High Performance Hardware Architectures for A Hexagon-Based Motion Estimation Algorithm

Ozgur Tasdizen[1,2,a], Abdulkadir Akin[1,2,b], Halil Kukner[1,2,c], Ilker Hamzaoglu[1,d], H. Fatih Ugurdag[3,e]

[1]Electronics Engineering, Sabanci University, 34956 Tuzla, Istanbul, Turkey
[2]Vestek Electronic Research & Development Corp., 34469 Maslak, Istanbul, Turkey
[3]Faculty of Engineering, Bahcesehir University, 34353 Besiktas, Istanbul, Turkey
[a]tasdizen@su.sabanciuniv.edu, [b]abdulkadir@su.sabanciuniv.edu, [c]shalil@su.sabanciuniv.edu.tr,
[d]hamzaoglu@sabanciuniv.edu, [e]fatih.ugurdag@bahcesehir.edu.tr

Abstract—**Motion Estimation is the most computationally intensive part of video compression and video enhancement systems. For the recently available high definition frame sizes and high frame rates, the computational complexity of full search motion estimation algorithm is prohibitively high, while the PSNR obtained by fast search algorithms is low. Therefore, in this paper, we propose a hexagon-based motion estimation algorithm and two high performance hardware architectures for implementing this algorithm. The proposed algorithm has lower computational complexity than full search algorithm, and the simulation results showed that the PSNR obtained by this algorithm is better than the PSNR obtained by other fast search algorithms. Both hardware architectures are implemented in VHDL and mapped to Xilinx FPGAs. Both hardware architectures can run at 144 MHz when implemented on an XC3S1200E-5 FPGA, and they can process 25 1920x1080 frames per second for the largest search range (±32, ±16). Various fast search algorithms can be implemented using the first hardware architecture. But, it uses 80 Block RAMs. Only the hexagon-based algorithm proposed in this paper can be efficiently implemented using the second hardware architecture. However, it uses 16 Block RAMs and fits into XC3S1200E-5, a low cost Xilinx Spartan-3E FPGA. In addition, a novel data reuse method is used in the second architecture to reduce the number of internal memory accesses, and it has low control overhead because of its regular data flow. Therefore, it can be used in consumer electronics products.**

I. INTRODUCTION

Motion Estimation (ME) is the most computationally intensive part of video compression and video enhancement systems. ME is used to reduce the bit-rate in video compression systems by exploiting the temporal redundancy between successive frames, and it is used to enhance the quality of displayed images in video enhancement systems by extracting the true motion information. ME is used in video compression standards such as ITU-T H.261/263/264 and ISO MPEG-1/2/4, and it is used in video enhancement algorithms such as frame rate conversion, de-interlacing and de-noising.

Block Matching (BM) is the most preferred method for ME. BM partitions current frame into non-overlapping NxN rectangular blocks and tries to find a block from a reference frame in a given search range that best matches the current block. Sum of Absolute Differences (SAD) is the most preferred block matching criterion because of its suitability for hardware implementations.

Full Search (FS) ME algorithm finds the reference block that best matches the current block by computing the SAD values for all search locations in a given search range. The computational complexity of FS algorithm is very high, especially for the recently available consumer electronic devices such as High Definition (HD) digital video broadcasting and high resolution & high frame rate flat panel displays. Because there are fast movements between successive frames in these applications which requires an increased search range and the frame sizes in these applications are very large.

Several fast search ME algorithms are developed for low bit-rate applications, which use small frame sizes and require small search ranges, in consumer electronics products. These algorithms try to obtain the same PSNR as FS algorithm by computing the SAD values for fewer search locations in a given search range. The most popular fast search algorithms are New Three Step Search (NTSS) [1], Four Step Search (FSS) [2], Diamond Search (DS) [3] and Hexagon Based Search (HEXBS) [4].

Fast search ME algorithms perform very well for low bit-rate applications such as video phone and video conferencing. However, fast search ME algorithms do not produce satisfactory results for the recently available consumer electronic devices such as HD digital video broadcasting and high resolution & high frame rate flat panel displays, because, in these applications, there are fast movements between successive frames. Among the fast search algorithms, DS and HEXBS may find satisfactory enough motion vectors if the motion between successive frames are very small. However, for fast moving objects, they find motion vectors which give a locally minimum block distortion due to the sequential nature of these algorithms.

Therefore, in this paper, we propose a ME algorithm which is a generalization of HEXBS ME algorithm proposed in [4] and two high performance hardware architectures for implementing this algorithm. The proposed algorithm has lower computational complexity than FS algorithm and the simulation results showed that the PSNR obtained from this algorithm is better than the PSNR obtained from other fast search algorithms.

Both hardware architectures are implemented in VHDL and mapped to Xilinx FPGAs using Xilinx ISE 9.2.04. Both hardware architectures can run at 144 MHz when implemented on an XC3S1200E-5 FPGA and, for the largest search range (±32, ±16), they can process 127 frames per second (fps), 57fps, and 25fps for 720x576, 1280x720, and 1920x1080 resolutions, respectively.

Various fast search algorithms can be implemented using the first hardware architecture. But, it uses 80 Block RAMs (BRAMs). Only the hexagon-based ME algorithm proposed in this paper can be efficiently implemented using the second hardware architecture. However, it uses 16 BRAMs and fits into XC3S1200E-5, a low cost Xilinx Spartan 3E FPGA. In addition, a novel data reuse method is used in the second architecture to reduce the number of internal memory accesses, and it has low control overhead because of its regular data flow. Therefore, it can be used in consumer electronics products.

Many hardware architectures for FS ME algorithm are proposed in the literature [5-8]. However, only a small number of hardware architectures for fast search ME algorithms are proposed [9, 10]. To the best of our knowledge, no hardware architecture is proposed for HEXBS ME algorithm in the literature.

The rest of the paper is organized as follows. Section II describes the proposed hexagon-based ME algorithm and presents the simulation results. The proposed hardware architectures are presented in Sections III and IV. Section V concludes the paper.

II. Proposed Hexagon Based ME Algorithm

We propose a ME algorithm which is a generalization of HEXBS ME algorithm proposed in [4]. Our algorithm consists of main and fine search patterns. The main search patterns consist of all the search locations that can be checked by HEXBS algorithm during several iterations in the horizontal and vertical directions.

We propose 10x9, 12x12, 14x15 and 32x16 main search patterns. The search locations of 10x9 search pattern are shown in Fig. 1. 32x16 search pattern consists of all the search locations that can be checked by HEXBS algorithm during 16 iterations in the horizontal direction and 8 iterations in the vertical direction. The difference between 32x16 pattern and the other patterns is that the other patterns have a gap of two pixels in the vertical direction compared to the one pixel gap of 32x16 pattern and they have less computational load than 32x16 pattern.

We tried the three fine search patterns shown in Fig. 2 and the results of our analysis show that "DoubleCross" fine search pattern improves the results up to 1% over the "Plus" fine search pattern, which is used in [4]. Therefore we used this fine search pattern with our main search patterns.

We compared the performance of our search patterns with the performance of FS, DS and HEXBS algorithms based on Mean Absolute Difference (MAD) metric. We used Flowers, Mobile Calendar, Table Tennis, Susie, Spider and Irobot videos for simulation. Each video has 100 frames. Spider and Irobot videos, which have large motion between frames, are taken from "Spiderman 2" and "Irobot" films respectively. The resolution of these two videos is 720x576 pixels and their frame rate is 25 fps. The other videos have a resolution of 704x480 pixels and a frame rate of 29 fps. The simulations are done using 8 bit luminance data and for a 16x16 block size.

The simulation results are shown in Table I and Table II. The results show that our search patterns outperform DS and HEXBS algorithms. The reason for this achievement is that our patterns are able to find the search location that has globally minimum block distortion by checking more search locations in the search range than DS and HEXBS algorithms.

Our search patterns perform better than DS and HEXBS algorithms especially for videos that have large motion. In order to show this, we analyzed the performance of the algorithms for different Frame Distances (FD). FD is the gap between the frames on which motion estimation is done. Since increasing FD is identical to lowering the frame rate of the video, large movements between successive frames are introduced by increasing FD.

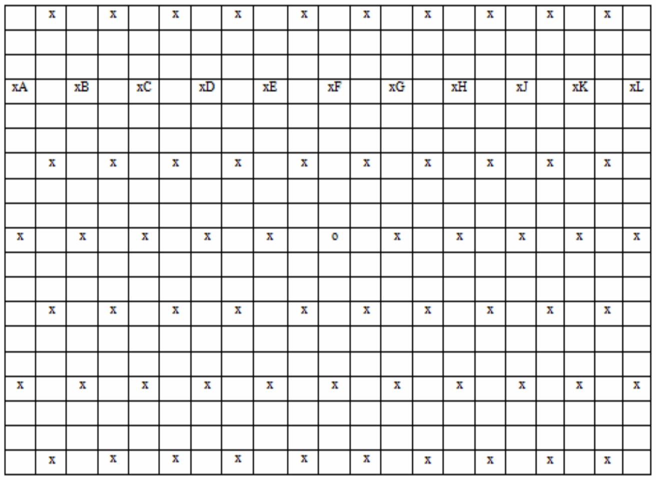

Figure 1. Search Locations of 10x9 Search Pattern

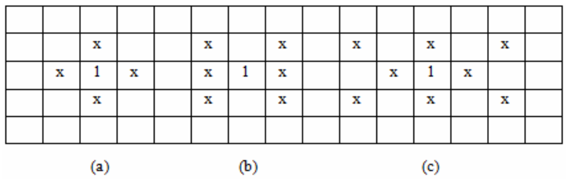

Figure 2. Fine Search Patterns: (a) Plus (b) Side (c) DoubleCross

TABLE I. MAD Simulation Results for FD=2

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie
FS	±10,±9	8.4195	11.2900	4.6473	4.3346
DS	±10,±9	9.8227	12.6452	4.8269	4.6230
HEXBS	±10,±9	10.3674	13.4550	4.8955	4.8455
10x9	±10,±9	8.7374	12.8067	4.7380	4.5742
10x9 Pattern's Improvement over HEXBS		%15.72	%4.82	%3.22	%5.60
FS	±12,±12	8.3397	11.2614	4.5486	4.0801
DS	±12,±12	9.7948	12.6405	4.7750	4.4327
HEXBS	±12,±12	10.3355	13.4469	4.8189	4.5951
12x12	±12,±12	8.6715	12.8671	4.6367	4.3139
12x12 Pattern's Improvement over HEXBS		%16.10	%4.31	%3.78	%6.12
FS	±14,±15	8.3248	11.2413	4.4929	3.9142
DS	±14,±15	9.7930	12.6386	4.7524	4.3191
HEXBS	±14,±15	10.3330	13.4436	4.7898	4.4681
14x15	±14,±15	8.6679	12.9005	4.5955	4.1723
14x15 Pattern's Improvement over HEXBS		%16.11	%4.04	%4.06	%6.62
FS	±32,±16	8.3118	11.1220	4.4157	3.5597
DS	±32,±16	9.7925	12.6360	4.7334	4.1450
HEXBS	±32,±16	10.3324	13.4396	4.7623	4.2753
32x16	±32,±16	8.4789	11.9386	4.4449	3.6496
32x16 Pattern's Improvement over HEXBS		%17.94	%11.17	%6.66	%14.64

TABLE II. MAD SIMULATION RESULTS FOR FD=1

Algorithm	Search Range	Spider	Irobot
FS	±10,±9	9.2939	7.5873
DS	±10,±9	9.7210	8.1208
HEXBS	±10,±9	10.2413	8.4511
10x9	±10,±9	9.3473	7.6989
10x9 Pattern's Improvement over HEXBS		%8.73	%8.90
FS	±12,±12	8.2719	7.1467
DS	±12,±12	8.9874	7.7959
HEXBS	±12,±12	9.3317	8.0455
12x12	±12,±12	8.3012	7.2488
12x12 Pattern's Improvement over HEXBS		%11.04	%9.90
FS	±14,±15	7.4357	6.8214
DS	±14,±15	8.4644	7.5768
HEXBS	±14,±15	8.8051	7.8295
14x15	±14,±15	7.4848	6.9352
14x15 Pattern's Improvement over HEXBS		%14.99	%11.42
FS	±32,±16	5.4335	5.6620
DS	±32,±16	7.6574	6.9723
HEXBS	±32,±16	7.9547	7.2144
32x16	±32,±16	5.5317	5.7265
32x16 Pattern's Improvement over HEXBS		%31.23	%20.62

III. PROPOSED GENERIC ARCHITECTURE

The block diagram of the proposed generic architecture for a 16x16 block size is shown in Fig. 3. In order to calculate the SAD of a 16x16 block in one clock cycle, we used 256 Processing Elements (PEs) and the outputs of all PEs are added using an adder tree. This hardware has 7 pipeline stages.

In order to calculate the SAD of a 16x16 block in one clock cycle, the pixels in the memories are organized to be able to bring the pixels belonging to a particular search location to the datapath in one clock cycle. The data layout in BRAMs is shown in Fig. 4. In the figure, each box represents a pixel and the number in a box shows the BRAM storing that pixel. We use 5 BRAMs to store one line of search range in order to avoid data collisions that occur during the access of a search location that is not aligned with BRAMs. Since the maximum word length of BRAMs can be configured as 32 bit wide, each memory location stores four pixels.

In order to be able to access the reference data of an arbitrary search location, outputs of the BRAMs have to be aligned. This is done by horizontal and vertical rotators. Dark shaded area in Fig. 4 shows the required reference data for an example search location. In this example, the horizontal rotator must rotate the outputs of BRAMs left by 10 pixels and the vertical rotator, which combines the output of the horizontal rotators, must rotate its inputs left by 6 lines.

The proposed hardware architecture is implemented in VHDL and mapped to Xilinx XC3S1200-5 FPGA using Xilinx ISE 9.2.04. The implementation is verified with RTL simulations using Mentor Graphics Modelsim 6.3c. RTL simulation results matched the results of a MATLAB model of the proposed hexagon-based ME algorithm.

The proposed hardware can work at 144 MHz on a Xilinx XC3S1200-5 FPGA. Therefore, for the largest search range (±32, ±16), it can process 206743 16x16 blocks per second. Therefore, it can process 127 fps, 57 fps, and 25 fps for 720x576, 1280x720 and 1920x1080 resolutions respectively.

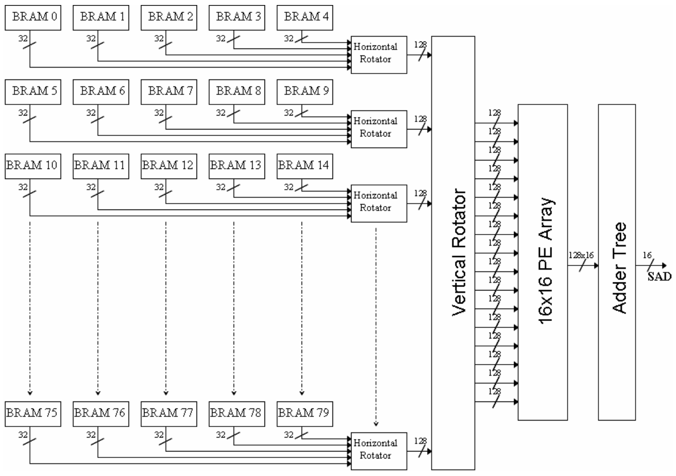

Figure 3. Proposed Generic Architecture

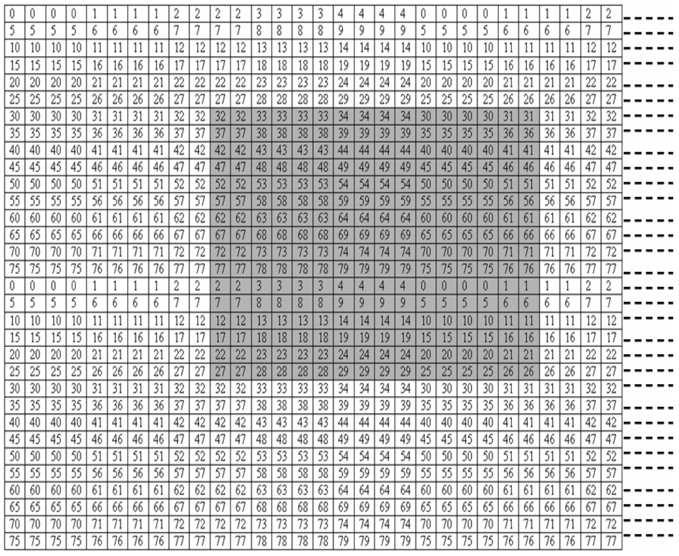

Figure 4. Data Layout in BRAMs

Instead of calculating the SAD of a 16x16 block in one clock cycle, if a 16x16 block is divided into subblocks and one subblock is processed in a clock cycle, the datapath of the generic architecture (the number of PEs, adders and rotators) and the number of BRAMs required can be reduced at the expense of increasing the number of clock cycles required for calculating the SAD value of a 16x16 block. The datapaths for different block sizes are implemented in VHDL and mapped to a Xilinx Spartan-3E FPGA using Xilinx ISE 9.2.04. The datapath area and the required number of BRAMs for each block size are shown in Table III.

The bottleneck for a high performance motion estimation hardware design that will be mapped to an FPGA is the number of BRAMs available in the FPGA. Since 16x16 and 16x8 generic architectures use large number of BRAMs, it is not possible to implement them using a low cost FPGA family. Although 16x4 and 16x2 generic architectures can be implemented using a low cost FPGA family, they are not suitable for real-time implementation of high frame size and high frame rate applications, since they require more clock cycles to compute an SAD value. Therefore, in the next section, we propose a systolic array architecture which is suitable for real-time implementation of high frame size and high frame rate applications and can be implemented using a low cost FPGA family.

TABLE III. COMPARISON OF GENERIC ARCHITECTURES FOR VARIOUS BLOCK SIZES

Block Size	Number of BRAMs	Number of PEs	SAD of a 16x16 Block (Cycles)	Total PE Area with Adder Tree (LUTs)	Total Area (LUTs)
16x16	80	256	1	6940	31416
16x8	40	128	2	3463	14675
16x6	30	96	3	2580	9447
16x4	20	64	4	1726	6304
16x2	10	32	8	857	2889

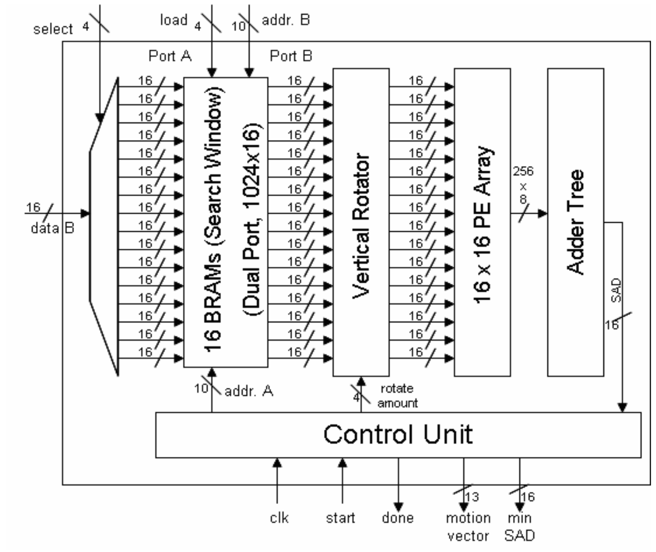

Figure 5. Proposed Systolic Array Architecture

TABLE IV. SEARCH PATTERNS

Search Range	Number of Search Locations	Required Clock Cycles
±10,±9	73	122
±12,±12	113	176
±14,±15	159	236
±32,±16	553	672
Fine Search Pattern	**Number of Search Locations**	**Required Clock Cycles**
Plus	4	25
Side	6	27
DoubleCross	8	29

IV. PROPOSED SYSTOLIC ARRAY ARCHITECTURE

The block diagram of the proposed systolic array architecture for a 16x16 block size and its datapath are shown in Fig. 5 and Fig. 6 respectively. In order to calculate the SAD of a 16x16 block in one clock cycle, we used 256 PEs and the outputs of all PEs are added using an adder tree. This hardware has 6 pipeline stages.

The main difference between this architecture and the architecture proposed in the previous section is that all PEs do not receive their reference data directly from BRAMs. In order to fit the ME hardware into a low cost Xilinx Spartan-3E FPGA, we used 16 BRAMs. These BRAMs are configured for a port width of 16 bits and they are connected to 32 PEs. The remaining 224 PEs receive their reference data from their neighboring PEs. Reference data is shifted to the right in the PE array. Loading the reference data of a search location has a start-up cost of 8 cycles. After the PE array is loaded, SAD values of the search locations in the same line is obtained in each clock cycle.

BRAMs are configured as dual port RAMs for overlapping motion estimation of current 16x16 block with loading of search range of next 16x16 block. One port is used only for write operations and the other port is used only for read operations.

The number of search locations checked by each search pattern and the number of clock cycles required to complete checking these search locations using the proposed systolic array architecture is shown in Table IV.

The data flow through the proposed systolic array architecture is shown in Table V. Let A - L shown in Fig. 1 denote the pixels in these columns. A1 denotes the pixels in the column A and A2 denotes the pixels in the column next to A (the right neighbor). Assuming that D is the first search location in the line, in the first clock cycle, the PE array is filled with the pixels in columns L1 and L2. In the second clock cycle, these pixels are shifted to the right in the PE array by two and the pixels in columns K1 and K2 are loaded into two left end columns of the PE array, and so on. Therefore, in the 8th clock cycle, the SAD value of search location D is obtained. In the 9th, 10th and 11th clock cycles, SAD values of search locations C, B and A are obtained.

The proposed hardware architecture is implemented in VHDL and mapped to Xilinx XC3S1200-5 FPGA using Xilinx ISE 9.2.04. The implementation is verified with RTL simulations using Mentor Graphics Modelsim 6.3c. RTL simulation results matched the results of a MATLAB model of the proposed hexagon-based ME algorithm.

The proposed architecture uses 6648 LUTs and 16 BRAMs, and it fits into a state of the art low cost Xilinx Spartan-3E FPGA. Because, in the proposed architecture horizontal rotators are not used, the number of pixels in each input line of vertical rotator is 16, and due to the regular data flow control unit uses only 265 LUTs.

The proposed hardware can work at 144 MHz on a Xilinx XC3S1200-5 FPGA. Therefore, for the largest search range (±32, ±16), it can process 206743 16x16 blocks per second. Therefore, it can process 127 fps, 57 fps, and 25 fps for 720x576, 1280x720 and 1920x1080 resolutions respectively.

V. CONCLUSIONS

In this paper, we proposed a hexagon-based ME algorithm which has lower computational complexity than FS ME algorithm, and the simulation results showed that the PSNR obtained by this algorithm is better than the PSNR obtained by other fast search algorithms. We also proposed two high performance hardware architectures for implementing this algorithm. Both of these hardware architectures are implemented in VHDL and mapped to Xilinx FPGAs. Both hardware architectures can run at 144 MHz when implemented on an XC3S1200E-5 FPGA, and they can process 25 1920x1080 fps for the largest search range (±32, ±16). Various fast search ME algorithms can be implemented using the first architecture. But, it uses 80 BRAMs. Only the hexagon-based ME algorithm we proposed can be efficiently implemented using the second architecture. However, it uses 16 BRAMs and fits into XC3S1200E-5, a low cost Xilinx Spartan-3E FPGA. In addition, a novel data reuse method is used in the second architecture to reduce the number of internal memory accesses, and it has low control overhead because of its regular data flow. Therefore, it can be used in consumer electronics products.

REFERENCES

[1] R. Li, B. Zeng, and M.L. Liou, “A new three-step search algorithm for block motion estimation,” IEEE Trans. Circuits and Systems for Video Technology, vol. 4, pp. 438–442, 1994.

[2] L.M. Po and W.C. Ma, “A novel four-step search algorithm for fast block motion estimation,” IEEE Trans. Circuits and Systems for Video Technology, vol. 6, pp. 313–317, 1996.

[3] S. Zhu and K.-K. Ma, “A new diamond search algorithm for fast block matching motion estimation,” IEEE Trans. Image Processing, vol. 9, pp. 287–290, 2000.

[4] C. Zhu, X. Lin, and L.P. Chau, “Hexagon-based search pattern for fast block motion estimation,” IEEE Trans. Circuits and Systems for Video Technology, vol. 12, pp. 349–355, 2002.

[5] T. Komarek and P. Pirsch, “Array Architectures for Block Matching Algorithms,” IEEE Trans. Circuits and Systems for Video Technology, vol. 36, pp. 301-1308, 1989.

[6] K.M. Yang, M.T. Sun, and L. Wu, “A family of VLSI designs for the motion compensation block-matching algorithm,” IEEE Trans. Circuits and Systems for Video Technology, vol. 36, pp. 1317- 1325, 1989.

[7] Y.S. Jehng, L.G. Chen, and T.D. Chiueh, “An efficient and simple VLSI tree architecture for motion estimation algorithm,” IEEE Trans. on Signal Processing, vol. 41, no. 2, 1993.

[8] L. Vos and M. Schobinger, “VLSI architecture for a flexible block matching processor,” IEEE Trans. Circuits and Systems for Video Technology, vol. 5, pp. 417-428, 1995.

[9] D. Xu, J.M. Noras, and W. Booth, “A simple and efficient VLSI architecture for a very fast high performance three step search algorithm,” IEE Colloquium on High Performance Architectures for Real-Time Image Processing, pp. 6/1 - 6/6, Feb. 1998.

[10] S.-T. Jung and S.-S. Lee, “A 4-way Pipelined Processing Architecture for Three-Step Search Block-matching Motion Estimation,” IEEE Trans. Consumer Electronics, vol. 50, pp. 674-681, 2004.

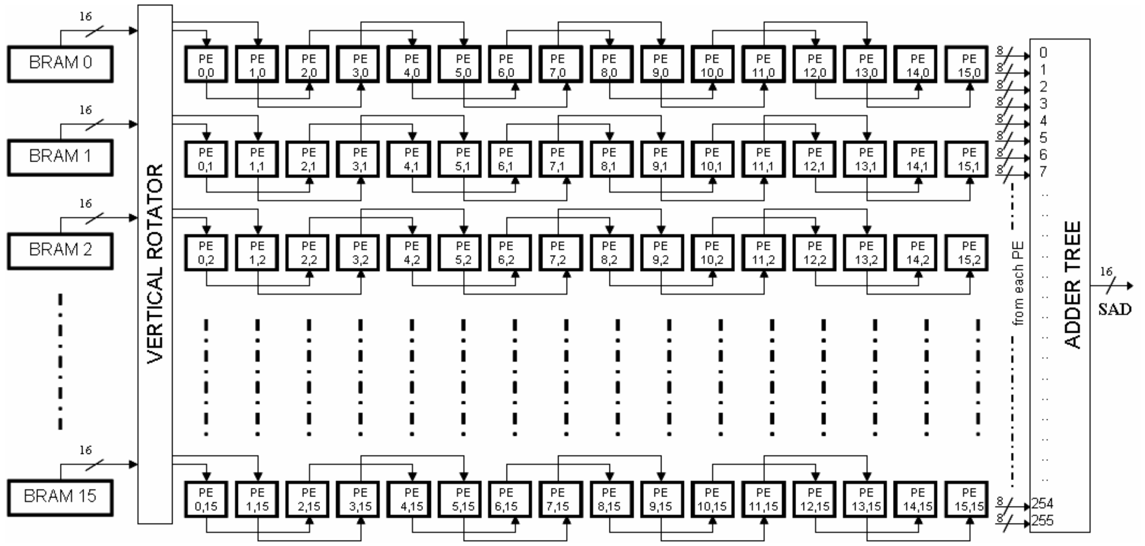

Figure 6. Datapath of Proposed Systolic Array Architecture

TABLE V. DATA FLOW THROUGH SYSTOLIC ARRAY

Clock Cycles	Processing Elements															
	(0,0) to (0,15)	(1,0) to (1,15)	(2,0) to (2,15)	(3,0) to (3,15)	(4,0) to (4,15)	(5,0) to (5,15)	(6,0) to (6,15)	(7,0) to (7,15)	(8,0) to (8,15)	(9,0) to (9,15)	(10,0) to (10,15)	(11,0) to (11,15)	(12,0) to (12,15)	(13,0) to (13,15)	(14,0) to (14,15)	(15,0) to (15,15)
1	L1	L2														
2	K1	K2	L1	L2												
3	J1	J2	K1	K2	L1	L2										
4	H1	H2	J1	J2	K1	K2	L1	L2								
5	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2						
6	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2				
7	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2		
8	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2
9	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2
10	B1	B2	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2
11	A1	A2	B1	B2	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2