Implementing 2D Memory Buffers for MPEG

M.B. Haverkamp
Electronic Engineering,
Hogeschool van Amsterdam,
Amsterdam, The Netherlands
MartienHaverkamp@hotmail.com
http://www.hva.nl

G. Kuzmanov S. Vassiliadis
Computer Engineering Lab,
Electrical Engineering Dept., EEMCS,
TU Delft, The Netherlands
{G.Kuzmanov, S.Vassiliadis}@EWI.TUDelft.NL
http://ce.et.tudelft.nl/

Abstract—**In MPEG applications, many of the algorithms are *data intensive* and require high levels of *data locality* and *data reusability*. A crucial performance bottleneck is the enormous data bandwidth the involved algorithms require. We focus on improving the speed of hardware MPEG decoders by using a 2-dimensional storage structure as part of a dedicated memory organization. The 2D storage makes the accesses to *rectangular blocks* of data more efficient. This is achieved by reduced number of memory accesses and improved data bandwidth utilization. The paper presents a generic structural design of the 2D storage, realized in VHDL. Feasible dimensions of the storage structure and the corresponding speed-ups get particular emphasis in the presented research effort. Results are obtained after the VHDL code is synthesized for the recent platform FPGA technology of Xilinx – Virtex II Pro. Reported data are related to the feasible sizes of the 2D buffer in terms of reconfigurable hardware resources consumed. Experimental data are compared for a 24x24 bytes 2D data storage and block patterns of 8x8 bytes versus linear memory with data bandwidth of 8, 16 and 32 bits. At reasonable hardware costs, the speed-up, estimated by simulations, may reach in some of the experimental cases up to a factor of 39. Structured tabular data are presented and can be utilized for taking design decisions with respect to different initial constraints and requirements.**

Keywords— **MPEG, memory hierarchy, memory buffer, VHDL, FPGA**

I. Introduction

The fast development pace of new visual data compression standards is dramatically increasing the system performance requirements. Very intensive data transfers cause the memory access bandwidth to be a severe performance bottleneck in a contemporary multimedia processing system. In this paper, we discuss some FPGA implementation tradeoffs regarding the design of a fast memory structure with increased bandwidth. We aim at speeding up MPEG processing in hardware and especially focus on the most demanding motion estimation and motion compensation algorithms involved. By Exploiting two important issues of these algorithms, namely *data locality* and *data reusability*, we have implemented a specific 2D-organized memory buffer between a conventional *linearly addressable memory* (LAM) and the dedicated processing block(s). We investigate feasible sizes of the 2D memory in terms of hardware resources required for an FPGA implementation. Another important estimation criterion of the design is the achievable data access speedup for considered MPEG algorithms.

The investigated designs have been described in VHDL and synthesized for the Virtex II Pro technology of Xilinx with the ISE 5.1 synthesis tools. Tabular data are presented to illustrate various synthesis results with respect to different design parameters.

- Results indicate reasonable hardware costs between 0,1% and 60% of the 2VP50 FPGA resources for a range of implementations with output bandwidth of 4, 8, 16, 4x4, 8x8 and 16x16 bytes.
- We estimate speedups between 2 and 39 versus pure LAM organizations with data words ranging between 8 and 32 bits.

The remainder of the paper is organized as follows. Section II presents some minimal required background information and introduces the problem by means of a motivating example. Section III describes the proposed design. Synthesis results for different design implementations are reported in Section IV. Finally, Section V concludes the discussion.

II. Background and Problem Description

Typically, video information is presented as a scanned sequence of pixels in a two dimensional visual plane. In digital video systems, this information is usually stored in LAM and displayed later as two-dimensional frames. In MPEG standards, video data are processed and modified between the scan and display phases. Most of the data processing in these standards, however, is not performed over individual pixels, but rather over certain

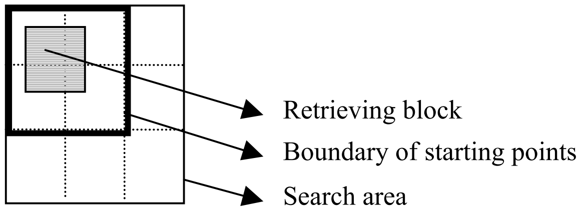

Figure 1. Search area and 'retrieving block' in MPEG.

regions (blocks) of pixels in a frame [5]. This may lead to some performance drawbacks due to problems with data alignment and accessibility into systems memory. We consider the full search for determining motion vectors in MPEG, which is the most data throughput-demanding algorithm. In full search, the 'retrieving block' is placed on every possible location inside a predefined search area. These locations are not restricted to any block position. A 'retrieving block' is a block of data, which will be compared in the search algorithm with a block from another frame. As motion estimation criteria, the Sums of Absolute Differences between the 'retrieving block' and the blocks within the search area are calculated. The bold square shows the boundary of the starting points (upper-left pixel) of the retrieving block in Figure 1. The following example illustrates the block addressing problems and motivates our research.

Motivating example. Assume a LAM and a pixel plane divided into 2x2 blocks, where each pixel is represented by a byte (see Figure 2a)). Note that in linear addressing spaces the basic addressable units are bytes and words. Further, assume that video information is memory aligned in a typical scan-line manner and the system is capable of accessing 4 bytes in a cycle. Obviously, neither of the blocks containing pixels 0, 1, 16, 17 and 17, 18, 33, 34 is accessible by a single memory transfer, because they are not aligned into consecutive memory locations (see Figure 2b)). Even though a 32-bit memory bandwidth is available, redundant data fetches cannot be avoided. Another approach may be to reorder data into the LAM. If we align each block into consecutive bytes (Figure 2c)), we will be able to access some blocks in a single memory cycle (e.g., pixels 0, 1, 16, 17). In MPEG, however, some of the most demanding algorithms (e.g., motion estimation, referred earlier) require accessing block data at an arbitrary position in the frame, thus in memory. Figure 2c) suggests that in such cases data fetching may become even less effective than the scan-line alignment scheme (e.g., accessing block 17, 18, 33, 34). Further, for conciseness, we will refer to blocks like 0, 1, 16, 17 on Figure 2a) as aligned, and to the remaining blocks (like 17, 18, 33, 34) as non-aligned blocks.

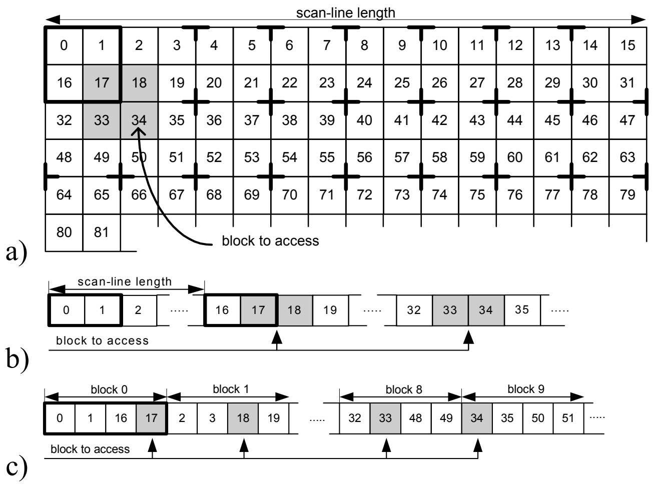

Figure 2. Block accessing problem example

The example indicates that different memory organizations and addressing modes may be vital for the performance of visual data processing algorithms. Obviously, a new memory organization should be utilized to solve the discussed problems.

III. Proposed Design

Figure 3 depicts a general view of the memory hierarchy considered for our design. The width of the *LAM* data bus is denoted by *w,* while *m* denotes the data width of the interfacing bus between the *2 Dimensionally Addressable Memory (2DAM*) and the processing elements (*Dedicated process*). T_{LAM} is the time to access LAM and T_{2DAM}- time to access the 2D memory. The 2D memory storage is a buffer that allows the reduction of redundant data transfers from the LAM, thus the available memory bandwidth is utilized more efficiently. On the other hand, the most severe bandwidth demands are imposed on transfers over highly localized data in the *2D addressable memory* (2DAM). Therefore, data bandwidth between the 2DAM and the dedicated processing block (i.e., w < m in Figure 2) should be increased. Two design issues help the performance improvement:

1. More efficient bandwidth utilization.
2. Reduced number of data transfers between memory hierarchy levels.

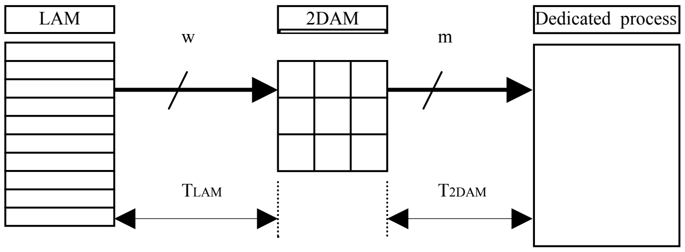

Figure 3. Considered memory hierarchy

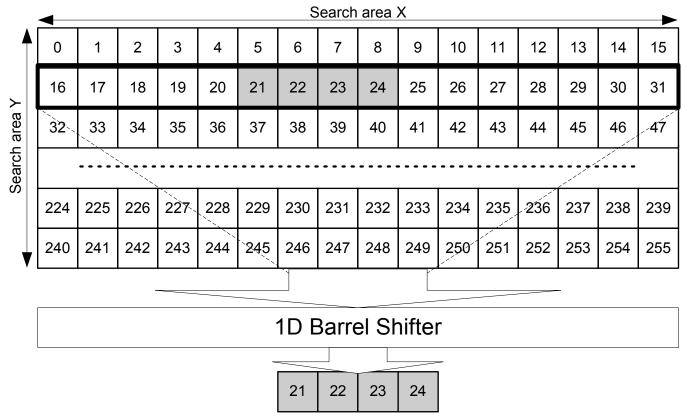

Figure 4. A single 1D barrel shifter implementation.

To implement the 2DAM, several approaches can be chosen. Many designs, originally dedicated for vector processors and display systems based on multiple memory module organizations can be adopted [1],[3],[4],[6]. Such designs, however, are suitable for large storage volumes. We consider motion estimation search algorithms, which are typically dealing with relatively smaller search areas. Therefore, we decided to use register storage with the dimensions of the search area, which can be easily implemented with the available FPGA flip-flops. 2 D addresses, horizontal X, and vertical Y, determine the location where a particular piece of data should be stored into the 2D register storage. Barrel shifters are used to access the output data. Since barrel shifters are relatively expensive to implement, we investigated a range of output designs. One extreme is to implement a single, 1D barrel shifter with input length equal to one of the search area dimensions, as depicted in Figure 4. Another option is to implement two levels of barrel shifters, thus implementing a 2D barrel shifting operation. Such a design includes in essence several 1D barrel shifters and its extreme implementation will include *D+1* barrel shifters, where *D* x D are the dimensions of the search area. Figure 5 sketches the 2D barrel shifter structure.

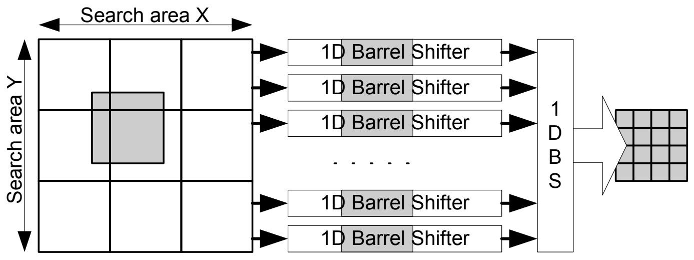

Figure 5. 2D barrel shifter implementation.

IV. Synthesis Results and Evaluation

The design was described in VHDL and synthesized for Virtex II Pro FPGA technology using different design parameters. Synthesis results for a 24x24 search area implementation are presented in Table I. These results indicate reasonable implementation costs.

Speedup evaluation. Assume a LAM with word length of w bits (*w*= 8, 16, 32) and time for a LAM access T_{LAM}. Consider square *n x n* data patterns to be accessed. We note that the following equations can be generalized for rectangular patterns, as well. The time (in [LAM cycles]) to access *N n x n* blocks of 8-bit pixels each varies depending on the block alignment:

All N blocks aligned $$\frac{8 \cdot n^2}{w} \cdot N \qquad (1)$$

Neither of the N aligned: $$\left(\frac{8 \cdot n^2}{w} + n \right) \cdot N \qquad (2)$$

Mixed: $$\left(\frac{1}{n} \cdot \frac{8 \cdot n^2}{w} + \frac{n-1}{n} \cdot \left(\frac{8 \cdot n^2}{w} + n \right) \right) \cdot N =$$
$$= \left(\frac{8 \cdot n^2}{w} + n - 1 \right) \cdot N \qquad (3)$$

By *mixed* access scenario we mean accessing both aligned and non-aligned blocks considering the full search algorithm. To derive (3), we assumed the number of accesses to aligned blocks in LAM to be 1/n from all accesses. Table II presents the normalized numbers of *LAM* access per block for some typical values of *n* and *w*. Three scenarios are considered: worst case (WC) – neither of the N blocks is aligned; (Mix) - Mixed scenario; and best case (BC) - all blocks are aligned.

TABLE I

SYNTHESIS RESULTS FOR VIRTEX II PRO 50 FPGA

D	1D = **24**						2D = **24x24**					
M	**4**		**8**		**16**		**4x4**		**8x8**		**16x16**	
# of Slices:	0,91%	224	1,19%	293	1,56%	384	15,5%	3822	31,3%	7715	62,6%	15428
# of Slice Flip Flops:	0,06%	32	0,13%	64	0,26%	128	0,1%	32	0,3%	128	1,0%	512
T_{2DAM} (in ns)		5,4		7,8		7,5		25,5		25,6		25,8

TABLE II.

NUMBER OF LAM ACCESSES PER N X N BLOCK.

N	W	WC	Mix	BC
8	**8**	72	71	64
	16	40	39	32
	32	24	23	16
16	**8**	272	271	256
	16	144	143	128
	32	80	79	64

Assuming a square search area with dimensions *D x D*, the number of all possible searches Nfs is calculated as:

$$N_{fs} = (D - n + 1)^2 \tag{4}$$

Consider the full search algorithm, i.e., a mixed access scenario. In a pure LAM implementation, the total number of memory cycles, required to perform one full search in the entire area is:

$$N_{LAM} = \left(\frac{8 \cdot n^2}{w} + n - 1 \right) \cdot N_{fs} \tag{5}$$

In the considered memory hierarchy, we should load all required aligned blocks into the 2DAM from the LAM first and then perform the full search accesses. The number of aligned blocks (N_{ab}) in the 2D memory is estimated to be:

$$N_{ab} = \left(\frac{D}{n} \right)^2 \tag{6}$$

Thus, the number of cycles required for loading the 2DAM from the LAM is:

$$\frac{8 \cdot n^2}{w} \cdot \left(\frac{D}{n} \right)^2 \tag{7}$$

The total number of cycles, spent for accessing N_{fs} blocks out of the 2DAM storage depends on the implemented width of its output bus, i.e., on the design parameter *m*. Thus, the number of cycles for a full search in the considered memory hierarchy when 1D barrel shifting access is implemented will be:

$$N_{2DAM} = \frac{8 \cdot n^2}{w} \cdot \left(\frac{D}{n} \right)^2 + N_{fs} \cdot \frac{n^2}{m} \tag{8}$$

For the 2D barrel-shifting access, when n barrel shifters are implemented in parallel, we assume that the 2DAM output bandwidth is *m x m pixels*, thus the total number of access cycles will be:

$$N_{2DAM} = \frac{8 \cdot n^2}{w} \cdot \left(\frac{D}{n} \right)^2 + N_{fs} \cdot \frac{n}{m} \tag{9}$$

We define technology independent speedup S_N assuming T_{LAM}=T_{2DAM} :

$$S_N = \frac{N_{LAM}}{N_{2DAM}}$$

Taking into account the implementation technologies and utilizing (8), the actual speedup for 1D implementation is:

$$S_{ACTUAL} = \frac{N_{LAM} \cdot T_{LAM}}{\frac{8 \cdot n^2}{w} \cdot \left(\frac{D}{n} \right)^2 \cdot T_{LAM} + N_{fs} \cdot \frac{n^2}{m} \cdot T_{2DAM}}$$

The actual speedup achievable by 2D barrel shifting, considering (9), is estimated to be:

$$S_{ACTUAL} = \frac{N_{LAM} \cdot T_{LAM}}{\frac{8 \cdot n^2}{w} \cdot \left(\frac{D}{n} \right)^2 \cdot T_{LAM} + N_{fs} \cdot \frac{n}{m} \cdot T_{2DAM}}$$

Both S_N and S_{ACTUAL} are presented in Table III. Obviously, the highest speedup can be observed when the ratio $\frac{w}{m}$ is greater.

This means, *the greater the 2DAM bandwidth is compared to the LAM bandwidth, the greater the speedup*. For the 24x24 implementation, the speed up can reach up to 39, when a 16x16 pattern is accessed and the LAM data bus is 8 bits wide. The same pattern will be accessed almost 13 times faster than a pure LAM implementation, if the LAM is 32-bit wide.

V. CONCLUSIONS

We presented scalable memory storage with a high output bandwidth, which is dedicated for MPEG hardware implementations. The storage has been designed as an integral part of a specific memory hierarchy and is capable of accessing randomly aligned data patterns out of a virtual 2D organized data. Synthesis results from an FPGA implementation indicate considerable speedups, compared to traditional linearly addressable memories at reasonable hardware costs. The design can be utilized to support specific memory architectures like [2], and can be especially beneficial in new FPGA based custom computing machines, for example see [7].

ACKNOWLEDGMENTS

This research has been supported by PROGRESS, the embedded systems research program of the Dutch organization for scientific research NWO, the Dutch Ministry of Economic Affairs, and the Technology Foundation STW (project AES.5021).

TABLE III.

SPEEDUP ESTIMATIONS FOR T_{LAM}=10NS, N X N=8 X 8, AND D X D=24 X 24

W		8			16			32		
M		4	8	16	4	8	16	4	8	16
N_{LAM}		20519	20519	20519	11271	11271	11271	6647	6647	6647
N_{2DAM}	1D	5200	2888	1732	4912	2600	1444	4768	2456	1300
	2D	1154	865	720,5	866	577	432,5	722	433	288,5
S_N	1D	3,9	7,1	11,8	2,3	4,3	7,8	1,4	2,7	5,1
	2D	17,8	23,7	28,5	13,0	19,5	26,1	9,2	15,4	23,0
T_{2DAM}, [ns]	1D	5,4	7,8	7,5	5,4	7,8	7,5	5,4	7,8	7,5
	2D	25,5	25,6	25,8	25,5	25,6	25,8	25,5	25,6	25,8
S_{ACTUAL}	1D	6,68	9,77	20,41	3,67	5,37	11,21	2,17	3,17	6,61
	2D	10,01	19,95	39,74	5,50	10,96	21,83	3,24	6,46	12,87

REFERENCES

[1] P. Budnik and D. J. Kuck. The organization and use of parallel memories. *IEEE Transactions on Computers*, 20(12):1566-1569, December 1971.

[2] G.Kuzmanov, S.Vassiliadis, J. van Eijndhoven “A 2D Addressing Mode for Multimedia Applications”, Proc. *"Workshop on System Architecture Modeling and Simulation (SAMOS 2001)"*, Samos, Greece, July 2001, pp. 291-306. (in LNCS 2268)

[3] D.H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, C-24(12):1145-1155, December 1975.

[4] D. Lei Lee. Scrambled Storage for Parallel Memory Systems. In *Proc.IEEE International Symposium on Computer Architecture*, pages 232-239, Honolulu, HI, USA, May 1988.

[5] Y.Q. Shi and H. Sun *Image and Video Compression for Multimedia Engineering.* Boca Raton CRC Press, 2000.

[6] D.C. van Voorhis and T. H. Morrin. Memory systems for image processing. *IEEE Transactions on Computers*, C-27(2):113-125, February 1978.

[7] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN ρμ -coded processor. In *11th International Conference on Field Programmable Logic and Applications (FPL)*, pages 275-285, Belfast, Northern Ireland, UK, August 2001.