EURASIP Journal on Image and Video Processing

Novel data storage for H.264 motion compensation: system architecture and hardware implementation

EURASIP Journal on Image and Video Processing 2011, **2011**:21 doi:10.1186/1687-5281-2011-21

Elena Matei (Elena.Matei@intec.ugent.be)
Christophe van Praet (Christophe.VanPraet@intec.ugent.be)
Johan Bauwelinck (Johan.Bauwelinck@intec.UGent.be)
Paul Cautereels (Paul.Cautereels@alcatel-lucent.com)
Edith Gilon de Lumley (Edith.Gilon@alcatel-lucent.com)

ISSN	1687-5281
Article type	Research
Submission date	30 March 2011
Acceptance date	19 December 2011
Publication date	19 December 2011
Article URL	http://jivp.eurasipjournals.com/content/2011/1/21

Novel data storage for H.264 motion compensation: system architecture and hardware implementation

Elena Matei*[1], Christophe van Praet[1], Johan Bauwelinck[1],

Paul Cautereels[2] and Edith Gilon de Lumley[2]

[1]Intec_design IMEC Laboratory, Ghent University,

Sint Pietersnieuwstraat 41, 9000-Ghent, Belgium

[2]Alcatel Lucent-Bell, Copernicuslaan 50,

Antwerpen, Belgium

*Corresponding author: Elena.Matei@intec.ugent.be

Email addresses:

CvP: Christophe.VanPraet@intec.ugent.be

JB: Johan.Bauwelinck@intec.UGent.be

PC: Paul.Cautereels@alcatel-lucent.com

EGdL: Edith.Gilon@alcatel-lucent.com

Abstract

Quarter-pel (q-pel) motion compensation (MC) is one of the features of H.264/AVC that aids in attaining a much better compression factor than what was possible in preceding standards. The better performance however also brings higher requirements for computational complexity and memory access. This article describes a novel data storage and the associated addressing scheme, together with the system architecture and FPGA implementation of H.264 q-pel MC. The proposed architecture is not only suitable for any H.264 standard block size, but also for streams with different image sizes and frame rates. The hardware implementation of a stand alone H.264 q-pel MC on FPGA has shown speeds between 95.9 fps for HD1080p frames, 229 fps for HD 720p and between 2502 and 12623 fps for CIF and QCIF formats.

Keywords: motion compensation; quarter-pel; address; memory; H.264 decoder; FPGA.

1 Introduction

H.264.AVC [1] is one of the latest video coding standards which can save up to 45% of a stream's bit-rate compared with the previous standards. The coding efficiency is mainly the result of two new features: variable block-size MC and quarter-pel (q-pel) interpolation accuracy. More precisely, the H.264 standard proposes several partition sizes for each macroblock (MB is a group of 16×16 pixels). In the inter-prediction approach, each partitioned block takes as estimation a block in the reference frame that is positioned at integer, half or quarter pixel location. This fine granularity provides better estimations and better residual compression. Unfortunately, the better performance brings also higher requirements with respect to computational complexity and memory access. The H.264 decoder is about four times more complex than the MPEG-2 decoder and about two times more complex than the MPEG-4 Visual Simple Profile decoder [2]. These higher requirements, together with the huge amount of video data that have to be processed for an HDTV stream, make the implementation of a 1080p real-time MC in a H.264 decoder a challenging task.

In a H.264 decoder, there are several modules that require intensive use of the off-chip memory. Wang [2] and Yoon [3] concluded that MC requires 75% of all memory access in a H.264 decoder, in contrast with only 10% required for storing the frames. This high memory access ratio of the MC module demands for highly optimized memory accesses to improve the total performance of the decoder.

The tree structured MC assumes the use of various block sizes. In H.264 4:2:0, the 4×4 luma block size is considered to provide the best results with respect to image quality, but it is also the most demanding with respect to data accesses for q-pel motion vectors (MV) [2]. The proposed implementation focuses on this 4×4 block size scenario in MC, which is using the highest amount of data and is computationally the most intensive. This is done to prove the efficiency of the proposed method. However, the presented addressing scheme and implementation are not limited to the 4×4 block, but can be used on any H. 264 standard block size.

A linear data mapping approach is a natural raster scan order image representation in the memory. In this representation, all neighboring pixels in an image remain neighbors in the memory also. This is the typical way of saving the reference frame on an external memory, also used in [3–5].

At the moment, the DDR3 memories are preferred for such implementations thanks to their fast memory access, high bandwidth, relatively large storage capability, and affordable price. The major bottlenecks of external SDRAM memory in a H.264 decoder are numerous accesses to implement the motion compensation (MC) and accesses to multiple memory rows to reach columns of pixels. This last bottleneck, known as cross-row memory access, is a problem for both access time and power utilization. The row precharge and row opening delay for DDR3 SRDAM are memory and clock frequency dependent. For a 64-bit 7-7-7 memory it takes about three times more time to read a data from an unopened row than from an already opened one [6]. This, together with the DDR3 optimized burst access are the facts that drove us to look into a more efficient memory access for MC.

The already mentioned problems motivate us to propose a vectorized memory storage scheme and the associated addressing scheme, which were both designed for the specific needs of the q-pel MC algorithm. The proposed method may be used at both the Encoder and the Decoder sides for performing q-pel H.264 MC. The most demanding scenario for MC uses the 4×4 block size data and assumes an unpredictable access pattern. This is why using only a caching mechanism as shown in [3] or [4] is not very efficient because it does not minimize the number of external memory row openings. A caching mechanism is compatible with the proposed data organization and addressing scheme. The proposed data vectorization and the specific addressing scheme presented in this article not only provide a faster access to all the requested data, hide the overhead produced by the 6-tap FIR filter, but also minimize the number of addresses on the address bus and the number of row precharges and row activations. The proposed system is able to provide the required data for any q-pel interpolation case with only one or two row opening penalties and it is suitable for streams with different image sizes and frame rate. This implementation is optimized for a 64-bit wide memory bus SDRAM, but it can easily be adapted for other types of memories and supports different image dimensions. Further on in this article the proposed method is also named the vectorized method.

The practical q-pel MC implementation was done in hardware using VHDL for design, simulation, and verification. Further on, this implementation is independent of the platform, being able to map to any available FPGA. For the proof of concept, a Stratix IV EP4SGX230KF40C2 has been used. A stand alone H.264 q-pel MC block has achieved speeds between 95.9 fps for HD1080p frames, 229 fps for HD 720p and between 2502 and 1262 fps for CIF and QCIF formats. These results are obtained using a single instance of the MC block, but multiple instances are possible if the resources allow it.

The rest of this article has the following structure: Section 2 presents the MC algorithm for H.264. In the next section, the memory addressing in SDRAM is briefly presented. Section 4 reveals the problems that a standard decoder faces with regard to its most demanding algorithm. Section 5 comes with the proposed solution for the previously presented problems and describes data mapping, reorganization, and the associated address mapping and read patterns. The memory address generation is also presented in this section. In Section 6, the system's architecture and hardware implementations are described. Next, in Section 7, the method results and a discussion focused on comparing the proposed approach to the existing work are presented. The conclusions section summarizes the conducted research.

2 MC in H.264

The presented implementation handles 4×4 luma and 2×2 chroma blocks for 4:2:0 Baseline Profile H.264 YUV streams. The efficiency of our method will be proved for this case, however, the proposed method is not limited to this specific block dimension but can be used on any H.264 standard block size.

Each partition in an inter-coded macroblock is predicted from an area of the reference picture. The MV between the two areas has sub-pixel resolution. The luma and chroma samples at sub-pixel positions do not exist in the reference picture and so it is necessary to create them using interpolation from nearby image samples.

For estimating the fractional luma samples, H.264 adopts a two-step interpolation algorithm. The first step is to estimate the half samples labeled as b, h, m, s, and j in Figure 1. All pixels labeled with capital letters, from A to

U, represent integer position reference pixels. The second step is to estimate quarter samples labeled as a, c, d, e, f, g, i, k, n, p, q, and r, based on the half sample values.

H.264 employs a 6-tap FIR filter and a bilinear filter for the first and the second steps, respectively [1].

In H.264, the horizontal or vertical half samples are calculated by applying a 6-tap filter with the following coefficients $(1, -5, 20, 20, -5, 1)/32$ on six adjacent integer samples as shown in Equation 1. In a similar way, half-pel positions labeled aa, bb, cc, dd, ee, ff, gg, hh are calculated. Half samples labeled as j are calculated by applying the 6-tap filter to the closest previously calculated half sample positions in either horizontal or vertical direction.

$$b = ((E - 5F + 20G + 20H - 5I + J) + 16)/32 \tag{1}$$

For estimating q-pel positions, first all the half-pel positions have to be computed. Then, quarter samples at position e, g, p, and r are generated by averaging the two nearest half samples, as shown in Equations 2 and 3.

$$e = (b + h + 1)/2 \tag{2}$$

Samples at positions g, p, and r are generated in the same way. Quarter samples at positions a, c, d, f, i, k, n, and q are generated by averaging the two nearest integer or half positions:

$$a = (G + b + 1)/2 \tag{3}$$

Samples at positions c, d, f, i, k, n, and q are generated in the same way.

For calculating the chroma samples, an 8-pel bilinear interpolation is executed on four of the nearest pixels.

3 Memory addressing in SDRAM

DDR3 SDRAM memories combine the highest data rate with improved latencies. A key characteristic of SDRAM memories is their organization in rows, columns, and banks. The access to several columns of the same row is very efficient, as it is the access on different banks. The access of different rows in the same bank however takes more time, as this new row must first be precharged and opened. This precharge can happen in advance if the row is located in another bank but it cannot be hidden when the new row is in the same bank. For an efficient data access, the information requested at a read or given at a write command should have a certain locality to prevent high delays because of bank opening, row precharge, and row activation. The access of several consecutive locations on the same row is also known as burst-oriented accesses.

Row precharge and row opening delay for DDR3 SDRAM are memory and clock frequency dependent. For a 64-bit 7-7-7 memory, the delay because of a row opening and precharging is three times higher than that of a column access. One feature of the burst accesses is that the subsequent column access time for consecutive locations is hidden and the only case where this access time is influencing the data retrieval delay is for the first column from the burst.

4 Problem definition

Many application and video providers migrate toward H.264 for making use of the high quality and lower datarate that it offers. The difficulty to implement real-time 1080p H.264 systems relies mainly in the fact that q-pel inter-prediction is very memory and computing intensive.

Since the luma 4×4 block represents the most demanding case with respect to memory accesses [3] and computational intensity for q-pel MC, the focus will be put on this type of block and its associated operations to prove the efficiency of the proposed method for a standard H.264 decoder.

The address to which the MV points in the reference image may be an integer position, a half-pel, or a q-pel displacement. H.264 luma MC has several steps to fulfill: first a relevant block of reference data is retrieved from the SDRAM memory, second the 6-tap FIR filtering either horizontal or vertical and third a linear interpolation takes place. In the first phase, the following algorithm is executed: if the MV set points to integer positions, retrieve one 4×4 block; if the MV set points to a half-pel position, retrieve either a 4×9 (rows $\times$ columns) block for horizontal displacement, or a 9×4 for vertical, or a 9×9 for both half-middle point and q-pel positions [5].

The main problems that exist when sub-pixel MC is implemented are because of several causes:

- the 6-tap FIR filter increases the memory bandwidth because of the overhead of extra pixel fetch beyond the 4×4 block;

- in the linear address translation approach there are minimum four and maximum nine row opening actions that are both time and energy consuming when working with off-chip memories;
- because of unpredictable access pattern in the reference image there is a high overhead when retrieving useful data;
- increased number of read commands on the address bus toward the memory.

The vectorized data storage scheme is further described in next section.

5 Vectorized data storage

The chosen DDR3 memory is a 64-bit memory location memory and consists of 8 banks. Since the DDR3 memory access is optimized for bursts, let us take the example of a burst length (BL) of 2. When such a read command is issued, the memory responds with a $\times 4$ (for bus clock multiplier) double data rate $\times 64$ bits for a given clock frequency. This results in returning 8 consecutive memory locations, which represent one line of data from 16 consecutive 4×4 pixel blocks.

Considering the DDR3 64-bit memory location, for a linear data mapping one could group 8 values together on one location and use V/8 number of columns from the physical memory. The linear data mapping without bank optimization is shown in Figure 2.

To calculate any of the interpolation steps needed, the maximum reference block is 9×9 pixels. So, for acquiring the reference block for a q-pel interpolation using a linear address mapping the system will issue: nine read commands for data that is located on nine different rows. For $BL = 1$, the memory will return 32 pixels per row from which only nine are useful. This results in a large data overhead and a considerable time penalty. The linear address mapping approach is presented in Figure 2 without any optimization and in Figure 3 with a bank optimization technique. With this optimization, every line of pixels is saved in a different bank. The linear address mapping is not optimal with respect to physical memory accesses, suffers from a large data overhead and does not tackle the problems stated in the previous section.

In this article, first a different image mapping in the memory is proposed. This different image mapping also demands for a different addressing scheme. Both are described in more depth in the following sections.

5.1 Data mapping and reorganization

As shown in Figures 4 and 5, a different manner is used to store the data in memory. This approach regroups the pixels for the filtering phase to reduce the off-chip memory accesses and the number of read commands on the memory address bus. Pixels that are statistically more likely to be requested together are stored on the same row. Each 4×4 luma block is vectorized as a one-dimensional structure and saved on two consecutive columns on the memory. This allows using one row activation for accessing all the information from a given 4×4 reference block.

The blocks' order is kept, so consecutive blocks in the image plane will remain consecutive in the memory both horizontally and vertically, as shown in Figures 4 and 5. Just the internal arrangement of the 4×4 blocks is changed. Keeping in mind how the physical memory works, a better result with respect to the row access time is obtained if the row 0 of image sub-blocks is saved in memory on row 0 from bank 0, row 1 of image on row 0 from bank 1, and so on, as shown in Figure 6. This is called the bank optimization approach and is similar with the presented organization from Figure 4 with the difference that the consecutive rows of sub-macroblocks will be saved in consecutive banks.

Since the MVs point from the current block address to any other block in the reference frame, the presented data reorganization has specific requirements for the addressing method and start address pixel which will be further explained in the next section.

5.2 Address mapping and read patterns

The presented data mapping and reorganization creates a different relationship between neighboring blocks. The following cases are explaining what the changes are to address the needed data and how the addresses are generated.

Case 1.0—Integer: Suppose that for the current block the corresponding set of MVs has integer values. That means that for this block there will be no interpolation and the output of the MC operation will be a block similar to the one that is retrieved from the reference frame. The addresses where this block is located are given by composing the current address with the displacement given by the MV on both directions. This can for example coincide with the start address of Block 5 (see Figure 7). In the same image, the memory read pattern is shown. It can be observed that only one read request is needed for retrieving a full block of 4×4 luma reference. This is however a particular case and does not represent the majority of the possible types of requests.

Case 1.1—half-pel horizontal: Taking this assumption one step further, assume that MC has to perform a horizontal half-pel interpolation and thus a 4×9 block is retrieved. Using linear address mapping (figured on the left side of Figure 7a), nine consecutive pixels from four rows need to be fetched from the memory. Based on the new data organization, it is easily observable that only one row of the SDRAM memory needs to be accessed to get all the requested data. The data are requested from the off-chip memory issuing a single read request with $BL = 2$. The data retrieved from the SDRAM are then Blocks 4, 5, 6, and 7.

Case 1.2—half-pel vertical: Similar to the previous case, for a vertical displacement a block of 9×4 is requested. Using the proposed new reordering there are three different rows from different banks are accessed to provide the MC with the required input block. In this case, Blocks 1, 5, and 9 need to be totally retrieved from the memory and further rearranged. The 6-tap FIR filter receives within 2 clock cycles (after a memory specific delay) all the data needed for calculating half-pel interpolation on all 16 pixel positions in the same time (this is the case also for the half-pel horizontal).

Case 1.3—half-pel middle or q-pel: A more complex step is imposed for these cases and 9×9 block is required from the memory. Although a more complex block is requested the read commands that will be issued are the same as in the previous case, only three rows from three different banks are accessed, issuing only one row activation delay when using the vectorized method. Similar, all the data is available for the FIR filter to start working.

Case 16.0—integer with different start point: The MVs are not necessarily multiple of 4. They can point to any start position for the reference block. Let us consider the case where the reference address is located on the last position of Block 5 (see Figure 7a). This case is similar to the previous ones, but more complex for the memory addressing scheme. For getting the necessary block, two rows need to be opened from consecutive banks, as shown Figure 7b.

With the proposed addressing scheme the method has a high degree of generality and is able to serve any quarter-pixel interpolation request by only opening one or maximum three consecutive rows, as shown in Table 1. When using the data spreading over different banks for any case of interpolation only one row opening penalty is associated with the data retrieval, the rest being hidden. The address generation system becomes intuitive when looking at the proposed data organization and is described in the following section.

5.3 Memory address generation

The reference image is saved into memory keeping the same order. Consecutive blocks in the image will be consecutive in the memory both horizontally and vertically when using the vectorization method. When adding the bank optimization, consecutive rows of vectorized blocks will be written in consecutive banks.

It would be of little interest if the addressing scheme could only serve frames of a given dimension. The proposed approach is designed to overcome this issue and offers the flexibility of computing MC on any image dimension up to full HD on the chosen memory. Once again let us take the worst case scenario to explain how the addressing scheme works.

The standard H.264 imposes that the image is organized in uniform blocks of 16×16 pixels called MB and further down to 4×4 sub-blocks. Taking a HD image of 1920 pixels, there are 120×68 MBs that are composed from 480×272 sub-blocks that have to be saved in the memory. The address mapping is based on this partitioning scheme.

Going one step further, a parallel address mapping between image space and memory space is done. In image space, every pixel is independent and can be addressed individually. As already explained, this is not optimal for a physical memory where the locations are 64 bit. The use of a DDR3 memory not only offers a high throughput, but also imposes some specific rules for addressing. One memory location may be addressed given a certain row-bank-column address. For the column address, the last significant 2 bits must be discarded when sending the address to the memory controller and interface block. This means that the addressing scheme will point to the column addresses multiple of 4 and that all the data from that location and the next three locations are available in one clock cycle for one read command. This is where the addressable columns of DDR3 memory are marked by arrows on Figure 7b.

For the given image, the total number of occupied rows in the memory will be equal to the $number\_sub\_blocks \div 4$ and the number of columns will be $number\_sub\_blocks \times 2$ on horizontal axis because one vectorized block occupies two physical memory locations. The chosen DDR3 memory has eight banks available. When saving consecutive rows of vectorized blocks on consecutive banks the least significant 3 bits of the row address represent the bank address. Each bank contains 2^{13} row addresses and 2^{10} column addresses, so it can accommodate images of maximum 128 MBs width using the same scheme [6].

Equations (4) and (5) show how the physical memory locations can be addressed, starting from the image space arrangement. The proposed addressing scheme treats MBs and sub-blocks individually and allocates separate address bit ranges for them. For the MB address, 7 bits are sufficient both horizontally (68 MBs $\times 2$ memory locations column address) and vertically (120 MBs $\div 8$ banks row address). The $\text{Sub\_block}_{\text{Addr}}$ and $\text{pixel}_{\text{Addr}}$ are fields of 2 bits each, representing the number of 4×4 blocks in a MB and the number of pixels in a 4×4 block along the two dimensions.

Always, the address vector is padded with $'0'$ values on the most significant bit locations for the case where the image is saved starting with row 0 in the memory, or any other displacement can be added to the given scheme for a different starting points.

$$\text{Row}_{\text{Addr}} = \text{MB}_{x\text{Addr}}\&\text{Sub\_block}_{x\text{Addr}}\&\text{pixel}_{x\text{Addr}} \tag{4}$$

$$\text{Col}_{\text{Addr}} = \text{MB}_{y\text{Addr}}\&\text{Sub\_block}_{y\text{Addr}}\&\text{pixel}_{y\text{Addr}} \tag{5}$$

Being that the proposed design vectorizes the pixels of a sub-block, this part of the address is only needed locally for selecting the data when retrieved from the memory. This takes us to Equations (6) and (7), where a division by 4 of the address starting from the image plane address is executed.

$$\text{Row}'_{\text{Addr}} = \text{Row}_{\text{Addr}} \div 4 \tag{6}$$

$$\text{Col}'_{\text{Addr}} = \text{Col}_{\text{Addr}} \div 4 \tag{7}$$

At this point it has been established how to generally address any sub-block from the image space. The memory row address when saving the reference frame on one bank is given by Equation (6). If the bank optimization is used, the bank address and row address are given by Equations (8) and (9), respectively.

$$\text{Row}''_{\text{Addr}} = \text{Row}'_{\text{Addr}} \div 8 \tag{8}$$

$$\text{Bank}_{\text{Addr}} = \text{Row}'_{\text{Addr}} \mod 8 \tag{9}$$

One sub-block is saved on two columns, thus a multiplication by a factor of 2 is required. This operation is shown in Equation (10). This is the full column address used for pointing to any column in the memory. The least significant bit is always zero when addressing one vectorized block.

$$\text{Col}''_{\text{Addr}} = \text{Col}'_{\text{Addr}} \times 2 \tag{10}$$

As shown in Figure 7 and explained earlier, the memory controller accepts column addresses in a format where the two least significant bits of the address are omitted. So the real column address that has to be put on the bus has the format shown in Equation (11)

$$\text{Col}'''_{\text{Addr}} = \text{Col}''_{\text{Addr}} \times 2 \tag{11}$$

Bit 0 of $\text{Col}''_{\text{Addr}}$ is zero always for addressing a start of a vectorized block. Bit 1 of $\text{Col}''_{\text{Addr}}$ together with the pixel address bits represent a select mechanism for further pointing to a pixel position in the retrieved data from the memory.

The same addressing system is kept for the 2×2 Chroma blocks. They are saved in an interlaced way in the memory as used also in [2]. The data are vectorized using the proposed method in a similar way. Being that the Chroma blocks contain four times less bits than the luma and that there are two of them, Cb and Cr, the same physical memory organization can be used. Of course, there will be a penalty for using the memory space inefficiently, but the same addressing scheme can be reused and only one memory access will provide both Cb and Cr at the same time.

6 System architecture and hardware implementation

Further, the Block level architecture that was conceived for the hardware implementation of q-pel MC using the vectorized data storage is described.

The system's architecture is presented in Figure 8. The MC block is implemented on the FPGA. The inputs to this block are on the right-hand side of the FPGA input frame to be interpolated that contains luma and 2 chroma components, MV map, and the request to inter-predict either a certain area of the image or the full image. These inputs can be provided from outside the FPGA. The reference image and the MV map are written through the memory controller and interface to the external SDRAM memory (figured on the left-hand side of the FPGA). For the proof of concept, the following inputs have been chosen. A sequence of images that has the pattern IPPP... frames, see Table 2. Here, all the P frames are inter-predicted based on the previous frame, and the requests are made for a full frame inter-prediction. After inter-predicting, the first P frame, this one becomes the new reference frame for the next P frame. Here, there are two possibilities: either output the obtained inter-predicted frame or feed it back to the SDRAM memory as the new reference frame. The second variant is chosen for this setup. The operation of writing the new reference frame to the memory does not add extra delay as the blocks are vectorized by reordering, which on a hardware platform can be performed without penalties.

The requests are buffered in a FIFO. Then they are processed one at a time. A read request is sent to the memory for retrieving the corresponding MV sets for every MB to be inter-predicted. After that the reference block start addresses are calculated by composing the address of the current position and the displacement introduced by the MV. These raster scan order addresses are mapped to vectorized addresses using the mechanism described in Section 5.3. Then the corresponding luma and chroma blocks are retrieved from the memory.

This design is highly pipelined. It performs simultaneously the inter-prediction for luma and 2 chromas. The luma pipeline has a higher processing delay than the pipeline serving the 2 chromas. Further on, after having retrieved from the SDRAM memory all the data necessary for one block inter-prediction, the filtering operations are performed according to the standard (FIR filtering for luma, bi-linear interpolation for chroma). The MC's results are also buffered into FIFOs.

7 Method results and comparison to existing work

Several YUV sequences of different dimensions were chosen for the proof of concept. The sequences range from the small QCIF format up to HD, as shown in Table 2.

The proposed method has been implemented using VHDL on a Stratix IV FPGA EP4SGX230KF40C2. The implementation of the MC block reached a 215-MHz maximum frequency. The resource usage for Logic utilization is 10% out of which 12988 combinational ALUTs and 6282 dedicated logic registers. 577 dedicated 18-bit DSP blocks are used. This is comparable to the resource usage from [5] The proposed design aimed for speed performance and that is why dedicated DSP blocks were used. The operations that are executed on these DSP blocks do not fully utilize their potential, but offer an increase in speed. Also, the available area on the chosen FPGA allows multiple instances of the proposed MC block to be used. The obtained MC processing speeds are listed in Table 2. The proposed method and implementation achieves over 217 fps for

HD 720p and up to 95.9 fps when using the full HD 1080p streams. This is over 50% more than a maximum of 60 fps for HD frames obtained in [7]. For smaller image formats, the framerate is considerably higher: between 12623 fps for QCIF and 2502 fps for CIF format.

The bandwidth toward the memory is visibly improved by an optimal data arrangement and efficient retrieval scheme. When compared to a linear address approach the proposed scheme realizes 2× up to 3× more efficient retrieval. This is obtained by grouping together on the same row data that are statistically more likely to be requested together and using a bank optimization. The number of read commands on the address bus is drastically reduced when using the proposed method as compared to a linear addressing approach. In Table 1, it is shown that the proposed method achieves the minimum row opening delay possible for any operation required by the MC algorithm.

In [4, 5], the same minimum required data reference loading scheme are used. This is not enough for making the requested data retrieval as efficient as possible. In [4], every MB takes between 492 and 304 clock cycles to complete, as opposed to our implementation that takes between 172 and 276.6 clock cycles for both luma and the 2 chromas. When compared to a similar VLSI implementation [8], the proposed implementation processes one MB of the "flower" movie in 216 cycles, as opposed to 288 or 247 cycles, respectively. It is also a noticeable fact that the proposed implementation gives the full cycle count for a MB with luma, Cb, and Cr components, and [8] offers results only for luma MC.

Another advantage of the proposed method is that all the necessary data for a certain interpolation is available after two or maximum three read actions with only one row opening penalty for any block dimension. Thus, all the pixels for one 4×4 block are ready to be processed in the same time. This makes it possible to obtain all 16 output pixels simultaneously and only have the specific processing delay for one output pixel calculation. This requires of course a higher hardware usage.

Other articles choose a cache memory approach for tackling the MC problem in H.264 decoders. The strategies that are proposed in [3, 4] offer a good data locality but the reusage of pixels in MC is quite low and unpredictable. A cache approach does not tackle the external memory access penalty. The proposed vectorization method could bring a higher benefit to caches when used together because the proposed method pays the minimum row opening penalty possible. The fact that cache memory is very expensive in a hardware implementation when compared to SDRAM memories represents also an issue for some implementations. As explained in Section 5.3 the proposed new address generation algorithm is fast and will not bring any extra delay in a hardware implementation.

The described system is able to perform MC also on bi-directional (B) slices, with small adjustments toward the reference list for MV, operations to be performed and processing order of the frames (for the case where a B frame depends on future P frames). The performance in a system using both P and B frames will be lower than in a system that only uses P frames. A system that can perform the proposed method on P and B frames is reserved for future research.

8 Conclusions

In this article, an innovative data reordering method and its associated addressing scheme for MC in a H.264 decoder were presented. Also the system's architecture and the hardware implementation are presented. The data vectorization makes the retrieval from the external memory faster by grouping pixels that are statistically more likely to be used together on the same memory row. This also results in fewer commands on the memory bus, thus energy wise more efficient. The associated addressing scheme allows the use of this method in a multitude of systems designed for different image sizes and formats. The proposed implementation is able to perform H.264 q-pel MC at speeds between 229.9 fps HD720p and 95.5 fps for 1080p frames and between 12623 and 2187.7 fps for QCIF and CIF, respectively. The proposed method is useful in any physical implementation of a H.264 decoder that aims for improving the most demanding part of the decoder, the MC block, making it possible to serve several streams in parallel in real time.

9 Competing interests

The authors declare that they have no competing interests.

Acknowledgments

The authors would like to thank the Agency for Innovation by Science and Technology in Flanders (Agentschap voor Innovatie door Wetenschap en Technologie in Vlaanderen) for funding the Vampire project and Alcatel Lucent-Bell (ALU) for financial and technical supports as well as their cooperation.

References

[1] Draft ITU-T Recommendation and Final Draft Internationa Standard of Joint Video Specification (ITU-T Rec. H.264/ ISO/ IEC14496-10 AVC), March 2003.

[2] R.G. Wang, J.T. Li, C.H. Huang, Motion compensation memory access optimization strategies for H.264/AVC decoder, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. 97–100, March 2005

[3] S. Yoon, S.-I. Chae, Cache optimization for H.264/AVC motion compensation, ISSN:1745-1361, 0916-8532, in *IEICE Transactions on Information and Systems,* vol. E91-D, no. 12, pp. 2902–2905 (IEICE, 2008)

[4] Y. Li, Y. Qu, Y. He, Memory cache based motion compensation. 1-4244-0921-7/07, 9518760, in *IEEE International Symposium on Circuits and Systems, ISCAS,* 2007

[5] D.-Y. Shen, T.-H. Tsai, A 4X4-block level pipeline and bandwidth optimized motion compensation hardware design for H.264/AVC decoder, 978-1-4244-4291-1/09, in *IEEE International Conference on Multimedia and Expo, ICME*, 2009

[6] DDR3 SDRAM Component Data Sheet: MT41J64M16LA-187E

[7] D. Zhou, P. Liu, A hardware-efficient dual-standard VLSI Architecture for MC Interpolation in AVS and H.264. 1-4244-0920-9, in *IEEE International Symposium on Circuits and Systems, ISCAS*, 2007

[8] N.-R. Zhang, M. Li, W.-C. Wu, High performance and efficient bandwidth motion compensation VLSI design for H.264/AVC decoder, 1-4244-0161-5/06 (IEEE, 2006)

Table 1: **Comparison between a linear address mapping and the vectorized data mapping for the operations required by MC**

Number of memory rows opening penalty	**Integer**	**Half-pel horizontal**	**Half-pel vertical/ middle**	**q-pel**
Linear data storage	4	4	9	9
Vectorized data storage	2	2	3	3
Linear bank optimization	1	1	2	2
Vectorized bank optimization	1	1	1	1

Table 2: **MC framerate for different image dimensions**

Sequence	**Type**	**Image dimensions pixels**	**MC framerate fps@ 215 MHz**	**Cycle count /MB luma, Cb, Cr**
News	QCIF	176x144	12623	172
Train	QCIF	176x144	8015.5	270.9
Bridge	CIF	352x288	2187.7	248
Flower	CIF	352x288	2502.2	216
Mobcall	HD720p	1280x720	229.9	259.6
Parkrun	HD720p	1280x720	217.9	274
Riverbed	HD1080p	1920x1080	95.9	276.6

Figure 1: **Integer and fractional samples' positions for quarter sample luma interpolation.**

Figure 2: **Linear data storage, no bank optimization.**

Figure 3: **Linear data storage with bank optimization.**

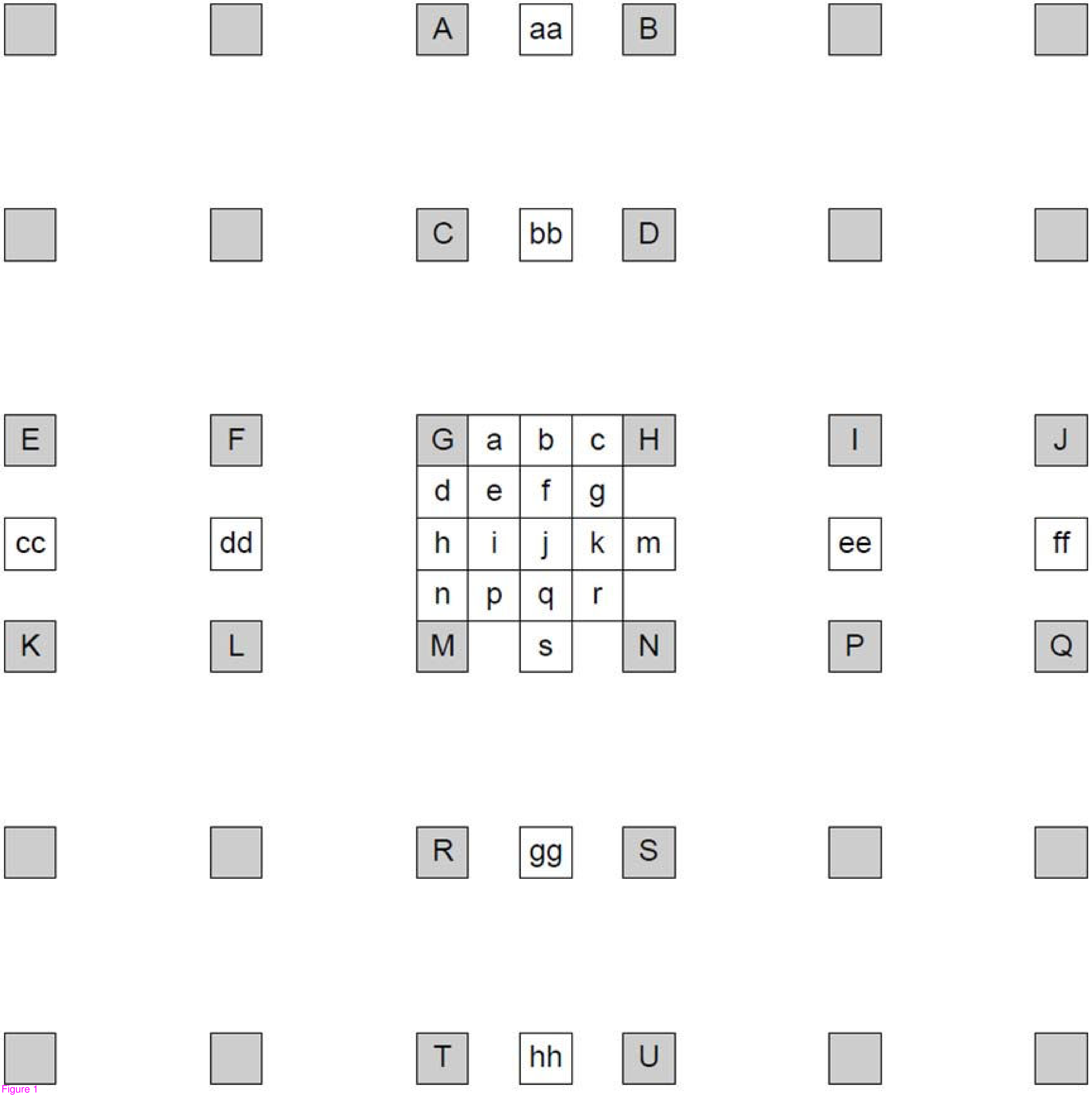
Figure 4: **Vectorized data storage**: **(a)** Image plane 8-bit pixels; **(b)** sub-block natural order, **(c)** vectorized luma 4×4 sub-block, **(d)** DDR3 SDRAM internal image storage.

Figure 5: **Vectorized data storage, no bank optimization.**

Figure 6: **Vectorized data storage with bank optimization.**

Figure 7: **Data mapping and read commands needed for any MC data retrieval**: **(a)** Image plane pixel map and the minimum required reference pixels for different interpolation types, **(b)** equivalent vectorized SDRAM read accesses.

Figure 8: **Block-level architecture for hardware implementation.**


A
aa
B
C
bb
D
E
F
G
a
b
c
H
I
J
d
e
f
g
cc
dd
h
i
j
k
m
ee
ff
n
p
q
r
K
L
M
s
N
P
Q
R
gg
S
T
hh
U
Figure 1

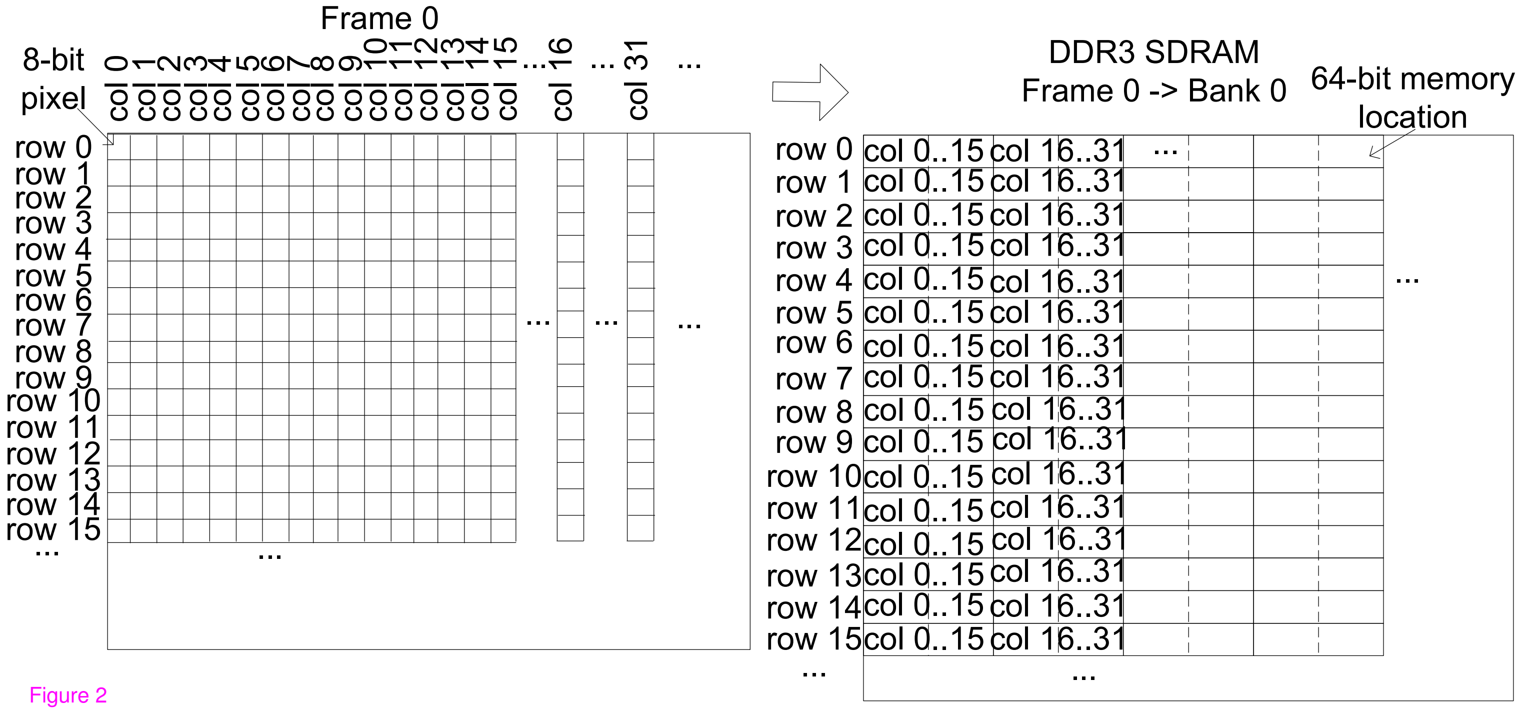
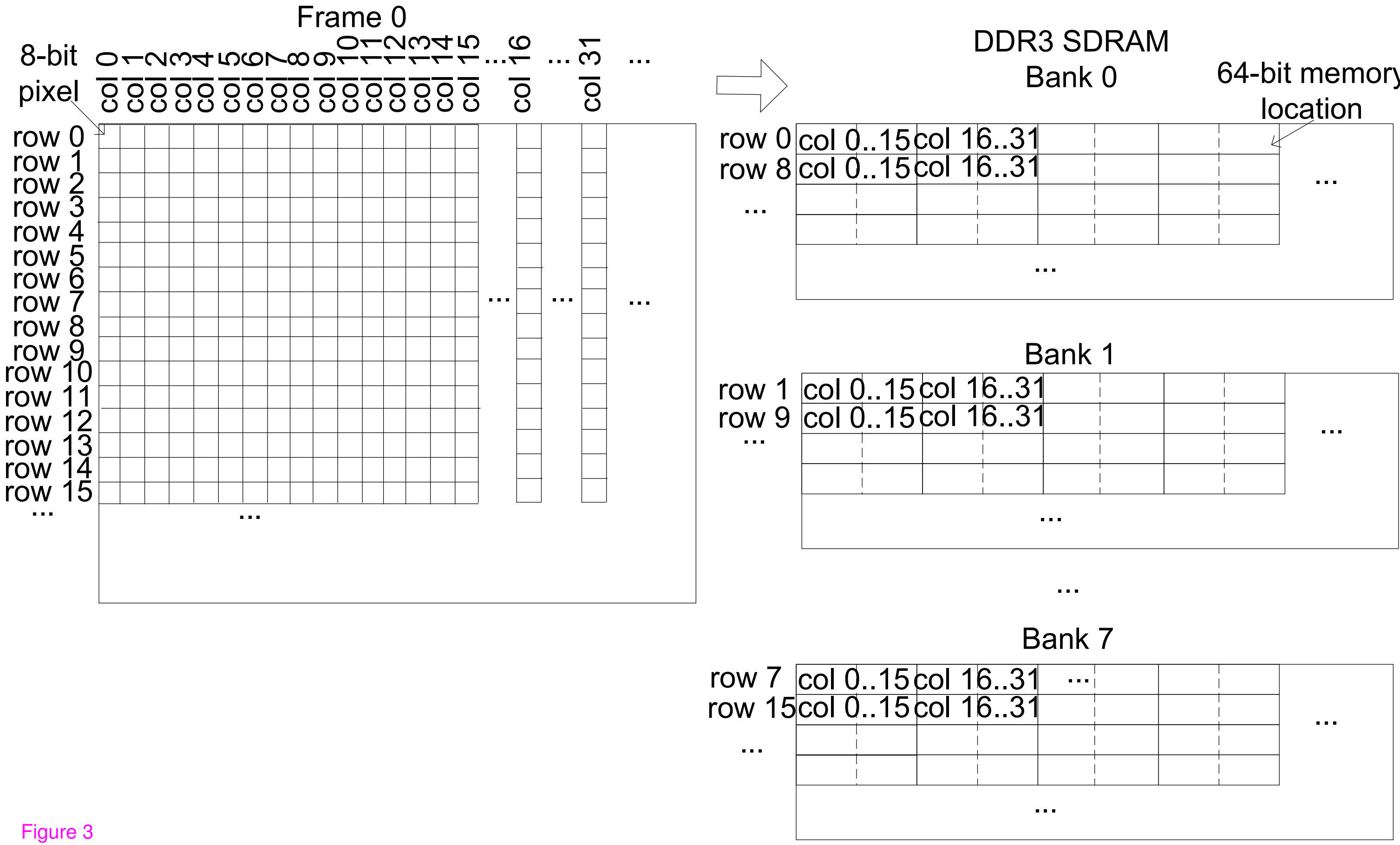
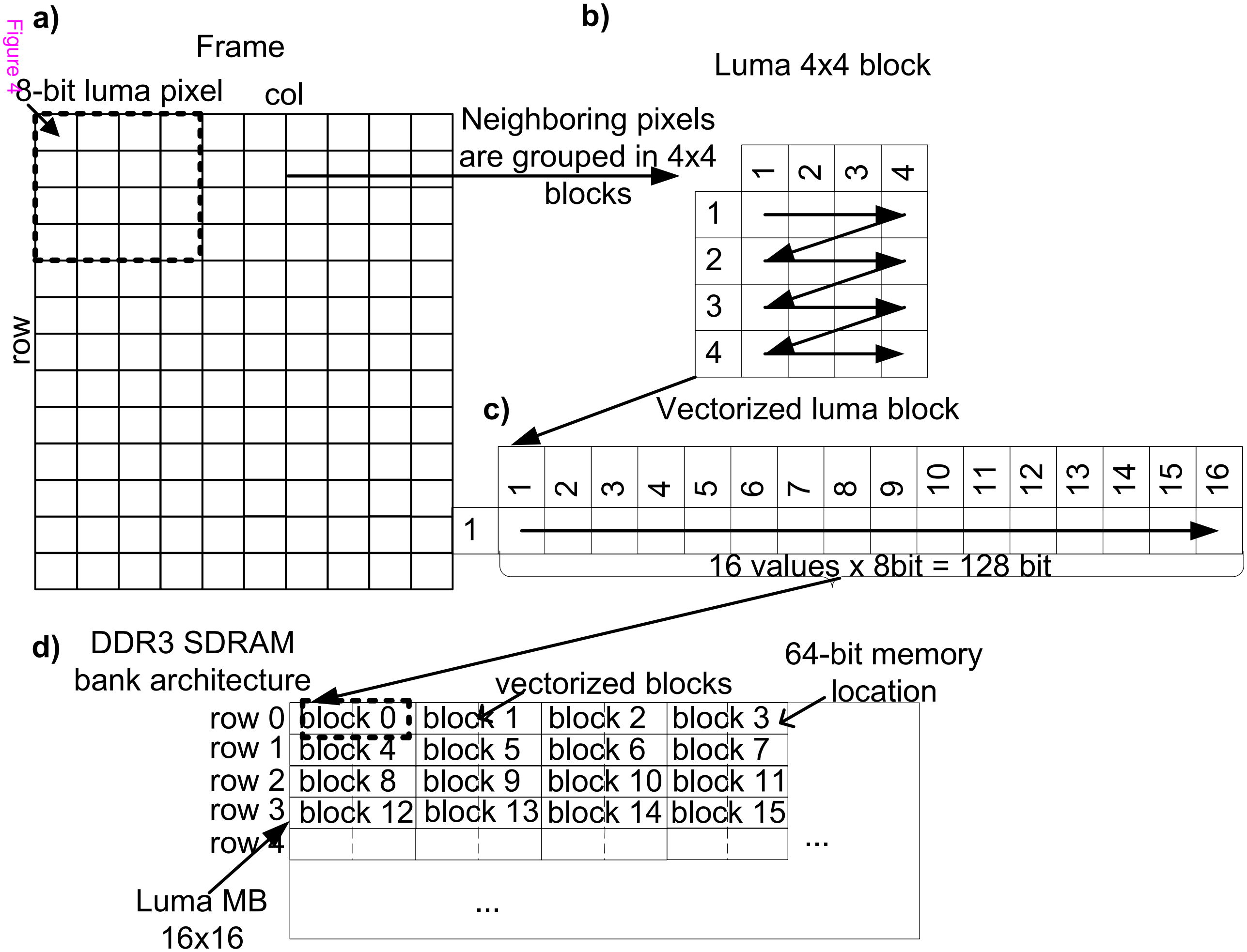

Frame 0
8-bit pixel
col 0
col 1
col 2
col 3
col 4
col 5
col 6
col 7
col 8
col 9
col 10
col 11
col 12
col 13
col 14
col 15
col 16
col 31
row 0
row 1
row 2
row 3
row 4
row 5
row 6
row 7
row 8
row 9
row 10
row 11
row 12
row 13
row 14
row 15
DDR3 SDRAM
Frame 0 -> Bank 0
64-bit memory location
col 0..15
col 16..31
...

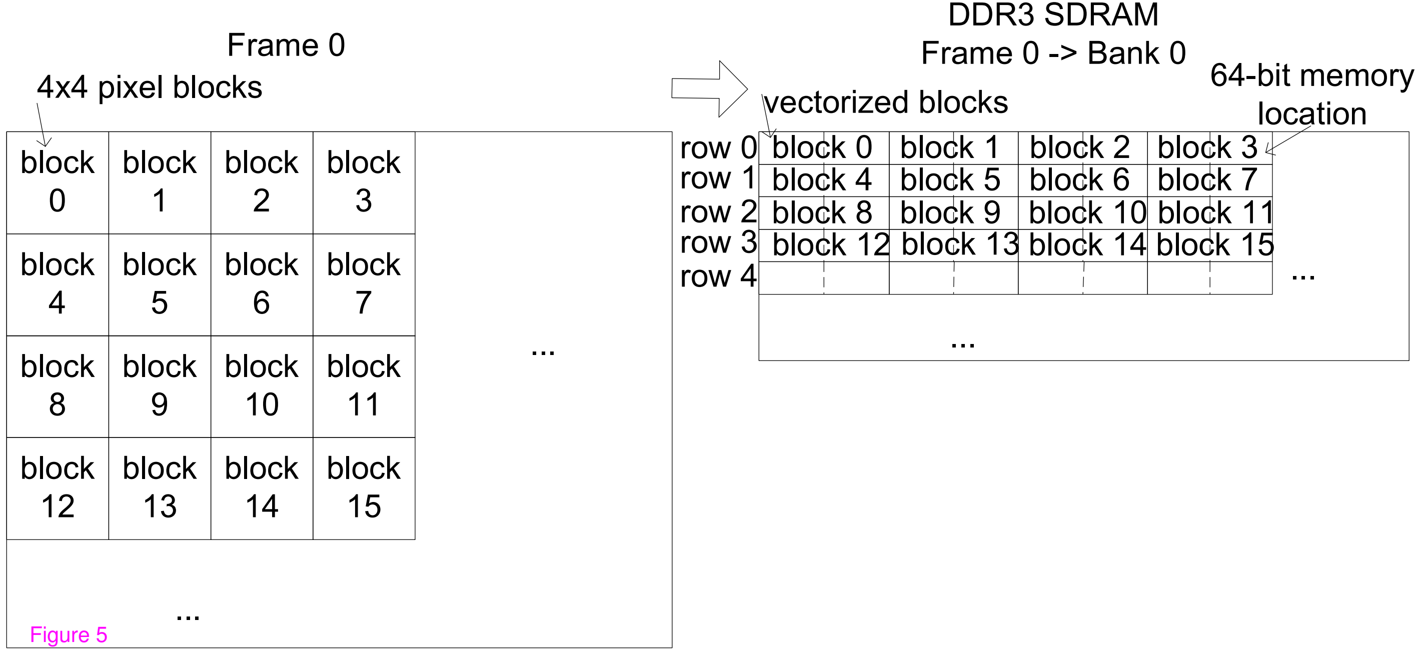
Figure 2

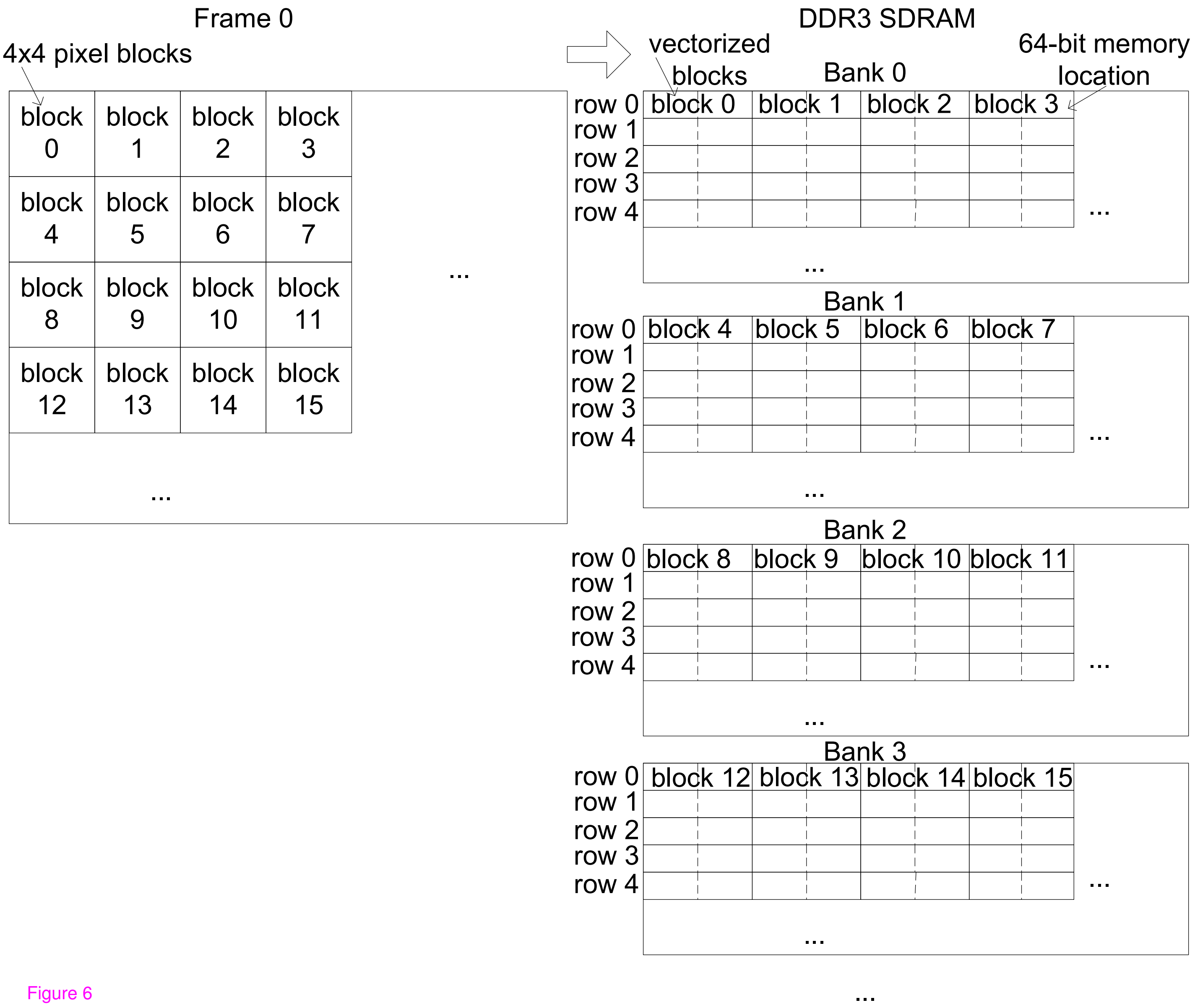


Frame 0
8-bit pixel
col 0
col 1
col 2
col 3
col 4
col 5
col 6
col 7
col 8
col 9
col 10
col 11
col 12
col 13
col 14
col 15
...
col 16
...
col 31
...
row 0
row 1
row 2
row 3
row 4
row 5
row 6
row 7
row 8
row 9
row 10
row 11
row 12
row 13
row 14
row 15
...
DDR3 SDRAM
Bank 0
64-bit memory location
row 0
row 8
col 0..15
col 16..31
Bank 1
row 1
row 9
Bank 7
row 7
row 15

Figure 3

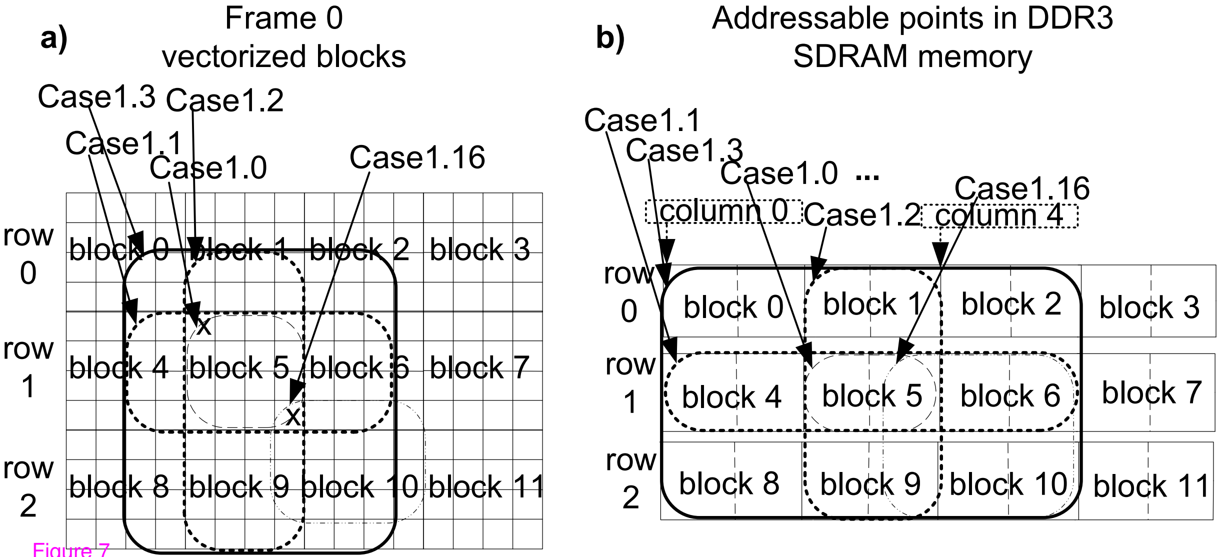

a)
Frame
8-bit luma pixel
col
row
Neighboring pixels are grouped in 4x4 blocks
b)
Luma 4x4 block
1 2 3 4
1 2 3 4
c)
Vectorized luma block
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1
16 values x 8bit = 128 bit
d)
DDR3 SDRAM bank architecture
vectorized blocks
64-bit memory location
row 0
row 1
row 2
row 3
row 4
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11
block 12
block 13
block 14
block 15
...
...
Luma MB 16x16


Frame 0
4x4 pixel blocks
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11
block 12
block 13
block 14
block 15
...
...
DDR3 SDRAM
Frame 0 -> Bank 0
vectorized blocks
64-bit memory location
row 0
row 1
row 2
row 3
row 4
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11
block 12
block 13
block 14
block 15
...
...
Figure 5



Frame 0
4x4 pixel blocks
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11
block 12
block 13
block 14
block 15
...
...
DDR3 SDRAM
vectorized blocks
Bank 0
64-bit memory location
row 0
row 1
row 2
row 3
row 4
block 0
block 1
block 2
block 3
...
...
Bank 1
row 0
row 1
row 2
row 3
row 4
block 4
block 5
block 6
block 7
...
...
Bank 2
row 0
row 1
row 2
row 3
row 4
block 8
block 9
block 10
block 11
...
...
Bank 3
row 0
row 1
row 2
row 3
row 4
block 12
block 13
block 14
block 15
...
...
...

Figure 6



a)
Frame 0
vectorized blocks
Case1.3
Case1.2
Case1.1
Case1.0
Case1.16
row 0
row 1
row 2
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11
x
x
b)
Addressable points in DDR3
SDRAM memory
Case1.1
Case1.3
Case1.0 ...
Case1.16
column 0
Case1.2
column 4
row 0
row 1
row 2
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 7
block 8
block 9
block 10
block 11

Figure 7

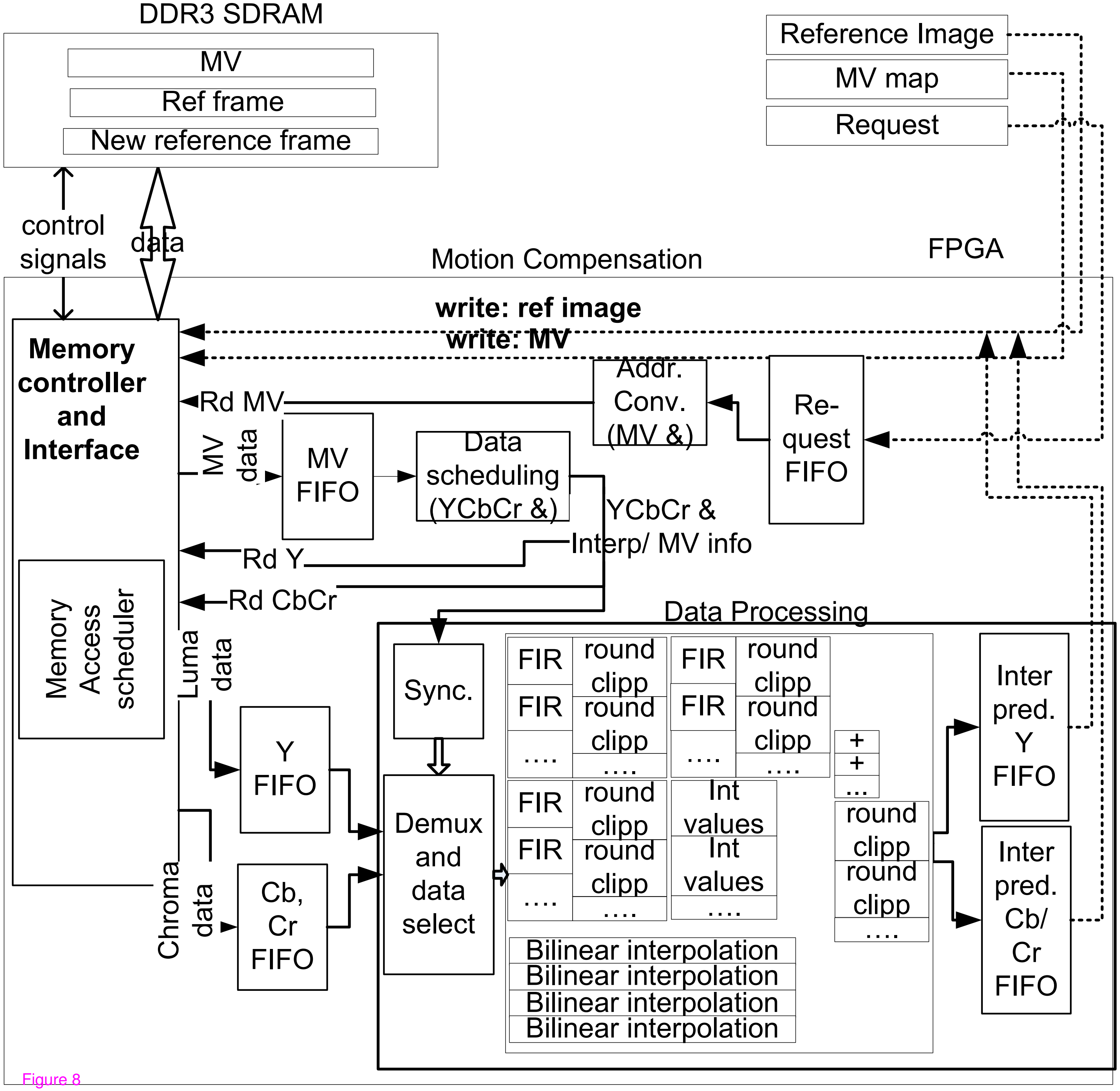

DDR3 SDRAM
MV
Ref frame
New reference frame
control signals
data
Reference Image
MV map
Request
FPGA
Motion Compensation
write: ref image
write: MV
Memory controller and Interface
Memory Access scheduler
Rd MV
Addr. Conv. (MV &)
Re-quest FIFO
MV data
MV FIFO
Data scheduling (YCbCr &)
YCbCr & Interp/ MV info
Rd Y
Rd CbCr
Data Processing
Luma data
Y FIFO
Chroma data
Cb, Cr FIFO
Sync.
Demux and data select
FIR
round clipp
FIR
round clipp
....
....
Int values
Int values
+
+
...
Bilinear interpolation
Bilinear interpolation
Bilinear interpolation
Bilinear interpolation
Inter pred. Y FIFO
Inter pred. Cb/ Cr FIFO

Figure 8