Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 5091519, 11 pages
http://dx.doi.org/10.1155/2016/5091519

Research Article

Hardware Efficient Architecture with Variable Block Size for Motion Estimation

Nehal N. Shah,[1] Harikrishna Singapuri,[2] and Upena D. Dalal[2]

[1]*Sarvajanik College of Engineering and Technology, Surat, India*
[2]*S V National Institute of Technology, Surat, India*

Correspondence should be addressed to Nehal N. Shah; nehal.shah@scet.ac.in

Received 27 July 2016; Revised 2 November 2016; Accepted 27 November 2016

Academic Editor: Jar Ferr Yang

Video coding standards such as MPEG-x and H.26x incorporate variable block size motion estimation (VBSME) which is highly time consuming and extremely complex from hardware implementation perspective due to huge computation. In this paper, we have discussed basic aspects of video coding and studied and compared existing architectures for VBSME. Various architectures with different pixel scanning pattern give a variety of performance results for motion vector (MV) generation, showing tradeoff between macroblock processed per second and resource requirement for computation. Aim of this paper is to design VBSME architecture which utilizes optimal resources to minimize chip area and offer adequate frame processing rate for real time implementation. Speed of computation can be improved by accessing 16 pixels of base macroblock of size 4×4 in single clock cycle using z scanning pattern. Widely adopted cost function for hardware implementation known as sum of absolute differences (SAD) is used for VBSME architecture with multiplexer based absolute difference calculator and partial summation term reduction (PSTR) based multioperand adders. Device utilization of proposed implementation is only 22k gates and it can process 179 HD (1920 × 1080) resolution frames in best case and 47 HD resolution frames in worst case per second. Due to such higher throughput design is well suitable for real time implementation.

1. Introduction

Digital video processing has been applied to a large number of consumer electronics products such as digital video recorders (DVR), personal digital assistants (PDA), digital cameras, and set top boxes. Motion estimation (ME), which plays most important role in video compression, is applied to evaluate the movement of blocks in the current frame. It aims to remove temporal redundancies that exist in video sequences, which results in substantial bit rate reductions. The block matching algorithm (BMA) is widely adopted for ME as it fits well with rectangular video frames as well as block based transforms and provides a reasonably effective temporal model.

In BMA, previous frame $f_{(k-1)}$ is considered as reference frame and frame f_k is called current frame. Macroblock (MB) of size $M \times N$ from current frame will look for its best match in region having maximum probability called search region in reference frame. Usually size of search region is considered as $[-p, +p]$ in x as well as in y direction which results in evaluation of $(2p+1)^2$ candidate macroblocks. The difference between the coordinates of current macroblock from current fame and best match candidate macroblock from reference frame is called displacement vector or motion vector (MV). Popular cost function in hardware implementation to identify best match is sum of absolute differences (SAD) which is described by

$$\mathrm{SAD}(u, v) = \sum_{x=1}^{M} \sum_{y=1}^{N} \left| f_k(x, y) - f_{(k-1)}(x + u, y + v) \right|. \quad (1)$$

Existing video coding standards offer variable block size video motion estimation to improve quality of encoding. Variable block size (VBS) motion compensated prediction (MCP) provides significant rate distortion performance gain over conventional fixed block size MCP but it involves massive computation and adds an extra burden to any ME architecture, in the form of additional hardware complexity,

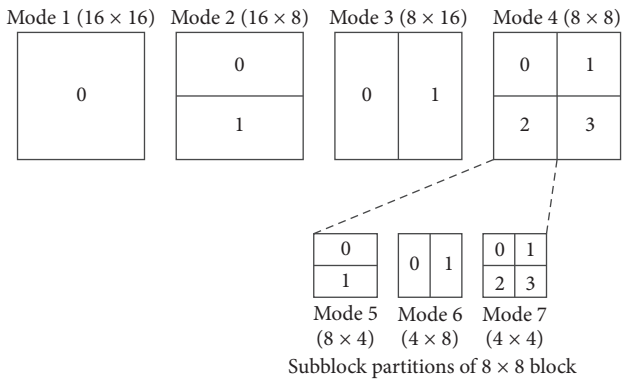

Figure 1: Macroblock modes [1].

extra computation time, or a combination of both. In H.264 standard of compression a typical macroblock has a dimension of 16 × 16 pixels which can be segmented in the smallest block size of dimension of 4 × 4 (base block) as shown in Figure 1. This division is represented as macroblock mode in Figure 1 and hence VBSs contain 16 × 16, 16 × 8, 8 × 16, 8 × 8, 8 × 4, 4 × 8, and 4 × 4 size blocks which results in 41 possible combinations of variable size. Due to block size ranging from 64 × 64 to 4 × 4 in recently developed HEVC standard, there are multifarious combinations of variable size.

To generate SAD value for all 41 possible combinations of 16 × 16 macroblock, 256 pixels are processed for current macroblock as well as for each candidate macroblock. There are several overlapping candidate macroblocks depending on the size of search area memory. Before SAD computation, reading pixels of macroblocks from different memory is most significant task. To serve the purpose, raster scan [4], meander scan [5], z scan [3], or spiral scan patterns are used. Based on pixel reading mechanism, architecture will perform absolute difference and accumulation of difference, and finally comparator will identify which block size is best suited for particular macroblock among various candidate macroblocks. In this paper Section 2 surveys existing VBSME architectures and their scanning patterns. Architecture based on z pattern is presented in Section 3. Section 4 describes simulation and synthesis results and comparison with existing architecture which is followed by conclusion.

2. Macroblock Scanning Pattern and VBSME Architectures

There has been large development done by researchers in the field of variable size block matching. VBSME with 41 possible combinations of variable size is highly time consuming and quite complex from hardware implementation perspective due to huge computation. In this section existing architectures for VBSME are discussed. Full search VBSME architectures [2–9] are able to perform a full motion search on various size of macroblocks.

VBSME unit initially reads current macroblock from current frame and candidate macroblocks from reference frame, divided into 3 stages. The very 1st stage is used to compute absolute difference between corresponding element of current macroblock data and reference macroblock data. The second stage is to calculate intermediate results to generate 41 different SAD values. The data is partially stored in buffer and also forwarded to third stage which is used to generate all SAD values which are useful for the generation of MVs. Various architectures with different scanning pattern gives a variety of performance results for motion vector (MV) generation showing tradeoff between macroblock processed per second and resource requirement for computation. To generate SAD value for all possible combinations of macroblocks all pixels are read using traditional raster scan pattern for 16 × 16 macroblock as shown in Figure 2 for architectures presented in [2, 4, 6, 7]. On the other hand, architectures presented in [5, 9] use meander scan and architecture presented in [3] uses z scan pattern as shown in Figure 3. Based on pixel reading mechanism architecture will perform absolute difference and accumulation of difference and finally comparator will identify which block size is best suited for particular macroblock among various candidate macroblocks.

16 × 16 macroblock can be segmented into 16 small blocks of size 4 × 4 as indicated in Figure 4 where various small blocks are labels with b0 to b15. In horizontal raster scan pattern of Figure 2(a), first row of blocks b0, b1, b2, and b3 are read while in vertical raster scan pattern of Figure 2(b) first column of blocks b0, b4, b8, and b12 are read. However both types of scan, horizontal and vertical, provide same results in context of resource utilization as well as number of clock cycles required for reading pixels. In VBSME architectures 1, 4, or 16 pixels are read simultaneously and processed in processing elements (PEs) to generate SAD combinations. For parallel processing of pixels architectures prefer multiple PEs which can be 4, 16, 64, or even 256. Most of architectures use 16 × 16 search range which is extended to 32 × 32 in few of the architectures. The VBSME architecture presented in [2] is based on 16 PEs. The current macroblock data is arranged in a raster scan sequence and search region data is arranged in a dual raster scan sequence. 16 SAD values are being computed, each with block size 4 × 4. The stored SAD values are then reused to compute SAD values for other block sizes. This is done by shuffling and combining the computed subblock SAD values appropriately to derive SAD for each of the other larger block sizes. This avoids the need to compute each of these from scratch and allow up to 41 SAD values to be processed in a single processor. Architectures presented in [2–4] read single pixel at a time and can process only one pixel of current macroblock and candidate macroblock using particular PE in single clock cycle and hence consume 282 clock, 271 clock, and 262 clock cycles, respectively, to generate 41 SAD combinations. Architecture presented in [4] uses 18 × 1 multiplexers as well as latches and eliminates the intermediate buffer requirement need compared to architecture presented in [2]. PEs are arranged in 4 × 4 array in architecture explained in [3] and it uses single pixel z scan for reading pixel from reference and current frame. The pixel values are fed through shift registers to 16 PEs which are arranged in 4 × 4 array. Concept

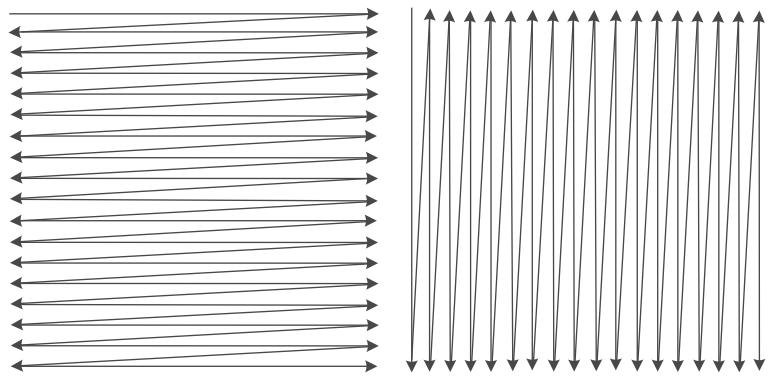

Figure 2: Scanning order for 16 × 16 macroblock. (a) horizontal raster scan [2]. (b) vertical raster scan.

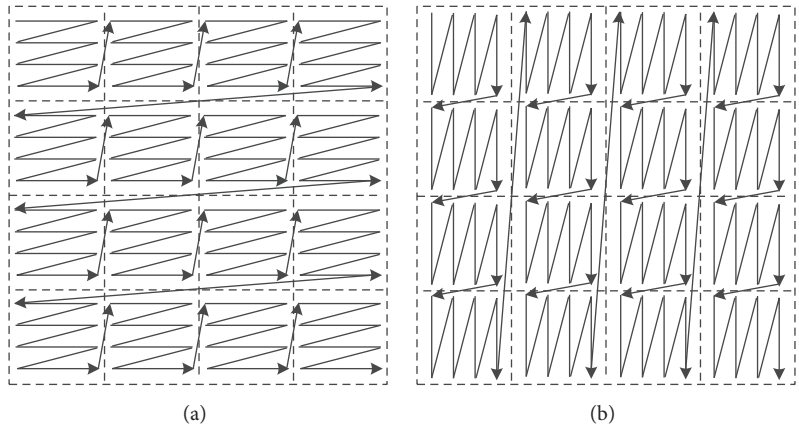

Figure 3: Scanning order for 16 × 16 macroblock. (a) Horizontal z scan [3]. (b) Vertical z scan.

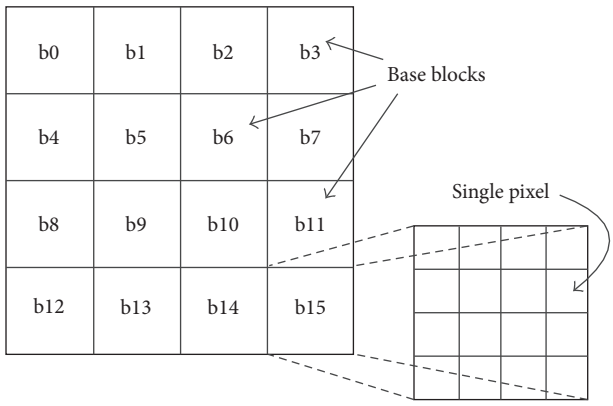

Figure 4: 16 × 16 macroblock segmented into 16—4 × 4 submacroblock [2].

is replicated several times to compute multiple candidate macroblocks in given search window. By using scanning pattern of [4] and reading 4 pixels at a time clock cycles required to generate 41 combinations reduce to 70 which is approximately 4 times lesser as indicated in [7]. Same author has also presented the extended version of architecture for 16-pixel processing in which the number of clock cycles required to generate the same 41 combinations is reduced to 20 which is lesser by factor 16. Architecture proposed in [5] deals with 16 pixels at each clock cycle with 16 computing units. Each computing unit has 16 PEs. Thus total 256 PEs are used for generation of SAD values for 16 × 16 macroblock size. It uses meander like scan pattern for search area. After surveying various architectures, with variety of scanning patterns we can summarize that at least 20 clock cycles are needed to compute 41 SAD combinations.

3. Proposed Architecture

3.1. Pixel Reading Pattern. In this section VBSME architecture is presented with aim of generating 41 SAD combinations of variable size macroblock in optimal clock cycles with reduced resource utilization. Instead of using conventional raster scan pattern, proposed architecture uses z scan pattern,

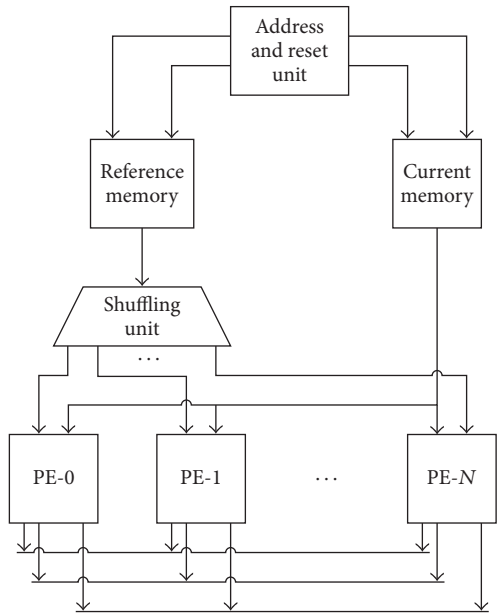

Figure 5: Proposed hardware implementation of VBSME.

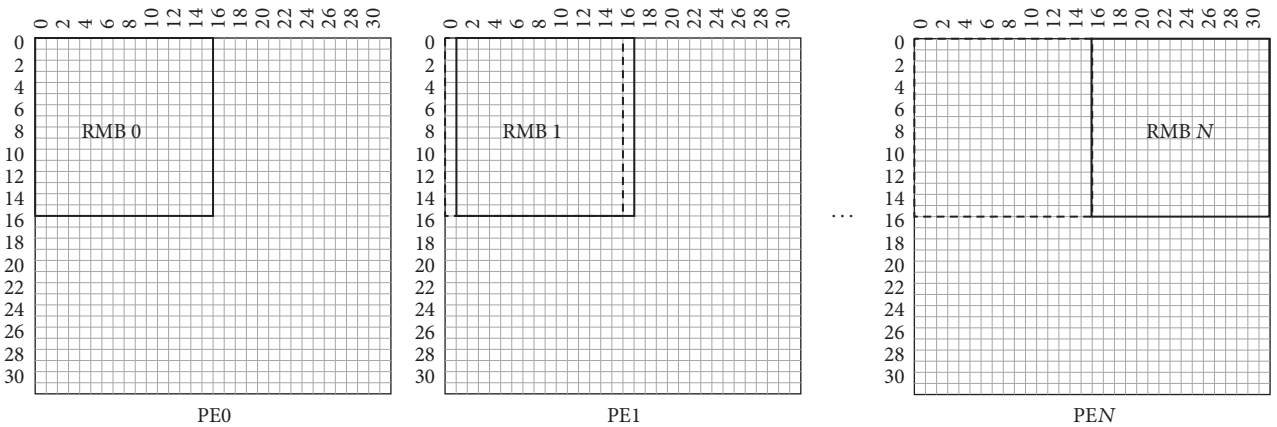

Figure 6: Location of RMBs in search area.

to read 16 pixels at a time from memory as shown in Figure 4. Due to such pattern smallest block of size 4×4 can be read at a time. Once base block is available in very next cycle SAD for that block is computed. Hence in two clock cycles blocks b0 and b1 are available and first 4×8 combination can be computed. Such scanning pattern will eliminate need of storing pixel values of intermediate row or column.

3.2. Architecture Description. Figure 5 shows multiple processing elements (PEs) of proposed VBSME architecture. Each PE computes 41 SAD combinations of current macroblock and corresponding candidate macroblock from reference memory called reference memory block (RMB). For window size of p there will be $(2p + 1)^2$ candidate RMBs that need to be processed. By choosing $N = (2p + 1)$, architecture can calculate SAD of current macroblock and $(2p + 1)$ RMBs together and by repeating process $(2p + 1)$ times SAD values for all candidate macroblocks are available. Figure 6 shows location of RMBs for various processing unit and Table 1 shows the data scheduling for the proposed architecture with 17 PEs.

As shown in Table 1, in very 1st cycle submacroblock b0 is read from both reference and current memory and fed to the processing element PE0. At the same time all other PEs also get same submacroblock from current memory but 1 column shifted submacroblock from reference memory. Due to proposed scanning pattern sixteen pixels are scanned together and their SAD values will be available in next clock cycle. Buffer is needed to store SAD value of this smallest size 4×4 submacroblock.

The processing element used in Figure 5 is represented in detail in Figure 7. The architecture is divided into multiple stages, namely, absolute difference calculation (ADC), addition of absolute difference, and generation of 41 SAD combinations. To compute absolute difference, multiplexer

Table 1: Pixel data scheduling for VBSME architecture.

Clock cycle	PE0	PE1	⋯	PE15	PE16
0	C(0:3, 0:3), R(0:3, 0:3)	C(0:3, 0:3), R(0:3, 1:4)	⋯	C(0:3, 0:3), R(0:3, 15:18)	C(0:3, 0:3), R(0:3, 16:19)
1	C(0:3, 4:7), R(0:3, 4:7)	C(0:3, 4:7), R(0:3, 5:8)	⋯	C(0:3, 4:7), R(0:3, 19:22)	C(0:3, 4:7), R(0:3, 20:23)
⋮	⋮	⋮	⋮	⋮	⋮
14	C(12:15, 8:11), R(12:15, 8:11)	C(12:15, 8:11), R(12:15, 9:12)	⋯	C(12:15, 8:11), R(12:15, 23:26)	C(12:15, 8:11), R(12:15, 24:27)
15	C(12:15, 12:15), R(12:15, 12:15)	C(12:15, 12:15), R(12:15, 13:16)	⋯	C(12:15, 12:15), R(12:15, 27:30)	C(12:15, 12:15), R(12:15, 28:31)
16	C(0:3, 0:3), R(1:4, 0:3)	C(0:3, 0:3), R(1:4, 1:4)	⋯	C(0:3, 0:3), R(1:4, 15:18)	C(0:3, 0:3), R(1:4, 16:19)
⋮	⋮	⋮	⋮	⋮	⋮
30	C(12:15, 8:11), R(13:16, 8:11)	C(12:15, 8:11), R(13:16, 9:12)	⋯	C(12:15, 8:11), R(13:16, 23:26)	C(12:15, 8:11), R(13:16, 24:27)
31	C(12:15, 12:15), R(13:16, 12:15)	C(12:15, 12:15), R(13:16, 13:16)	⋯	C(12:15, 12:15), R(13:16, 27:30)	C(12:15, 12:15), R(13:16, 28:31)

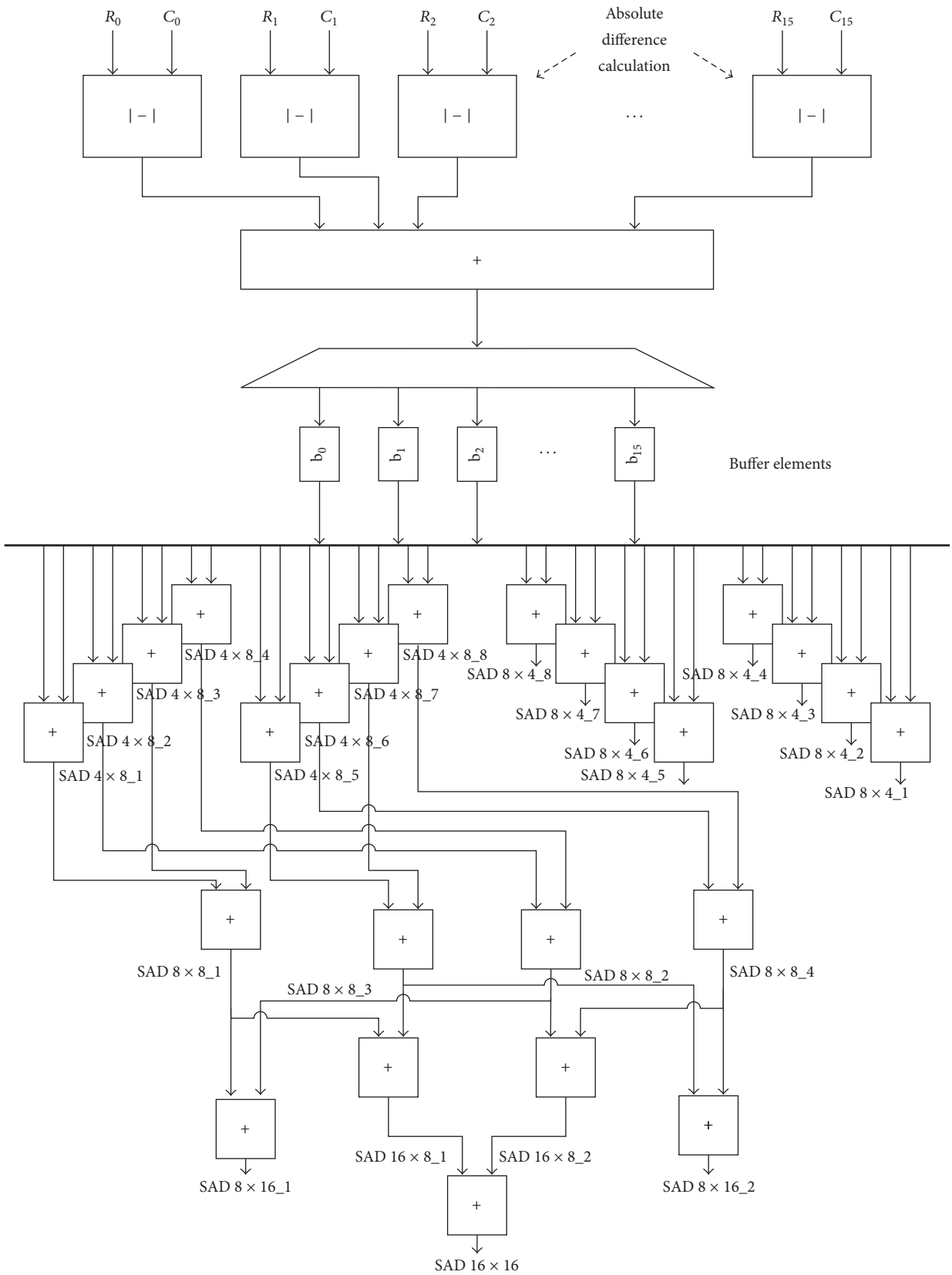

Figure 7: Detailed PE structure.

based ADC presented in [10] and, for addition of operands, adder presented in [11] are used. 16 reference macroblock pixels and 16 current macroblock pixels are fed to the ADC unit and result is forwarded to adder block. Adder block sums up all the difference values and stores them to the respective intermediate buffer labelled as b0 to b15. 1×16 demultiplexer is used to select respective buffer to compute 4×8, 8×4, 8×16, 16×8, and 16×16 combination further using multilevel addition. Summation of macroblock sizes less than 16×16 is kept on respective data buses for further computation and finally 41 combinations for VBSME are ready.

At the end of 16 clock cycles according to schedule of Table 1 all 4×4 submacroblocks are read and their individual SAD values are available as shown in Table 2. At very next, that is, on 17th clock, the remaining 25 combinations are computed. Thus all 41 SAD values are available in total 17 clock cycles in all PEs. Immediately RMBs are shifted to next rows and computation of $(2p + 1)$ combinations of that particular row is started.

Once all SAD values are available in $(2p + 1)$ PEs, comparators identify best possible combination for $(2p + 1)$ RMBs which is stored and compared with next row of RMBs. After evaluation of all $(2p + 1)^2$ RMBs, best match macroblock is identified which is followed by motion vector computation. Then, next macroblock from current frame is evaluated. Latency between two consecutive macroblocks of current frame depends on time required to read search area. Due to 128-bit data bus 16 pixels are read from reference frame concurrently, which takes 48 clock cycles for very first macroblock and 64 clock cycles for the rest of the macroblocks if single search area memory is used. In this work three search area memories are incorporated which are used in round robin fashion. When $p = 8$ is chosen, then 50% search areas for two consecutive macroblocks are overlapped; hence at the time of filling one memory, pixels are filled in next memory also. Due to this arrangement, at the time of motion vector computation for any macroblock, search area memory is prepared for next macroblock; hence there is no latency between successive macroblocks.

3.3. Synthesis Results of Proposed VBSME Architecture. Proposed VBSME hardware architecture is implemented and tested in terms of various evaluation metrics. Architectures have been implemented using VHDL and synthesized using Xilinx FPGA family Spartan3 and Virtex5 with chip XC3s400 and XC5vlx50, respectively. Current memory size is chosen as 16×16 pixels due to macroblock size of 16×16 while reference memory size is 32×32 pixels by considering search window parameter p as 8. Table 3 shows macrostatistics for proposed implementation. Architecture is optimized for adder subtractors and other resources hence demonstrating very low gate count of only 22k. Synthesis delay of design is only 2.543 ns offering maximum frequency of 393.16 MHz. At maximum frequency it can process 179 HD (1920×1080) frames in one second. Post place and route delay is 9.72 ns which is considered as worst case delay in which 47 HD (1920×1080) frames can be processed per second at frequency of 102 MHz.

Table 2: SAD output schedule for VBSME architecture.

Clock	Block	Size
1	0	4×4
2	1	4×4
3	0, 1	4×8
	2	4×4
4	3	4×4
5	2, 3	4×8
	4	4×4
6	0, 4	8×4
	5	4×4
7	6	4×4
	1, 5	8×4
	4, 5	4×8
	0, 1, 4, 5	8×8
8	7	4×4
	2, 6	8×4
9	8	4×4
	3, 7	8×4
	6, 7	8×4
	2, 3, 6, 7	8×8
	0, 1, 2, 3, 4, 5, 6, 7	8×16
10	9	4×4
11	10	4×4
	8, 9	4×8
12	11	4×4
13	12	4×4
	10, 11	4×8
14	13	4×4
	8, 12	8×4
15	14	4×4
	9, 13	8×4
	12, 13	4×8
	8, 9, 12, 13	8×8
	0, 1, 4, 5, 8, 9, 12, 13	16×8
16	15	4×4
	10, 14	8×4
17	11, 15	8×4
	14, 15	4×8
	10, 11, 14, 15	8×8
	8, 9, 10, 11, 12, 13, 14, 15	8×16
	2, 3, 6, 7, 10, 11, 14, 15	16×8
	Full macroblock	16×16

Table 4 indicates the comparison between the existing VLSI implementation of VBSME and proposed implementation. Similar comparison between the existing FPGA implementation of VBSME and proposed implementation is shown in Table 5. Most of architectures are implemented with variable block sizes from 16×16 to 4×4 presented in [14] which is limited to block size between 16×16 and 8×8. Architectures presented in [7, 16] are demonstrated for search range 16×16;

Table 3: Macrostatistics for VBSME architecture.

Adders/subtractors	1343
12-bit adder	255
13-bit adder	136
14-bit adder	68
15-bit adder	34
16-bit adder	17
4-bit subtractor	17
8-bit adder	816
Comparators	2
6-bit comparator equal	1
6-bit comparator greater	1
Counters	21
4-bit up counter	17
5-bit up counter	2
6-bit up counter	2
Registers	76
16-bit register	16
8-bit register	60
12-bit latches	272

therefore they can evaluate only one candidate macroblock. The rest of architectures are tested with search range 32×32 or 33×33. Most of VLSI implementations are 180 nm or 130 nm technology while FPGA implementations are using Virtex series. Implementation parameters like search area, pixel scanning pattern, data bus width to read pixels, and number of PEs are diverse for various designs; hence to evaluate their performance number of macroblocks processed per second and frame processing rates are an important criterion.

The architecture proposed in this design works on 16 pixels' scanning which results in higher throughput compared to not only 1-pixel scan and 4-pixel scan architecture but also existing 16-pixel scan architectures. In comparison with 16-pixel raster scan architecture of Warrington et al. [7] proposed architecture can process 3 times more HD frames even in worst case and offers 7 times lesser gate count while compared to 16-pixel meander scan architecture of Wei et al. [5] it can process more than 2 times HD frames with 16 times less processing elements. Gate count of López et al. [6] architecture is comparable with proposed architecture but it offers frame rate of only 60 fps for CIF resolution which in actuality is very less. Gate count of [15] is lesser compared to proposed design but frame processing rate is not given and therefore is not adequate for comparison. Architecture presented by Olivares [12] can process 21.42 HD (1920×1080) resolution frames with 256 PEs; still this frame rate is not sufficient for real time implementation. From comparison among FPGA implementation of VBSME architectures also we can observe that number of LUTs used by proposed design is higher but at same time design offers higher frame processing rate. From overall comparison with various 16 pixels' scan architectures we can derive that proposed architecture outperforms in terms of throughput.

For the advance comparison of architecture, in addition to frame processing rate, hardware efficiency E_H [5] is used which is defined as the ratio of data throughput rate TP over hardware cost in terms of resource utilization or gate count. TP is defined by the number of macroblocks processed by architecture per second. Equation (2) indicates hardware efficiency and its unit is macroblocks per second per gate. To evaluate the architecture efficiency in terms of power, E_P can be defined as ratio of TP over the power as shown in (3). Unit of E_P is macroblocks per second per mW. With higher E_H and E_P, architecture is more efficient.

$$E_H = \frac{\text{TP}}{G} = \frac{\text{Number of macroblock/sec}}{G}, \tag{2}$$

$$E_P = \frac{\text{TP}}{\text{Power}} = \frac{\text{Number of macroblock/sec}}{\text{Power}}. \tag{3}$$

As per (2) and (3) hardware and power efficiency are computed for existing and proposed VBSME implementation and shown in Table 6. Hardware efficiency of proposed architecture in comparison with existing architectures is more than 5 times enhanced in worst case while it is more than 19 times superior in best case. In terms of power efficiency, proposed implementation produces similar results as implementation presented by Fatemi et al. [13]. Other than that power efficiency of proposed architecture is better than other architectures in best case. In comparison of some of the architectures, proposed design uses somewhat more gates but throughput of proposed design is higher compared to all existing architectures. Overall comparison indicates that proposed VBSME architecture is hardware efficient and power efficient.

4. Conclusion

In this paper, architecture for full search variable block size motion estimation is described. Architecture makes calculation for all 41 combinations of variable block size motion vector considering 289 candidate macroblocks in search area of 32×32. Architecture described in this paper uses 16-pixel z scan pattern to access pixels of current macroblock and 17 candidate macroblocks and can compute all 41 combinations of 16×16 macroblock in only 16 clock cycles. Process is repeated 17 times using 17 processing elements, hence in 272 clock cycles all the combinations of all candidate macroblocks are available based on which best match and motion vector is computed. Device utilization of proposed implementation is only 22k and it can process 179 HD (1920×1080) resolution frames in best case and 47 HD resolution frames in worst case per second. Implementation results show that proposed VBSME architecture outperforms in area utilization compared to existing 1-pixel scan, 4-pixel scan, and 16-pixel scan architectures due to 16-pixel z scanning pattern. VBSME architecture demonstrates 19 times better hardware efficiency in comparison with other VBSME implementations. Power efficiency of proposed VBSME architecture is either better or comparable with existing implementations. Architecture can be configured with more PEs to suffice need of extended

Table 4: Comparison among VLSI implementations of VBSME architectures.

VBSME architecture	Search range	# of PEs	# of pixels	# of clock cycles to generate 41 SAD	# of clock cycles to generate MV	Frequency (MHz)	Frame processing rate (fps)	Technology	Gate count
Yap and McCanny [4]	32 × 32	16	1	281	4496	100	52 @CIF	130 nm	108k
Yap and McCanny [2]	32 × 32	16	1	262	4096	294	181 @CIF	130 nm	61k
Wei et al. [5]	33 × 33	256	16	40	1129	180	409 @CIF 45 @720p	180 nm	160k + 3.328 kB SRAM
López et al. [6]	31 × 31	16	16	—	—	100	60 @CIF	250 nm	21.3k
Warrington et al. [7]	16 × 16	16	16	20	—	155	90 @SD	180 nm	155k
Kim and Park [3]	32 × 32	16	1	262	16384	416	256 @CIF	180 nm	39.2k
Ruiz and Michell [9]	32 × 32	64	4	65	1207	300	30 @1080p	180 nm	32.3k + 59 kB SRAM
Olivares [12]	32 × 32	256	16	—	4913	380.1	21.42 @1080p	130 nm	54k + 2.76 kB SRAM
Fatemi et al. [13]	32 × 32	256	4	90	5120	207	30 @SD	180 nm	31.5k
Tung et al. [14]	—	16	16	18	—	546.4	—	180 nm	149.2k
Parandeh-Afshar et al. [15]	—	4	4	64	—	285	—	130 nm	18k
Proposed	32 × 32	17	16	17	272	393.16	179 @1080p	130 nm	22k

Table 5: Comparison among FPGA implementations of VBSME architectures.

VBSME architecture	Search range	# of PEs	# of pixels	# of clock cycles to generate 41 SAD	# of clock cycles to generate MV	Frequency (MHz)	Frame processing rate (fps)	FPGA	LUTs
Olivares [12]	32 × 32	256	16	—	4913	380.1	21.42 @1080p	Virtex 5	3768
Elhamzi et al. [16]	16 × 16	16	16	—	4096	436	13 @1080p	Virtex 6	1281
Parandeh-Afshar et al. [15]	—	4	4	64	—	285	—	Virtex 2	1431
Proposed	32 × 32	17	16	17	272	393.16	179 @1080p	Virtex 5	9486

Table 6: Comparison of hardware and power efficiency for VBSME architectures.

Architecture	Frame processing rate (fps)	Gate count (k)	Power (mW)	TP in kMB/sec	E_H in MB/sec/gate	E_P in MB/sec/mW
Yap and McCanny [2]	181 @CIF	61	570 mW	71.676	1.175	125.75
Wei et al. [5]	409 @CIF 45 @720p	163.32	423 mW	162	0.992	383
López et al. [6]	60 @CIF	21.3	—	23.76	1.11	—
Warrington et al. [7]	90 @SD	155	68 mW/70 kMB/s	324	2.09	1029.41
Kim and Park [3]	256 @CIF	39	—	101.38	2.6	—
Ruiz and Michell [9]	30 @1080p	91.3	115 mW	243	2.66	2113.04
Olivares [12]	21.4 @1080p	56.76k	314 mW	173.5	3.06	552.55
Fatemi et al. [13]	30 @SD	31.5	40.07 mW	108	3.43	2695.3
Parandeh-Afshar et al. [15]	—	18k	7.7 mW	9.615	0.53	1248.70
Proposed	179 @1080p	22k	540 mW	1449.9	65.9	2685

search area. With adequate frame processing rate architecture is well suited for real time implementation.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[2] S. Y. Yap and J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 7, pp. 384–389, 2004.

[3] J. Kim and T. Park, "A novel VLSI architecture for full-search variable block-size motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 728–733, 2009.

[4] S. Y. Yap and J. V. McCanny, "A VLSI architecture for advanced video coding motion estimation," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '03)*, pp. 293–301, IEEE, June 2003.

[5] C. Wei, H. Hui, T. Jiarong, L. Jinmei, and M. Hao, "A high-performance reconfigurable VLSI architecture for VBSME in H.264," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1338–1345, 2008.

[6] S. López, G. M. Callicó, F. Tobajas, J. F. López, and R. Sarmiento, "A flexible template for H.264/AVC block matching motion estimation architectures," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 845–851, 2008.

[7] S. Warrington, W.-Y. Chan, and S. Sudharsanan, "Scalable high-throughput variable block size motion estimation architecture," *Microprocessors and Microsystems*, vol. 33, no. 4, pp. 319–325, 2009.

[8] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 3, pp. 578–593, 2006.

[9] G. A. Ruiz and J. A. Michell, "An efficient VLSI processor chip for variable block size integer motion estimation in H.264/AVC," *Signal Processing: Image Communication*, vol. 26, no. 6, pp. 289–303, 2011.

[10] S. Rehman, R. Young, C. Chatwin, and P. Birch, "An FPGA based generic framework for high speed sum of absolute difference implementation," *European Journal of Scientific Research*, vol. 33, no. 1, pp. 6–29, 2009.

[11] N. N. Shah, K. R. Agarwal, and H. M. Singapuri, "Implementation of sum of absolute difference using optimized partial summation term reduction," in *Proceedings of the International Conference on Advanced Electronic Systems (ICAES '13)*, pp. 192–196, IEEE, September 2013.

[12] J. Olivares, "A low cost architecture for variable block size motion estimation," *Journal of Signal Processing Systems*, vol. 68, no. 1, pp. 127–138, 2012.

[13] M. R. H. Fatemi, H. Ates, and R. Salleh, "Analysis and design of low-cost bit-serial architectures for motion estimation in H.264/AVC," *Journal of Signal Processing Systems*, vol. 71, no. 2, pp. 111–121, 2013.

[14] D. M. Tung, T. Le, and T. Dong, "A VLSI architecture for H.264/AVC variable block size motion estimation," *Journal of Automation and Control Engineering*, vol. 3, no. 1, pp. 51–55, 2015.

[15] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Scalable and low cost design approach for variable block size motion estimation (VBSME)," in *Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT '09)*, pp. 271–274, April 2009.

[16] W. Elhamzi, J. Dubois, J. Miteran, and M. Atri, "An efficient low-cost FPGA implementation of a configurable motion estimation for H.264 video coding," *Journal of Real-Time Image Processing*, vol. 9, no. 1, pp. 19–30, 2014.