BANDWIDTH REDUCTION FOR VIDEO PROCESSING IN CONSUMER SYSTEMS

Egbert G.T. Jaspers [1] *and Peter H.N. de With* [2]

[1]Philips Research Labs., Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands.
[2]CMG Eindhoven / Eindhoven University of Technology, Den Dolech 2, 5600 MB, The Netherlands

Abstract—**The architecture of present video processing units in consumer systems is usually based on various forms of processor hardware, communicating with an off-chip SDRAM memory (see Fig. 1). Examples of these systems are currently available MPEG encoders and decoders, and high-end television systems. Due to the fast increase of required computational power of consumer systems, the data communication to and from the off-chip memory has become the bottleneck in the overall system performance (memory wall problem). This paper presents a strategy for mapping pixels into the memory for video applications such as MPEG processing, thereby minimizing the transfer overhead between memory and the processing. A novelty in our approach is that the proposed communication model considers the statistics of the application-dependent data accesses in memory. With this technique, a 26% reduction of the memory bandwidth was obtained in an MPEG decoding system containing a 64-bit wide memory bus. For double-data-rate SDRAM (DDR SDRAM), the proposed mapping strategy reduces the bandwidth in the system with even 50%. This substantial performance improvement can readily be used for extending the quality or the functionality of the system.**

Keywords— **bandwidth reduction, memory interface, memory mapping, burst access, DDR SDRAM, MPEG.**

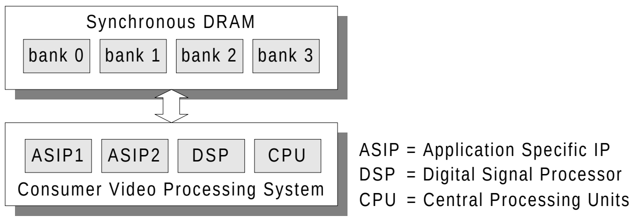

Synchronous DRAM
bank 0
bank 1
bank 2
bank 3
ASIP1
ASIP2
DSP
CPU
Consumer Video Processing System
ASIP = Application Specific IP
DSP = Digital Signal Processor
CPU = Central Processing Units

Fig. 1. Consumer system architecture.

I. Introduction

THE architecture of present advanced video processing systems in consumer electronics commonly uses various processor modules that communicate with an off-chip SDRAM memory (see Fig. 1). For example an MPEG decoder requires a background memory to store reference pictures for prediction of successive video frames. When a large variety of processors desire communication with a standard off-chip memory configuration, a communication bottleneck will be exposed.

This above-mentioned bottleneck was already recognized at an earlier stage in the design of general-purpose computing systems. Moore predicted that the performance density (i.e. the performance per unit area and per unit power) of systems on chip would double every 18 months. This has become particularly noticeable in the computer market where the performance of the CPU has increased proportionally with the number of integrated transistors on chip. Hennessy & Patterson [1] showed this by measuring the performance of microprocessors that were developed in the last decades. The performance is defined as the time that is necessary to execute a well-defined benchmark set [2]. Unfortunately, the performance of off-chip memory communication has not evolved in the same speed as the CPU performance. For example, in [1] it is shown that the CPU performance increases 60% per year, whereas external memory bandwidth improves only with 20%. Concluding, there is an increasing gap between computational power and memory bandwidth. Besides bandwidth, the performance of memory is also determined by its *latency*, i.e. the time between request for data and the actual reception.

For media processing in consumer systems, the bandwidth problem also exists. However, the latency problem can be solved by using efficient pipelining and prefetching techniques [3]. This is caused by the property that the computing operations on one data element are largely independent of operations on other elements so that parallelism can be exploited.

Let us return to the bandwidth problem. In recently developed systems this problem was combated by communicating to several memory devices in parallel. Currently, the smallest available double-data-rate synchronous DRAM (DDR SDRAM) has a 64-Mbit capacity with a 16-bit data bus or smaller, providing a peak bandwidth of 0.53 GB/s [4]. However, significantly more bandwidth is required for media applications, such as simultaneous High-Definition MPEG decoding, 3-D graphics rendering and field-rate conversion. The Imagine processor [3] features four memory controllers with a 32-bit data bus each. The

Emotion Engine processor [5] contains a dual 16-bit memory bus at 800 MHz, providing 3.2 GB/s with 32 MByte in Direct Rambus DRAMs (RDRAM). However, this solution of parallel memory devices introduces more memory capacity than required by the system, leading to a lower cost efficiency. For the above mentioned systems, 256 Mbit of SDRAM memory is the minimal available capacity which is more than required by most consumer systems. In this paper we focus on the reduction of the memory bandwidth, thereby contributing to a reduction of parallel memory devices. This is important because the use of parallel memory controllers leads to high systems costs such as, increased power dissipation, substantially larger silicon areas and more expensive chip packages.
In our study we will concentrate on MPEG decoding as a pilot application. This application features block-based video processing and memory access. Such memory access to optimize bandwidth was already addressed in [6], where a mapping of video data units into the memory is proposed. This work is related to analyzing the application software model only, without considering data dependencies such as the set of requested data blocks including their probability of occurrence. In this paper, we determine an optimal mapping of the video into the memory by analyzing the actual memory accesses, so that data dependencies are taken into account.

To understand the optimizations for bandwidth efficiency, Section II elaborates on the architecture of SDRAM-based memories and introduces the concept of data units. Section III derives the size of data units depending on the access timing parameters and the memory bus width. Subsequently, Section IV explains how the pixels are mapped onto the physical memory addresses considering the application-dependent accesses and the organization in memory banks, rows and columns. In Section V we will discuss the optimization calculations and the implementation in a realistic simulation model. All application-specific issues that have an impact on the simulation results will be discussed in Section VI by means of an MPEG decoding application example.Finally, Section VII presents the results and conclusions.

II. SDRAM-based memory architecture and the introduction of data units

The increasing demand for more memory bandwidth requires the use of sophisticated memory devices like DDR SDRAM or RDRAM. To obtain high performance, these devices use two main features: the burst-access mode and the multiple-bank architecture. A block diagram of a DDR SDRAM memory device is

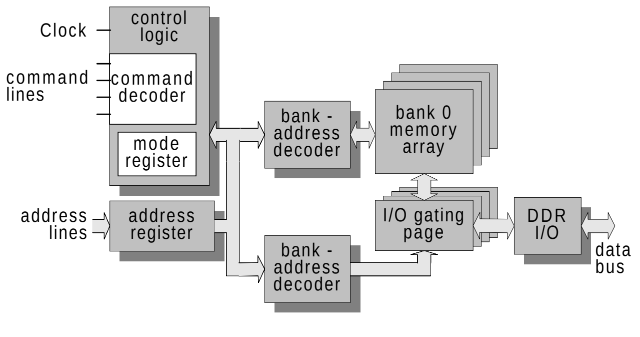


Fig. 2. Block diagram of a DDR SDRAM.

shown in Fig. 2. The burst-access mode enables access to a number of consecutive data words by giving a single read or write command. Because the reading of dynamic memory cells is destructive, the content in a row of cells in the memory bank is copied into a row of static memory cells (the page registers). Subsequently, read and write access to this copied row is provided. The result after the required accesses in the row has to be copied back into the (destructed) dynamic cells, before a new row in the memory bank can be accessed. These actions, referred to as row-activation and precharge respectively, consume valuable time in which the array of memory cells (a bank) cannot be accessed. In order to overcome this problem and access the memory device also during row-activations and precharges, a multiple-bank architecture is used, where each bank can be accessed alternately. Hence, a bank can be accessed while other banks are activated or precharged. Furthermore, high throughput is achieved by dividing the device into stages using pipelining (at the expense of increased latency).

Let us now concentrate on the consequences of the burst-access mode, using the previously described SDRAM architecture. To optimize the utilization of the memory-bus bandwidth, data can only be accessed

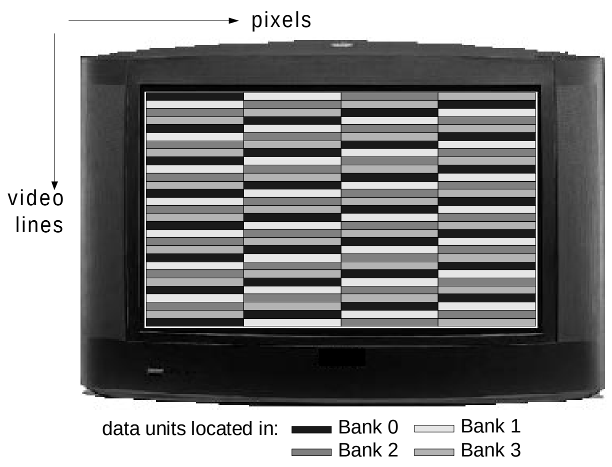
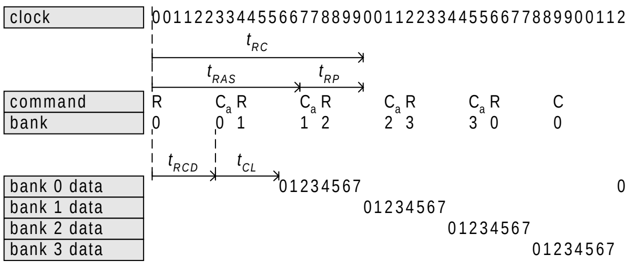


Fig. 3. Video pixels mapped onto data units, each located in a particular memory bank.

at the grain size of a data burst (e.g. eight words). If the memory configuration provides a 64-bit bus and is programmed for a burst length of eight words, one data burst contains 8×64 bit = 64 bytes of data. These data bursts represent non-overlapping blocks in the memory which can only be accessed as an entity. In the remainder of this paper, these blocks are referred to as *data units*. Fig. 3 shows an example how the pixels of a video picture are mapped onto data units in the memory. Each colored rectangle represents one data unit. A required group of pixels (e.g. a macroblock in MPEG) might be partly located in several data units and therefore results in the transfer of all corresponding data units. Hence, significantly more pixels than required are transferred. In the sequel we call these extra pixels *pixel overhead*. This overhead becomes particularly noticable if the size of the data units is relatively large compared to the requested group of pixels. This paper describes the partitioning of data units into memory banks and determines the optimal dimensions of the data units to maximize the available memory bandwidth. The optimization includes statistical analysis of the data to be accessed. A mapping of video data units into the memory is already proposed in [6]. However, this paper proposes to analyzing the application software model only without considering data dependencies such as the set of requested data blocks including their probability of occurrence. For example, the type of data blocks that are fetched for motion compensation in an MPEG decoder, strongly depends on the motion-estimation strategy applied by the encoder. In this paper, we determine an optimal mapping of the video into the memory by measuring and analyzing the actual memory accesses, so that data dependencies are taken into account. Another consideration that is important for bandwidth efficiency is the organization into memory banks, which is provided in all modern memory devices. It will become clear that both aspects contribute to a substantial improvement of the available memory bandwidth.

III. Derivation of the data-unit size

To access a data unit in the memory, first a row-activate command also called Row Address Strobe (RAS) has to be issued for a bank to copy the addressed row into the page (static-cell registers) of that bank. After a fixed delay t_{RCD} (RAS to CAS delay), a read or write command also called Column Address Strobe (CAS) for the same bank can be issued to access the required data units in the row. When all required data units in the row are accessed, the corresponding bank can be precharged, which takes t_{RP} time. The time from row-activate command until the precharge may not be less than the t_{RAS}, which



Note: R is row-activation command
C_a is column command followed by an auto-precharge
t_{RC} is minimal row cycle time
t_{RAS} is minimal row active time
t_{RP} is precharge time
t_{RCD} is minimal RAS to CAS delay
t_{CL} is CAS to data latency

Fig. 4. Timing of the memory command.

is the minimum time a row is active. After access of a row, precharging is required to prepare the bank for the subsequent row addressing in the same bank. Hence a minimum time after activation of a row in a bank ($> t_{RAS}$), the precharge time t_{RP} elapses before that bank can be accessed again. Consequently, the time between two subsequent row-activate commands (referred to as the row cycle time) for the same bank is at least $t_{RC} = t_{RAS} + t_{RP}$ time and is typically 10 cycles for current DDR SDRAMs (see Fig. 4).
The clock numbers are indicated twice because both positive and negative clock transitions are used to access data. The memory commands are provided at each positive clock transition only. The bottom of the figure shows how four bursts of eight data words are read by four successive accesses, each in a different bank. Obviously, the elapse time between the first data word from the first bank until the last data word from the fourth bank, consumed 32 double-data-rate (DDR) cycles and is equivalent with 16 single-data-rate (SDR) input cycles. Because this time exceeds the row cycle time t_{RC}, a new row-activate command can be issued immediately, without wasting valuable memory-bus cycles. It can be concluded, that interleaved access of the memory banks provides optimal utilization of the memory-bus bandwidth.

Let us now return to the primary objective of this section and determine the best choice for the size of the data unit from the above-mentioned memory properties. Amongst others, it depends on the size of the burst length. To minimize the pixel overhead the data units are preferred to be small. However, if the burst length is too small, the time that elapses after accessing all four banks does not exceed t_{RC} and causes some waiting cycles in which no data is transferred over the bus. Apparently there is a tradeoff between bus utilization and pixel overhead. To determine the minimal burst length BL for which a full utilization of the bus bandwidth can be achieved, the number of data words transferred within t_{RC} (t_{RC} cycles at DDR) is divided by the number of banks, thus:

$$BL \geq 20/4 \qquad (1)$$

Because the burst length is a multiple of two due to the double-data-rate and because the burst length can only be programmed for the values 2, 4 or 8, the burst length is set to $BL = 8$. Because this value is larger than the determined lower bound of 5 (see Eq. (1)), it is not required to interleave all four banks before the first bank is accessed again. Note that access to three successive banks occupies 3×8 cycles, thus already exceeds t_{RC}.

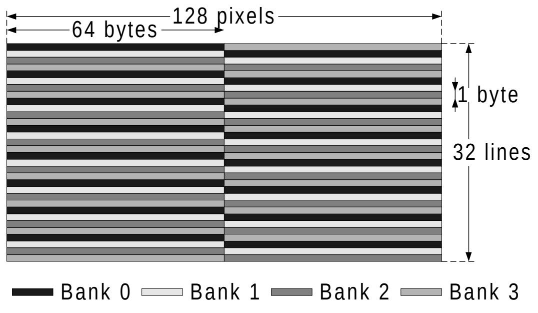


Fig. 5. Mapping of 64×1 adjacent pixels onto data units.

IV. The mapping of pixels into the memory

In this section we discuss the key parameters that determine the optimal dimensions of the data units. Firstly, we will describe the partitioning of data units into memory banks, while considering an interleaved usage of all banks. It will be shown how this is achieved for both progressive and interlaced video signals.

Let us discuss a few examples for data unit dimensions and the corresponding pixel overhead. For this purpose, we assume a system that requires a 64-bit bus SDRAM configuration to provide sufficient bandwidth. Consequently, the data units in the memory contain 64 bytes. For the mapping of pixels, several options can be recognized. The most straightforward way is to map 64 successive pixels of a video line into one data unit as depicted in Fig. 5. The figure shows how each consecutive block of 64 pixels is interleaved in the banks in both horizontal and vertical direction. If for such interleaved mapping the pixel data is sequentially read or written (given by the application), the memory banks are accessed alternately. However, when a data block of 16×16 pixels is requested from the memory, the amount of data that needs to be transferred is much more. If the data block is horizontally positioned within one data unit, 64×16 pixels are transferred. If the data block overlays two data units in horizontal direction, the amount of transferred data is 128×16,

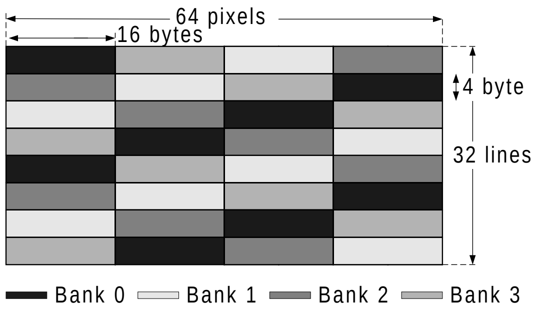


Fig. 6. Mapping of 16×4 adjacent pixels onto data units.

resulting in 700% pixel overhead. Fig. 6 shows a much more appropriate mapping of pixels onto the memory for this data-block request. Blocks of 16 horizontal by 4 vertical pixels are stored in a single data unit, resulting in less pixel overhead when accessing a data block of 16×16 pixels. However, when a data block of 128×1 is requested, Fig. 5 provides a better mapping strategy. Let us now discuss the effect of interlacing. For several applications in a multi-media system, it is necessary to read the video data both progressively and interlaced, e.g. for frame prediction and field prediction in MPEG decoding. However, when subsequent odd and even lines are mapped onto the same data unit, it is not possible to access only odd or even lines without wasting memory bandwidth. Therefore, the odd and even lines are positioned in different banks of the memory. As a result, the data units are interleaved in the memory when the vertical size is larger than one. The resulting mapping strategy for data units of 16×4 pixels is shown in Fig. 7. For this mapping the 16×4

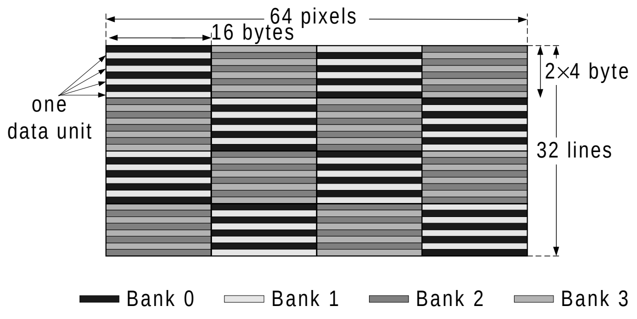


Fig. 7. Mapping of interlaced video onto memory data units.

pixels are not adjacent. Four line pieces of 16×1 which are interlaced in the video frame are mapped onto one data unit. Note that for retrieval of data blocks with *progressive* video lines, the size of the smallest data packet to be accessed as an entity has become eight lines high (two vertically adjacent data units), whereas for access to data blocks with *interlaced* video, the size is four lines (one data unit).

For efficient access of interlaced video data, the mapping of odd and even video lines into odd and even

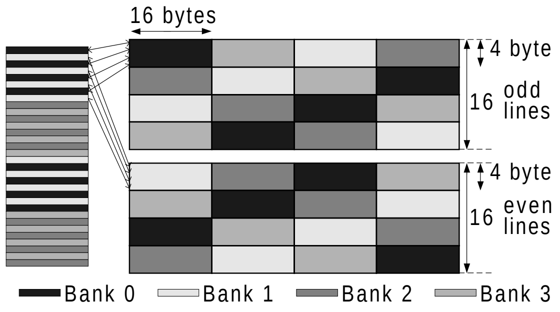


Fig. 8. Decomposition into mappings for separate video fields.

banks is toggled after four units in vertical direction (a reversal of the bank parity when looking to the global checkerboard pattern in Fig. 7). In the first group of 16 video lines, the odd lines are mapped onto bank 0 and 2, while the even lines are mapped onto bank 1 and 3. In the following 16 video lines (two checkerboard blocks lower), the odd lines are mapped onto bank 1 and 3 and the even lines are mapped onto bank 0 and 2. For progressive video this gives only a minor difference, but for interlaced video, this results in addressing of all banks instead of only odd or even banks. This is shown in Fig. 8 where the mapping of Fig. 7 is decomposed into separate video fields. The left part of the figure shows only one column of data units from Fig. 7.

Concluding all above-mentioned system aspects from the previous examples in this section, the optimal mapping strategy depends on the following parameters (see Fig. 9 for the definition of the size parameters).

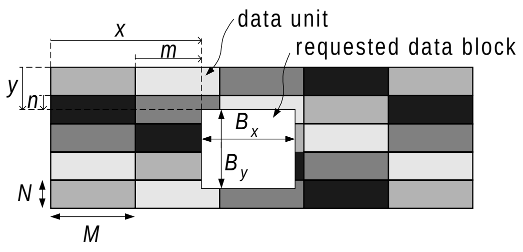


Fig. 9. Definition of the size parameters.

- *The dimensions of the requested data blocks, $B_x \times B_y$.* MPEG-2 decoding contains a large variety of different data-block accesses: due to interlaced and progressive video, field and frame prediction, luminance and chrominance data and due to the sub-pixel accurate motion compensation (all these processing issues are addressed in the MPEG standard).
- *The interlace factor of the requested data blocks.* Progressive data blocks require accesses in pairs of data units in vertical direction. Consequently, the smallest data entity to access is two data units. Hence, the calculations are slightly different for progressive and interlaced video.
- *The probability of their occurrence, $P(B_x \times B_y)$.* For example, if only 16×1 data blocks are accessed (100% probability), the optimal data-unit dimension will also be very much horizontally oriented. Obviously, the probability of each data-block type dependents very much on the application. Moreover, it depends on the implementation. For example if the color components in an MPEG decoder are multiplexed before storage in the reference memory, some data-block types for chrominance and luminance are equal, thereby increasing their probability.
- *The probability distribution of their positions, $P_{B_x \times B_y}(m,n)$.* For this function, the parameter $m = x \bmod M$, $n = y \bmod N$, M is the horizontal data-unit size and N the vertical data-unit size. Thus (x,y) are the global coordinates of the requested data block, (m,n) denote the local coordinates within the corresponding data unit. If a requested data block is aligned with the boundaries of the data units, it overlays the minimum amount of data units, resulting in the minimum pixel overhead. Data blocks that overlay many data units cause much pixel overhead. Note that the 16×16 macroblock grid for MPEG and the high probability of the zero-motion vectors have a positive influence on reducing the pixel overhead.

The last two parameters indicate that the statistics of the memory access are considered, because all requested data blocks (e.g. location and usage frequency) are retrieved with varying probability. The probability distributions introduced in this section will be inserted into an architecture model which is discussed in the next section.

V. Architecture model for simulation

To measure the statistics as indicated in the previous section, an MPEG-2 decoder was modeled including the main-memory interface that transfers data to and from the SDRAM memory when requested by the decoder. The interface between the MPEG-2 decoder and the memory interface is defined as follows:

```
void transfer(
  boolean  read,   // a read (TRUE) or write transfer
  integer  Bx,     // horizontal data-block size
  integer  By,     // vertical data-block size
  boolean  interl, // interlaced (TRUE) or progressive
  integer  x,      // horizontal data-block position
  integer  y,      // vertical data-block position
  integer  line,   // horizontal size of a video line
  u char   *mem,// pointer to the memory
  u char   *buf)  // pointer to the read/write buffer
```

The implementation of the interface translates the input parameters to a set of corresponding data-unit addresses in the memory. Depending on the state of the memory banks, arbitration is provided between read and write requests from the MC unit, and the output unit (VO) that displays the data. Subsequently, it translates all data-unit requests to memory commands and schedules the memory commands to satisfy all timing parameters for an optimal data-bus utilization. In addition, the memory interface generates function calls to the communication analyzer. The communication analyzer as depicted in Fig. 10 analyzes the requests and updates the statistics which were mentioned in the previous section into a database. After decoding of a representative set of bit streams, the optimal data-unit dimensions can be calculated from the statistics in the database.

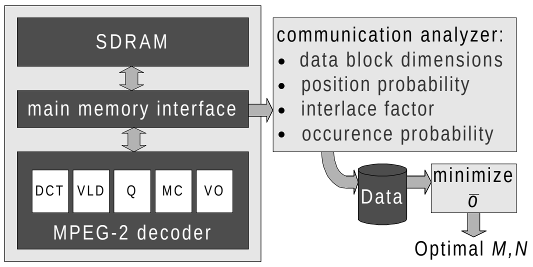


Fig. 10. Architecture model for simulation.

The optimal data-unit dimensions are calculated by minimizing the pixel overhead as function of the data-unit dimensions. The pixel overhead $\bar{o}_i$ for interlaced data-block requests is calculated as:

$$\bar{o}_i(M,N,V_i) = \frac{\sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) H(M,N,V_i)}{\sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) \cdot B_x \cdot B_y} - 1 \quad (2)$$

with

$$H(M,N,V_i) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} P_{B_x \times B_y}(m,n) \bullet M \bullet N \bullet \left(1 + \left\lfloor \frac{B_x+m-1}{M} \right\rfloor\right) \bullet \left(1 + \left\lfloor \frac{B_y+n-1}{N} \right\rfloor\right), \quad (3)$$

where V_i is the set of possible requested data blocks $B_x \times B_y$, $P(B_x \times B_y)$ the probability of the data block, M the horizontal size of the data unit and N the vertical size of the data unit (see Fig. 9 for the definition of the parameters). Probability $P_{B_x \times B_y}(m,n)$ is equal to the probability that the upper left corner pixel of a requested data block $B_x \times B_y$ is positioned at any location (x,y) that satisfies the following condition: x mod $M = m$ AND y mod $N = n$. The numerator in Eq. (2) represents the amount of transferred data including the pixel overhead. The denominator represents the amount of requested data without the pixel overhead. For progressive data-block requests, the data has to be partitioned into two interlaced data-block requests. Therefore, the overhead calculation for progressive data-block requests is slightly different. V_i becomes V_p in Eq. (2) and $H(M,N,V)$ in Eq. (3) is defined as:

$$H(M,N,V_p) = \sum_{m=0}^{M-1} \sum_{n=0}^{2N-1} P_{B_x \times B_y}(m,n) \bullet M \bullet N \bullet \left(1 + \left\lfloor \frac{B_x+m-1}{M} \right\rfloor\right) \bullet \left(2 + \left\lfloor \frac{\lceil B_y/2 \rceil + \lfloor n/2 \rfloor - 1}{N} \right\rfloor + \left\lfloor \frac{\lfloor B_y/2 \rfloor + \lceil n/2 \rceil - 1}{N} \right\rfloor\right). \quad (4)$$

When a system that uses a combination of progressive and interlaced video has to be considered, the set of requested data blocks has to be separated into a set of interlaced data blocks and a set of progressive data blocks. Subsequently, Eq. (2) has to be applied with both Eq. (3) and (4). Thus

$$\bar{o}(M,N,V) = \bar{o}_i(M,N,V_i) + \bar{o}_p(M,N,V_p), \quad (5)$$

with $V = V_i \cup V_o$.

Note that Eq. (5) is a non-weighted sum of averages, because each individual term covers only a part of the overall occurrence probabilities (thus already statistically weighted). For example, if the occurrence ratio between interlaced and progressive data-block requests is 1:3, the value of $\bar{o}_i$ is only one quarter of the actual pixel overhead for interlaced data-block requests.

VI. MPEG decoding as application example

As mentioned in the previous section, we modeled an MPEG-2 decoder as a pilot application. In our simulations, we consider the reading of data for prediction of macroblocks (MBs), the writing of reconstructed MBs and the reading of data for display.

A. Reading of prediction data

Let us consider the sets V_i and V_p that are used for prediction of the MBs.

$$\begin{aligned} V_p &= \{(16\times16),(17\times16),(16\times17),(17\times17),\\ &\quad (16\times8),(18\times8),(16\times9),(18\times9)\} \\ V_i &= \{(16\times16),(17\times16),(16\times17),(17\times17),\\ &\quad (16\times8),(18\times8),(16\times9),(18\times9),\\ &\quad (17\times8),(17\times9),(16\times4),(18\times4),\\ &\quad (16\times5),(18\times5)\} \end{aligned}$$

The numbers $2^p \pm 1$ for the luminance data blocks originate from sub-pixel accurate motion compensation. For the chrominance, the Cr and Cb components

TABLE I
Probability of occurrence, $P(B_x \times B_y)$.

Luminance frame prediction		**Luminance field prediction**		**Chrominance frame prediction**		**Chrominance field prediction**	
block type	prob. [%]	block type	prob. [%]	block type	prob. [%]	block type	prob. [%]
$16 \times 16 \in V_p$	1.59	$16 \times 8 \in V_i$	2.28	$16 \times 8 \in V_p$	12.95	$16 \times 4 \in V_i$	7.85
$17 \times 16 \in V_p$	3.12	$17 \times 8 \in V_i$	6.84	$18 \times 8 \in V_p$	4.47	$18 \times 4 \in V_i$	6.24
$16 \times 17 \in V_p$	4.75	$16 \times 9 \in V_i$	3.06	$16 \times 9 \in V_p$	4.05	$16 \times 5 \in V_i$	6.16
$17 \times 17 \in V_p$	14.86	$17 \times 9 \in V_i$	13.50	$18 \times 9 \in V_p$	2.86	$18 \times 5 \in V_i$	5.43
Total	**24.32**		**25.68**		**24.32**		**25,68**

are multiplexed in the horizontal direction. Each odd sample is a Cr value, each even sample is a Cb value. Therefore, the sub-pixel accurate motion compensation of chrominance data blocks may result in the numbers $2^p \pm 2$ for the horizontal direction. The probability distribution of the position of a requested block $B_x \times B_y$ that satisfies the condition $x \bmod M = m$ AND $y \bmod N = n$ was measured during decoding a representative test-set of MPEG-2 bit streams. Fig. 11 and 12 show two examples of a probability distribution of the positions. Fig. 11 shows high probabilities at the corner positions. At these positions, a progressive block of 17×17 is aligned with boundaries of the 8×8 data units and occurs when the block has a zero-motion vector (or half-pel). Apparently, zero or very low-speed motion macroblocks (MBs) have a high occurrence probability. If data blocks are aligned with the boundaries of the data units, the amount of pixel overhead is minimal. Consequently, the high probability of zero-motion has a positive effect on the transfer bandwidth. Fig. 12 shows the position probabilities of an interlaced 18×4 block. From the zero probability of the odd horizontal positions, it can be concluded that it concerns a chrominance block in which the C_r and C_b samples are multiplexed in the horizontal direction. Because the requested block contains interlaced video, the probability of the odd vertical positions are very similar to the probabilities of the even vertical positions.

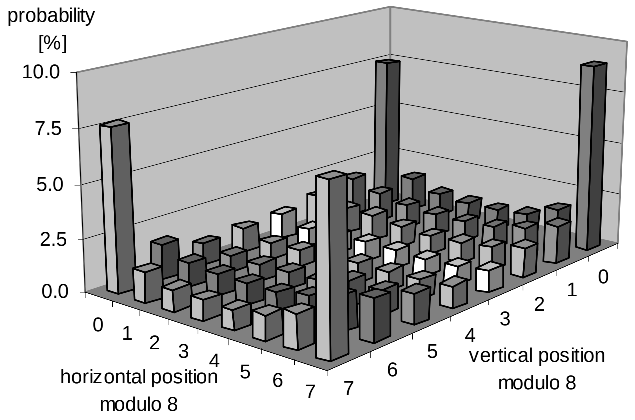


Fig. 11. Example of probability distributions for luminance of $P_{17\times17}(n,m)$ from set V_p with $(M,N) = (8,8)$.

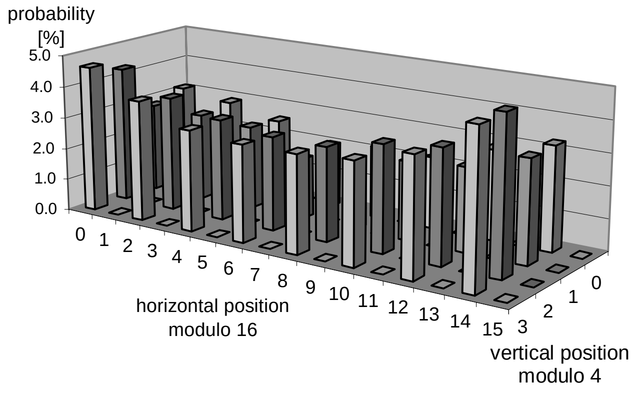


Fig. 12. Example of probability distributions for chrominance of $P_{18\times4}(m,n)$ from set V_i with $(M,N) = (16,4)$.

Besides the probability distribution of the positions, also the probability of occurrence of all block types are measured (see Table I). Note that the amount of block requests for luminance is equal to the amount of block requests for chrominance. Furthermore, the table shows that the blocks of $\{(16 \times 16), (17 \times 16), (16 \times 17), (17 \times 17)\}$ from set V_i are absent. This indicates that no field-based decoding is carried out by the MPEG decoder. Hence, only frame-based pictures are used. Since most commercially available MPEG encoders perform coding of frame-based pictures only, this is a realistic measurement. Because the motion vectors for the luminance and the chrominance in a MB are equal (apart from scaling), the probability of each chrominance block type can mathematically be determined from the probabilities of the luminance block types. However, this is not true for all applications. Hence, the occurrence probability of all block types was measured to generalize our optimization approach in this paper for all applications.

B. Writing of the reconstructed data

In an MPEG decoder application, the reconstructed pictures are written in memory for output display, or as reference pictures to predict new pictures using motion compensation. This writing is done on MB basis and consumes part of the memory bandwidth. Also for this kind of block-based access, the pixel-overhead considerations as discussed above are valid. However, the access for writing reconstructed pictures is very regular. The MBs of the pictures are written sequentially from left to right and from top to bottom at fixed grid positions of 16×16. Consequently, the probability distribution of the positions can be determined easily. Let us assume that the 16×16 grid of the MBs are always aligned to some boundaries of the $M \times N$ grid. With this restriction, the following probability distribution holds:

$$P_{(16\times16\in V_p)}(m,n) = \qquad (6)$$
$$\begin{cases} \frac{1}{\lceil\frac{M}{16}\rceil\cdot\lceil\frac{N}{16}\rceil} & , m \bmod 16 = 0 \wedge n \bmod 16 = 0 \\ 0 & , \text{elsewhere} \end{cases},$$

with $m = x \bmod M$ AND $n = y \bmod N$. Because the bit streams in the test set only contain frame pictures, the written MBs are only contained in the set V_p. Because the occurrence probability is a relative measure, it highly depends on the amount of data requests for the prediction. This is determined by, amongst others, the amount of field and frame predictions, the structure of the Group Of Pictures (GOP), the amount of forward, backward and bi-directional predicted MBs in a B-picture, etc. However, experiments have shown that the final results as presented in Section VII, are not very sensitive for these differences.

C. Reading of data for display

Besides the reading of prediction data and the writing of MBs, also reading of video data for display has to be taken into account. In contrast with the previous memory accesses, the reading of video data for display is done line-wise instead of block-based. Conversion of the block-based data in memory into line-based data is another factor that influences the mapping strategy. To optimize the mapping strategy as function of the pixel overhead calculated with Equations (2)-(5), the requests for display of the video have to be included into the data set. For the dimensions of the requested data for display, the following options should be considered:

- reading of video lines by means of block transfers, thereby accepting a significant penalty in memory bandwidth;
- usage of embedded video-line memories in the architecture to convert data blocks into video lines.

For the first option, line-based requests are used with data blocks of size $M \times 1$ and are added to the set of data blocks. The probability distribution of the position depends on the data-unit dimensions by:

$$P_{(M\times1\in V_i)}(M,N) = \begin{cases} \frac{1}{N} & \text{for } m \bmod M = 0, \\ 0 & \text{elsewhere.} \end{cases} \qquad (7)$$

It is easy to derive that the pixel overhead for such transfers equals

$$o(M, 1, \{(M \times 1)\}) = (N-1) \bullet 100\%. \qquad (8)$$

Because the ratio between requests for writing the output MBs and reading for video display is fixed, the probability of occurrence for the line-based requests can be calculated as follows:

$$P(M \times 1 \in V_i) = \frac{16 \times 16}{M \times 1} \bullet P(16 \times 16 \in V_p). \qquad (9)$$

When video-line memories are embedded, the size of the requested data blocks is $M \times N$ with the following probability distribution of the position:

$$P_{M\times N\in V_p}(m,n) = \qquad (10)$$
$$\begin{cases} 1 & \text{for } m \bmod M = 0 \bigwedge n \bmod N = 0, \\ 0 & \text{elsewhere,} \end{cases}$$

with $m = x \bmod M$ AND $n = y \bmod N$. The probability of occurrence is:

$$P(M \times N \in V_p) = \frac{16 \times 16}{M \times N} \bullet P(16 \times 16 \in V_p). \qquad (11)$$

It is also possible to have a combination of the above-described options. For example, an MPEG decoder may use data units of 16×4 pixels instead of 32×2, thereby reducing pixel overhead for block-based accesses. In this case, embedded video-line memories for $N = 2$ are used to convert the blocks into video lines. Consequently, the pixel overhead for reading video lines is not zero, but much smaller than 300% as in case without video-line memories.

D. Overall occurrence probabilities

Table II shows the occurrence probability of each block type, considering all memory requests performed by the MPEG-2 decoder. The table results from using a decoder that performs line-based requests for the output display data and has a mapping of 16×2 pixels into data units. Note that the occurrence probability of the memory access for display is significant. Although it is much higher than the occurrence probability of the write requests for reconstructed MBs, the amount of data that is requested is equal. This is caused by the relation between the data-block size of $M \times 1 \in V_i$ for

TABLE II
Probability of occurrence, $P(B_x \times B_y)$.

prediction data-block requests			
block type	prob. [%]	block type	prob. [%]
$16 \times 16 \in V_p$	0.21	$16 \times 8 \in V_i$	0.30
$17 \times 16 \in V_p$	0.41	$17 \times 8 \in V_i$	0.89
$16 \times 17 \in V_p$	0.62	$16 \times 9 \in V_i$	0.40
$17 \times 17 \in V_p$	1.94	$17 \times 9 \in V_i$	1.76
$16 \times 8 \in V_p$	1.69	$16 \times 4 \in V_i$	1.02
$18 \times 8 \in V_p$	0.58	$18 \times 4 \in V_i$	0.80
$16 \times 9 \in V_p$	0.53	$16 \times 5 \in V_i$	0.81
$18 \times 9 \in V_p$	0.37	$18 \times 5 \in V_i$	0.71
write MB requests			
$16 \times 16 \in V_p$	5.12		
output read requests			
		$M \times 1 \in V_i$	81.85
Total	**11.46**		**88.54**

display of the video and $16 \times 16 \in V_p$ for writing the constructed MBs. Eq. (9) shows that the size of the requested data-blocks times the occurrence probability is constant, thus:

$$M \times 1 \bullet P(M \times 1 \in V_i) = 16 \times 16 \bullet P(16 \times 16 \in V_p).$$

It can be concluded that the pixel overhead is not merely determined by the occurrence probability of the data-block requests, but also by their size. However, the size may have an impact on the utilization of the data bus. As shown in Section III, the scheduling of all memory commands is constrained by several timing parameters. Relatively small data-block requests (<3 data units), will result in a decreased memory efficiency. Although memory command scheduling for small data block requests is beyond the scope of this paper, it can be concluded that a large amount of small data-block requests for display has a negative influence on the utilization of the memory bus.

VII. Results and conclusions

We have simulated the architecture of Fig. 10 based on an SDRAM interface for determining an optimal mapping of the video into the memory with the objective to minimize the overall memory bandwidth. The mapping is optimized for reducing the transfer overhead by measuring and analyzing the actual memory accesses, so that data dependencies are taken into account. Another issue that is important for bandwidth efficiency is the organization into memory banks, which is provided in all modern memory devices. The proposed mapping strategy increases the memory efficiency, thereby contributing to a decreased memory-bandwidth requirement.
The experiments, conducted with a large test-set of bit streams, were performed for architectures featuring a 32-bit and a 64-bit memory bus and for architectures with and without line-memories for conversion of block-based storage to line-based output. For each architecture configuration, the measured statistics were stored in a database for off-line calculation of the optimal data-unit dimensions. Subsequently, Equations (2)-(5) were applied to calculate the average pixel overhead for a given dimension (M, N) of the data units. The tables below show the simulated bandwidth numbers for various data-unit dimensions.

Table III shows the final bandwidth results for 32-byte data units, where the requests for video display are line-based. From the table it can be concluded that the mapping of 16×2 results in the smallest pixel overhead. If the reading from memory for display of the video is $(M \times N)$-based, the optimal data-unit dimensions have a more vertical preference.

TABLE III
Bandwidth results for 32-byte data units and line-based request for output.

data unit dimensions	**requested[1] data [%]**	**transferred[1] data [%]**
(32×1)	100	100 + 101
(16×2)	100	**100 + 69**
(8×4)	100	100 + 118
(4×8)	100	100 + 246

[1] 100% equals 240 MB/s for 25 Hz High-Definition video.

TABLE IV
Bandwidth results for 32-byte data units and $(M \times N)$-based request for output.

data unit dimensions	**requested data [%]**	**transferred data [%]**
(32×1)	100	100 + 101
(16×2)	100	100 + 55
(8×4)	100	**100 + 49**
(4×8)	100	100 + 65

TABLE V
Bandwidth results for 64-byte data units and line-based request for output.

data unit dimensions	**requested data [%]**	**transferred data [%]**
(64×1)	100	100 + 241
(32×2)	100	100 + 144
(16×4)	100	**100 + 142**
(8×8)	100	100 + 262

TABLE VI
Bandwidth results for 64-byte data units and $(M \times N)$-based request for output.

data unit dimensions	**requested data [%]**	**transferred data [%]**
(64×1)	100	100 + 241
(32×2)	100	100 + 130
(16×4)	100	**100 + 72**
(8×8)	100	100 + 81

For this scenario, the 8 × 4 mapping outperforms the 16 × 2 mapping. This is shown in Table IV. Table V and VI show the results for 64-byte data units. For these systems, the usage of 16 × 4 pixels for data units results in the optimal solution. However, the 32 × 2 mapping for line-based reading at the output shows a similar performance.

In recent proposals for multimedia computing architectures (e.g. [7][8][9]) video data is written line by line into the address space. This can be regarded as block-based data units with a vertical size of one ($N = 1$). For such systems, the results of the first row in the tables apply. Hence, the system with 64-byte data units consume a factor of 3.4 more memory bandwidth than requested. The proposed mapping with the optimal data-unit dimension reduces the amount of memory bandwidth for such system with 50%. For systems with 32-byte data units, the bandwidth reduces with 26% (see Fig. 13). For HDTV MPEG decoding, these numbers result in a reduction of 405 MB/s and 125 MB/s, respectively. This substantial performance improvement corresponds with a bandwidth magnitude of a complete function or application such as the display of a secondary SDTV channel or the addition of an advanced 2-D graphics application. Moreover, the presented results can be exploited to reduce the continuously growing gap between required computational power and memory bandwidth is.

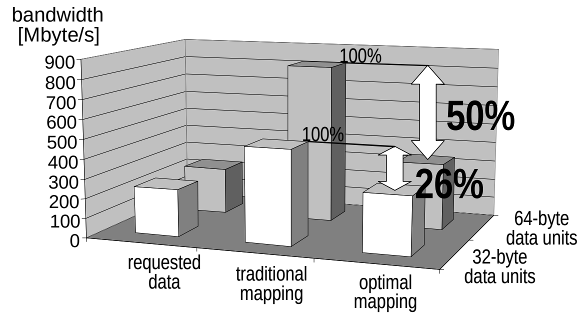


Fig. 13. Bandwidth reduction of the proposed mapping strategy.

References

[1] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach*, p. 374, Morgan Kaufmann, 2nd edition, 1996, ISBN 1-55860-372-7.

[2] www.specbench.org.

[3] B. Khailany *et al.*, "Imagine: Media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, March-April 2001.

[4] B. Davis, T. Mudge, B. Jacob and V. Cuppu, "DDR2 and low latency variant," in *Proceedings of the Workshop on Solving the Memory Wall Problem*, June 2000, www.ece.neu.edu/wall2k.html.

[5] M. Oka and M. Suzuoki, "Design and programming the emotion engine," *IEEE Micro (USA)*, vol. 19, no. 6, pp. 20–28, Nov. 1999.

[6] H. Kim and I.C. Park, "Array address translation for SDRAM-based video processing applications," in *Proc. of SPIE: Vis. Comm. and Image Proc.*, June 2000, vol. 4067, pp. 922–931.

[7] S. Rixner, *et al.*, "Memory access scheduling," *Computer Architecture News*, vol. 28, no. 2, pp. 128–138, May 2000.

[8] S. Rathnam and G. Slavenburg, "Processing the new world of interactive media," *IEEE Signal processing Magazine*, vol. 15, no. 2, pp. 108–117, March 1998.

[9] S. Dutta, D. Singh and V. Mehra, "Architecture and implementation of a single-chip programmable digital television and media processor," in *Proc. IEEE Workshop on Sig. Proc. Systems, SiPs 99, Design and Implementation*, Oct. 1999, pp. 321–330.

Egbert Jaspers was born in Nijmegen, The Netherlands, in 1969. He graduated in electrical engineering from the Venlo Polytechnic in 1992 and subsequently, he joined Philips Research Laboratories in Eindhoven. He continued his education at the Eindhoven University of Technology, and graduated (MSc) in electrical engineering in 1996. Afterwards, he joined Philips Research Labs Eindhoven, where he became a member of the TV Systems Department. There he worked on video compression for digital HDTV recording. Currently he is involved in the research of programmable architectures and their implementation for consumer systems. In 2000 he received a IEEE Consumer Electronics Section Paper Award.

Peter H.N. de With graduated in electrical engineering from the University of Technology in Eindhoven. In 1992, he received his Ph.D. degree from the University of Technology Delft, The Netherlands, for his work on video bit-rate reduction for recording applications. He joined Philips Research Labs Eindhoven in 1984, where he became a member of the Magnetic Recording Systems Department. From 1985 to 1993 he was involved in several European projects on SDTV and HDTV recording. In this period he contributed as a coding expert to the DV standardization. In 1994 he became a member of the TV Systems group, where he was leading the design of advanced programmable video architectures. In 1996, he became senior TV systems architect and in 1997, he was appointed as full professor at the University of Mannheim, Germany, at the faculty Computer Engineering. In 2000, he joined CMG Eindhoven as a principal consultant and he became professor at the University of Technology Eindhoven, at the embedded systems institute (EESI). He has written numerous papers on video coding, architectures and their realization. Regularly, he is a teacher of the Philips Technical Training Centre and for other post-academic courses. In 1995 and 2000, he co-authored papers that received the IEEE CES Transactions Paper Award. In 1996, he obtained a company Invention Award. In 1997, Philips received the ITVA Award for its contributions to the DV standard. Mr. de With is a senior member of the IEEE, program committee member of the IEEE CES and board member of various working groups.