IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 13, 2019, accepted September 3, 2019, date of publication September 11, 2019,
date of current version September 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940351

Adaptive Linear Address Map for Bank Interleaving in DRAMs

JAE YOUNG HUR[1], (Member, IEEE), SANG WOO RHIM[1], BEOM HAK LEE[1], AND WOOYOUNG JANG[2], (Member, IEEE)

[1]Faculty of Engineering, Vietnamese-German University, Ho Chi Minh 75114, Vietnam
[2]Department of Electronics and Electrical Engineering, Dankook University, Yongin 16890, South Korea

Corresponding author: Wooyoung Jang (wyjang@dankook.ac.kr)

ABSTRACT The conventional linear address map can degrade memory utilization and system performance when an access pattern is not linear. To improve memory system performance, the adaptive bank-interleaved linear address map for a DRAM technology is proposed. In our approach, the addresses are efficiently rearranged using the bank-flipping technique for a given application and a memory configuration. The system can configure the address map based on the bank interleaving metric in the systematic way when an application is invoked. Considering image processing applications, the algorithm, the analysis, the design, and the evaluation of the proposed address map are presented. The experimental results show that the presented method can effectively improve the performance with a moderate hardware cost.

INDEX TERMS Address mapping, DRAM, embedded system, architecture, performance.

I. INTRODUCTION

A modern system on a chip (SoC) increasingly embeds high bandwidth image processing components together with high resolution displays and cameras. Typically, a dynamic random access memory (DRAM) is attached to the SoC as a main memory due to its high density and low cost. As a display resolution increases, the memory bandwidth of multimedia traffic accordingly increases. However, the capacity of a DRAM is still limited and usually is the bottleneck of system performance. From the architecture perspective, the low utilization (or the bottleneck) is related to bank access patterns in a DRAM. A DRAM typically accommodates multiple banks. A bank is organized with two dimensional array of rows and columns. When a different row in the same bank is accessed, the previously accessed row must be precharged and a new row must be activated. This operation (called a bank conflict) requires significant cycles. Frequent bank conflicts degrade memory utilization and system performance. Therefore, it is important to minimize bank conflicts [1]–[3].

In an image processing application, a pixel is associated with an address. A master generates memory transactions using an address map and accesses memory locations using a memory map. Typically, memory addresses of image pixels are organized in the row-wise linear and sequential manner. If the memory access pattern of an application is linear, high utilization can be maintained in a DRAM. This is because bank conflicts are minimized. Accordingly, the linear address map performs well for sequential address patterns such as a raster-scan display operation. In some cases, however, an access pattern is not linear. As an example, when an image is accessed in the vertical direction, adjacent transactions can incur bank conflicts and degrade memory utilization. The conventional method to alleviate this problem is to design a memory map in a slave for particular traffic patterns [2]–[4]. Nevertheless, traffic patterns vary with different use cases, applications, or masters. A memory map for particular access patterns may not be suitable for the other patterns.

A bank interleaving technique is one of the widely used methods to improve memory utilization [1]–[3]. Using this technique, adjacent transactions access multiple banks in the statistically interleaved manner. Subsequently, activating and precharging latencies can be possibly hidden. In some cases, the conventional address map provides desirably interleaved patterns. However, in the other cases, the conventional address map incurs undesirably interleaved patterns. Depending on an application and a memory configuration, bank access patterns are not always interleaved as desired. It is expected that overall performance can be improved by customizing an addressing scheme for those undesirable cases.

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

TABLE 1. Summary of related works.

Attributes		[1]	Our	[2]	[3]	[4]
Design approach		Bank-interleaved address generation in a master		Bank-interleaved memory map in a slave		Configurable memory map in a slave
Address map	Format	Adaptive tiled map for *RBC*	Adaptive linear map	Conventional map for *RBC*	Conventional map for *RCBC*	Conventional map for *RBC* or *BRC*
	Operation	Superpage level	Grouped-superpage level			
Memory map		*RBC*	Various	*Permuted RBC*	*Permuted RCBC*	*RBC* or *BRC*

Based on these motivations, an adaptive approach is proposed. To do this, a configurable address map is devised to enhance bank interleaving. Combining the advantages of the legacy linear address map and the customized address map, the proposed approach can effectively improve memory system performance. The main contributions of this paper are:

1) It is proposed that the linear address map is adaptively rearranged using the bank-flipping technique in the grouped-superpage level. The generalized criteria to apply the proposed method is derived.
2) The method to apply our address map to various memory maps is presented.
3) The performance of the proposed address map is evaluated.
4) A hardware design is presented and the overheads are evaluated.

The paper is organized as follows. In Section II, related work is described. In Section III, the conventional designs are presented. In Section IV, the proposed design is presented. In Section V, the experimental results are described. Finally, conclusions are drawn in Section VI.

II. RELATED WORK

1) BANK-INTERLEAVED ADDRESS MAP

In [1], an address map that is rearranged to enhance bank interleaving is presented. This paper is similar to [1] in that the *bank-flipping* technique is utilized to rearrange the address map. Additionally, the metric analysis similar to [1] is conducted in this paper. This paper differs from [1] in the following ways. First, the adaptive linear address map is presented in this paper, whereas a tiled map is claimed in [1]. Second, the system in this paper operates in the grouped-superpage level, whereas the system in [1] operates in the superpage level. As a result, our method can outperform [1]. Third, our design can be utilized for arbitrary image sizes without an additional memory space compared to the conventional design. In [1], an additional amount of memory space can be required. Fourth, the design method to adapt an address map to various memory maps is presented, whereas only the fixed row-bank-column (*RBC*) map is considered in [1]. Fifth, the hardware design, the overhead evaluation, and the performance evaluation are described.

2) MEMORY MAPPING IN A SLAVE

A number of works to reduce bank conflicts are reported. In [2], a memory map using the permutation-based page interleaving is proposed. In [3], a memory map for the minimal open page is proposed. In [2] and [3], bank indices are generated using the well-known XOR permutation with the row bit patterns. In [4], multiple classes of memory components in a system are described. Different types of configurable memory maps are presented in [4]. These permutation-based and configurable approaches are typically used in practice. Moreover, these approaches require an insignificant hardware cost. Table 1 summarizes the reference schemes. In Table 1, a row, a bank, a column are denoted by *R*, *B*, *C*, respectively.

In [5], the bit-reversal address mapping scheme is presented. The order of the high address bits is reversed in [5]. In [6], the memory technology is presented to enable both row-wise and column-wise activations such that the memory device is adapted to an access pattern. In [7], an application-specific memory controller is presented. The DRAM access patterns of an application are analyzed using a combinatorial optimization. Then a customized memory map is generated based on the analysis. In [8], the DRAM subsystem design space exploration framework is presented. The framework in [8] generates a customized memory map. To do this, the DRAM access trace of an application is analyzed to detect hot-spot row bits. Then XOR permutations (similar to [2]) are conducted. In our approach, the trace analyses in [7] and [8] are not required.

In [9], the run-time rearrangement of a memory map is presented. The memory controller contains the on-line prediction hardware that detects workload-specific address mapping using a counter. When an address mapping is changed, the existing data are migrated to new locations in a DRAM. In our work, such a data migration and its overhead are not required. These methods [2]–[9] are implemented in a memory controller or a DRAM device in the slave side. An issue is that it is difficult for a single slave to meet various requirements of different masters and exploit their access patterns. Contrast to [2]–[9], our adaptive address map in a master exploits its own pattern for a given slave configuration. Our approach does not require the hardware modification of a memory controller.

3) SOFTWARE APPROACH

In [10], The memory allocator of an operating system (OS) for bank privatization is presented. In [10], expecting a thread has a high spatial locality, physical frames mapped to the same DRAM bank can be exclusively allocated to a single

thread. In [11], the per-core bank privatization is presented. The memory allocator can map an OS page into a specific memory bank. In [10] and [11], bank conflicts between processes can be reduced because a process can exclusively access a bank. However, when multiple applications concurrently run, it can be difficult for a single process to privatize a bank because the number of banks is limited. In our work, using the legacy OS allocator, a hardware rearranges an address layout when an application is invoked and bank conflicts are expected.

4) APPLICATION-SPECIFIC DESIGN

In [12], to exploit regular multimedia data access patterns, the system accommodates the prediction logic that monitors an access pattern. Then the memory controller conducts a prefetch using the monitored information. An issue is that, when multiple applications concurrently run, it is difficult to analyze multiple type of patterns. Additionally, a hardware can be complex to support the run-time prediction. In [13], the processor architecture that accommodates a customized instruction for multimedia applications is presented. In this architecture, multiple pixel data are computed in parallel using the instruction. In our work, the customized address map for bank-level parallelism in a DRAM is presented. In [14], conflict-free addressing schemes for the memory-based FFT processor is presented. In [14], data are distributed to different memory banks and the data can be concurrently accessed. Our work differs from [14] in that an addressing scheme to exploit the characteristics of DRAM banks is presented.

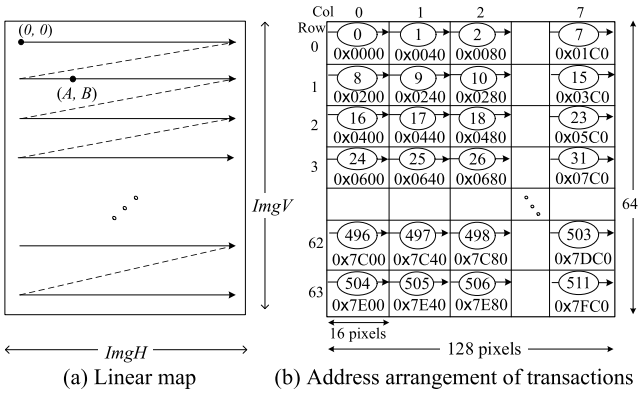
5) ADDRESS LAYOUT IN A MASTER

In the master side, a number of address maps for efficient memory accesses are reported. In [15]–[17], a tiled address layout is presented. In [15], the image tile sizing algorithm using the constraint programming is presented. In [16], the 4D tile format is presented. In [17], the 4-level Z-order tile format for 2D data processing is presented. These formats can be suitable for a particular access pattern or a system configuration. An issue is that it is difficult to exploit the characteristics of different access patterns because the tile formats are fixed. In our work, the configurable address layout is presented.

In [18], to reduce bank conflicts, the data-to-DRAM bank mapping algorithm is presented. To do this, a graph traversal analysis is required. In [19], the hashed addressing for the banked scratchpad in a GPU using the configurable bit-vector is presented. To configure hash functions, a heuristic (exhaustive) search is conducted. Similar to our work, an address layout is reorganized for an application. Our work differs from [18] and [19] in that the address map is selected using a metric criteria in the deterministic way. In our work, the metric criteria is represented by relatively simple formulations. Our approach does not require the off-line analysis.

III. CONVENTIONAL DESIGN

In this section, the conventional address map and the memory map are described as a background. Then an image rotation application is described as a motivational use case. Image pixels are mapped onto a transaction address using an address map. There are three general rules to associate an image pixel with an address. First, a unique image pixel is mapped to a unique address. Second, the address value should be within the allocated memory space. Third, when multiple masters (or processes) communicate using a shared address space, the masters should use the same address map to maintain consistent data.



(a) Linear map (b) Address arrangement of transactions

FIGURE 1. Linear address map (LIAM).

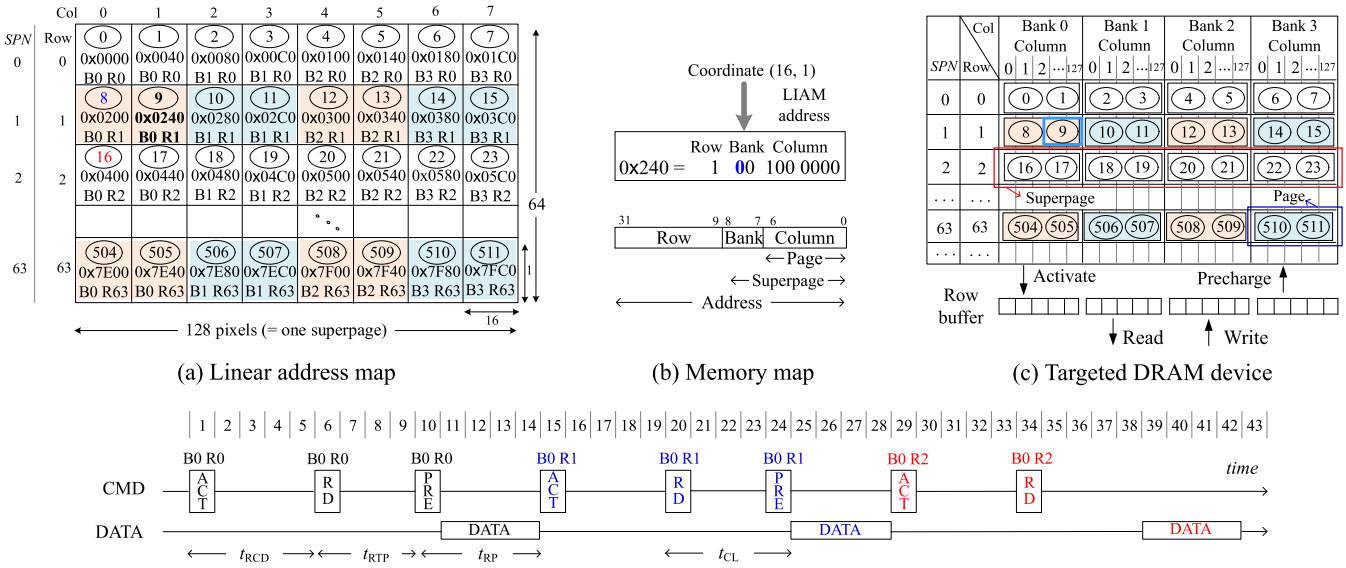
A. LINEAR ADDRESS MAP

The linear address map (LIAM) is one of the most widely used address maps because of its simplicity and clarity. A software developer typically uses the linear map by default. Fig. 1(a) shows the linear map. Addresses sequentially increase in the horizontal direction. *ImgH* is an image horizontal size and *ImgV* is an image vertical size in the number of pixels. The address of the coordinate (A, B) is the following:

$$\text{Address} = BaseAddr + A \times BytePixel + B \times ImgHB \quad (1)$$

where *BaseAddr* denotes the base address of an allocated memory space. *BaseAddr* is the address of the coordinate $(0, 0)$. *BytePixel* denotes a pixel size in bytes. *ImgHB* denotes an image horizontal size in bytes. Suppose an image size is 128×64. When *BytePixel* is 4 B, *ImgHB* is 512 B ($=128 \times 4$ B). In case *BaseAddr* is 0, the address of the coordinate (16, 1) is 576 ($= 0 + 16 \times 4 + 1 \times 512$) or 0×240.

Image pixel data are written to or read from memory in the granularity of a transaction. A transaction is typically a burst of data transfers. A data transfer can contain multiple pixels. Suppose the datawidth of a system bus is 16 B. In case a pixel is represented in 4 B, a data transfer contains 4 ($=\frac{16\text{ B}}{4\text{ B}}$) pixels. When a transaction has 4 transfers, the transaction contains



(a) Linear address map (b) Memory map (c) Targeted DRAM device

(d) Timing diagram in DRAM interface for three transactions ⓪, ⑧, ⑯.
DDR3 timing parameters t_{CL}-t_{RCD}-t_{RP} = 5-5-5 cycles.

FIGURE 2. **Mapping example in LIAM.**

16 (=4x4) pixels or 64-byte data. Fig. 1(b) shows the example of a linear map in the transaction granularity. The number in a circle is the transaction number. This number indicates the order of addresses.

B. MEMORY MAP

A transaction address is mapped onto a DRAM location (row, bank, column) using a memory map. Fig. 2 shows the case study that is referred to throughout this paper. Fig. 2(a) shows a linear address mapping example. *ImgHB* is 512 B (=128 x 4 B). Fig. 2(b) shows a memory map. A physical address is organized in the order of row, bank, and column. This is called an *RBC* memory map. As an example, the address 0x240 of the transaction ⑨ is mapped to the row 1, the bank 0, and the column 0x40. Fig. 2(c) shows a DRAM organization. The number of banks is 4. A row in a bank is called a *page*. An entire row of all banks is called a *superpage*. A superpage size (*SPS*) is calculated by page size x number of banks. In this example, page size is 128 B. Therefore, *SPS* is 512 B (=128 B x 4). In Fig. 2, the entire image data are mapped to 64 superpages with superpage numbers (*SPNs*) 0 to 63 in the sequential manner.

A memory interface has a command bus, an address bus, and a data bus. To access the data of a DRAM cell, the row of a bank must be activated and copied to the row buffer of the bank. The row buffer stores the currently activated row. Then read or write bursts are issued to the row buffer. When a different row of the bank is accessed, the previously activated row should be precharged and stored back into the memory cell. Then a new row must be activated. This operation (bank conflict) requires typically tens of cycles depending on memory devices. During this period, any data in the same bank can not be accessed. Therefore, it is important to minimize bank conflicts. Additionally, from the utilization perspective, it is desired to access an activated row as much as possible.

C. MOTIVATIONAL USE CASE

Memory performance is in general significantly affected by address patterns. A rotation application is considered as an example since it is widely used in modern mobile devices. Typically, a hardware accelerator conducts this operation because the application requires high memory bandwidth. Fig. 3 shows the *rotated preview* scenario example. First, a camera captures an image in the raster-scan order and conducts the rotation. Second, the camera controller writes the image in the vertical direction. Third, a display controller reads the image in the raster-scan order and finally displays the rotated image.

In the linear map, a system often suffers from bank conflicts when an access pattern is not linear. In Fig. 2(a), suppose an image is vertically accessed in the order of ⓪, ⑧, ⑯, and so on. The first transaction ⓪ accesses the bank 0 and the row 0 (represented by B0R0). The second transaction ⑧ accesses the bank 0 and the row 1 (B0R1). To do this, the previously activated row 0 should be precharged and the new row 1 should be activated. Fig. 2(d) shows the timing diagram of the transactions ⓪, ⑧, ⑯. The consecutive transactions ⑧ and ⑯ incur bank conflicts. Subsequently, the precharge (*PRE*) and the activation (*ACT*) operations are serialized. This bank conflict overhead is undesirable.

Another issue is an adaptivity. Fig. 4 shows various memory maps. Specific memory maps (for example [2], [3] to

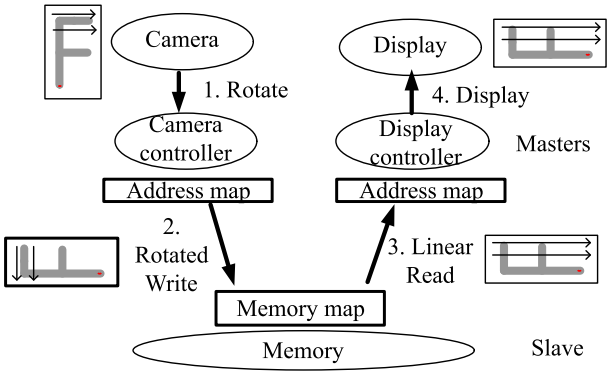


FIGURE 3. Image *rotated preview* scenario.

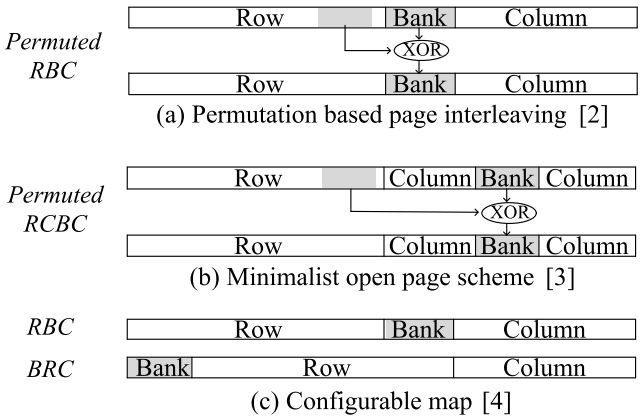


FIGURE 4. Various memory maps.

enhance bank interleaving and [4] to provide configurability) can be designed for particular access patterns. However, applications running in different masters can have different requirements. Even when a master runs a single application, access patterns can vary. An access pattern is often not well matched with the underlying memory map. Moreover, a memory map is typically fixed at design time or configured at booting time. As an underlying memory map is fixed, it is difficult to exploit the characteristics of different access patterns. In [4], a rule to configure the memory maps is not presented.

IV. BANK-INTERLEAVED LINEAR ADDRESS MAP

The aim is to improve the performance of an application using an adaptive approach. An I/O device accelerator that operates dedicated image processing tasks and generates physical addresses in embedded systems is mainly considered. In this case, the address map is implemented in hardware. Therefore, it is desired to design the hardware with an insignificant cost.

A. OVERVIEW

The bank-interleaved address map that adapts itself to an application for a given memory configuration is devised. The main approach is to *exchange bank addresses* of the linear map. An example is shown in Fig. 5. In Fig. 5(a), a set of

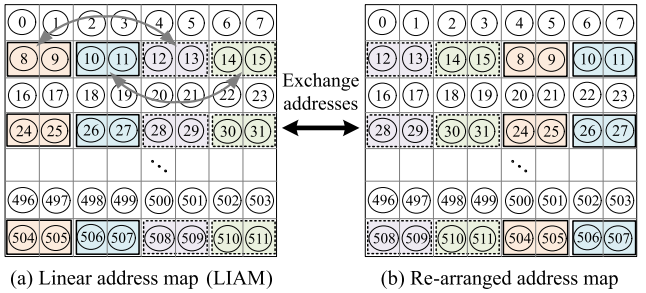


FIGURE 5. The general approach.

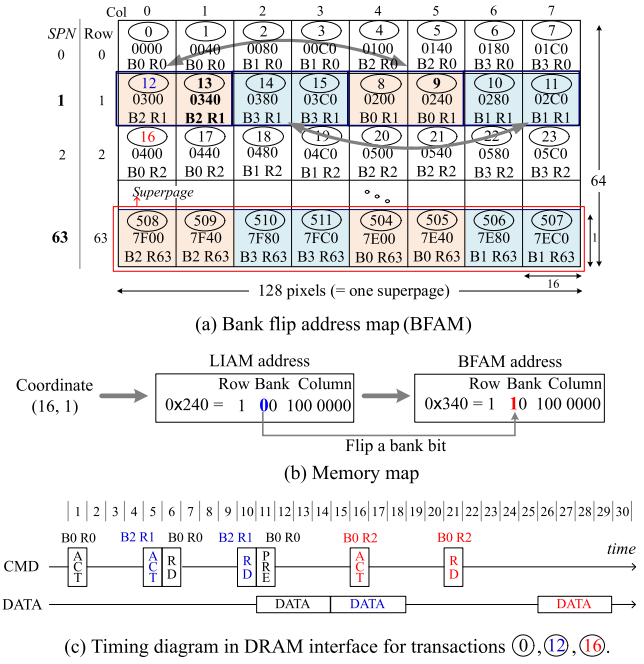


FIGURE 6. Mapping example in BFAM.

addresses (the solid and the dotted rectangles) are flipped in every other row. Then the address map is rearranged by exchanging the addresses. This is shown in Fig. 5(b).

Fig. 6(a) shows the rearranged address map. The addresses in the odd-numbered rows are rearranged. When an image is vertically accessed, the order of addresses is 0x0, 0x300, 0x400, and so on. In this case, the order of mapping patterns is B0R0, B2R1, B0R2, and so on. As clearly can be found, the adjacent addresses are bank interleaved. An implementation technique to rearrange the addresses is to flip the (most significant) bank bit of a LIAM address. This is shown in Fig. 6(b). The address of the coordinate (16, 1) becomes 0x340. The LIAM address 0x240 is exchanged with the address 0x340 by flipping a bank bit. This is called *bank flip address map* (BFAM). In this example, BFAM is expected to be beneficial in terms of bank interleaving. Fig. 6(c) shows the timing diagram of the transactions ⓪, ⑫, ⑯. The consecutive transactions ⓪ and ⑫ access different banks. The activation (*ACT*) command for ⑫ is issued in advance (in cycle 5) while ⓪ is served. In this way, the bank conflict latency

overhead can be hidden. In Fig. 6(c), BFAM requires 29 cycles. For comparison, in Fig. 2(d), LIAM requires 42 cycles. In this case, BFAM can significantly reduce bank conflicts and improve memory utilization compared to LIAM. It is noted that, depending on an image size and a superpage size, BFAM may not be always beneficial. It is necessary to clarify the cases where BFAM is expected to be better.

B. METRIC ANALYSES

To compare BFAM with LIAM, three metrics are defined. Then a comparative analysis is conducted.

1) METRIC OF ADJACENT TRANSACTIONS

To compare bank access patterns, the metric of adjacent transactions is defined by the following:

$$\text{Metric}_{adj} = \begin{cases} -1, & \text{if same bank and different row} \\ 2, & \text{if same bank and same row} \\ 1, & \text{if different banks} \end{cases} \quad (2)$$

where the (low or undesirable) metric -1 is given when adjacent transactions incur bank conflicts. The (higher) metric 2 is given when adjacent transactions access the same bank and the same row. This is because a bank is better utilized in this case. The (high) metric 1 is given when adjacent transactions access different banks. As an example, for the adjacent transactions ⑮ and ⑧ in Fig. 6(a), Metric_{adj} is 1 because they access different banks. It is noted that these metric values are given for the relative analysis.

2) METRIC OF A TRANSACTION

Considering an image is accessed in both vertical and horizontal directions, the metric of a transaction is defined by the following:

$$\text{Metric}_{i,j} = \sum_{N,S,E,W} \text{Metric}_{adj} \quad (3)$$

where $\text{Metric}_{i,j}$ indicates the metric of a transaction in the row i and the column j. $\text{Metric}_{i,j}$ is calculated by summing up Metric_{adj} in the northern (N), the southern (S), the eastern (E), and the western (W) directions. As an example, for the transaction ⑮ in Fig. 6(a), $\text{Metric}_{1,3}$ is 5 (= 1+1+1+2). Fig. 7 shows $\text{Metric}_{i,j}$ values of LIAM and BFAM. The numbers in the shaded rectangles indicate Metric_{adj} values for $\text{Metric}_{1,3}$.

3) AVERAGE METRIC

An average metric for the total number of transactions is derived by the following:

$$\text{Average metric} = \frac{\sum_{i,j} \text{Metric}_{i,j}}{\text{Total number of transactions}} \quad (4)$$

In Fig. 7, the total number of transactions is 512 (=64x8). The average metrics of LIAM and BFAM are 0.8 and 4.7, respectively. The higher metric suggests that bank accesses are highly interleaved. In this case, BFAM is expected to be better.

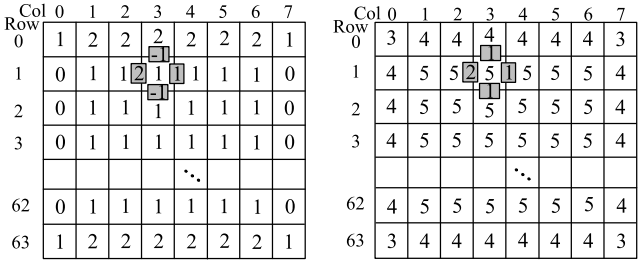


FIGURE 7. Metric of a transaction in LIAM (Fig. 2) and BFAM (Fig. 6).

In Fig. 2, *ImgHB* is 512 B. This is equal to *SPS*. When an image is vertically accessed, significant bank conflicts are expected to occur. However, when *ImgHB* is 1.5 times *SPS*, bank conflicts will be minimized. It is observed that the bank interleaving pattern varies with the ratio of *ImgHB* to *SPS*. Subsequently, the average metrics are derived for different image sizes and superpage sizes. This is shown in Table 2. T denotes the ratio of *ImgHB* to *SPS*. In Table 2, the bold numbers indicate higher metrics.

TABLE 2. Average metrics of LIAM and BFAM for different image sizes.

ImgHB (bytes)	(a) Superpage size 4 kB			(b) Superpage size 8 kB		
	T	LIAM	BFAM	T	LIAM	BFAM
2048	0.5	**5.31**	4.28	0.25	**5.38**	**5.38**
4096	1	3.16	**5.38**	0.5	**5.45**	4.34
6144	1.5	**5.43**	4.26	0.75	**5.50**	**5.50**
8192	2	2.99	**5.47**	1	3.05	**5.53**
10240	2.5	**5.50**	3.55	1.25	**5.57**	**5.57**
12288	3	2.80	**5.54**	1.5	**5.60**	4.24
14336	3.5	**5.57**	3.19	1.75	**5.64**	**5.64**
16384	4	2.62	**5.60**	2	2.68	**5.67**

Similar to Table 2, average metrics versus T values are further derived. This is depicted in Fig. 8. The main observations are:

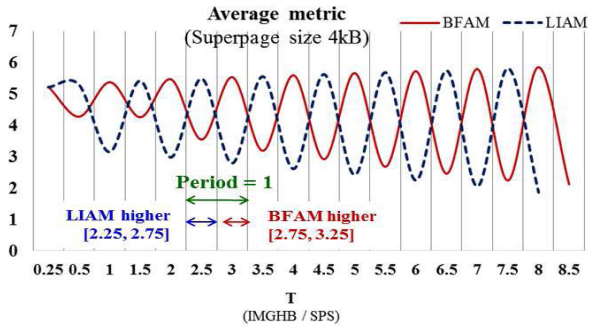
1) The metric is periodic. The period of T is 1.
2) The metrics of LIAM and BFAM are mutually supplementary. The metric of BFAM is higher in one half of a period. The metric of LIAM is higher in the other half of a period.
3) If T is within the ranges of [0.75, 1.25], [1.75, 2.25], [2.75, 3.25], and so on, the metric of BFAM is higher than LIAM.

C. CONDITION TO SELECT BFAM

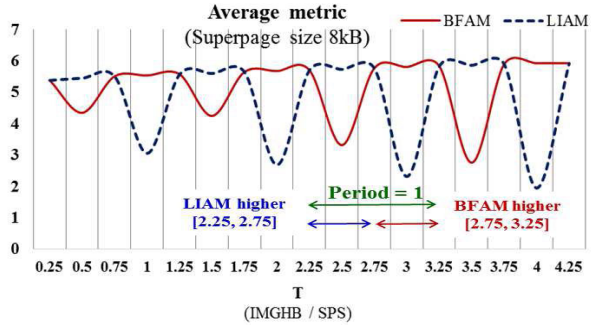
Based on the observation in Fig. 8, the condition to select BFAM is derived as follows:

$$k - \frac{1}{4} \le T \le k + \frac{1}{4} \quad (5)$$

where k is a positive integer. If the condition is met, the metric of BFAM is higher than LIAM. This means that BFAM



(a) Superpage size 4kB



(b) Superpage size 8kB

FIGURE 8. Average metrics versus T values.

Adaptive address map
Input: Coordinate
Output: Address

1. Calculate LIAM address.
2. **if** BFAM is selected **then** // Eq. (5) is true.
3. Calculate BFAM address. // Apply bank-flipping algorithm.
4. **else**
5. Use LIAM address.
6. **end if**

FIGURE 9. Operation of the address map.

Bank-flipping algorithm
Input: LIAM address, *SPS*, *k*
Output: New bank number

1. // Get superpage number.
2. *SPN* = **floor** (LIAM address / *SPS*)
3. // Get grouped-superpage number.
4. *GSPN* = **floor** (*SPN* / *k*)
5. // Conduct bank flipping in every other *GSPN*.
6. **if** *GSPN* is odd number **then**
7. Flip the most significant bank bit in LIAM address.
8. **end if**

FIGURE 10. Bank-flipping algorithm.

is expected to provide better bank interleaving. Otherwise, LIAM is expected to provide better interleaving. Typically, the memory access pattern of an image processing application is regular and has certain strides. A stride is defined as an address distance between adjacent memory transactions.

$$Stride = \begin{cases} ImgHB & \text{if vertical access} \\ TransSize & \text{if hotizontal access} \end{cases} \quad (6)$$

where *TransSize* denotes a transaction size in bytes. In a raster-scan operation, an access pattern is horizontally linear. In this case, *Stride* is a transaction size, for example 64 B. If an access pattern is vertical, *Stride* is *ImgHB*. The condition in Eq. (5) is affected by *Stride*. Accordingly, T is redefined as follows:

$$T = \frac{Stride}{SPS} \quad (7)$$

In Eq. (5), k is defined by a *round* operation as follows:

$$k = \begin{cases} 1 & \text{if } Stride < SPS \\ round\,(T) & \text{otherwise} \end{cases} \quad (8)$$

The T value can be determined when an application is invoked. Based on this value, the system checks the condition in Eq. (5). If the condition is met, the system selects BFAM. Otherwise, the system selects LIAM.

D. OPERATION OF THE ADDRESS MAP

When an application runs, the address map operates as shown in Fig. 9. This is summarized by the following:

1) Taking an image coordinate as an input, the address map calculates a LIAM address using Eq. (1).
2) If BFAM is selected, the LIAM address is rearranged to obtain a BFAM address. To do this, the bank-flipping algorithm (in the next section) is applied.
3) If BFAM is not selected, the LIAM address is used.

E. BANK-FLIPPING ALGORITHM

The bank-flipping algorithm is indicated in the line 3 of Fig. 9. In this section, the algorithm is described. As an image can be arbitrarily sized, a generic algorithm is presented. In the proposed approach, superpages can be grouped. The grouping is conducted to enhance bank interleaving when an image is large sized. A grouped superpage contains k superpages. If superpages are not grouped, k is 1. In this case, the bank flipping operates similar to [1]. For a given LIAM address, our algorithm *modifies a bank number* as shown in Fig. 10. This is summarized by the following:

1) Taking a LIAM address as an input, a grouped-superpage number (*GSPN*) is calculated. *GSPN* is defined by *floor* ($\frac{SPN}{k}$).
2) If *GSPN* is an odd number, the bank flipping is conducted to obtain a new bank number.
3) If *GSPN* is an even number, the bank flipping is not conducted. In this case, a BFAM address is same as a LIAM address.

Fig. 11 shows the mapping example for the image size 256 x 64. In this example, T is 2 ($=\frac{1\text{ kB}}{512\text{ B}}$). Accordingly, k is 2 ($=round\,(T)$). This means a grouped superpage contains

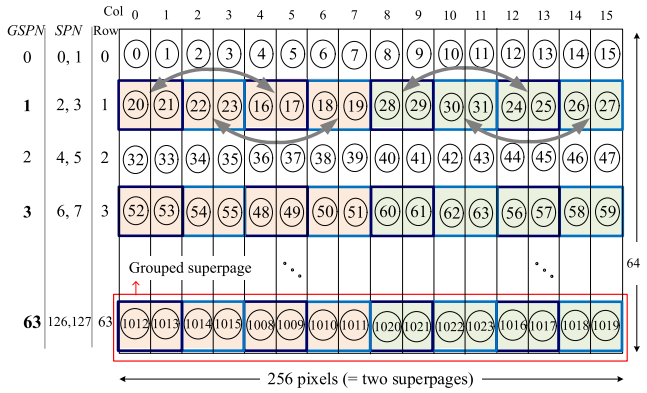


FIGURE 11. Grouping example in BFAM.

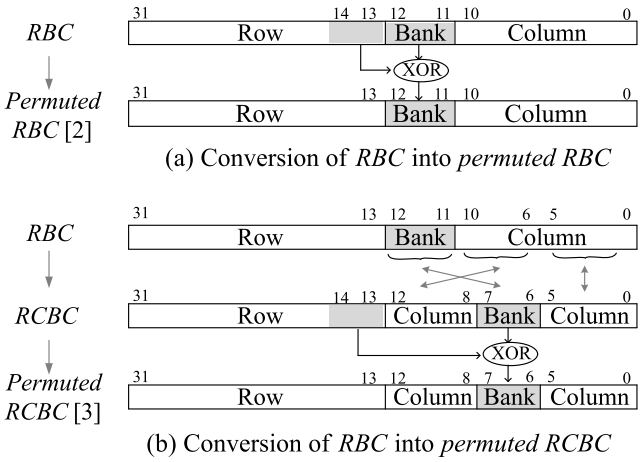


FIGURE 12. Adapting to various memory maps.

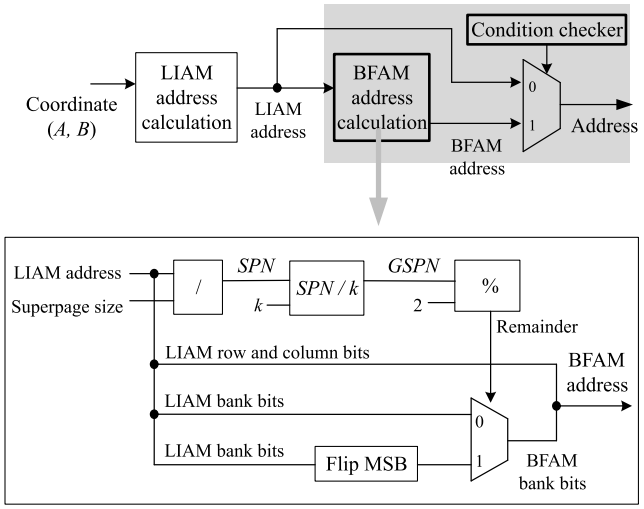


FIGURE 13. Address map hardware.

2 superpages. As T is within the range [1.75, 2.25], BFAM is selected. In this case, the bank flipping is applied to the odd-numbered *GSPNs* 1, 3, 5, and so on. In this way, the addresses are rearranged in every other grouped superpage as shown in Fig. 11.

Compared to LIAM, BFAM does not require extra memory space. In [1], it is inherently assumed that an image horizontal size is aligned with a superpage size. This implies that a significant amount of unused memory space can be required in [1]. By using a grouped-superpage number, the presented technique can be utilized for arbitrary image sizes without an additional memory space. In this work, image processing applications (related to image display) are mainly considered. The presented method can be used in applications (for example, 2D data processing) that have the regular address patterns with certain strides. The usage of the presented method in other applications can be further investigated. This investigation is left for future work.

F. ADAPTING TO MEMORY MAPS

In the previous sections, it is assumed that an underlying memory map is *RBC* format for the sake of simplicity. In case the address starts with a row (for example, *RBC*, *RCBC*), these memory maps are called *R-star* formats. The proposed method can be applied to various *R-star* formats by changing the bank bit positions. However, the assumed (*RBC*) map and an underlying memory map may be different. In this case, the bank interleaving pattern highly relies on the underlying memory map. The generated addresses may not be efficient as intended. The approach to handle this issue is to convert the *RBC* format into the underlying memory map format. This is shown in Fig. 12. The format is modified with simple bit operations. In this way, the presented design can be adapted to various memory maps. Then memory banks are possibly accessed as intended by a master.

G. HARDWARE DESIGN

Fig. 13 shows the adaptive address map hardware. The hardware implements the operation depicted in Fig. 9. First, the condition checker selects LIAM or BFAM. Second, taking an image coordinate as an input, the hardware calculates a LIAM address. Third, if BFAM is selected, the hardware calculates a BFAM address. The shaded rectangle in Fig. 13 shows the additional logic to implement the proposed method.

1) CONDITION CHECKER

Eq. (5) is implemented in this logic. It is possible that software (or device driver) checks Eq. (5), selects an address map, and configures the hardware. In this work, this is implemented in hardware as follows. First, the hardware calculates the remainder, *ImgHB % SPS*. Considering *SPS* is 2^n bytes, the remainder is *ImgHB*[n-1:0]. Second, the hardware checks whether it belongs to +25% or -25% range from an integer. This can be efficiently implemented by taking two bits and checking the following:

$$ImgHB[\text{n-1:n-2}] \text{ is } 00_2 \text{ or } 11_2 \tag{9}$$

If this value is 00_2, T belongs to +25% range. If this value is 11_2, T belongs to -25% range. In case Eq. (9) is true, the system selects BFAM. Otherwise, the system selects LIAM.

2) BFAM ADDRESS CALCULATION

This hardware logic calculates a BFAM address as shown in Fig. 13. The bank-flipping algorithm shown in Fig. 10 is implemented in this logic. The hardware calculates the remainder, *GSPN % 2*. If the remainder is 1, *GSPN* is an odd number. Then the bank flipping is conducted to obtain a new bank number. In this logic, $\frac{SPN}{k}$ requires certain cost because it conducts a division. The $\frac{1}{k}$ is a constant when an image size is determined and the value does not change during run time. Software can configure $\frac{1}{k}$ using a control register. Then hardware can conduct an integer multiplication. In this work, a divider is used to conservatively measure the hardware area. Other logics are implemented in simple gates and multiplexers.

TABLE 3. Hardware cost. The number of slice look-up tables (LUTs) in Xilinx Virtex-7 xc7vx980t.

Address map	LIAM	BFAM	Δ
Number of LUTs	448	1585	1137

3) OVERHEADS

The presented adaptivity feature is added in the address map hardware component. Accordingly, the hardware complexity of the address map itself increases. Two overheads are evaluated as follows. First, to measure the hardware area overheads, the address map is implemented in Verilog, synthesized, placed, and routed in Xilinx FPGA device. Table 3 shows the results. BFAM additionally requires 1137 slice look-up tables (LUTs). BFAM requires 3.5**x** more area than LIAM. The targeted device contains 612000 LUTs in total. Though BFAM has some area overheads, the additional logic occupies less than 1% of the targeted FPGA device. Accordingly, the area overhead is considered to be moderate. Second, the latency overhead is evaluated. Based on the logic implementation, the presented design has one more cycle latency compared to the conventional design. It is observed that the additional latency is insignificant because the design operates in the pipelined way. In our experiment, the latency overhead is less than 1%.

An image size and a memory configuration significantly affect the degree of bank interleaving in a DRAM. Accordingly, the degree of adaptivity can vary with image sizes and memory configurations. The presented address map is configured using the generic criteria shown in Eq. (5). This criteria is applied to arbitrary image sizes and superpage sizes. The hardware cost (shown in Table 3) to implement the criteria does not vary with image sizes and superpage sizes. Subsequently, the variable degree of adaptivity does not affect the hardware cost (or other design parameters) of the address map. It is noted that the presented approach does not require the modification of other system components. The design parameters of other components are not affected by the presented adaptivity feature.

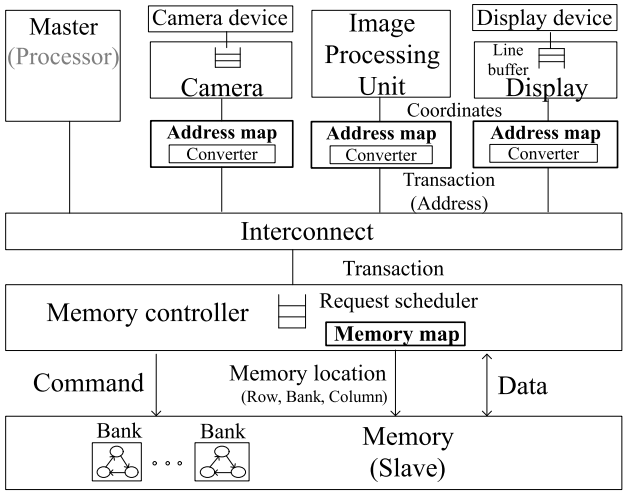


FIGURE 14. System organization.

TABLE 4. System configuration.

Components	Item	Configuration
Interconnect	Arbitration	Round-robin
	Multiple outstanding	Max. 16
Memory controller	Memory map (optional)	*RBC*, *BRC*, *Permuted RBC*, *Permuted RCBC*
	Request queue	16 entries
Memory (DRAM)	Model	DDR3-800
	Timing	t_{CL}-t_{RCD}-t_{RP} = 5 - 5 - 5
	Interface	One channel single chip
	Banks	4

V. EXPERIMENTAL RESULTS

The experimented system is shown in Fig. 14. The system is configured as shown in Table 4. The interface of a component operates with AXI bus protocol [21]. Traffic generators are implemented to represent the memory access behavior of masters. A single datum is 128 bits wide. A pixel is represented in 4-byte sized RGB format. A transaction contains 64 bytes of data. A physical address is 32 bits wide. The double data rate 3 (DDR3) timing parameters in [22] are used. To conduct the conservative performance experiment, the device that has the least t_{CL}, t_{RCD}, and t_{RP} values is used. To evaluate the performance of the presented address map, the cycle-based transaction-level performance model is implemented in C++. The model is integrated in the simulation environment of [20]. Execution cycles are significantly affected by memory efficiency. Memory efficiency is highly determined by address patterns and scheduling policies of a memory controller. To improve memory efficiency, high priority is typically given to a request that accesses an activated row in the requested bank [23]. The memory controller model is implemented to increase the memory efficiency [20].

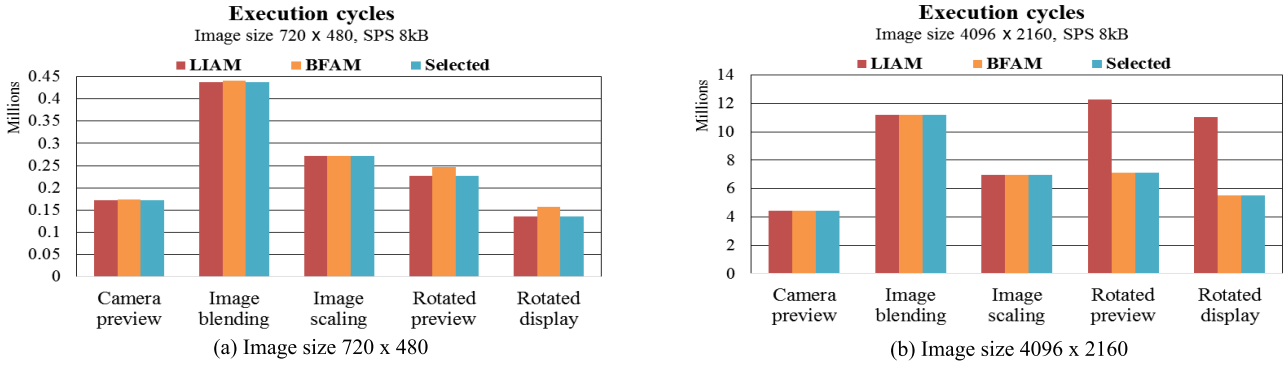


FIGURE 15. Execution cycles for different image sizes. A superpage size is 8 kB. A memory map is *RBC*.

TABLE 5. Workload scenarios.

Workloads	Component	Operation	Stride	Pattern
Camera preview	Camera	Write	64 B	Horizontal
	Display	Read	64 B	
Image blending	Scaler	Read, Read, Write	64 B	Horizontal
Image scaling x1.5	Camera	Write	64 B	Horizontal
	Scaler	Read, Write	64 B	
	Display	Read	64 B	
Rotated preview	Camera	Write	*ImgHB*	Vertical + Horizontal
	Display	Read	64 B	
Rotated display	Display	Read	*ImgHB*	Vertical

The workloads for the simulation are shown in Table 5. The *rotated preview* scenario is described in Section III-C. The *image scaler* resizes an image. The *image blender* combines two images to generate a composite image. It is noted that a scenario is associated with multiple processes that concurrently operate in multiple masters. Accordingly, transactions from multiple masters are mixed in the shared main memory. To evaluate the performance of the proposed approach, execution cycles to run a single frame are measured. Three parameters (an image size, a superpage size, a memory map) are explored.

First, different image sizes for a given superpage size and a memory map are experimented. Fig. 15 shows the results for the superpage size 8 kB and the *RBC* memory map. When an access pattern is (raster-scan) linear, the stride is the transaction size 64 B. In this case, T is 0.008 ($=\frac{64}{8192}$) and LIAM is selected. As shown in Fig. 15, LIAM and BFAM provide similar performances. This means BFAM does not degrade performance because bank interleaving is conducted in a superpage. When an access pattern is vertical, the stride is *ImgHB*. As shown in Fig. 15, the performances of LIAM and BFAM vary with image sizes. When an image size is 720 **x** 480, T is 0.35 ($=\frac{720\text{x}4}{8192}$) and LIAM is selected. In this case, LIAM performs up to 14% better than BFAM. When an image size is 4096 **x** 2160, T is 2 ($=\frac{4096\text{x}4}{8192}$) and BFAM is selected. In this case, BFAM performs up to 50% better than LIAM. The results indicate that the proposed design can improve performance by adapting to an image size.

Second, different superpage sizes for a given image size and a memory map are experimented. Fig. 16 shows the results for the image size 1080 **x** 1920 and the *RBC* memory map. When an access pattern is linear, the performance difference between LIAM and BFAM is insignificant similar to the previous experiment. When an access pattern is vertical, the performances of LIAM and BFAM vary with superpage sizes. When a superpage size is 4 kB, T is 1.06 and BFAM is selected. In this case, BFAM performs up to 57% better. When a superpage size is 8 kB, T is 0.53 and LIAM is selected. In this case, LIAM performs up to 13% better. The results indicate that the proposed design can improve performance by adapting to a superpage size.

Third, different memory maps for a given image size and a superpage size are experimented. In [1], the tiled map operates in the superpage level. For fair comparison, a tile in [1] is sized by 16**x**1 because it operates as a linear map. Fig. 17 shows the results. The performances of [1] and our design are the same when k is 1. Our design can perform better than [1] when k is greater than 1. In Fig. 17, our design performs on average 18% better than [1].

In the reference designs of [2] and [3], LIAM operates over the permuted memory maps. In the experimented image sizes, *ImgHB* vary from 2880 B (for 720 **x** 480) to 16384 B (for 4096 **x** 2160). When an access pattern is vertical, bit positions 13 and 14 are hot-spots in that these bits frequently change. Therefore, these bits are chosen for the permutation to increase bank interleaving in [2] and [3]. In Fig. 17, when an access pattern is linear, the performance difference between [2] and ours is insignificant. Our design performs up to 37% better than [3]. This is because memory accesses from multiple masters incur interferences and frequent bank conflicts in [3]. In a certain configuration, the traffic pattern is well matched with [2] and [3]. In Fig. 17(b), when an access pattern is vertical, bit positions 13 and 14 are well utilized in [2] and [3]. In this case, our performance is comparable to [2] and [3]. However, in Fig. 17(a), these bit positions in [2] and [3] are not well utilized because of the superpage size. In Fig. 17, our design performs on average 10% better than [2] and 35% better than [3].

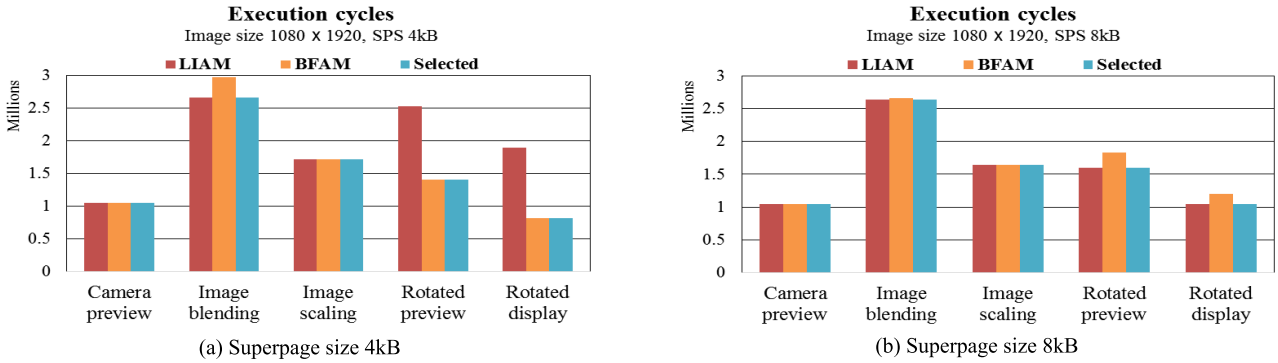


FIGURE 16. Execution cycles for different superpage sizes. An image size is 1080 x 1920. A memory map is *RBC*.

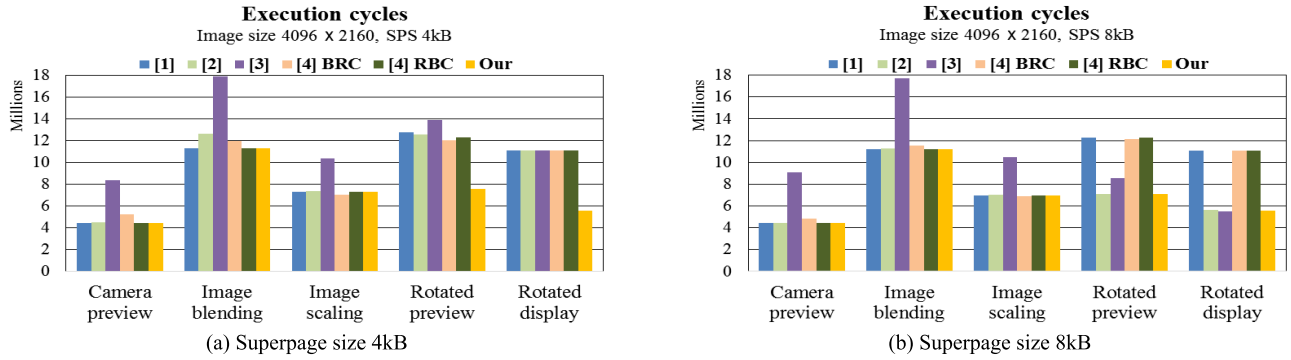


FIGURE 17. Execution cycles for various memory maps. An image size is 4096 x 2160.

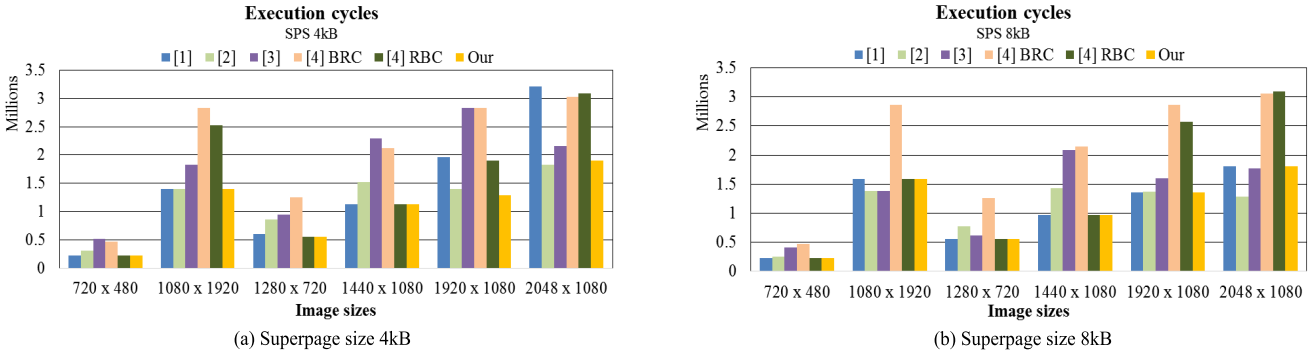


FIGURE 18. Execution cycles of the *rotated preview* scenario for various image sizes and memory maps.

The design in [4] can be configured in *BRC* or *RBC*. Accordingly, LIAM operates over *BRC* or *RBC* map. When an access pattern is linear, our performance is comparable to [4]. When an access pattern is not linear, our design performs significantly (up to 42%) better than [4]. When [4] is configured in *BRC*, only a single bank is utilized. When [4] is configured in *RBC*, it operates in the same way as the LIAM shown in Fig. 2. In both cases, significant bank conflicts occur. In Fig. 17, our design performs on average 19% better than [4]. The results indicate that the proposed design can improve performance by adapting to a memory map.

Fourth, various image sizes and memory maps for a given superpage size are experimented. Fig. 18 shows the results of the *rotated preview* scenario for various image sizes. Similar to the previous experiment, the performance is significantly affected by a memory map. It is noted that a performance is affected by traffic interferences from multiple masters. In a particular configuration, the address patterns match well with [2] and [3]. In these cases, the memory maps in [2] and [3] provide well interleaved bank access patterns. When an image size is 2048 x 1080 and a superpage size is 8 kB, our design performs 39% worse than [2]. When an image size is 1080 x 1920 and a superpage size is 8 kB, our design performs 15% worse than [3]. In overall, however, our design performs well by adapting to a given memory map. In Fig. 18, our design performs on average 7%, 9%, 29%, 33% better

than [1]–[4], respectively. The results indicate that the proposed design can improve performance by adapting to both image sizes and memory maps.

TABLE 6. Allocated memory size.

Image sizes	Allocated memory size (Mega-bytes)		Δ
	[1]	Our (BFAM)	(%)
1080 x 1920	15.0	7.9	47.3
1920 x 1080	8.4	7.9	6.0
2048 x 1080	8.4	8.4	0
4096 x 2160	33.8	33.8	0

Fifth, the allocated memory size is measured to compare the memory footprint with [1]. In [1], when the bank flipping is conducted, an image horizontal size is assumed to be aligned with a superpage size. In BFAM, there is no additionally required memory space compared to LIAM. Table 6 shows the results when BFAM is selected. Δ indicates the improvement of our design in percentage compared to [1]. Our design occupies up to 47% less memory space than [1].

VI. CONCLUSION

The adaptive bank-interleaved linear address map to reduce bank conflicts is presented. The presented address map is rearranged in a master to explicitly control bank access patterns. To achieve the adaptivity, the configurable scheme that selects LIAM or BFAM is devised. A main advantage is the increased performance. The experimental results indicate that the proposed method performs on average 21% better than the reference designs. Additionally, the presented BFAM occupies the same memory space as LIAM. No extra memory space is required. The presented scheme adapts to an application when the application is invoked. No off-line trace analysis is required to configure the address map. A disadvantage is the increased complexity of the address map. BFAM requires more area than LIAM. The area overhead can be reasonable in that the address map occupies a small portion of the entire device.

REFERENCES

[1] J. Y. Hur, S. W. Rhim, and B. H. Lee, "Method and apparatus for performing adaptive memory bank addressing," U.S. Patent 9 390 007 B2, Jul. 12, 2016.

[2] Z. Zhang, Z. Zhu, and X. Zhang, "Breaking address mapping symmetry at multi-levels of memory hierarchy to reduce DRAM row-buffer conflicts," *J. Instruct.-Level Parallelism*, vol. 3, pp. 1–35, Oct. 2001.

[3] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Porto Alegre, Brazil, Dec. 2011, pp. 24–35.

[4] M. S. Smith, "Multiple class memory systems," U.S. Patent 8 930 647, Jan. 6, 2015.

[5] J. Shao and B. T. Davis, "The bit-reversal SDRAM address mapping," in *Proc. Workshop Softw. Compil. Embedded Syst.*, Dallas, TX, USA, Sep./Oct. 2005, pp. 62–71.

[6] W. Jang, "Screen orientation aware DRAM architecture for mobile video and graphic applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 12, pp. 3527–3538, Dec. 2018.

[7] M. Jung, D. M. Mathew, C. Weis, N. Wehn, I. Heinrich, M. V. Natale, and S. O. Krumke, "Congen: An application specific dram memory controller generator," in *Proc. 2nd Int. Symp. Memory Syst.*, Alexandria, VA, USA, Oct. 2016, pp. 257–267.

[8] M. Jung, C. Weis, and N. Wehn, "DRAMSys: A flexible dram subsystem design space exploration framework," *IPSJ Trans. Syst. LSI Des. Methodol.*, vol. 8, pp. 63–74, Aug. 2015.

[9] M. Ghasempour, J. D. Garside, A. Jaleel, and M. Luján, "DReAM: Dynamic re-arrangement of address mapping to improve the performance of DRAMs," in *Proc. 2nd Int. Symp. Memory Syst.*, Alexandria, VA, USA, Oct. 2016, pp. 362–373.

[10] M. K. Jeong, D. H. Yoon, D. Sunwoo, M. Sullivan, I. Lee, and M. Erez, "Balancing DRAM locality and parallelism in shared memory CMP systems," in *Proc. IEEE 18th Int. Symp. High Perform. Comput. Archit. (HPCA)*, New Orleans, LA, USA, Feb. 2012, pp. 1–12.

[11] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Berlin, Germany, Apr. 2014, pp. 155–166.

[12] T. A. Alawneh and A. Elhossini, "A prefetch-aware memory system for data access patterns in multimedia applications," in *Proc. 15th ACM Int. Conf. Comput. Frontiers*, Ischia, Italy, May 2018, pp. 78–87.

[13] S. Khan, M. Rashid, and F. Javaid, "A high performance processor architecture for multimedia applications," *Comput. Electr. Eng.*, vol. 66, no. C, pp. 14–29, Feb. 2018.

[14] K.-F. Xia, B. Wu, T. Xiong, and T.-C. Ye, "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 6, pp. 1919–1929, Jun. 2017.

[15] V. Schwambach, S. Cleyet-Merle, A. Issard, and S. Mancini, "Image tiling for embedded applications with non-linear constraints," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Krakow, Poland, Sep. 2015, pp. 1–8.

[16] S. Hettiaratchi and P. Y. K. Cheung, "A novel implementation of tile-based address mapping," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Paris, France, Feb. 2004, pp. 306–310.

[17] B. Wang, Y. Fukazawa, T. Kondo, and T. Sasaki, "Tile/line access cache memory based on a multi-level Z-order tiling data layout," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 9, May 2018, Art. no. e4375.

[18] H.-K. Chang and Y.-L. Lin, "Array allocation taking into account SDRAM characteristics," in *Proc. Asia South Pacific Design Automat. Conf.*, Yokohama, Japan, Jan. 2000, pp. 497–502.

[19] G.-J. van den Braak, J. Gómez-Luna, J. M. González-Linares, H. Corporaal, and N. Guil, "Configurable XOR hash functions for banked scratchpad memories in GPUs," *IEEE Trans. Comput.*, vol. 65, no. 7, pp. 2045–2058, Jul. 2016.

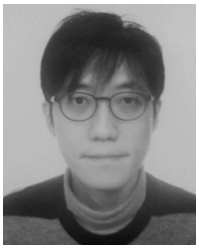
[20] J. Y. Hur, "Contiguity representation in page table for memory management units," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 1, pp. 147–158, Jan. 2019.

[21] ARM. *ARM Architecture Reference Manual, ARMv7-A Edition*. Accessed: May 20, 2014. [Online]. Available: http://www.arm.com

[22] Micron. *1Gb DDR3-800 SDRAM Specification*. Accessed: Nov. 18, 2018. [Online]. Available: http://www.micron.com

[23] B. Akesson and K. G. W. Goossens, *Memory Controllers for Real-Time Embedded Systems: Predictable and Composable Real-Time Systems*. New York, NY, USA: Springer-Verlag, 2012, ch. 3.

JAE YOUNG HUR received the B.S. degree in electronics engineering from Cheju National University, Cheju, South Korea, in 1995, the M.S. degrees in electronics engineering from Sogang University, Seoul, South Korea, in 1998, and also from the Munich University of Technology, Munich, Germany, in 2002, and the Ph.D. degree in computer engineering from the Delft University of Technology, Delft, The Netherlands, in 2011. He was an Engineer, from 1999 to 2000 and a Senior Engineer, from 2008 to 2016 with the Semiconductor Division, Samsung Electronics Ltd., South Korea. He is currently a Researcher with Vietnamese-German University, Binh Duong, Vietnam. His research interests include embedded system architectures, VLSI design, and reconfigurable computing.



SANG WOO RHIM received the B.S. degree in electronics engineering from Hongik University, Seoul, South Korea, in 1998, and the M.S. degree in electrical engineering from the University of Florida, USA, in 2002. From 2002 to 2008, he was a Researcher with the Samsung Advanced Institute of Technology (SAIT), South Korea. From 2009 to 2013, he was a Senior Engineer with the Semiconductor Division, Samsung Electronics Ltd., South Korea. He is currently an Adjunct Researcher with Vietnamese-German University, Binh Duong, Vietnam. His research interests include embedded system design, memory system architectures, and networks-on-chip.

BEOM HAK LEE received the B.S. and M.S. degrees in electronics engineering from Hongik University, Seoul, South Korea, in 1997 and 2000, respectively. From 2001 to 2008, he was a Senior Researcher with the Samsung Advanced Institute of Technology (SAIT), South Korea. From 2009 to 2015, he was a Senior Engineer with the Semiconductor Division, Samsung Electronics Ltd., South Korea. He is currently an Adjunct Researcher with Vietnamese-German University, Binh Duong, Vietnam. His research interests include computer architectures and networks-on-chip.

WOOYOUNG JANG (S'08–M'11) received the B.E. degree in radio science and technology from Kyunghee University, South Korea, in 1998, the M.S. degree in electrical and computer engineering from Yonsei University, South Korea, in 2000, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, USA, in 2011.

From 2000 to 2013, he was with the System LSI Division, Samsung Electronics, as a Senior Engineer. He is currently an Associate Professor with the Department of Electronics and Electrical Engineering, Dankook University. He has published more than 37 papers in highly referred conferences and journals, and holds four U.S. patents. His current research interests include interconnection networks, computer architecture, low-power SoC design, and machine learning.

He has served in the Technical Program Committee for the IEEE International Conference on Computer-Aided Design (ICCAD), the Technical Program Committee of IEEE International Conference on Computer Design (ICCD), the International Symposium on VLSI Design, Automation and Test (VLSI-DAT), and the International Program Committee of International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics).