A PROGRAMMABLE MOTION ESTIMATION PROCESSOR FOR FULL SEARCH BLOCK MATCHING

Alain Pirson[1], Fathy Yassa[1*], Philippe Paul[1], Barth Canfield[2], Friedrich Rominger[3], Andreas Graf[3], Detlef Teichner[3]

[1]Thomson Consumer Electronics Components, Meylan, France
[2]Thomson Consumer Electronics, Indianapolis, USA
[3]Thomson Consumer Electronics, VS-Villingen, Germany

ABSTRACT

This paper describes a programmable motion estimation processor applying a block matching technique on large search windows.

Developed in the context of an MPEG-2 video encoder, its use can be extended to any application where fast motion estimation is required. Its high performance (17 Gops peak) and its ability to work in parallel make it ideal for real time applications like video compression.

The *Block Matching Processor (BMP)* consists of a CPU associated with several specific units including a fast motion estimator, a DRAM interface, IO ports and some on-chip memory. This approach allies the flexibility of a CPU to the efficiency of dedicated hardware. A DRAM controller minimizes the impact of data transfer on the computing power.

1. INTRODUCTION

The growing need for handling digitized video sequences is met using compression techniques. The high interest in such techniques led the International Organization for Standardization (ISO) to define some standards. A very important one is the recently emerged MPEG-2 standard which aims at providing high compression ratio while keeping good picture quality [1].

MPEG-2 rests on an algorithm which puts together a group of techniques specially adapted to video compression. It implements among others a very efficient block-based motion compensation algorithm which involves a rich set of prediction modes (using field or frame data, past or future pictures...). The compression ratio and the quality of the decoded pictures depend on the quality of the prediction which is tightly related the quality of the motion estimation. On the other hand, real-time video processing requires high performance systems.

* This work was completed while with TCEC (Thomson Consumer Electronics Components). Currently joined Motorola IC development Centre, Geneva, Switzerland.

In conclusion, a motion estimation processor for MPEG-2 encoding must provide good quality estimation at high performance. It must also be flexible to support various modes. These general requirements can be applied to a wide number of applications. They constitute the major characteristics of the processor described in this paper.

2. IMPLEMENTED ALGORITHM

Among motion estimation techniques, block matching seems to be one of the most appropriate methods. It is consistent with the block-based approach of the MPEG-2 standard. It is also one of the most suitable techniques for VLSI implementation which is a major concern for real-time video applications.

This technique consists in moving a source block around in a search window. For each position, an error is calculated. The best matching corresponds to the position leading to the smallest error. The search strategy and the range determine the quality of the estimation. Our requirements led us to consider a full search strategy on large search windows. However, a simple calculation shows that the computing power is excessively high for suitable ranges. That is why we decided to adopt a hierarchical approach. It consists of a multi-level motion estimation with refined search accuracy at each level. At coarse levels, the block matching is applied on decimated pictures. This approach reduces the amount of calculation while maintaining the precision.

The motion estimation technique implemented in our processor is a 4x8, 8x8, 8x16, 16x16 pels block matching using a regular full search strategy. The matchings are carried out on a search area limited by a maximum range in both directions. This range is programmable. The error function is a *sum of absolute errors (SAE)* between matched pixels.

3. MOTION ESTIMATION HARDWARE

3.1. Block matching array

Systolic arrays seem to be ideal candidates for a VLSI implementation of block matching algorithms using regular search

0-7803-2431-5/95 $4.00 © 1995 IEEE

strategies. Different architectures have been proposed in the literature. In our MPEG-2 encoder, the size of the source block is fixed and small. This means the source block can be stored on-chip during the whole search process. This has a strong impact on the architecture since the whole IO bandwidth can be allocated to the search window input. It led to the choice of the structure presented on figure 1.

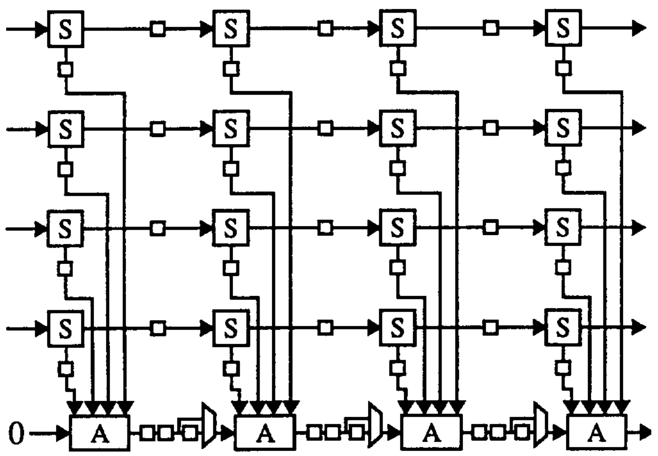
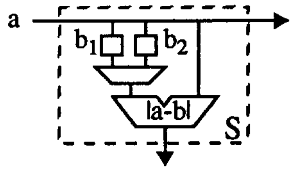


figure 1: block matching array

This structure is derived from the systolic approach. It contains two types of identical processing elements (S and A) spatially organized in a regular way. All the processing elements work synchronously. Each S processing element is loaded with a source block pixel. In fact, two source blocks can be stored in this way. The search windows are input stripe by stripe on the left side of the array. Each S processing element calculates the absolute error between an incoming search pixel (a) and one of the source pixels (b_1 or b_2) stored locally (figure 2.a). The A cells are 5-input adders used to sum the intermediate results.



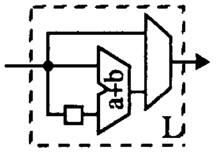


figure 2.a: S cell figure 2.b: L cell

The block matching array produces one result every cycle. The consecutive SAEs *(sum of absolute errors)* correspond to the motion resulting from a horizontal shift of the source block one position to the right every cycle (figure 3). A number of stripes may be submitted to the array to cover the entire search window.

There is no predifined order to scan the stripes. However, it is advantageous to adopt a progressive order where each new stripe corresponds to the previous one shifted one row down. The data exchange can be significantly reduced by using on-chip line memories since only one row needs to be updated between consecutive stripes.

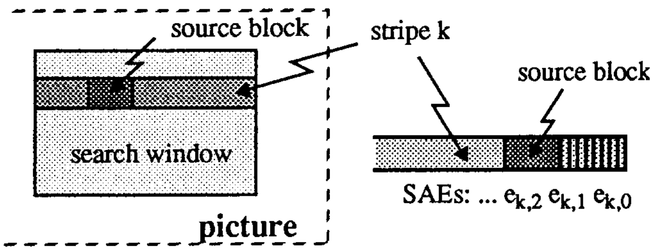


figure 3: computation with the array

The structure presented on figure 1 allows two source blocks to be stored at the same time in the network, thanks to a double buffering (figure 2.a). A control signal determines which block is actually used in the current computation. This capability offers two advantages:

- Any search stripe present in local memories can be used twice, once for each block. This feature reduces again the bandwidth by a factor of 2.

- It is possible to interlace two block matching operations at pixel level, by switching from one source block to the other every cycle and loading alternatively the columns of the search stripes. This *"interlaced"* mode allows block matching operations with source blocks twice as big as the array. For example, a 4x8 array can achieve 8x8 block matching.

In normal mode, two registers are necessary between the A cells. A third register is required for the interlaced mode. In this mode, a final stage is also needed at the output of the array to add together the contribution of the interlaced matchings. This last stage (L) is shown in figure 2.b. It incorporates a bypass for the normal mode.

3.2. Mincell

The SAEs produced by the array are forwarded to a block, the **mincell**, in charge of finding the smallest SAE and its related motion vector. It compares the consecutive SAEs and keeps the minimum. In the case of equality, the vector having the smallest magnitude is kept.

The mincell needs the relative coordinates of the upper left corner of the search window as seed vector. This seed vector is loaded before starting a search process.

4. BLOCK MATCHING PROCESSOR

Up to now, we have described the architecture of a block matching unit. Let us go one step further and integrate this unit into a block matching processor.

Our major requirement is computing power coupled to a flexible control architecture. Having this in mind, we decided to adopt a classical DSP approach based on a CPU core with the following dedicated units associated to it:

- a block matching unit (BMU),
- a motion vector storage buffer (MV buffer),
- an 8-bit input port,
- a serial port to output motion vectors (output port),
- a micro-processor interface for control.

This architecture is shown on figure 4.

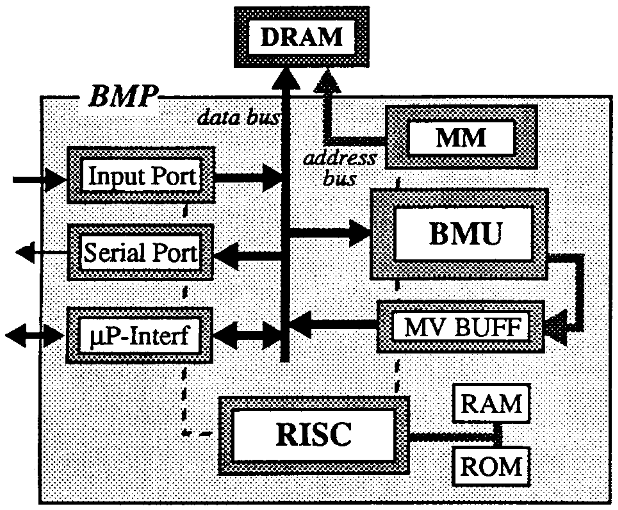


figure 4: BMP architecture

The dedicated units share a common bus connected to an external memory. They all have some local storage available (RAM or FIFO). The bus allocation is handled by a memory manager (MM) under control of the CPU.

Each BMP can address a dynamic memory (DRAM). Page mode is intensively used to maximize the bandwidth.

The consecutive pictures are loaded through the input ports and stored in the DRAM. These pictures can be used as source pictures (set of blocks to be matched) or as anchors (search pictures).

The source blocks are processed by the BMU. For that purpose, a group of them are fetched from the DRAM and loaded in the BMU. After having written the seed vectors, a search process is launched. It performs motion estimation with anchor data read from the DRAM. When the search process is completed, the computed vectors are stored in the motion vector buffer to be sent to the DRAM, while the procedure is restarted with other source blocks.

The motion vectors will be output on the serial line when requested. They are also accessible through the micro-processor interface.

4.1. On chip controller

The heart of the BMP is a RISC (Reduced Instruction Set Computer) processor. This processor is a one-stage pipe-line Harvard machine[1], executing one instruction every clock cycle. It contains a register file as well as a data memory. The operative unit consists of an ALU associated with a barrel shifter. Each instruction is fetched from one of two on-chip program memories: a ROM and a RAM down-loaded through the micro-processor interface.

The RISC processor controls all the specific units of the BMP. It activates them by launching tasks and responds to their requests (i.e. end of task, data ready...). For this purpose, some special instructions were added to its instruction set. The RISC processor is also in charge of computing the parameters the specific units need, like address of data for the memory manager, search range or seed coordinates for the motion estimator...

4.2. Block matching unit

The block matching unit implements the above concepts as follows (figure 5).

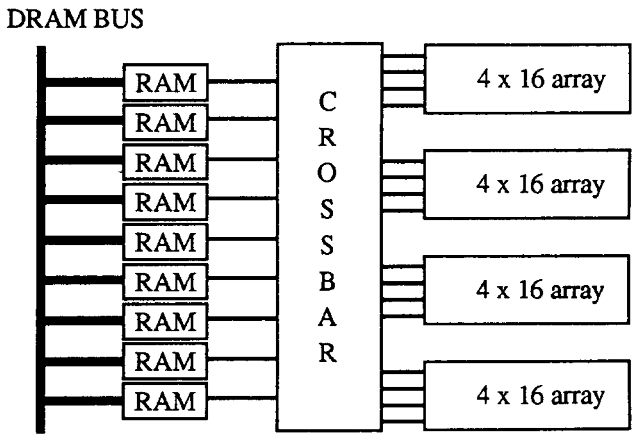


figure 5: block matching unit

1. A Harvard machine is a processor with separated data and program flows (they do not share the same bus and the same memory)

This architecture implements four 4x16 arrays. This organization is attractive because it supports all the target matching sizes. Indeed, the global structure can be configured as:

- eight 4x8 arrays (4 cascades of two 4x8 arrays),
- four 8x8 arrays (2 cascades of two 8x8 arrays),
- two 8x16 arrays,
- two 8x16 arrays in interlaced mode to achieve two 16x16 block matchings[1].

The four arrays are connected to 9 line memories through a crossbar. The size of the memories determines the widest search window which can be processed.

The crossbar connects the right memories to the arrays. In mode 4xX, only 5 memories are used at the same time. Four of them feed the network whilst the fifth one is loaded with data from external memory. In mode 8xX the 9 memories are used, 8 of them to feed the network. The 16x16 interlaced mode refers to the same organization as mode 8xX. The search range is twice as small.

4.3. Input ports

The BMP implements an 8-bit input port used to load the source and the anchor pictures. This port includes a decimation device necessary for hierarchical motion estimation. The data sent to the port are stored in a buffer and shifted to the DRAM by the memory manager in fast burst mode.

4.4. Serial port

The BMP also includes a serial port used to output the computed motion vectors. This port consists of a buffer connected to a synchronous serial line.

4.5. UP interface

The BMP includes an 8-bit bidirectional micro-processor interface which allows an external processor to control the BMP. This interface enables the exchange of information between the external processor and several units of the BMP.

It permits the down-loading of the RISC program. It allows the external processor to pass some parameters to the RISC processor via some shared registers mapped in the interface. The RISC processor can also send some information to the external processor through a dedicated interface register.

1. The 16x16 mode is based on two 8x16 arrays using interlaced mode and working in parallel because only 8 input buses are available. The performance is halved. However, the global performance remains unchanged since two 8x16 arrays are used in parallel.

5. CONCLUSION

The processor described in this paper attains the expected performance of 17 Gops peak. Of course, the real performance is application dependent. It is related to the user's ability to exploit the integrated hardware. For example, the need to optimize the code is of capital importance.

REFERENCES

[1] Coding of moving pictures and associated audio, MPEG-2/System, ISO/IEC JTC1/SC29/WG11.

[2] Special issue on VLSI implementations for digital image and video processing applications, IEEE Transactions on Circuits and Systems, Oct. 1989.

[3] R. Saint Girons at al., "MPEG++ a robust compression and transport system for digital HDTV", IEEE International Symposium on Circuits and Systems, LA California, May 1992.

ACKNOWLEDGMENTS

We want to acknowledge F. Kazi (TCEC Meylan, France) for his help in preparing this paper.

We also want to thank M. Henri (SGT Thomson Grenoble, France) for his constructive criticism regarding the architecture.