World Academy of Science, Engineering and Technology
International Journal of Electrical and Computer Engineering
Vol:2, No:10, 2008

A Pipelined FSBM Hardware Architecture for HTDV-H.26x

H. Loukil, A. Ben Atitallah, F. Ghozzi, M. A. Ben Ayed, N. Masmoudi

***Abstract*—**In MPEG and H.26x standards, to eliminate the temporal redundancy we use motion estimation. Given that the motion estimation stage is very complex in terms of computational effort, a hardware implementation on a re-configurable circuit is crucial for the requirements of different real time multimedia applications. In this paper, we present hardware architecture for motion estimation based on "Full Search Block Matching" (FSBM) algorithm. This architecture presents minimum latency, maximum throughput, full utilization of hardware resources such as embedded memory blocks, and combining both pipelining and parallel processing techniques. Our design is described in VHDL language, verified by simulation and implemented in a Stratix II EP2S130F1020C4 FPGA circuit. The experiment result show that the optimum operating clock frequency of the proposed design is 89MHz which achieves 160M pixels/sec.

***Keywords*—**SAD, FSBM, Hardware Implementation, FPGA.

I. Introduction

IN the last few years, video coding systems have been assuming an increasingly important role in several application areas tied in with digital television, video-phone and video-conference, video-surveillance and with the storage of video data. Several video compression standards have been established for these different application [1], exploiting both spatial and temporal redundancies of video sequence to achieve the required compression rates. Among these technique, motion estimation has proved to be a fundamental technique to improve inter-frame prediction in video coding. It is often the case that video frames that are close in time are also similar. Therefore, when coding a video frame, it would be judicious to make as much use as possible of the information presented in a previously coded frame. One approach to achieve this goal is to simply consider the difference between the current frame and a previous reference frame, as shown in Fig. 1, and code the difference or residual. When the two frames are very similar, the difference will be much more efficient to code than coding the original frame. In this case, the previous frame is used as an estimate of the current frame. A more sophisticated approach to increase coding efficiency is to work at the macroblock (NxN pixels) level in the current frame, instead of processing the whole frame all at once as described above.

H. Loukil Is With University Of Sfax, National School Of Engineering, Bp W, 3038 Sfax, Tunisia. E-Mail : Hassenloukil@Gmail.Com

A. Ben Atitallah Is With University Of Sfax, High Institute Of Electronics And Communication, Bp 868, 3018 Sfax, Tunisia. E-Mail : Ahmad.Benatitallah@Isecs.Rnu.Tn

F. Ghozzi Is With University Of Sfax, High Institute Of Electronics And Communication, Bp 868, 3018 Sfax, Tunisia. E-Mail : Fahmi.Ghozzi@Isecs.Rnu.Tn

M. A. Ben Ayed Is With University Of Sfax, High Institute Of Electronics And Communication, Bp 868, 3018 Sfax, Tunisia. E-Mail : Mohamedali.Benayed@Isecs.Rnu.Tn

N. Masmoudi Is With University Of Sfax, National School Of Engineering, Bp W, 3038 Sfax, Tunisia. E-Mail : Nouri.Masmoudi@Enis.Rnu.Tn

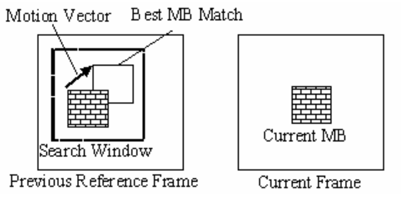

Fig. 1 Block-matching algorithm

The process is called motion compensated prediction, and is based on the assumption that most of the motion that the macroblocks (MB) undergo between frames is a translational motion. This approach attempts to find, for each NxN luminance block of a MB in the current frame, the best matching block in the previous frame. A search window is usually defined and bounds the area within which the encoder can perform the search for the best matching block. The motion of a MB is represented by a motion vector that has two components; the first indicating horizontal displacement, and the second indicating vertical displacement. Different criteria could be used to measure the closeness of two blocks [2]. The most popular measure is the Sum of Absolute Differences (SAD) [3], [4] defined by "(1)".

$$SAD = \sum_{i=0}^{15} \sum_{j=0}^{15} \left| Y_{k,l}(i,j) - Y_{k-u,l-v}(i,j) \right| \quad (1)$$

Where $Y_{k,l}(i,j)$ represents the $(i,j)th$ pixel of a 16 x 16 MB from the current picture at the spatial location (i,j) and $Y_{k-u,l-v}(i,j)$ represents the $(i,j)th$ pixel of a candidate MB from a reference picture at the spatial location (k,l) displaced by the vector (i,j).

To find the MB producing the minimum mismatch error, we need to compute SAD at several locations within a search window. This approach is called full search or exhaustive search, and is usually computationally expensive, but on the other hand yields good matching results. To perform FSBM algorithm, we must execute $(2p+1)^2$ SAD functions. As we can see that FSBM algorithm is very complex in term of computation, which can be a significant problem in a real time video coding using software solution [5], [6]. There are several block-matching algorithms (BMAs) [7]-[9] that can be used for motion estimation but the FSBM algorithm is preferred due to their relative simplicity, low-control overhead and achieves optimal performances in terms of PSNR (Peak Signal to Noise Ratio) for a given compression factor. Nowadays configurable Field Programmable Gate Array (FPGA) technology is able to execute complex embedded video processing in real time. Thus, to reduce complexity and to take advantage of the FSBM algorithm a pipelined hardware implementation in FPGA of this algorithm is proposed. In our study, we suppose that MB size is 16x16 and search area is 32x32 pixels wide. Therefore, around current MB in current frame, we insert 8 pixels (p=8). Generally, current MB and search area have an NxN an (N+2p)x(N+2p) pixel-size respectively as shown in Fig. 2.

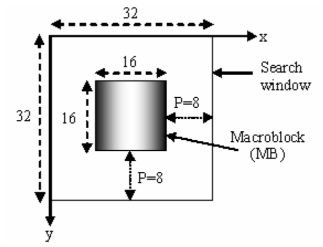

Fig. 2 The current MB position in the search area

This paper is organized as follows. Section 2 presents different hardware architectures for FSBM. Section 3 describes our proposed hardware architecture for FSBM algorithm. The simulation and synthesis results for all architectures are presented and discussed in section 4. Finally, section 5 concludes the paper.

II. DIFFERENT ARCHITECTURE FOR FSBM

In literature, different architectures are proposed to implement the FSBM algorithm [11]-[16], but these architectures have in important clock cycles number to compute the motion vector. This high number of clock makes these architectures unsuited to achieve for example the processing requirements of high definition TV (HTDV 1080i, 1920x1088@60Hz) which requires 125M pixels/sec. This section presents briefly these different architectures and our pipelined hardware architecture.

A. T. Komarek and P. Pirsch Architecture

All figures in this section are represented for N=3 and p=2. All architectures in this section are composed by 4 components:

- AD_i: calculate the Absolute difference value between tow pixels and accumulate present value with previous value.
- R : Registers allows the data synchronization
- A: Accumulator.
- M: Comparator.

1) AB1 Architecture

Fig. 3 represents AB1 architecture. It's composed by N "AD", (2xN +1) registers, one accumulator and one comparator. For calculate different SAD's (N=16 and p=8), we use 16 ADs, 33 registers, one accumulator and one comparator. Each AD has one input for MB and an others input for search window. To calculate $(2p+1)^2$ SAD's we use 9250 clock cycles. The detail of intermediate operation of this architecture is described in [11].

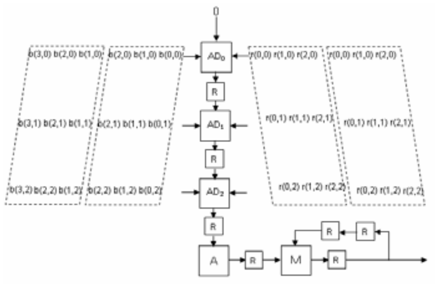

Fig. 3 AB1 Architecture

2) AB2 Architecture

Fig. 4 represent AB2 architecture. It's composed by NxN "AD", (NxN+(N-1)xN+2N+1) registers, N accumulators and one comparator.

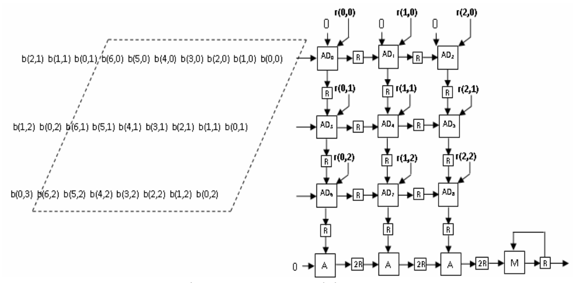

Fig. 4 AB2 Architecture

For N=16 and p=8, we use 256 ADs, 529 registers, 16 accumulator and one comparator for calculate $(2p+1)^2$ SAD's. Each AD have one input for search window. The interesting point in this architecture is the storage of MB pixels in Each AD. With this idea, for calculate all SAD's and the motion vector, we use 579 clock cycles. The detail of intermediate operation of this architecture is described in [11].

3) AS1 Architecture

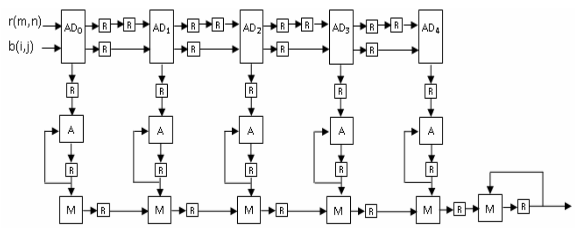

Fig. 5 AS1 Architecture

Fig. 5 represents AS1 architecture. It's composed by Ψ "AD" (Ψ is equal to number of displacement in search window), (3x (Ψ-1) + 3x Ψ +1) registers, (Ψ) accumulators and (Ψ+1) comparator. For N=16 and p=8, we use 17 ADs, 100 registers, 17 accumulators and 18 comparators. For all ADs, we have on input for MB and other input for search window. The detail of intermediate operation of this architecture is described in [11]. For calculate all SAD's and the motion vector, we use 8722 clock cycles.

4) Architecture AS2

Fig. 6 represents AS2 architecture. It's composed by Nx Ψ "AD", (3x(Ψ)xN-1 + Nx(Ψ-1)+ 3xΨ + 1+c) registers, Ψ accumulators and Ψ+1 comparators. (c = Ψ-(Ψ-1)+ Ψ-(Ψ-2) + …. +(Ψ-2)).

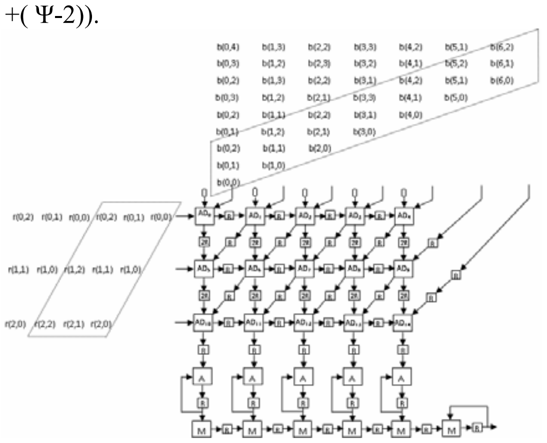

Fig. 6 AS2 Architecture

For this architecture we use 272 Ads, 685 registers, 17 accumulators and 18 comparators (N=16 and p=8). Each AD has one input for MB and an others input for search window. To calculate $(2p+1)^2$ SAD's we use 577 clock cycles. The detail of intermediate operation of this architecture is described in [11].

B. K. M. Yang and al. Architecture

This architecture is composed by N PEs, (N-1) flip-flop, N multiplexers (MUX) and one comparator for compute the minimum SAD and the motion vector. All this components is presents in fig. 7.

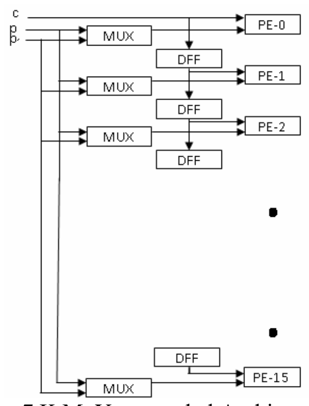

Fig. 7 K.M. Yang and al Architecture

For N=16 and P=8, we use 16 PEs, 15 DFF and 16 MUX. The intermediate operation of each PE and synchronization of data are presented in [12]. For calculate all SAD's and the motion vector, we use 4370 clock cycles.

C. H. Hsieh and al. Architecture

This architecture is composed by (NxN) processor element (PE), (NxP) Shift register (SR), Parallel Adder and comparator. Fig. 8 presents the connection between these elements.

For N=16 and P=8, we use 256 PEs, 128 SRs, one Parallel Adder and one comparator. We use 1028 clock cycles for calculate $(2p+1)^2$ SAD's and the motion vector. You can find all detail of this architecture in [13].

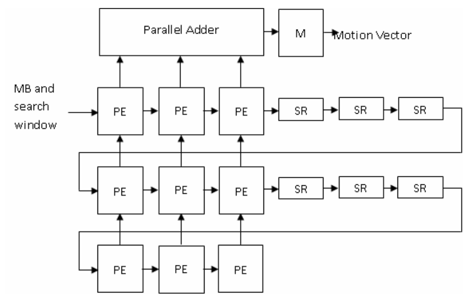

Fig. 8 C.H. Hsieh and al. Architecture

D. H. Yeo and al. Architecture

This architecture is present in fig. 9. It is composed by NxN PEs, 2 MUX, one input for MB and tow input for search window. In [14], you can find the intermediate operation of this architecture. For N=16 and P=8, we use 256 PEs and 2 MUX. The motion vector is outputted after 547clock cycles.

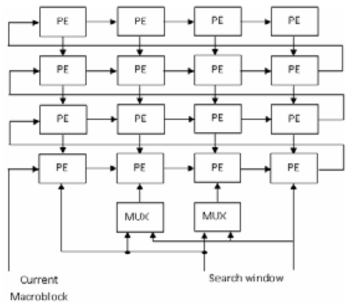

Fig. 9 H. Yeo and al Architecture

F. M. Yang and al. Architecture

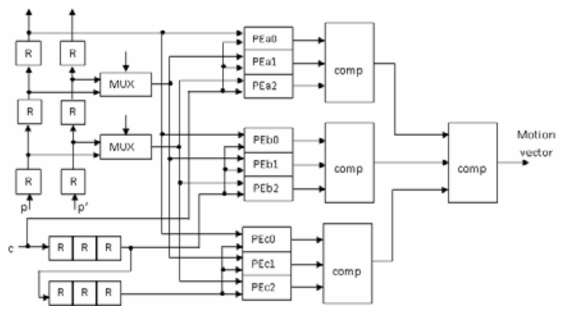

Fig. 10 F.M. Yang and al Architecture

In this architecture, you can find NxN PEs, N MUX, input for MB and tow input for search window, (N+1) comparator for obtained the motion vector. Therefore, for synchronization of data, we use N registers for MB and (2xN) registers for search window. Fig. 10 present the schematic of this architecture. In this architecture "C" is input for MB and (P,P') is input for search window.All detail for this architecture is presented in [15]. For N=16 and P=8, we use 256 PEs, 16 MUX and 48 registers. For calculate all SAD's and the motion vector, we use 534 clock cycles.

E. *Y. S. Jehng and al. Architecture*

Fig. 11 present the diagram of this architecture for N=4. It's composed by NxN "D" for computing the absolute value, (NxN -1) accumulator and one comparator. In addition, we find NxN input for MB and NxN input for search window. In [16], you can find the intermediate operation of this architecture.

For calculate all SAD's and the motion vector for N=16 and P=8, we use 256 "D", 255 accumulator and one comparator. The motion vector is outputted after 290 clock cycles.

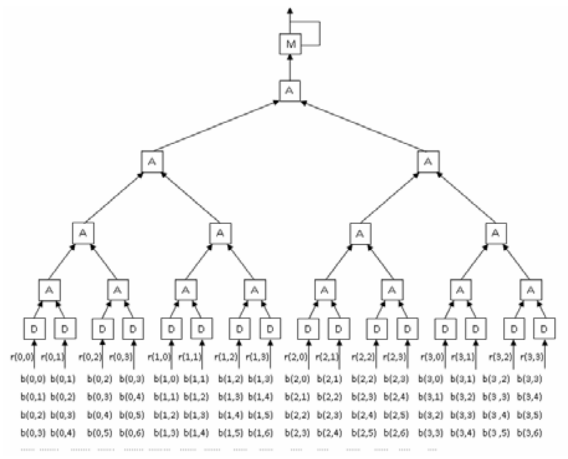
M
A
D
r(0,0) r(0,1) r(0,2) r(0,3) r(1,0) r(1,1) r(1,2) r(1,3) r(2,0) r(2,1) r(2,2) r(2,3) r(3,0) r(3,1) r(3,2) r(3,3)
b(0,0) b(0,1) b(0,2) b(0,3) b(1,0) b(1,1) b(1,2) b(1,3) b(2,0) b(2,1) b(2,2) b(2,3) b(3,0) b(3,1) b(3,2) b(3,3)

Fig. 11 Y.S. Jehng and al Architecture for N=4

III. PROPOSED ACHITECTURE

A. *Proposed structure*

In order to realize FSBM algorithm, various architectures have been proposed. By examining these architectures, we conclude that processing elements (PEs), address generator and data memories are indispensable and necessary components for FSBM algorithm implementation. In fact, PE accomplishes the computation of block distortion measure (SAD). Address generator generates the address to memories and transfer data from each memory block to the corresponding processing elements. Consequently, the proposed FSBM architecture is illustrated in Fig. 12. Our proposed architecture is composed of multiplexed registers, memories, Flip-flops, absolute difference components, accumulators, and a comparator for selecting the minimum SAD provided by each PE. Controller module contains the address generator engine that produces the memory addresses and transfers data from each memory block to the corresponding PEs.

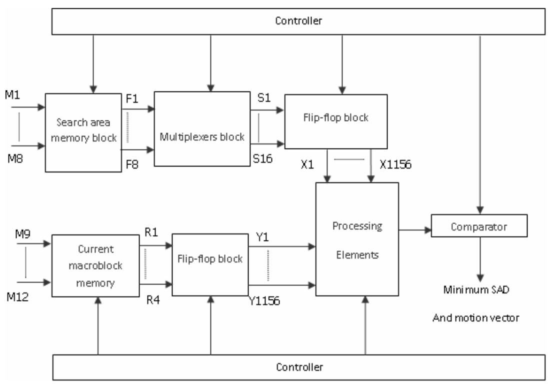

Fig. 12 The proposed architecture

B. *Data memories control*

For this architecture, we need a combination of 12 single-port memories that enables the reading of 12 pixels for every clock cycle (8 memories for the search area and 4 memories for the current MB). Each memory is generated by LPM mega-functions library in order to take advantage of the "StratixII" embedded RAM blocks. We must read pixels from input frames by using VHDL description with various "TEXTIO" instructions (camera entity) and store them in 12 memories. Then, the data coming from camera entity must be organized on these memories as shown in Table 1 and Table 2.

Table 1 shows that we have stored search area pixels in 8 separate memories in order to address 8 pixels at each clock cycle. These memories are presented by the search area memory block in the proposed architecture.

Table 2 shows that we have stored pixels of current MB in 4 memories in order to address 4 data's at each clock cycle. These memories take parts of the current MB memory block in the proposed architecture.

After memorizing search area and current MB, we will compute 289 SADs. That's why we must respect the memories read order presented in Table 3.

To test and simulate memory blocks, we use a test bench file allowing the mapping of each memory component with another component called "camera", which allows the reading of the pixels values from PGM frame files. For the search area, we use 8 memories of 128 bytes each. In fact, the search area has 32x32 pixels size and 25x25=210=1024 addresses. Each address consist of 8 bits data, thus we obtain 1024 bytes. For the current MB, we use 4 memories of 64 bytes each. Indeed, the size of a current MB is 16x16 pixels requiring 24x24 =28=256 addresses. Each address consists of 8 bits data, thus we obtain 256 bytes. Memories are synchronous and they can be used in writing or reading mode. In writing mode, the memories receive frame files data and store each one in the specified address.

C. *Multiplexers block diagram*

Multiplexers are used to select the data search area, and to allow horizontal movement for the execution of 17 SADs in parallel. Fig. 13 depicts the multiplexer's block.

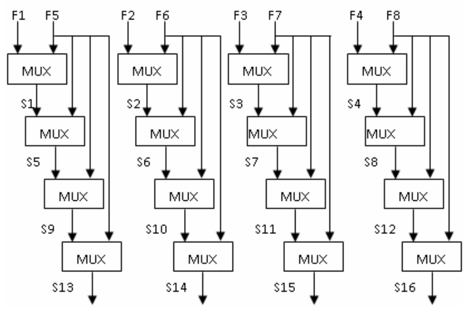

Fig. 13 Block of the multiplexers

F1 to F8 represent the pixels generated by the various memories of the search area and will be organized 2 by 2. This block consists of 16 multiplexers as shown in fig. 13. S1 to S16 represent the outputs of multiplexers, and correspond to the 16 horizontal movements (in the first horizontal movement we don't use any multiplexer). The Flip-Flop block allows during the processing of 289 SADs to read the different memories only one time and pass various data to each PE in order to compute the corresponding SAD.

D. *SAD calculation*

Processing elements block represented in fig. 12 is composed by 1156 inputs for the search area data (X) and 1156 inputs for the current MB data (Y). This block allows the computation of 289 SADs in a mixed mode: parallel and pipeline. It is possible to combine 4 processors elements (PE) in one engine in order to compute SAD 16x16 as shown on fig. 14.

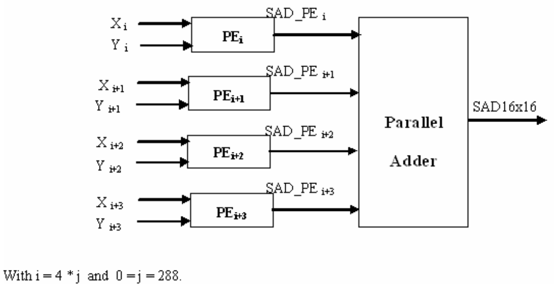

Fig. 14 Proposed SAD 16x16 diagram

Each PEi allows the computation of a SAD 16x4. The parallel adder performs the sum of 4 SADs 16x4 values for a given SAD 16x16. That's why we dispose of 1156 inputs (4*289) for both search area and current MB. Since we use 4 PEs to compute 289 SADs. Fig. 15 represents the internal structure of each PE. Our proposed algorithm allows the computation of 16x16 SAD in 64 cycles instead of 256 cycles for the existing architectures. Table 4 represents the necessary method for computing 289 SADs in mixed mode: parallel and pipeline.

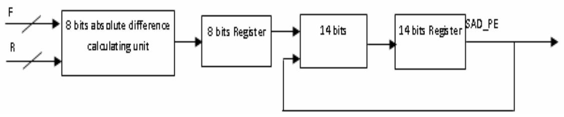

Fig. 15 Processor Element

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

In the previous section, we present an overview of all architecture. These architectures are described with VHDL language. Result of synthesis on the "STRATIXII family – EP2S130F1020C4" component is presented in the following table:

TABLE V
RESULT OF SYNTHESIS

Resources	*Avail*	*Used*	*Utilization*
Input/output	743	76	10%
Logic ports	106032	70004	66%
Memory Bits	6747840	10240	<1%

The maximum clock frequency is 89 MHz. The experimental results show that just 140 cycles is necessary to compute the final motion vector. With this result, we can execute 160M pixels/sec which is suited to process HTDV (1920x1088@60Hz) video sequences. The table 6 and 7 resumes the functional parameter and synthesis results for all presented architectures respectively. Form these tables, our pipelined FPGA architecture takes the minimum execution time for compute the motion vector.

V. CONCLUSION

In this paper, we have proposed the motion estimation architecture. Our pipelined architecture benefits from several PE engines executing in parallel and pipeline mode. This will solve the real time constraint and enable a better efficiency in HTDV video coding. It has been proved through our study that FPGA is an ultimate solution for the design of a motion estimation algorithm based on FSBM conception through the hardware description language (VHDL).

ACKNOWLEDGMENT

We would like to thank the reviewers and the editor for their valuable and fruitful comments and suggestions which contributed to greatly improve the quality of the original version of this paper.

REFERENCES

[1] G. Robert, "Représentation et codage de séquences vidéo par hybridation de fractales et d'éléments finis," Thèse / PhD, INPG Grenoble, 07 December 2000.
[2] S. Roux, "Adéquation algorithme – architecture pour le traitement multimédia embarqué," Thèse / PhD, 22 January 2002, TIMA, Institut National Polytechnique de Grenoble - INPG.
[3] S. Wong, B. Stougie, S. Cotofana, "An Investigation on FPGA based SAD Hardware Implementations," in Proceedings of the 13th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC2002), pp. 568-573, Veldhoven, The Netherlands, November 2002.
[4] Ja-Ling Wu, "Motion Estimation for Video Coding Standards," Department of Computer Science and Information Engineering, National Taiwan University.
[5] A. Ben Atitallah, P. Kadionik, N. Masmoudi, H. Levi "HW/SW FPGA Architecture for a Flexible Motion Estimation," IEEE ICECS '07, Marrakech, Morocco, 11-14, December 2007.
[6] J. Zhang, Y. He, S. Yang, and Y. Zhong, "Performance and Complexity Joint Optimization for H.264 Video Coding," Proceedings of the 2003 International Symposium on Circuits and Systems, Vol. 2. (2003) 888–891.

[7] C. Zhu, X. Lin, and L. P. Chau, " Hexagon-Based Search Pattern for Fast Block Motion Estimation ," IEEE Trans. On Circuits And Systs. For Video Technology, vol. 12, pp. 349-355, May 2002.

[8] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation ," IEEE Trans. Circuits Syst. Video Technol., vol. 8, pp. 369–377, Aug. 1998.

[9] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 4, pp. 419–423, Aug. 1996.

[10] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, Ph. Marchegay, "Optimization and implementation on FPGA of the DCT/IDCT algorithm ," IEEE ICASSP '06, Toulouse, France, 14-19 Mai 2006.

[11] T. Komarek, P. Pirsch, "Array architectures for block matching algorithms," IEEE Transactions on Circuits and Systems, Vol. 36, No. 10, October 1989.

[12] K. M. Yang, M. T. Sun, L. Wu, "A family of vlsi designs for the motion compensation block-matching algorithm," IEEE Transactions on Circuits and Systems, Vol. 36, No. 10, October 1989.

[13] C. H. Hsieh et al., "Vlsi architecture for block-matching motion estimation algorithm," IEEE Transactions on Circuits and Systems for video technology, Vol. 2, No. 2, June 1992

[14] F. M. Yang, S. Wolter, R. Laur., "Parallel implementation of a block-matching algorithm for hdtv motion estimation," Workshop on Design Methodologies for Microelectronicx and Signal Processing, pp. 73-80. October 1993.

[15] Y. S. Jehng, L. G. Chen, T. D. Chiueh, "An efficient and simple vlsi tree architecture for motion estimation Algorithms," IEEE Transactions on signal processing, Vol. 41, No. 2, October 1993.

[16] H. Yeo et al., "A novel modular systolic array architecture for full-search block matching motion estimation," IEEE Transactions on Circuits and Systems for video technology, Vol. No. 5, October 1995.

TABLE I
DATA MEMORIES STRUCTURE FOR SEARCH AREA

Memory F1	Memory F2	Memory F3	Memory F4	Memory F5	Memory F6	Memory F7	Memory F8
F(0,0)	F(1,0)	F(2,0)	F(3,0)	F(16,0)	F(17,0)	F(18,0)	F(19,0)
F(4,0)	F(5,0)	F(6,0)	F(7,0)	F(20,0)	F(21,0)	F(22,0)	F(23,0)
F(8,0)	F(9,0)	F(10,0)	F(11,0)	F(24,0)	F(25,0)	F(26,0)	F(27,0)
F(12,0)	F(13,0)	F(14,0)	F(15,0)	F(28,0)	F(29,0)	F(30,0)	F(31,0)
F(0,1)	F(1,1)	F(2,1)	F(3,1)	F(16,1)	F(17,1)	F(18,1)	F(19,1)
….	….	….	….	….	….	….	….
F(0,2)	F(1,2)	F(2,2)	F(3,2)	F(16,2)	F(17,2)	F(18,2)	F(19,2)
….	….	….	….	….	….	….	….
….	….	….	….	….	….	….	….
F(12,15)	F(13,15)	F(14,15)	F(15,15)	F(28,15)	F(29,15)	F(30,15)	F(31,15)
F(0,16)	F(1,16)	F(2,16)	F(3,16)	F(16,16)	F(17,16)	F(18,16)	F(19,16)
….	….	….	….	….	….	….	….
F(12,31)	F(13,31)	F(14,31)	F(15,31)	F(28,31)	F(29,31)	F(30,31)	F(31,31)

TABLE II
DATA MEMORIES STRUCTURE FOR CURRENT MB

Memory R1	Memory R2	Memory R3	Memory R4
R(0,0)	R(1,0)	R(2,0)	R(3,0)
R(4,0)	R(5,0)	R(6,0)	R(7,0)
R(8,0)	R(9,0)	R(10,0)	R(11,0)
R(12,0)	R(13,0)	R(14,0)	R(15,0)
R(0,1)	R(1,1)	R(2,1)	R(3,1)
…….	……	…….	…..
R(0,2)	R(1,2)	R(2,2)	R(3,2)
…..	…..	……	….
…..	….	…..	…..
R(12,14)	R(13,14)	R(14,14)	R(15,14)
R(0,15)	R(1,15)	R(2,15)	R(3,15)
…..	…..	…..	…..
R(12,15)	R(13,15)	R(14,15)	R(15,15)

TABLE III
MEMORIES ADDRESS GENERATOR

T	Current MB memory				Search area memory							
1+(4*0)	R(0,0)	R(1,0)	R(2,0)	R(3,0)	F(0,0)	F(1,0)	F(2,0)	F(3,0)				
2+(4*0)	R(4,0)	R(5,0)	R(6,0)	R(7,0)	F(4,0)	F(5,0)	F(6,0)	F(7,0)				
3+(4*0)	R(8,0)	R(9,0)	R(10,0)	R(11,0)	F(8,0)	F(9,0)	F(10,0)	F(11,0)				
4+(4*0)	R(12,0)	R(13,0)	R(14,0)	R(15,0)	F(12,0)	F(13,0)	F(14,0)	F(15,0)				
1+(4*1)	R(0,1)	R(1,1)	R(2,1)	R(3,1)	F(0,1)	F(1,1)	F(2,1)	F(3,1)	F(16,0)	F(17,0)	F(18,0)	F(19,0)
2+(4*1)	R(4,1)	R(5,1)	R(6,1)	R(7,1)	F(4,1)	F(5,1)	F(6,1)	F(7,1)	F(20,0)	F(21,0)	F(22,0)	F(23,0)
….	….	….	….	….	….	….	….	….	….	….	….	….
1+(4*2)	R(0,2)	R(1,2)	R(2,2)	R(3,2)	F(0,2)	F(1,2)	F(2,2)	F(3,2)	F(16,1)	F(17,1)	F(18,1)	F(19,1)
2+(4*2)	R(4,1)	R(5,1)	R(6,1)	R(7,1)	F(4,1)	F(5,1)	F(6,1)	F(7,1)	F(20,1)	F(21,1)	F(22,1)	F(23,1)
….	….	….	….	….	….	….	….	….	….	….	….	….
….	….	….	….	….	….	….	….	….	….	….	….	….
1+(4*15)	R(0,15)	R(1,15)	R(2,15)	R(3,15)	F(0,15)	F(1,15)	F(2,15)	F(3,15)	F(16,14)	F(17,14)	F(18,14)	F(19,14)
2+(4*15)	R(4,15)	R(5,15)	R(6,15)	R(7,15)	F(4,15)	F(5,15)	F(6,15)	F(7,15)	F(20,14)	F(21,14)	F(22,14)	F(23,14)

3+(4*15)	R(8,15)	R(9,15)	R(10,15)	R11,15)	F(8,15)	F(9,15)	F(10,15)	F(11,15)	F(24,15)	F(25,15)	F(26,15)	F(27,15)
4+(4*15)	R12,15)	R(13,15)	R(14,15)	R(15,15)	F(12,15)	F(13,15)	F(14,15)	F(15,15)	F(28,15)	F(29,15)	F(30,15)	F(31,15)
1+(4*16)					F(0,16)	F(1,16)	F(2,16)	F(3,16)	F(16,15)	F(17,15)	F(18,15)	F(19,15)
….					….	….	….	….	….	….	….	….
4+(4*30)					F(12,30)	F(13,30)	F(14,30)	F(15,30)	F(28,29)	F(29,29)	F(30,29)	F(31,29)
1+(4*31)					F(0,31)	F(1,31)	F(2,31)	F(3,31)	F(16,30)	F(17,30)	F(18,30)	F(19,30)
….									….	….	….	….
1+(4*32)									F(16,31)	F(17,31)	F(18,30)	F(19,31)
2+(4*32)									F(20,31)	F(21,31)	F(22,31)	F(23,31)
3+(4*32)									F(24,31)	F(25,31)	F(26,31)	F(27,31)
4+(4*32)									F(28,31)	F(29,31)	F(30,31)	F(31,31)

TABLE IV
ARCHITECTURE OPERATION

Clock cycle	SAD0	SAD1	SAD2	SAD3	SAD4		SAD287	SAD288

1+(4*0)	R(0,0)-F(0,0)
	R(1,0)-F(1,0)
	R(2,0)-F(2,0)
	R(3,0)-F(3,0)							

2+(4*0)	R(4,0)-F(4,0)
	R(5,0)-F(5,0)
	R(6,0)-F(6,0)
	R(7,0)-F(7,0)
		R(0,0)-F(1,0)
		R(1,0)-F(2,0)
		R(2,0)-F(3,0)
		R(3,0)-F(4,0)
			R(0,0)-F(2,0)
			R(1,0)-F(3,0)
			R(2,0)-F(4,0)
			R(3,0)-F(5,0)
				R(0,0)-F(3,0)
				R(1,0)-F(4,0)
				R(2,0)-F(5,0)
				R(3,0)-F(6,0)
					R(0,0)-F(4,0)
					R(1,0)-F(5,0)
					R(2,0)-F(6,0)
					R(3,0)-F(7,0)			

3+(4*0)	R(8,0)-F(8,0)
	R(9,0)-F(9,0)
	R(10,0)-F(10,0)
	R(11,0)-F(11,0)
		R(4,0)-F(5,0)
		R(5,0)-F(6,0)
		R(6,0)-F(7,0)
		R(7,0)-F(8,0)
			R(4,0)-F(6,0)
			R(5,0)-F(7,0)
			R(6,0)-F(8,0)
			R(7,0)-F(9,0)
				R(4,0)-F(7,0)
				R(5,0)-F(8,0)
				R(6,0)-F(9,0)
				R(7,0)-F(10,0)
					R(4,0)-F(8,0)
					R(5,0)-F(9,0)
					R(6,0)-F(10,0)
					R(7,0)-F(11,0)			

4+(4*0)	R(12,0)-F(12,0)
	R(13,0)-F(13,0)
	R(14,0)-F(14,0)
	R(15,0)-F(15,0)
		R(8,0)-F(9,0)
		R(9,0)-F(10,0)
		R(10,0)-F(11,0)
		R(11,0)-F(12,0)
			R(8,0)-F(10,0)
			R(9,0)-F(11,0)
			R(10,0)-F(12,0)
			R(11,0)-F(13,0)
				R(8,0)-F(11,0)
				R(9,0)-F(12,0)
				R(10,0)-F(13,0)
				R(11,0)-F(14,0)
					R(8,0)-F(12,0)
					R(9,0)-F(13,0)
					R(10,0)-F(14,0)
					R(11,0)-F(15,0)			

1+(4*1)	R(0,1)-F(0,1)
	R(1,1)-F(1,1)
	R(2,1)-F(2,1)
	R(3,1)-F(3,1)
		R(12,0)-F(13,0)
		R(13,0)-F(14,0)
		R(14,0)-F(15,0)
		R(15,0)-F(16,0)
			R(12,0)-F(14,0)
			R(13,0)-F(15,0)
			R(14,0)-F(16,0)
			R(15,0)-F(17,0)
				R(12,0)-F(15,0)
				R(13,0)-F(16,0)
				R(14,0)-F(17,0)
				R(15,0)-F(18,0)
					R(12,0)-F(16,0)
					R(13,0)-F(17,0)
					R(14,0)-F(18,0)
					R(15,0)-F(19,0)			

2+(4*1)	R(4,1)-F(4,1)
	R(5,1)-F(5,1)
	R(6,1)-F(6,1)
	R(7,1)-F(7,1)
		R(0,1)-F(1,1)
		R(1,1)-F(2,1)
		R(2,1)-F(3,1)
		R(3,1)-F(4,1)
			R(0,1)-F(2,1)
			R(1,1)-F(3,1)
			R(2,1)-F(4,1)
			R(3,1)-F(5,1)
				R(0,1)-F(3,1)
				R(1,1)-F(4,1)
				R(2,1)-F(5,1)
				R(3,1)-F(6,1)
					R(0,1)-F(4,1)
					R(1,1)-F(5,1)
					R(2,1)-F(6,1)
					R(3,1)-F(7,1)			

1+(4*2)	R(0,2)-F(0,2)
	R(1,2)-F(1,2)
	R(2,2)-F(2,2)
	R(3,2)-F(3,2)
		R(12,1)-F(13,1)
		R(13,1)-F(14,1)
		R(14,1)-F(15,1)
		R(15,1)-F(16,1)
			R(12,1)-F(14,1)
			R(13,1)-F(15,1)
			R(14,1)-F(16,1)
			R(15,1)-F(17,1)
				R(12,1)-F(15,1)
				R(13,1)-F(16,1)
				R(14,1)-F(17,1)
				R(15,1)-F(18,1)
					R(12,1)-F(16,1)
					R(13,1)-F(17,1)
					R(14,1)-F(18,1)
					R(15,1)-F(19,1)			

2+(4*2)	R(4,2)-F(4,2)
	R(5,2)-F(5,2)
	R(6,2)-F(6,2)
	R(7,2)-F(7,2)
		R(0,2)-F(1,2)
		R(1,2)-F(2,2)
		R(2,2)-F(3,2)
		R(3,2)-F(4,2)
			R(0,2)-F(2,2)
			R(1,2)-F(3,2)
			R(2,2)-F(4,2)
			R(3,2)-F(5,2)
				R(0,2)-F(3,2)
				R(1,2)-F(4,2)
				R(2,2)-F(5,2)
				R(3,2)-F(6,2)
					R(0,2)-F(4,2)
					R(1,2)-F(5,2)
					R(2,2)-F(6,2)
					R(3,2)-F(7,2)			

4+(4*15)	R(12,15)-F(12,15)
	R(13,15)-F(13,15)
	R(14,15)-F(14,15)
	R(15,15)-F(15,15)
		R(8,15)-F(9,15)
		R(9,15)-F(10,15)
		R(10,15)-F(11,15)
		R(11,15)-F(12,15)
			R(8,15)-F(10,15)
			R(9,15)-F(11,15)
			R(10,15)-F(12,15)
			R(12,15)-F(13,15)
				R(8,15)-F(11,15)
				R(9,15)-F(12,15)
				R(10,15)-F(13,15)
				R(11,15)-F(14,15)
					R(8,15)-F(12,15)
					R(9,15)-F(13,15)
					R(10,15)-F(14,15)
					R(11,15)-F(15,15)			

1+(4*16)		R(12,15)-F(13,15)
		R(13,15)-F(14,15)
		R(14,15)-F(15,15)
		R(15,15)-F(16,15)
			R(12,15)-F(14,15)
			R(13,15)-F(15,15)
			R(14,15)-F(16,15)
			R(15,15)-F(17,15)
				R(12,15)-F(15,15)
				R(13,15)-F(16,15)
				R(14,15)-F(17,15)
				R(15,15)-F(18,15)
					R(12,15)-F(16,15)
					R(13,15)-F(17,15)
					R(14,15)-F(18,15)
					R(15,15)-F(19,15)			

1+(4*19)							R(0,0)-F(15,16)	R(0,0)-F(16,16)

							R(1,0)-F(16,16)
							R(2,0)-F(17,16)
							R(3,0)-F(18,16)
								R(1,0)-F(17,16)
								R(2,0)-F(18,16)
								R(3,0)-F(19,16)

4+(4*34)							R(12,15)-
							F(27,31)
							R(13,15)-
							F(28,31)
							R(14,15)-
							F(29,31)
							R(15,15)-
							F(30,31)
								R(12,15)-F(28,31)
								R(13,15)-F(29,31)
								R(14,15)-F(30,31)
								R(15,15)-F(31,31)

4+(4*34)							R(12,15)-
							F(27,31)
							R(13,15)-
							F(28,31)
							R(14,15)-
							F(29,31)
							R(15,15)-
							F(30,31)
								R(12,15)-F(28,31)
								R(13,15)-F(29,31)
								R(14,15)-F(30,31)
								R(15,15)-F(31,31)

TABLE VI
FUNCTIONAL RESULTS FOR DIFFERENT ARCHITECTURES

N=16 et P=8	AB1	AB2	AS1	AS2	Heish	K. M. Yang	F. M. Yang	Jehng	Yeo	Proposed Architecture
Processor element number	16	256	17	272	256	17	289	512	289	289
Architecture topology	1-D	2-D	1-D	2-D	1-D	1-D	2-D	2-D	2-D	2-D
Input port number for search window	16	16	1	32	1	2	2	256	2	8
Input port number for MB	16	-	1	16	1	1	1	1	1	4
Clock cycles number for compute the motion vector	9250	579	8722	577	1028	4370	534	290	547	140

TABLE VII
SYNTHESIS RESULTS FOR ALL ARCHITECTURES

N=16 et P=8	AB1	AB2	AS1	AS2	Heish	K. M. Yang	F. M. Yang	Jehng	Yeo	Proposed Architecture
Logic element number	5153	17179	2068	20548	14010	1654	26799	69917	15866	70004
Pins number	62	62	62	62	62	62	52	62	62	76
Memory bits number	10240	10240	10240	10240	-	10240	-	-	-	10240
Maximum frequency (Mhz)	142.51	115.97	111.17	160.62	128.09	134.86	96.64	50.97	70.01	89
Latency time	2.6	0.922	2.173	2.593	2.41	0.973	8.844	10.278	8.717	7
Execution time (ns)	64909.85	4993.639	78456.563	3594.995	8028.006	32404.523	5534.676	5705.878	7399.781	1579.432

Hassen Loukil received electrical engineering degree from the National School of Engineering-Sfax (ENIS) in 2004. His received his MS degree in electronic engineering from the National School of Engineering-Sfax (ENIS) in 2005. He is currently researcher in the Laboratory of Electronics and Information Technology and an assistant at the University of Sfax, Tunisia. His research interests include signal and image processing, hardware implementation using FPGA, embedded systems technology.

Nouri Masmoudi received electrical engineering degree from the Faculty of Sciences and Techniques-Sfax, Tunisia, in 1982, the DEA degree from the National Institute of Applied Sciences-Lyon and University Claude Bernard-Lyon, France in 1982. From 1986 to 1990, he prepared his thesis at the laboratory of Power Electronics (LEP) at the National school Engineering of Sfax (ENIS). He received his PhD degree at the National school Engineering of Tunis (ENIT), Tunisia in 1990. From 1990 to 2000, he was an assistant professor at the electrical engineering department-ENIS. Since 2000, he has been an associate professor and head of the group 'Circuits and Systems' in the Laboratory of Electronics and Information Technology. Since 2003, He is responsible for the Electronic Master Program at ENIS. His research activities have been devoted to several topics: design, telecommunication, embedded systems and information technology, Video Coding (Motion Estimation, Mode Decision, Image Interpolation, and Denoising.