DOI: 10.21917/ijivp.2015.0142

SAD PROCESSOR FOR MULTIPLE MACROBLOCK MATCHING IN FAST SEARCH VIDEO MOTION ESTIMATION

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

[1]*Department of Electronics and Communication Engineering, Sarvajanik College of Engineering and Technology, India*
E-mail: nehal.shah@scet.ac.in

[2] *Department of Electronics and Communication Engineering, Sardar Vallabhbhai National Institute of Technology, India*
E-mail: udd@eced.svnit.ac.in

Abstract

Motion estimation is a very important but computationally complex task in video coding. Process of determining motion vectors based on the temporal correlation of consecutive frame is used for video compression. In order to reduce the computational complexity of motion estimation and maintain the quality of encoding during motion compensation, different fast search techniques are available. These block based motion estimation algorithms use the sum of absolute difference (SAD) between corresponding macroblock in current frame and all the candidate macroblocks in the reference frame to identify best match. Existing implementations can perform SAD between two blocks using sequential or pipeline approach but performing multi operand SAD in single clock cycle with optimized recourses is state of art. In this paper various parallel architectures for computation of the fixed block size SAD is evaluated and fast parallel SAD architecture is proposed with optimized resources. Further SAD processor is described with 9 processing elements which can be configured for any existing fast search block matching algorithm. Proposed SAD processor consumes 7% fewer adders compared to existing implementation for one processing elements. Using nine PE it can process 84 HD frames per second in worse case which is good outcome for real time implementation. In average case architecture process 325 HD frames per second.

Keywords:

Motion estimation (ME), Block Matching Algorithm (BMA), Sum of Absolute Difference (SAD), Processing Element (PE), Macroblock (MB), SAD processor, Diamond Search Architecture

1. INTRODUCTION

Video compression involves block based motion estimation (ME) [1] between successive frames for temporal redundancy reduction. It is defined as searching the motion vector which is the displacement of the coordinate of the best similar macroblock (MB) in reference frame for the macroblock in current frame. The most commonly used metric to calculate the resemblance is the Sum of Absolute Difference (SAD), which adds up the absolute differences between corresponding elements in the candidate and current block. Candidate MB having minimum distortion is treated as best match. Full search or fast search algorithm decides how many candidate MBs are searched which is most time consuming process in estimation.

SAD processor consists of N processing elements (PE) for N candidate MBs, each PE consists of three parts; absolute difference calculation between corresponding elements of MB, performing addition of all resultant operands and finding MB having minimum SAD value among all candidate blocks. The hardware implementation of multi operand adders has been addressed using various approaches among them most common approach is compressor trees utilizing different heuristics. Implementation of 8:4 and 9:4 compressors [2] based on full adders and mux demonstrates reduction in vertical critical path and number of stages in multi operand addition. Mux based adders offer less delay and consumes less power at cost of higher area. Based on carry save compressor tree, 9:2 as well 11:2 compressors are presented in [3] which claims better speed compared to carry propagate adders (CPA). High speed low power 15:4 compressor based on 5:3 compressor is discussed in [4]. Modern FPGA includes specific hardware dedicated for fast carry propagation. Efficient implementation of carry save adder on FPGA is discussed in [5] which offers less delay compared to Radix-4 CSA. Another variant of carry save adder on FPGA is discussed in [6]. Pipelined adder implementation based on FPGA is presented in [7]. SAD implementation on FPGA for 16 × 16 MB is discussed in [8]. SAD16 unit can be used to perform the complete16 × 16 operation, either by replicating the unit 16 times or exploiting its pipeline characteristic. The SAD16 implementation produces its first result after19 clock cycles. By replicating the SAD16 unit and adding another adder tree, the resulting fully parallelized implementation requires 27 clock cycles to produce the 16 × 16 SAD result. Another area efficient method utilizing the pipelined SAD16 unit requires 42 clock cycles to perform the 16 × 16 SAD operation. The study shows that sequential architecture uses less resources but it also gives lesser throughput which is not suitable for real time. While pipeline architecture uses moderate resources and offer reasonable throughput, and parallel architecture provides maximum throughput at the expense of highest resource utilization. In paper [9] comparative analysis for sequential, pipeline and parallel architectures for SAD 16 × 16 implementation is given and it proposes partial summation term reduction approach for multi operand addition. There is always trade-off between the maximum throughput and resource utilization. In paper [10], FPGA implementation of the Sum of Absolute Differences (SAD) algorithm is introduced which accomplish correlation based wavefront sensing. 4 × 4 SAD is implementation on FPGA is presented in [11] and 16 × 16 SAD implementation in 29 clock cycles is presented in [12].

In this paper configurable architecture for SAD processor using multiple processing elements for fast search block matching algorithm is presented which can process all candidate MBs in one clock cycle and offers optimal resource utilization. Section 2 shows top level module for SAD processor consisting multiple processing elements and evaluates various approaches for finding SAD of 8 × 8 MB based on existing multi operand addition schemes. Experimental results are shown in section 3 for one PE and then SAD processor with nine PEs configured for diamond search BMA and final remark is presented in section 4.

2. SAD PROCESSING ELEMENTS BASED ON VARIOUS APPROACHES OF MULTI OPERAND ADDITION

2.1 MOTION ESTIMATION ARCHITECTURE FOR FAST SEARCH BMA

Existing fast search BMAs are based on diamond or hexagon shape or their hybrid version and demands 7 to 9 processing elements in parallel to process multiple candidate MBs. The general block diagram for estimating motion in fast search BMAs is shown in Fig.1 with nine processing elements which can be configured according to requirement. According to graphical pattern of fast search BMAs the candidate blocks from search window can be organized in memory m0, m1 up to m8. As an example location of 9 candidate MBs in search area for diamond search pattern is shown in Fig.2. After memory organization the next step is to calculate the best match for the current block using SAD.

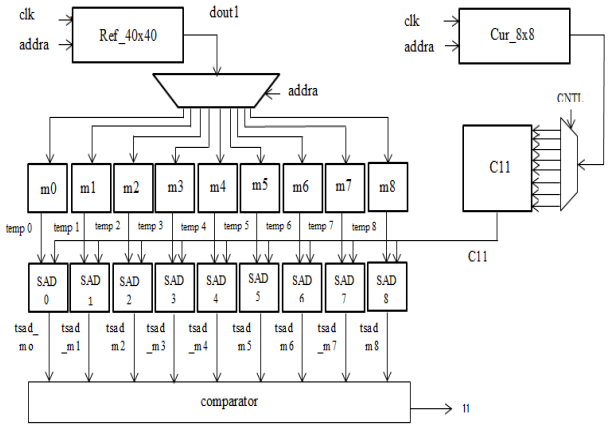

Fig.1. Configurable SAD processor for any fast search BMA

The equation for calculating the sum of absolute difference is given by Eq.(1).

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

$$ADC_{u,v}=\begin{Bmatrix} x-y=x+\overline{y}+1, MSB=0 \\ \overline{y-x=x+\overline{y}+1}+1, MSB=1 \end{Bmatrix} \qquad (2)$$

SAD, is calculated in three steps as shown in Fig.3. First computation of absolute differences between corresponding elements is done by A.D. block. Addition of all difference values of the one row is done by addition scheme block. Finally the addition of all individual eight rows is done. For absolute difference calculation Eq.(2) [11] is used. 2's complement of a signed complemented number results in the original number. First X-Y is calculated assuming that X > Y. If it is true then the MSB of X-Y will be zero, otherwise again calculate the 2's complement of (X-Y). This absolute difference computation scheme utilizes fewer resources and results in fast absolute difference. After calculating the absolute difference, addition schemes are used. Various multi operand addition schemes are available in literature as mentioned before. Few of them are modified for computation of fix size MB and incorporated in SAD processor.

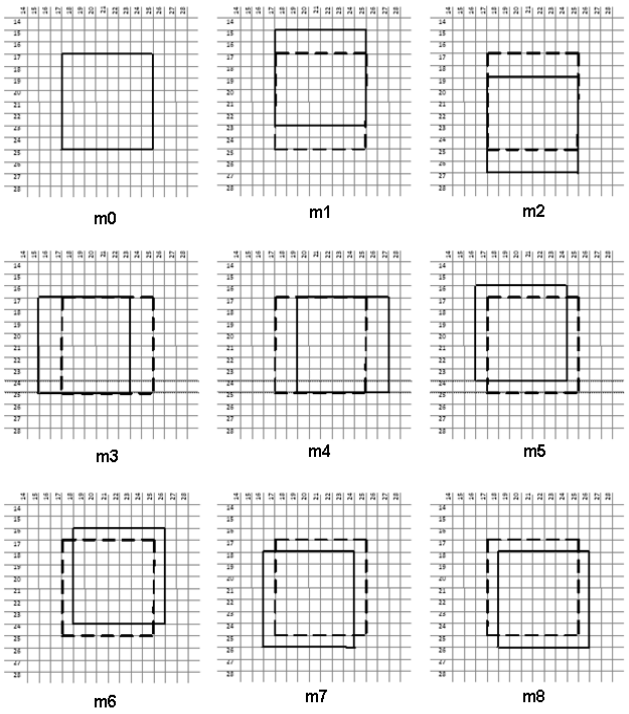

Fig.2. Location of candidate MBs in search area for diamond search pattern

2.2 MULTI OPERAND ADDITION SCHEMES FOR FIX SIZE MACROBLOCK

2.2.1 Addition using Hierarchical Adders:

Most familiar addition scheme is addition in hierarchy [9]. In this method SAD of one row of a macroblock is performed as shown in Fig.4. The difference of reference and current pixel is calculated in first step, then an absolute difference is obtained and finally these pixels are added in hierarchy. SAD calculation of one row is replicated as shown in Fig.3 and all eight rows are added.

2.2.2 Addition using 8:4 Compressor:

Literature exhibit column addition based multi operand addition schemes which are evaluated here. One row of 8 × 8 macroblock has 8 pixels, and each pixel has 8 bits as shown in Fig.5(a). The Fig.5(b) shows 4:2 compressor and Fig.5(c) shows 8:4 compressor [2] having 8 inputs (I0-I7), four outputs (X1-X4) both are based on full adder and half adder. This compressor uses counter property so that, output of compressor gives number of 1's at input. For example, if all input bits are 1, then output of the compressor is "1000". This design of 8:4 compressor takes three stages of adders to compress the input bits into four output bits. Totally, four full adders and three half adders are used. 4:2 compressor uses additional two full adders. 3:2 compressor itself is full adder. As shown in Fig.5(d) final addition of one column is generated, by combining all three compressors. Column addition is replicated 8 times to find sum of all 64 pixels of macroblock. In this method pixels are fed after absolute difference hence SAD is obtained as a result.

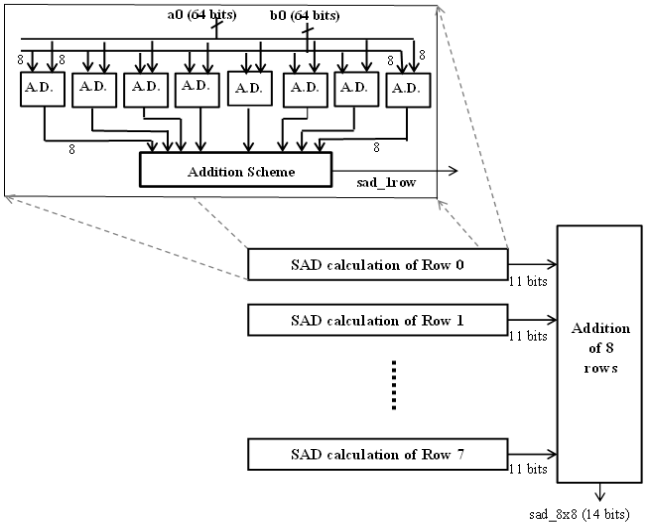

Fig.3. Sad calculation for one macroblock (8 × 8) [13]

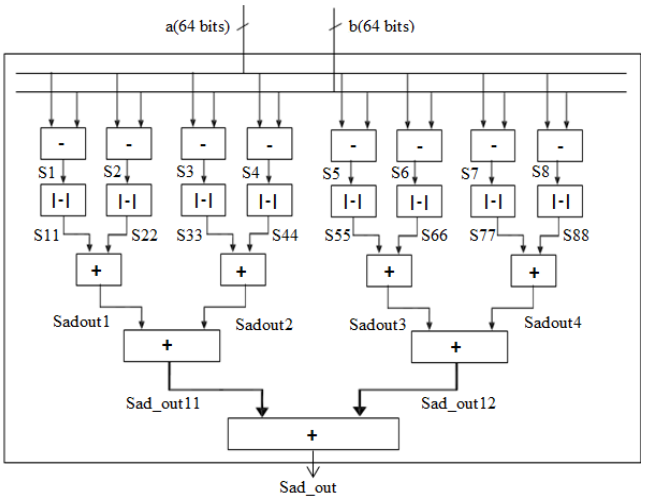

Fig.4. SAD calculation of one row using Hierarchical addition

2.2.3 Addition Using Carry Save Adder (CSA):

Another variant of addition scheme is possible based on 3:2 carry save adder as shown in Fig.6(a). In this method the sum of four bits of one column is calculated, the name of this module is given as CSA 4bits [5]. After that this CSA 4bits module is replicated eight times, so that addition of four rows are calculated, this is shown in Fig.6(a) with curly bracket and also shown in Fig.6(b). The name for this module is given as CSA 4 rows. Finally this CSA 4 rows module replicated as shown to perform addition of all eight pixels of macroblock which is SAD calculation of 1 row of Fig.3. The name of this module is given as CSA 8 rows as shown in Fig.6(b).

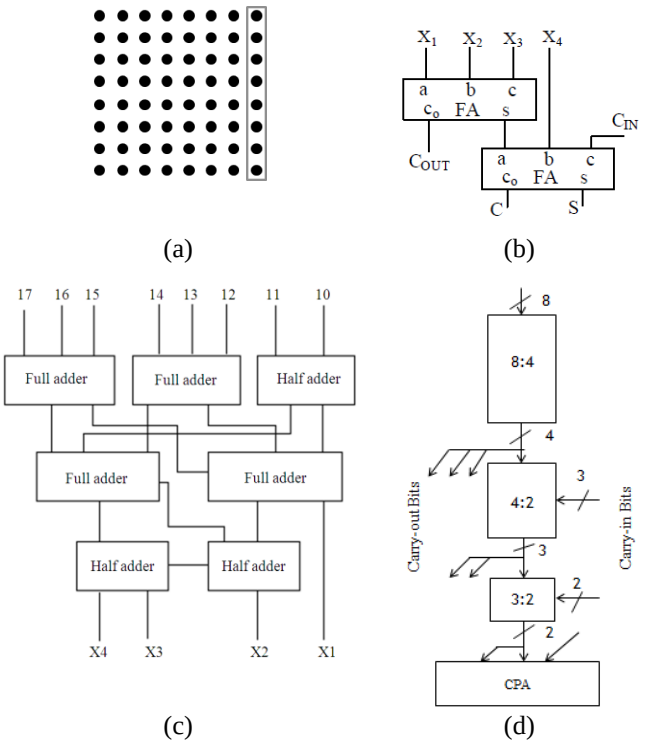

Fig.5(a). 8 × 8 macroblock bits, (b). 4:2 compressor, (c). 8:4 compressor [2] and (d). Sum module for 8 bits

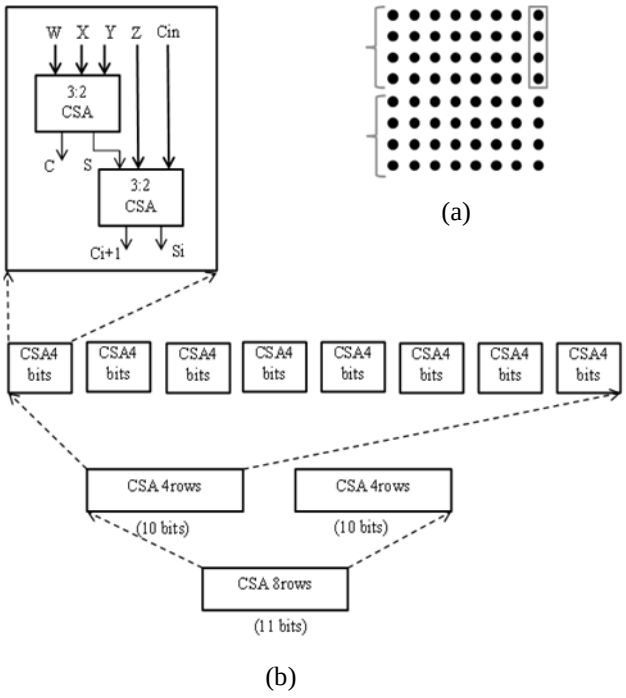

Fig.6.(a). Addition flow of 8 rows, (b). SAD calculation of one row using Carry Save Adder

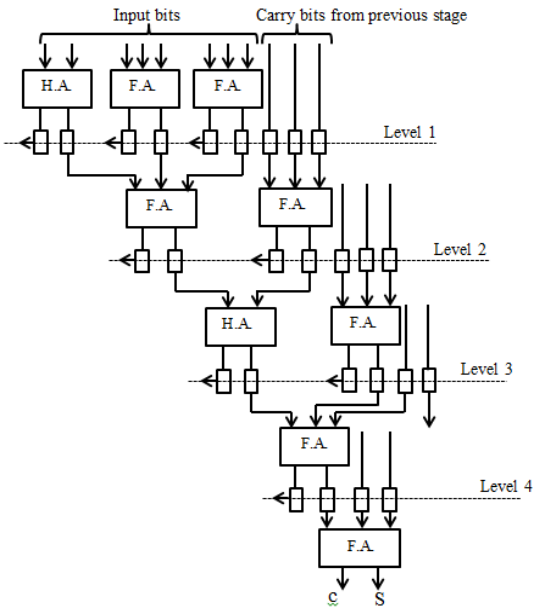

Fig.7. Addition of one column using PSTR [9]

2.2.4 Addition using Partial Summation Term Reduction (PSTR):

Instead of using various stages of compressors, if concept of partial summation term reduction (PSTR) [9] is used then addition can be speedy. In PSTR *N* number of layers are reduce to two layers where *N* numbers are operands to be added. This reduction is carried out using components like a full adder and a half adder. The two layers resulting thus are generally known as sum and carry. In final step there are various ways to add last two rows of sum and carry. In Fig.7 shows the flow of addition for 8 bits of one column, in different levels. At level 1 the addition of 8 bits of one column is shown and the sum bits are forwarded at the same column in next level and the carry bits are forwarded at the next column in the same level. This flow is repeated at different levels, until the level at which it reduces to the two rows. After reducing the eight rows in two rows, either the carry look ahead adder or ripple carry adder is used. This mechanism results in less number of adders compared to compressor based schemes.

3. SIMULATION AND RESULT FOR SAD PROCESSOR

Based on four addition schemes SAD processor is synthesis for Xilinx FPGA families Spartan3 and Virtex5. 'Foreman' frame is used for simulation in Xilinx 12.3i ISE and results are shown for 8 × 8 MB. The search range parameter p = 16, so the search window size is 40 × 40 in reference frame. SAD of 8 × 8 macroblock results are compared based on four different addition schemes using Virtex5 - xc5vlx50 and Spartan3 – XC3S400 FPGA in Table.1 and Table.2 respectively.

It can be observed that number of look-up-tables (LUTs) used are less than all other schemes in PSTR, and at same time post map static timing delay as well as post place and route static timing delay is also less. Table.3 indicates macro statistics for all four addition schemes. It is clearly observed that numbers of adders are less in PSTR based addition, that is the reason for having less LUTs in implementation. Table.3 shows, SAD of 8 × 8 macroblock based on PSTR addition scheme uses 29%, 23%, and 7% less number of adders compared to hierarchical, 8:4 compressor based, CSA based adders respectively. Both comparisons reveal that PSTR scheme outperforms in all aspects and can result in best addition scheme for SAD processor hence same is incorporated in proposed SAD processor architecture. This comparison is important due to usage of multiple processing elements in parallel. Marginal visible difference results in huge performance improvement when multiple processing elements are performed in parallel.

Table.1. Comparison of SAD of 8 × 8 macroblock using various addition schemes on Virtex5 FPGA

	Based on hierarchical adders	**Based on 8:4 compressor**	**Based on CSA**	**Based on PSTR adders**
Number of Slice Registers (28,800)	1033	1032	1032	1031
Number of Slice LUTs (28,800)	2064	1917	1804	1725
Synthesis Delay (ns)	2.154	2.154	2.154	2.154
Maximum Frequency (MHz)	464.296	464.296	464.296	464.296
Maximum output required time after clock (ns)	11.413	15.561	12.971	12.833
Post map Static Timing Delay (ns)	20.452	21.150	16.327	16.153
Post PAR Static Timing Delay (ns)	21.718	26.038	19.882	19.299
Total Power (mW)	528.28	530.21	529	529.97

Table.2. Comparison of SAD of 8 × 8 macroblock using various addition schemes on Spartan3 FPGA

	Based on hierarchical adders	**Based on 8:4 compressor**	**Based on CSA**	**Based on PSTR adders**
Number of Slices (3584)	1443	1528	1184	1126
Number of 4-input LUTs (7168)	2674	2518	2166	2105

Synthesis Delay (ns)	5.96	5.917	5.960	5.917
Maximum Frequency (MHz)	167.783	168.998	167.783	168.998
Maximum output required time after clock (ns)	31.280	40.049	32.201	15.770
Post map Static Timing Delay (ns)	21.864	21.150	21.132	19.352
Post PAR Static Timing Delay (ns)	34.254	40.802	34.789	19.352

The PSTR based 8 × 8 SAD implementation is now compared with existing implementations in Table.4. In [8] the computation of the SAD for 16 × 16 MB is implemented on a FPGA device. It uses less LUTs but consumes 26 more clock cycles compared to proposed implementation. The solution proposed in [12] uses highest LUTS and requires 29 cycles for a SAD computation at a frequency of 380 MHz. Rehman's implementation [11] exhibits lower utilization because of 4 × 4 macroblock size. Proposed SAD method outperforms from delay speed and area perspective for single cycle computation. Maximum frequency of proposed 8 × 8 SAD implementation is 464.3MHz on virtex5 xc5vlx50 FPGA.

Fast search BMA based on diamond or cross diamond shape uses nine candidate blocks hence SAD processor is configured for nine processing elements using PSTR based SAD of 8 × 8 macroblock and comparison of macro statistics is given in Table.6. Number of counter, registers, latches and comparator used with all four schemes are same but vast difference is in number of adders / subtracters. SAD processor based on PSTR uses less number of adders and that also of lower bits. Simulation results of SAD processor with 9 processing elements are shown in Fig.8. To organize nine candidate macroblocks from reference memory for diamond shape needs 34 clock cycles and as shown in Fig.8(a), sad_m0 to sad_m8 indicate SAD value of all eight rows of nine candidate MBs available in next clock cycle. Same time comparator provides result for best match among candidate MBs as indicated in Fig.8(b).

Table.3. Resource utilization for SAD of 8 × 8 macroblock

Macro Statistics	**Based on hierarchical adders**	**Based on 8:4 compressor**	**Based on CSA**	**Based on PSTR adders**
Adders/ Subtracters	191	145	175	135
14-bit adder	7	7	15	7
11-bit adder	8	--	--	--
10-bit adder	16	10	32	--
9-bit adder	32	--	--	--
8-bit adder	64	64	64	64
8-bit adder carry in	64	64	64	64
Registers	1024	1024	1024	1024
4x1Mux	--	600	256	--
1-bit xor2	--	381	--	224
1-bit xor3	--	195	256	408

Table.4. Comparison of SAD implementation for fix size macroblocks

Architectures	**Macroblock Size**	**FPGA**	**LUTs**	**Synthesis Delay (ns)**	**Maximum Frequency MHz)**	**Clock cycles**
carry–save adders	16 × 16	FLEX20KE Altera	1699	5.076	197	27
carry generator method [12]	16 × 16	STRATIX EP1S80	7765	2.632	380.7	29
Kincses's architecture [11]	8 × 8	Spartan 3 (3s400-5pq208)	3435	7.692	130	1
Dadda tree scheme [11]	4 × 4	Virtex 2 (2v1000bg575-4)	657	7.505	133.245	3
Proposed SAD implementation	8 × 8	Spartan 3 (xc3s400-5pq208)	2105	5.960	167.78	1
		Virtex 5 (xc5vlx50-3ff676)	1725	2.154	464.296	1

Table.5. Performance comparison for calculating motion vector (MV) using diamond search BMA

Architecture Used	**To fill All candidate macro blocks**	**Latency of PU**	**To calculate SAD**	**Comparator**	**Large DS Pattern LDSP**	**Small DS Pattern SDSP**	**Total cycle to generate MV**	**Frequency**	**FPS HDTV 1080p (worst case)**	**FPS HDTV 1080p (average case)**
PU [14]	26	4	7	5	42	20	62	272.6	58	240
PU-42 [14]	26	3	7	5	41	19	60	287.3	63	263

PU-82 [14]	26	2	7	5	40	18	58	283.0	65	269
Proposed	31	3	1		35	3	38	451.101	84	325

Table.6. Comparison of SAD processor for fast search BMA using nine SAD processing elements

Macro Statistics	**Based on hierarchical adders**	**Based on 8:4 compressor**	**Based on CSA**	**Based on PSTR**
Adders/ Subtractors	1719	1215	1575	1152
8-bit adder	--	576	576	576
8-bit adder carry-in	--	576	576	576
9-bit subtractor	576	--	--	--
10-bit adder	--	--	288	--
11-bit adder	1080	--	72	--
14-bit adder	63	63	63	--
Counters	2	2	2	2
Registers	4642	4642	4642	4642
Latches	133	133	133	133
Comparators	8	8	8	8
1-bit xor2	--	4104	--	2160
1-bit xor3	--	3816	2304	4428

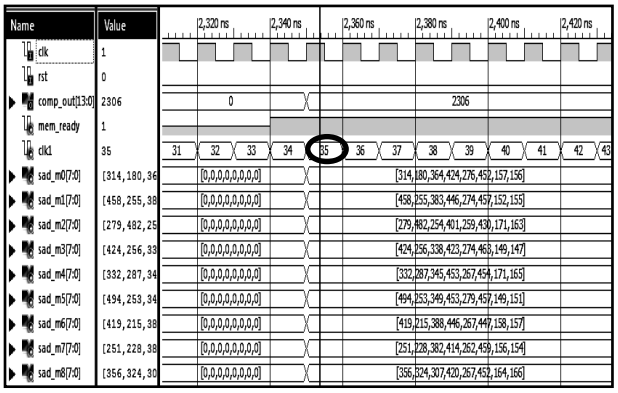

Fig.8(a). SAD values of eight rows of nine candidate MB

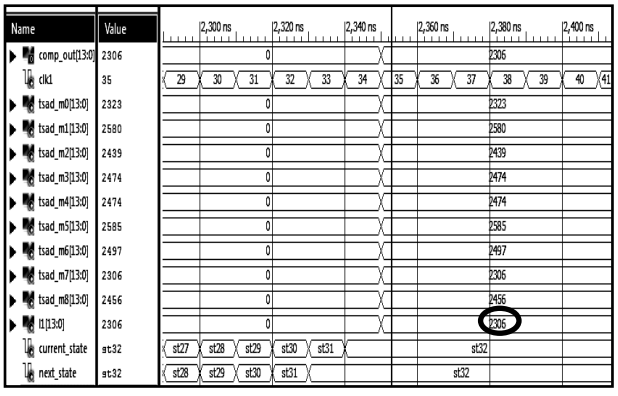

Fig.8(b). Comparator value

Fig.8. Results of SAD processor with 9 processing elements

Proposed architecture is compared with existing implementation of diamond search BMA [14] and results are indicated in Table.5. For large diamond search pattern (LDSP) proposed architecture requires only 35 clock cycles, which is less compared to Porto's architecture. For calculating small diamond search pattern (SDSP) proposed architecture requires only 3 clock cycles, so for generating motion vector (MV) for the best case 38 clock cycles are needed. The best case means best match is found in first iteration only and MV is on center of search window. Table.5 also indicates frequency of operation and number of frames processed per second of HD 1080p resolution in worse case and average case. The average case and worst case are obtained through the software implementation of the algorithm and it has been observed that for most of sequences use 3 iterations on an average. Worse case is considered as 20 iterations. Proposed architecture can process 84 frames per second in worst case and 325 frames per second in average case for HDTV 1080p resolution.

4. CONCLUSION

Motion estimation is used to minimize the temporal redundancies in a video sequence for the video compression. For estimating motion BMA is used, which uses SAD as cost function in most hardware implementations. SAD computation is challenging task in implementation. In this paper, four methods of multi operand addition based on carry save addition, 8 to 4 compressor, hierarchical addition and addition using partial summation term reduction (PSTR) are implemented and compared. SAD for 8 × 8 MB is computed using all four methods in 1 clock cycle. Among all these methods, SAD processor based on PSTR scheme outperforms in terms of delay and resource utilization which results in 84 and 325 frames per second in worse case and average case respectively for HDTV 1080p resolution. Due to such figure architecture is well suited for real time implementation. Compared to existing implementation of diamond search BMA proposed implementation save 34.5% clock cycles in best case implementation which is the case for approximately 75% MBs in real time video on an average.

REFERENCES

[1] Iain E. Richardson, "*H.264 and MPEG-4 Video Compression*", John Wiley & Sons, Inc., 2003.

[2] R. Marimuthu, D. Bansal, S. Balamurugan and P. S. Mallick, "Design of 8-4 And 9-4 Compressors for High Speed Multiplication", *American Journal of Applied Sciences*, Vol. 10, No. 8, pp. 893-900, 2013.

[3] J. Hormigo, J. Villalba and E. L. Zapata, "Multi-operand Redundant Adders on FPGAs", *IEEE Transactions on Computers*, Vol. 62, No. 10, pp. 2013-2025, 2013.

[4] S. R. Chowdhury, A. Banerjee, A. Roy and H. Hiranmay Saha, "Design, Simulation and Testing of a High Speed Low Power 15-4 Compressor for High Speed Multiplication Applications", *First IEEE International Conference on Emerging Trends in Engineering and Technology*, pp. 434-438, 2008.

[5] M. Ortiz, F. Quiles, J. Hormigo, F. J. Jaime, J. Villalba and E. L. Zapata, "Efficient implementation of carry-save adders in FPGAs", *20th IEEE International Conference on Application-specific Systems, Architectures and Processors Efficient*, pp. 207-210, 2009.

[6] S. Ravi Chandra Kishore and K. V. Ramana Rao, "Implementation of carry-save adders in FPGA", *International Journal of Engineering and Advanced Technology*, Vol. 1, No. 6, pp. 27-29, 2012.

[7] F. De Dinechin, H. D. Nguyen and B. Pasca, "Pipelined FPGA Adders", *International Conference on Field Programmable Logic and Applications*, pp. 422-427, 2010.

[8] Stephan Wong, Stamatis Vassiliadis and Sorin Cotofana, "A Sum of Absolute Differences Implementation in FPGA Hardware", *Proceedings of 28th Euromicro Conference*, pp. 1-5. 2002.

[9] N. N. Shah, K. R. Agarwal and H. M. Singapuri, "Implementation of sum of absolute difference using optimized partial summation term reduction", *International Conference on Advanced Electronic Systems*, pp. 192-196, 2013.

[10] Z. Kincses, Z. Nagy, L. Orzó, P. Szolgay and G. Mező, "Implementation of a parallel SAD based wavefront sensor architecture on FPGA", *European Conference on Circuit Theory and Design*, pp. 823-826, 2009.

[11] 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.

[12] S. Wong, B. Stougie and S. Cotofana, "Alternatives in FPGA-based SAD Implementations", *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 449-452, 2002.

[13] A. Ben Atitallah, P. Kadionik, N. Masmoudi and H. Levi, "HW / SW FPGA Architecture for a Flexible Motion Estimation", *14th IEEE International Conference on Electronics, Circuits and Systems*, pp. 30-33, 2007.

[14] M. Porto, A. Silva, S. Almeida, E. Costa, and S. Bampi, "Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding", *Journal Integrated Circuits Systems*, Vol. 5, No. 1, pp. 78-88, 2010.