and the product of the vertical size of windows and the innermost loop bound is greater than the minimum number of data needed for hiding the row-change overhead cycles.*

Since all the neighboring windows are stored in different banks, any row-activations of vertically contiguous windows can be overlapped with the burst-accesses of other windows. Since the product of the vertical size of windows and the innermost loop bound is greater than the overhead cycles of a row change, the overhead cycles can be hidden by column accesses of other banks. Therefore, a block-type access does not need any intra-overhead cycles unlike the 2-bank case, if it meets at least one of the above conditions.

Fig. 7 shows block-type accesses of two arrays. If we assume that each window is translated such that it meets one of the above conditions, no intra-overhead cycle is induced within a block-type access. However, the last data of Block1 and the first data of Block2 are stored in two different rows of Bank3, and thus the block-type access of Block2 induces inter-overhead

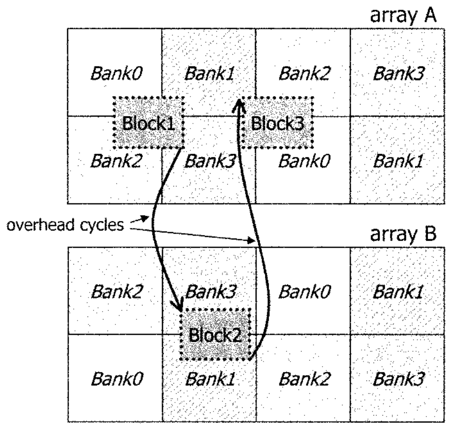

Fig. 7. Block-type accesses of two arrays that are stored in the same set of banks. In this case, overhead cycles are required in changing block-type accesses.

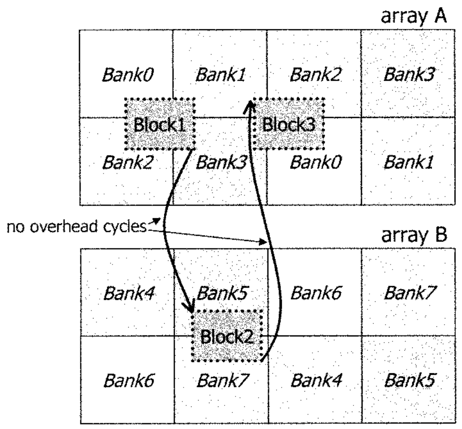

Fig. 8. Block-type accesses of two arrays that are stored in different sets of banks. In this case, no overhead cycle is required in changing block-type accesses.

cycles. We can minimize these inter-overhead cycles by array partitioning and memory-access simulation, which will be explained in the next subsection.

3) Number of Banks $= 2^m$ $(m > 2)$*:* As mentioned in the 4-bank case, four is the minimum number of banks required to access block data without any intra-overhead cycle. However, if multiple arrays are accessed in an interleaving manner, the interference may lead to inter-overhead cycles. To avoid the interference effects, we use an array-partitioning method [9]. Our method differs in that arrays are partitioned into $N/4$ groups, not N groups, where N denotes the number of banks in SDRAM, and four banks are exclusively assigned to each group. Arrays are partitioned such that the total interference among arrays in a group is minimized. In such a partitioning, most interferences occur between different groups that are stored in different bank sets, and thus the inter-overhead cycles of the interleaving array accesses can be minimized as shown in Fig. 8.

B. Searching Algorithm for an Optimal Window

Besides the regular block-type access, the variation of outer loop indices also plays an important role in determining memory-access patterns. The indices of outer loops determine only the start address of a block-type access, and do not change

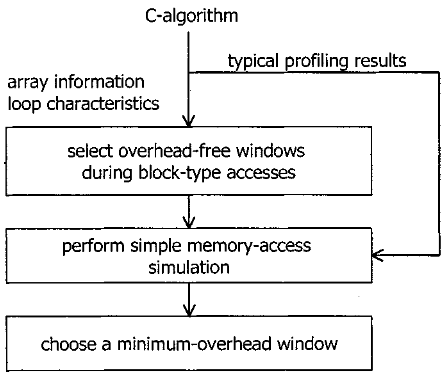
C-algorithm
typical profiling results
array information
loop characteristics
select overhead-free windows during block-type accesses
perform simple memory-access simulation
choose a minimum-overhead window

Fig. 9. Simplified flow of the proposed algorithm searching for the best window size.

the number of data accessed sequentially. Since the start addresses of block-type accesses are usually varying at run time in video-processing applications, memory-access simulations for all the candidate windows are needed to determine an optimal translation. The simulation estimates overhead cycles by calculating physical locations of memory operations of typical input vectors. To choose a robust address translation, we should determine the window size by performing the simulation repeatedly for various typical input vectors. Since the repetitive simulations are very time-consuming, we present an efficient algorithm, as outlined in Fig. 9.

The algorithm takes the C-description of an application as its input and generates an optimal translation of the given application considering the memory-access patterns and parameters of SDRAM, such as the number of banks and the size of a row. First, based on static memory-access patterns such as loop characteristics and array information contained in the C description, the algorithm evaluates the conditions for minimal intra-overhead cycle windows and selects candidate windows that meet the conditions. Second, repetitive simple memory-access simulations are performed for the candidate windows selected in the first step to consider dynamic memory-access patterns. By taking into consideration the dynamic access patterns, the simulation can consider the interleaving accesses of multiple arrays. The simple simulation calculates only the variation of start locations of block-type accesses, not the whole memory accesses, in order to reduce CPU time. As block-type accesses of the innermost loops do not induce any intra-overhead cycles in 2^m-bank $(m \geq 2)$ memories, the simple simulation that estimates the number of inter-overhead cycles is sufficient to calculate overhead cycles accurately. If the number of banks is less than 4, intra-overhead cycles are also considered, which are computed from the number of vertical windows visited during a block-type access. Lastly, the algorithm chooses the best window associated with the minimum overhead cycles.

C. Code Transformation

To exploit the burst access mode, a simple loop transform should be performed in the algorithm description. The transform separates memory-access operations from data processing operations, as shown in Fig. 10. After this transform, there are only data processing operations that fetch data from the local buffer in the innermost loop (lines 6–10), and a 16×16 block data is

```
      Simple_MPEG_Video_Decoding_Routine()
      {
        /* A : current frame   */
        /* B : previous frame */
        /* C : IDCT result     */
        /* LA, LB : local buffers */
1:      for (J = 0; J < VSIZE; J += 16 ) {
2:        for ( I = 0; I < HSIZE; I += 16 ) {
3:          Motion_Vector_Decode (&MVY, &MVX);
4:          C = IDCT ( );
5:          Memory_Block_Read (B, LB, MVY+J, MVX+I);
6:          for (Y = 0; Y < 16; Y ++ ) {
7:            for (X = 0; X < 16; X ++ ) {
8:              LA[J+Y][I+X] = LB[Y][X] + C[Y][X];
9:            } /* end of X-for */
10:         } /* end of Y-for */
11:         Memory_Block_Write (A, LA, J, I);
12:       } /* end of I-for */
13:     } /* end of J-for*/
```

Fig. 10. Transformed algorithm of the Fig. 4.

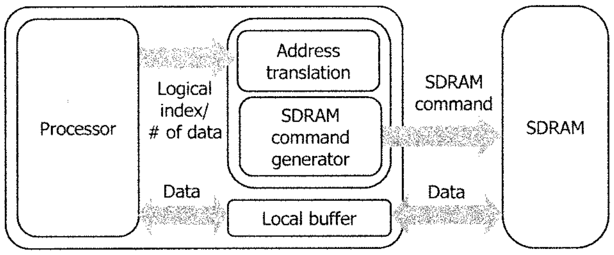

Fig. 11. Block diagram of the proposed memory-interface architecture.

transferred between the external memory and the local buffer by a sequence of burst access modes (lines 5 and 11).

VI. Proposed Memory Interface Architecture

To apply the proposed array address-translation technique, the memory-interface architecture should include an address-translation logic as shown in Fig. 11. The address-translation logic is placed in between the processor and the SDRAM command generator, and an on-chip local buffer is included to synchronize the burst data with data processing operations.

In this architecture, a memory operation is executed as follows. The processor requests a block-type memory access to the SDRAM controller. Then, the controller generates SDRAM commands after the array address translation. If the data of a block-type access are dispersed to several windows, many row-activations are required, but they can be hidden by other burst accesses. Although the address-translation logic is additionally required compared to the conventional architecture, it can reduce the overhead cycles significantly.

As shown in Fig. 12, the modified SDRAM controller contains the address-translation logic consisting of two blocks: First-Line block and Next-Line block. First-Line block calculates the physical location of the first data of each block-type access, such as bank, row,and column addresses. It also calculates the corresponding window, ($Iint$, $Jint$), and the position within the window where the first data is stored, ($Imod$, $Jmod$). Then, Next-Line block computes the physical addresses of the first data of every line in the block-type access. If the data of a block-type access are stored in two horizontally contiguous windows, Next-Line block must generate the physical address of the first data of every line in the latter window. Next-Line block also conveys timing information on when the commands should be issued to the SDRAM command generator.

Though the on-chip local buffer is assumed in this paper, a cache memory can be used to replace the local buffer as long as the cache size is large enough to cover a block-type access. The cache memory is placed before the address-translation logic and looked up with virtual addresses obtained by linearly translating the logical indices. In this case, the proposed method can improve the system performance by reducing overhead cycles of external SDRAM accessed for cache line filling as well as by eliminating external memory accesses for the data hit in the cache.

A. Hardware Implementation of Address-Translation Logic

First-Line block takes logical indices, window sizes and the number of data to be accessed as its inputs and generates the physical address of the first data element by dividing the logical indices (I, J) by window sizes (h_size, v_size). In case of 2-D arrays, two division operations are required for horizontal and vertical axes. To minimize the hardware overhead, a bit-serial divider is employed to generate quotient and remainder results simultaneously.

The Next-Line block calculates the physical address of the first data in each line. Since the video data are accessed in a block-type, the physical locations of the rest lines can be incrementally calculated from that of the first line. For the incremental calculation, we use three counters—a bank counter, a row counter and a column counter—which select their increments according to the relation between the windows of the current data and the previous data, as shown in (5)–(7). The bank counter can select its increment among 0, 1 and 2, while the row counter can select its increment among 0, 1, and $Hrow$, which denotes the number of rows that are needed to store as many array lines as two times of vertical window size (2*v_size). If the window of the current data is the same as that of the previously accessed data, the bank and the row addresses do not change. If the current window is horizontally contiguous to the previous window-like block-type accesses Fig. 13(a) and (b), the bank counter increases by 1 and the row counter increases by 0 or 1. Lastly, if the current window is vertically contiguous to the previous window like block-type accesses Fig. 13(c) and (d), the bank counter increases by 2 and the row counter increases by 0 or $Hrow$. Similarly, the column counter can select its operand among $Imod$, $Incr1$ and $Incr2$, where $Imod$ denotes the horizontal position of the first data within the window and $Incr1$ and $Incr2$ are calculated with the window size and the vertical increment in the innermost loop. As shown in Fig. 14, $Imod$, $Incr1$, and $Incr2$ are used for the column address calculations of the first data in the horizontally contiguous window, the next line data within the window and the next line data in

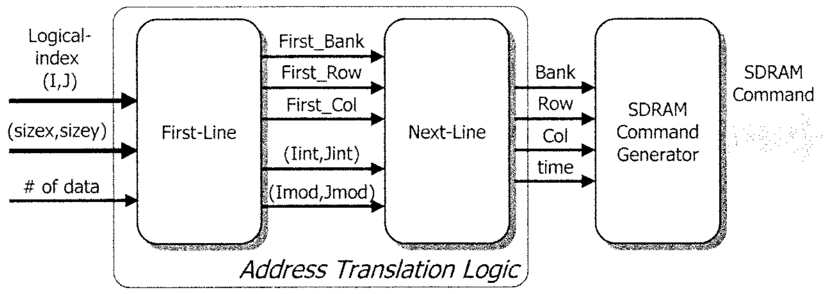

Fig. 12. Block diagram of the SDRAM controller.

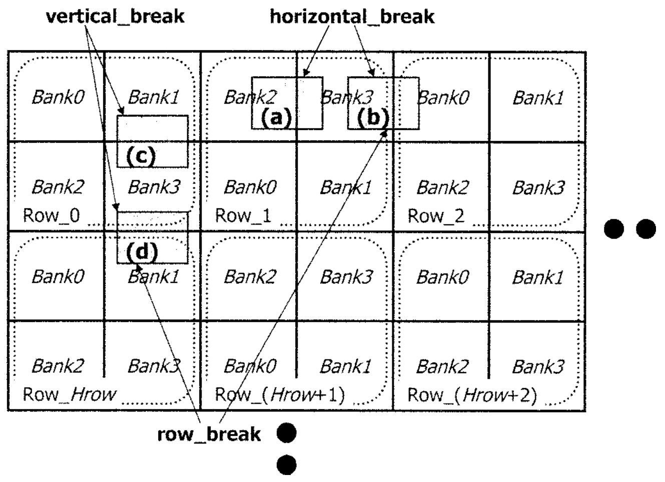

Fig. 13. Mapping of logical window to physical memory.

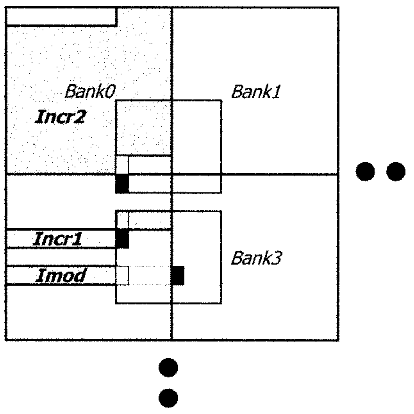

Fig. 14. Increments for column address calculation.

the vertically contiguous window. The numbers of the shaded data denote increments for calculating new addresses from old addresses

$$bank\_address(0) = bank\_counter(0) = the\_first\_bank\_address$$

$$bank\_address(n) = \begin{cases} bank\_counter(n-1)+1, & \text{if } horizontal\_break = TRUE \\ bank\_counter(n-1)+2, & \text{if } vertical\_break = TRUE \\ bank\_counter(n-1), & \text{otherwise} \end{cases}$$

$$bank\_counter(n) = \begin{cases} bank\_counter(n-1), & \text{if } horizontal\_break = TRUE \\ bank\_address(n), & \text{otherwise} \end{cases} \quad (5)$$

$$row\_address(0) = row\_counter(0) = the\_first\_row\_address$$

$$row\_address(n) = \begin{cases} row\_counter(n-1)+1, & \text{if } (horizontal\_break \,\&\, row\_break) = TRUE \\ row\_counter(n-1)+Hrow, & \text{if } (vertical\_break \,\&\, row\_break) = TRUE \\ row\_counter(n-1), & \text{otherwise} \end{cases}$$

$$row\_counter(n) = \begin{cases} row\_counter(n-1), & \text{if } (horizontal\_break \,\&\, row\_break) = TRUE \\ row\_address(n), & \text{otherwise} \end{cases} \quad (6)$$

$$column\_address(0) = column\_counter(0) = the\_first\_column\_address$$

$$column\_addres(n) = \begin{cases} column\_counter(n-1)-Imod, & \text{if } horizontal\_break = TRUE \\ column\_counter(n-1)+Incr2, & \text{if } vertical\_break = TRUE \\ column\_counter(n-1)+Incr1, & \text{otherwise} \end{cases}$$

$$column\_counter(n) = \begin{cases} column\_counter(n-1), & \text{if } horizontal\_break = TRUE \\ column\_address(n), & \text{otherwise.} \end{cases} \quad (7)$$

Since these increments depend on loop characteristics, array information and window size, it is sufficient to calculate them only once at start time. To deal with arrays in loops with different characteristics, the address-translation logic can include several tables of increments that are indexed by loop identifiers.

Hence, the address-translation logic is implemented with a divider, three counters, increment tables, and simple controllers.

As First-Line block is implemented with a bit-serial divider, a division operation takes about 12 cycles, if the bit width of logical indices is 12. Thus, it takes about 26 cycles to calculate the physical location of the first data in First-Line block. After the physical location of the first data is calculated, the Next-Line block can generate the column address of the burst access at every cycle. Finally, the SDRAM command generator generates the commands to hide row-activations as many as possible by using the burst accesses. Since the number of data in a block is usually large and the data are processed in a pipeline way, the latency of the address-translation logic does not degrade the throughput of memory operations.

We implemented the address-translation logic for a MP@HL MPEG-2 video decoder with a 0.35 -μm CMOS standard-cell library. To process a high-resolution video and loops that have the different loop characteristics, the bit-length of array indices is set to 12 and the increment tables have six entries. The hardware complexity of the address-translation logic is 7253 equivalent gates, which is negligible compared to that of the MP@HL MPEG-2 video decoder core.

VII. Energy Reduction by Address Translation

In this section, we explain the effect of the array address-translation technique on energy consumption in SDRAM. As explained in Section III, the memory-related power consumption consists of two components, the core, and the I/O power consumptions. Since row-activations and column data accesses are the major sources of the core power consumption, their power consumption must be reduced. As specified in [13], we assume that the dc currents of row-activation, burst access, and refresh operations are about 150, 180, and 210 mA, respectively. As the last compensation term of (2) is small enough to be ignored, the equation of average current can be simplified to (8), where we assume that $t_{\rm RAS(MIN)}$, $t_{\rm RP(MIN)}$, and $t_{\rm CK(MIN)}$ are 45, 20, and 7.5 ns, respectively. In practice, the number of refresh commands is quite small compared to other commands. Thus, the equation shows that row-activations and column data accesses are important sources of memory power consumption and that a row-activation command induces more current than a column data access by about six times. Therefore, the proposed technique can save the energy consumption by reducing the number of row-activations

$$I = \frac{1}{T}\big(8.40 \times 10^{-9} \times CN + 1.35 \times 10^{-9} \times DN + 13.7 \times 10^{-9} \times RN\big). \quad (8)$$

In addition to the power consumption of the memory core, switching of the interconnection bus connecting the processor and the memory is a crucial source of power consumption. As the switching activity is normally proportional to the number of data and commands, it is required to reduce the number of commands needed to access the same number of data. The proposed address-translation technique has a chance to reduce the power consumption significantly by reducing the number of row-activation commands. Hence, though the technique leads to additional power consumption in the address-translation logic, the reduction of I/O power and bit-line switching power pay off the additional power.

TABLE I
MEMORY PARAMETERS

CAS delay	3 cycles
Page size	1,024 bytes
Data bus width	8 bits
Number of banks	4
Minimum RAS-to-CAS delay ($t_{\rm RCD(MIN)}$)	20 ns
Minimum row active time ($t_{\rm RAS(MIN)}$)	45 ns
Minimum precharge time ($t_{\rm RP(MIN)}$)	20 ns
Last data in to row precharge ($t_{\rm RDL(MIN)}$)	2 cycles
Clock frequency	133 MHz
Refresh period	8k / 64 ms

VIII. Experimental Results

As a test application, we use the MPEG-2 video decoding algorithm [15], which processes 2-D video data and consists of several kernel operations such as variable-length decoding (VLD), de-quantization, IDCT, MC, and block addition (BA). In particular, the MC and BA routines intensively access the video data stored in external memory. Since the kernel operations in MPEG-2 decoding are processed with a group of data called a "macroblock," the memory-access patterns in the routines are almost deterministic, e.g., the burst length is determined to 8, 9, 16, or 17 and the stride of the vertical axis is fixed to 1 or 2 according to the MC mode.

We use a set of MPEG-2 video bit-streams each of which contains 15 frames. Table I shows the characteristics of memory parameters used in this experiment [13]. According to the memory-timing parameters, we assume that the number of cycles needed for each row-activation and pre-charge is in the range of 3–10, which varies with operation types such as read and write. To meet the refresh period, five refresh commands are generated at every 40 μs. With these parameters, we calculate the memory cycles of two address translations; the first stores the video data linearly in order of logical addresses (linear translation), and the second translates the logical addresses appropriately for the MPEG routine (proposed translation).

Table II shows the results of the two address translations. The first and the second columns show the MPEG-2 bit streams and the number of data accessed during MPEG-2 video decoding, respectively. The third and the fourth columns show the number of execution cycles for memory operations. The last column shows the speedup of the proposed method. As shown in the table, the video decoding under the linear translation consumes a large number of cycles for row-activations, while the proposed translation requires almost no additional cycles. On the average, the speedup of the proposed translation over the linear translation is 1.5.

As shown in Fig. 15, the overhead cycles computed by the proposed algorithm is consistent to the result of the exact simulation. The speedup of the proposed algorithm over the exact simulation is about 76 and the relative error is less than 5%, which indicates that our algorithm is efficient in finding an optimal window.

TABLE II
COMPARISON OF MEMORY CYCLES

MPEG-2 Data	Number of memory accesses	Cycles in linear translation	Cycles in proposed translation	Speedup
CAR	11,195,719	17,565,953	11,326,003	1.55
CHEER	12,834,865	19,157,171	12,985,812	1.48
FOOTBALL	13,124,461	19,949,417	13,281,479	1.50
MOBILE	13,362,354	20,368,411	13,490,880	1.51
SUSIE	14,595,623	21,815,309	14,746,552	1.48
Average	13,022,604	19,771,252	13,166,145	1.50

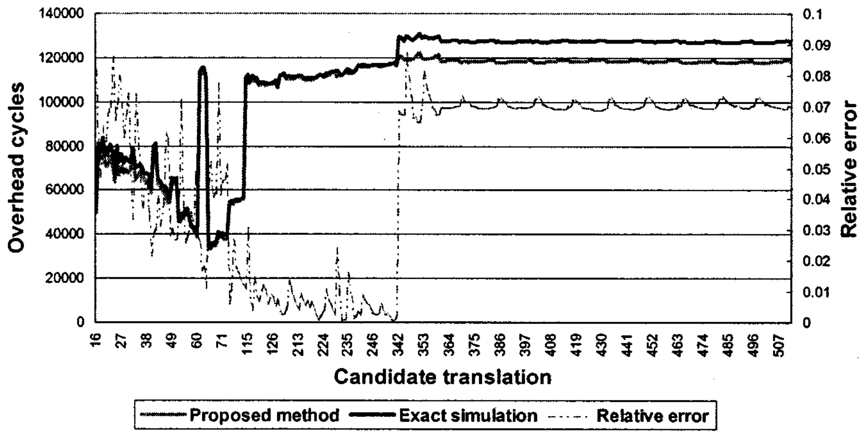

Fig. 15. Consistency of the results of the proposed algorithm and those of the exact simulation.

TABLE III
COMPARISON OF THE NUMBER OF SDRAM COMMANDS

MPEG-2 data	Number of commands in linear translation	Number of commands in proposed translation	Reduction ratio
CAR	1,998,772	1,365,068	31.70%
CHEER	2,101,936	1,519,370	27.72%
FOOTBALL	2,196,587	1,544,242	29.70%
MOBILE	2,246,087	1,574,399	29.90%
SUSIE	2,351,566	1,700,345	27.69%
Average	2,178,990	1,540,685	29.34%

TABLE IV
COMPARISON OF THE NUMBER OF ROW-ACTIVATION COMMANDS

MPEG-2 Data	Number of row-activation commands in linear translation	Number of row-activation commands in proposed translation	Reduction Ratio
CAR	969,850	93,816	90.33%
CHEER	969,784	107,834	88.88%
FOOTBALL	1,045,588	110,671	89.42%
MOBILE	1,078,684	114,364	89.40%
SUSIE	1,111,940	133,554	87.99%
Average	1,035,169	112,048	89.20%

Table III shows that the proposed translation reduces 29% of SDRAM commands which have a significant effect on the switching activity of the external address bus. Particularly, the number of row-activation commands that induce large energy consumption in SDRAM is reduced by a factor of ten, as shown in Table IV. Table V shows the active current, total execution time and energy consumption of the two translations. The average current is calculated by taking an average over the execution time. Since the execution time is decreased very much in the proposed translation, the average current is slightly increased.

TABLE V
COMPARISON OF ENERGY CONSUMPTION IN SDRAM CORE

MPEG-2 Data	Linear translation			Proposed translation			Reduction ratio of energy consumption
	Active current (mA)	Execution time (s)	Energy consumption (mJ)	Active current (mA)	Execution time (s)	Energy consumption (mJ)	
CAR	177.8	0.1321	77.50	188.4	0.0852	52.96	31.67%
CHEER	178.6	0.1440	84.87	188.4	0.0976	60.72	28.46%
FOOTBALL	178.4	0.1500	88.30	188.4	0.0999	62.10	29.67%
MOBILE	178.7	0.1531	90.29	189.0	0.1014	63.27	29.93%
SUSIE	178.8	0.1640	96.77	189.5	0.1109	69.35	28.34%
Average	178.4	0.1487	87.55	188.8	0.0990	61.68	29.61%

Nevertheless, the energy consumption of memory core that is a product of the average current and the execution time is saved by about 30% on the average, as shown in Table V. To take into consideration the power consumption of the address-translation logic, we estimate the power using PowerMill. As the estimated average current is 3.1 mA, the overhead is about 1.6%.

IX. CONCLUSION

We have proposed a new memory-interface architecture that translates memory addresses to improve memory bandwidth and power consumption in multi-dimensional video-processing applications. We have also presented an algorithm that searches for a suitable window size that can be stored in a row of SDRAM considering the memory-access patterns and memory parameters. Experimental results show that the proposed translation is very effective in increasing the memory bandwidth of SDRAM and reducing the number of memory cycles required for row changes and precharges. In addition, the proposed architecture can save energy consumption of SDRAM by reducing the switching activity of the address bus and the number of bit-line charging.

REFERENCES

[1] E. D. Greef, F. Catthoor, and H. D. Man, "Program transformation strategies for memory size and power reduction of pseudoregular multimedia subsystems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 719–733, Oct. 1998.

[2] Y. Takai, M. Nagase, M. Kitamura, Y. Koshikawa, N. Yoshida, Y. Kobayashi, T. Obara, Y. Fukuzo, and H. Watanabe, "250 Mbyte/s synchronous DRAM using a 3-stage-pipelined architecture," *IEEE J. Solid-State Circuits*, vol. 29, pp. 426–431, Apr. 1994.

[3] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. D. Man, "Global communication and memory optimizing transformations for low power signal processing systems," in *VLSI Signal Processing*, vol. VII, 1994, pp. 178–187.

[4] H. Kim and I.-C. Park, "Array address translation for SDRAM-based video processing applications," *Electron. Lett.*, vol. 35, pp. 1929–1931, Oct. 1999.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture a Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1990, pp. 420–490.

[6] P. R. Panda, N. D. Dutt, and A. Nicolau, "Memory organization for improved data cache performance in embedded processors," in *Proc. Int. Symp. System Synthesis*, 1996, pp. 90–95.

[7] ——, "Memory data organization for improved cache performance in embedded processor applications," *ACM Trans. Design Automat. Electron. Syst.*, vol. 2, pp. 384–409, Oct. 1997.

[8] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, and H. De Man, "Cache conscious data layout organization for embedded multimedia applications," in *DATE*, 2001, pp. 686–691.

[9] K. Asheesh, P. R. Panda, N. D. Dutt, and A. Nicolau, "High-level synthesis with synchronous and RAMBUS DRAMs," in *Proc. SASIMI'98*, 1998, pp. 186–193.

[10] M. Winzker, P. Pirsch, and J. Reimers, "Architecture and memory requirements for stand-alone and hierarchical MPEG2 HDTV-decoders with synchronous DRAMs," in *Proc. Int Symp. Circuits and Systems*, 1995, pp. 609–612.

[11] T. Takizawa, J. Tajime, and H. Harasaki, "High performance and cost effective memory architecture for an HDTV decoder LSI," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1999, pp. 1981–1984.

[12] P. R. Panda and N. D. Dutt, "Low-power memory mapping through reducing address bus activity," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 309–320, Sept. 1999.

[13] Samsung Electronics. (1999) Data book. [Online] Available: http://www.samsungelectronics/semiconductors/DRAM/DRAM.htm

[14] Elpida Memory Inc.. (1998) How to use SDRAM. [Online] Available: http://www.elpida.com/en/products/index.html

[15] *Generic coding of moving pictures and associated audio*, ISO/IEC (MPEG-2), Mar. 1994.

Hansoo Kim received the B.S. degree from Yonsei University, Seoul, Korea, in 1994, and the M.S. and Ph.D. degrees from Korea Advanced Institute of Science and Technology, Taejon, Korea, in 1996 and 2001, respectively, all in electrical engineering.

In 2001, he joined LG Electronics, Seoul, Korea, where he is currently a Senior Member of Engineering Staff. His research interests include algorithms and VLSI architectures for video compression.

In-Cheol Park received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1988 and 1992, respectively.

From 1995 to 1996, he was with IBM T.J. Watson Research Center, Yorktown Heights, NY, as a Member of Technical Staff in the area of circuit design. In 1996, he joined the faculty of the Department of Electrical Engineering, KAIST, where he is currently an Associate Professor. His research interests include CAD algorithms for high-level synthesis and VLSI architectures for general-purpose microprocessors.