VLSI Architecture Design of Motion Estimation Block with Hexagon-Diamond Search Pattern for Real-Time Video Processing

Atin Mukherjee
Department of Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela, India
email: mukherjeea@nitrkl.ac.in

2021 IEEE 18th India Council International Conference (INDICON) | 978-1-6654-4175-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/INDICON52576.2021.9691531

***Abstract*— In recent times, design of efficient video encoders is very important because of their ubiquitous use in mobile and small hand-held battery operated portable devices. Block based motion estimation algorithm is known to be the best and the simplest among various motion estimation techniques, whereas hexagon-diamond search (HEXDS) block matching algorithm is the most efficient block matching algorithms, due to its fewer number of computations and higher speed of searching. This method is equally important for the recently prevailing High Efficiency Video Coding (HEVC) standard for video compression along with the existing Advanced Video Coding (AVC) standard. In this paper, an architecture for the motion estimation block has been proposed along with the address generation for HEXDS algorithm. The architecture achieves maximum frequency of 200 MHz and have a gate count of 12.6 k, while implemented in Verilog HDL and mapped to Virtex-4 FPGA.**

Keywords— hexagon-diamond search, VLSI architecture, motion vector, motion estimation

I. INTRODUCTION

Block based motion estimation algorithms are the most popular ones in applications of different motion-compensated video-coding standards including mostly used ITU-T H.264 and the recent prevailing standard H.265. In such methods, current frame is partitioned into macroblocks, which are non-overlapping rectangular blocks of equal size. Then the best matched position in a reference frame (past or future frame) for each macroblock is found out by a block-matching method. In most of the cases, the mean of absolute difference between the pair of blocks is considered for matching. The block in the reference frame is moved to find out the best-matched block producing the minimum distortion. The motion vector (MV) is calculated by computing the change in position of the current frame with its best-matched frame position among all reference blocks.

Among different block matching algorithms (BMA), the full search (FS) algorithm is the simplest one in terms of hardware implementation. It searches exhaustively for the best MV within the specified search range, but is most compute-intensive making real-time implementation quite challenging [1]. For the quick calculation of motion vectors, many fast block matching motion estimation algorithms like three-step search (TSS) [1], four-step search (4SS) [2], diamond search (DS) [3], hexagon-based search (HEXBS) [4], test zone search (TZS) [5] etc. have been proposed over years.

Among various search algorithms, the DS algorithm considers a search pattern having shape of a diamond and needs less number of iterations to find out the MV in comparison with TSS and 4SS [3] with similar distortion. HEXBS algorithm is a modified version of the DS algorithm, but offers faster solution to its predecessor with similar distortion performance [4]. In [6], Ranjit et al. merged the advantages of HEXBS and DS and proposed a new algorithm called hexagon-diamond search (HEXDS) algorithm that uses repetitive HEXBS is used and finally it switches to small diamond search pattern (SDSP). HEXDS has fewer search points and hence a lower computational complexity compared to HEXBS. In [6], performance of HEXDS is compared with other search techniques and proved to be superior in terms of PSNR, speed improvement, required processing time and total number of search points than those of FS, TSS, original HEXBS and DS. Modified versions of HEXDS have been proposed in [7-8]. Significant amount of data compression is required for a wide range of modern video applications including online video streaming, digital TV/HDTV broadcasting, video conferencing, video database services, online video storage etc. In such cases, HEXDS algorithm is still used as the primary block matching motion estimation [9]. More recently hexagon based search algorithm has also been proved to be efficient for the High Efficiency Video Coding (HEVC) implementations [10] for 4K or 8K video resolutions. In [11], a new quadrant-based search algorithm based on HEXBS suitable for HEVC has been proposed. A digital video stabilization system using hexagonal search algorithm has been implemented in Xilinx Zynq XC7030 [12].

In this paper, VLSI architecture for block-based motion estimation incorporating HEXDS algorithm has been proposed. Area efficient architectures for processing element and motion vector generation unit have been shown in the next section. In Section III, architectural implementation of HEXDS algorithm has been discussed. Section IV deals with simulation results and comparisons with other existing implementations. The paper is concluded in Section V.

II. ARCHITECTURE OF MOTION ESTIMATION BLOCK

Fig. 1 shows our proposed architecture for the motion estimation (ME) block. In this architecture, the current frame is partitioned in 16×16 non-overlapping blocks. Base address of reference data generation unit depends on the search algorithm selected. We have considered the sum of absolute difference (SAD) as our matching criteria. SAD for each candidate location (u, v) is calculated as:

$$\mathrm{SAD}(u,v)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|f_t(x+i,y+j)-f_{t-1}(x+i+u,y+j+v)\right| \quad (1)$$

, where $-W \le (u, v) \le W$ (W is the maximum search range) and $f_t(x, y)$ is the pixel intensity at location (x, y) in the t-th frame. The motion vector is determined as: [$d1$, $d2$] = arg. min. [SAD (u, v)].

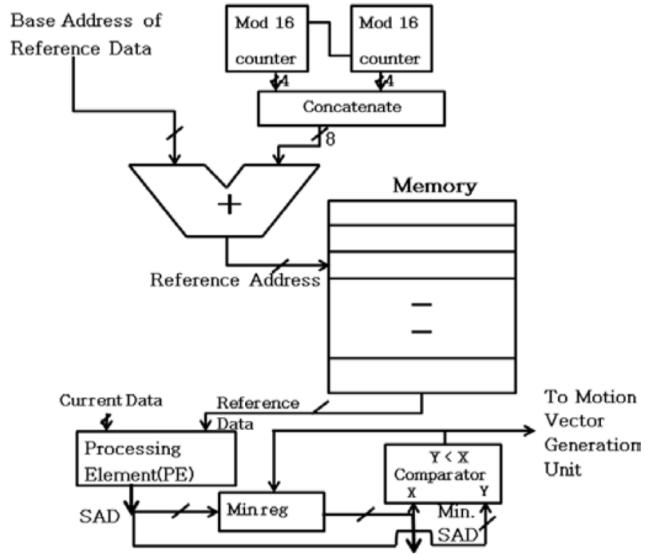

Fig. 1. Proposed architecture for the motion estimation block

We stored the reference data in RAM and retrieved the data as required to calculate the SAD values with data from the current image frame. The processing element (PE) calculates the SAD values corresponding to current data and each possible reference data depending on the search algorithm. Two's complement addressing has been used to reduce the hardware cost. The comparator compares different SAD values to find out the minimum one and also indicates the address for which the SAD value is minimum. The PE block consists of an accumulator and an absolute difference unit, which finds the absolute difference values between the current and reference macroblock pixel values and the adder to sum up all the absolute difference values calculated.

In the absolute difference calculator, we subtract the pixel values directly using a subtractor as shown in Fig. 2. If current pixel value is greater than the reference pixel value, no borrow is generated and the difference value is directly taken as the absolute difference value. But if the current pixel value is less than the reference pixel value, then B_{out} = 1, and 2's complement of the difference value is the desired absolute difference value.

Now the 2's complement block and the MUX can be combined into one unit which will give 2's complement of the difference value D if borrow B is generated by the subtractor or will pass the difference value directly, otherwise. The controlled 2's complement correction hardware with input Di's and Bout and output Yi's can be designed using the following logical expressions:

$$I_m = (D_m + I_{m-1})B_{out}$$
$$Y_m = D_m \oplus I_{m-1} \quad (2)$$

, where $I_0 = \overline{B_{out}}$.

For fast operation, the subtractor used in our design is a conditional difference subtractor (CDS). It has construction similar to that of conditional sum adder with modified conditional cell (CC) to perform the subtraction operation. The 256 absolute difference values generated are accumulated using an adder to get the required SAD value in the PE. The complete architecture for the PE block is shown in Fig. 3. We use a carry look ahead adder for fast operation. The SAD value computed for each block is compared with the previous minimum SAD value stored and replace the previous one if found smaller.

In Fig. 1, one comparator is used to compare the stored minimum SAD value with the current SAD value to find out the minimum one between the two and store the result in the minimum register. But we need only $Y<X$ output of the comparator and the comparator can be simply replaced by an adder consisting of the carry propagation part only. Fig. 4 shows the architecture for comparison of the two SAD values. $Cout = 0$ denotes $Y<X$, i.e., stored SAD value is larger than the current SAD value and the register is activated to store the new SAD value. The same EN signal is used to find out the MV at the minimum SAD.

Fig. 5 shows the architecture for motion vector generation unit, which calculates difference between best match position of the candidate block in the reference frame corresponding to the minimum SAD location and the current position of the candidate block. The enable signal here is the same as the EN signal that of Fig. 4, which indicates the minimum SAD location.

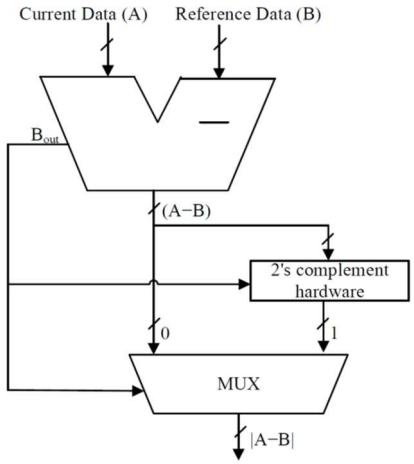

Fig. 2. Absolute difference calculation unit

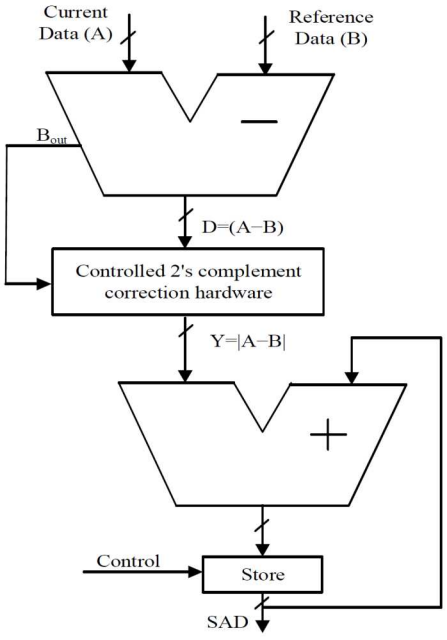

Fig. 3. The processing element (PE)

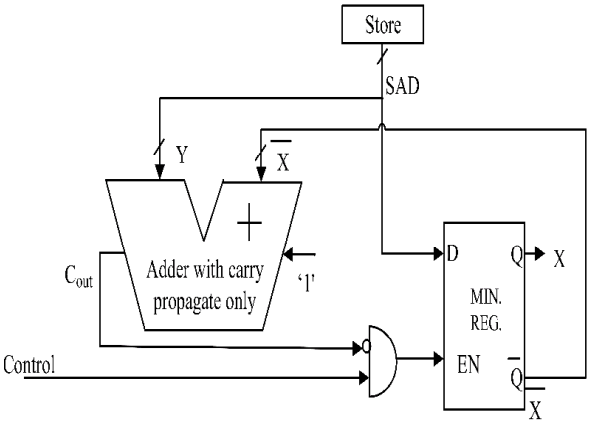

Fig. 4. Minimum SAD calculation block

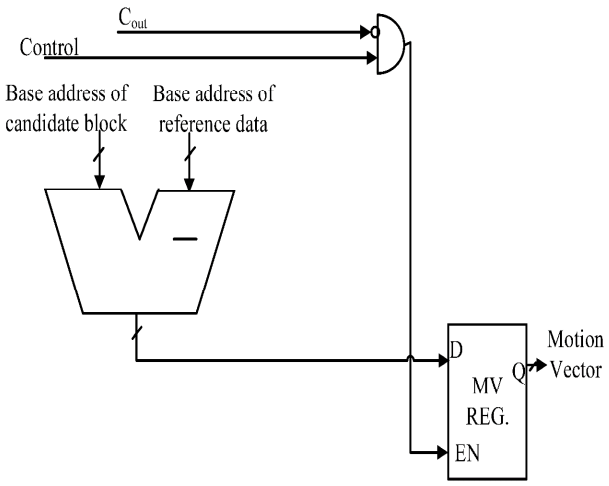

Fig. 5. The motion vector (MV) generation unit

III. HEXDS Architecture Implementation

Ideally, the fastest search speed with equal distribution of minimum number of search points is achieved through circle-shaped search pattern [4] and each search point can equally be utilized. The diamond search (DS) algorithm has proved to be useful because of its fewer search points to NTSS and 4SS having diamond-shaped circle-like approximated pattern. Hexagonal-based search (HEXBS) pattern is a modified version of DS and is also a circle approximated search pattern, where the search points are equally distributed. It has seven checking points which utilizes a centre biased search pattern with six points surround the central one composing a hexagon. There are six endpoints in the hexagon, among which four points have a distance of √5 is surrounding the centre point (refer to Fig. 6) and two horizontal points away from the centre to its both sides with distance 2. Moreover, each neighbouring pair of endpoints among the six endpoints have distance of either 2 or √5 between them. All the six endpoints of a hexagon are equally distributed around the centre minimizing the total number of search points. The hexagonal search pattern is simpler to 9-point DS search because of its two fewer checking points. At final step, when MBD is found to be same as previous step, smaller hexagon with four checking points (each having distance 1 from the centre) is used for finer inner search [4].

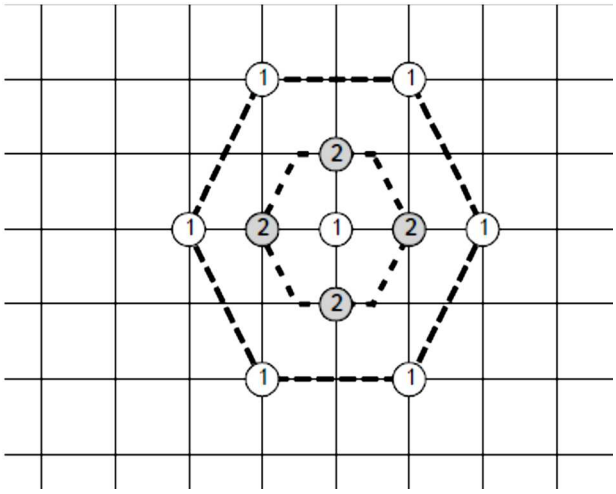

Fig. 6. Hexagon-diamond search: ① Large hexagon (HEXBS) and ② Small hexagon (DS) patterns

In case of HEXDS algorithm, at each step the point with minimum block distortion (MBD), i.e., point having the best motion vector, is considered as the new centre of the hexagonal search window. It is calculated by checking all seven points of the hexagon including the centre point. If the centre point of the hexagon is found to be the MBD point, we shift to the smaller search patter of DS to refine it more. Otherwise, the hexagonal search including the seven points is applied repeatedly until the MBD point becomes the same as that of the previous search. Fig. 6 shows the large and small hexagon patterns in HEXDS algorithm, where in the figure, ① represents the search points for large hexagon (HEXBS) and ② represents the search points for small hexagon (DS) patterns. With the new search point as the centre, the pattern has to calculate for three new search points with other three points overlapping with previous search points.

The MBD point is calculated at each step with the search point having least SAD value among all the points in the hexagon considered for the HEXDS pattern. At any step, if the MBD is found to be the same as that in the previous step, the algorithm automatically switches to smaller pattern to DS for finer search. The algorithm is flexible as it has adaptive number of search points inside the search window. n being the number of times the large hexagonal pattern has been searched, the total number of search points per block for the HEXDS algorithm is at most $N=3n+8$.

So, in the first step of HEXDS, we have to calculate a total of seven search points with centre as the base address of the current block and after that, in each step we have to search only three new points until we find the middle point as the minimum SAD point itself. If the minimum SAD point is the centre point of the current hexagon, then in the next step we should search the four points of the smaller hexagon corresponding to DS with the same centre point. Seven points of a hexagon are marked as shown in Fig. 7 for further references. In this figure, (x, y) displays the direction of movements with respect to the centre point of the hexagon. Next, we need to calculate the amount of displacement required to generate the six corner point addresses from the given centre point, which is same as that of the base address of the current block.

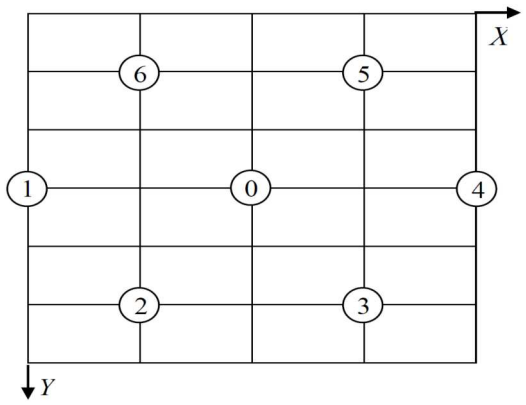

Fig. 7. Seven points of a Hexagon

Table I provides the displacement required in position to reach the point of the first column from the base address (marked as 0) and the next search locations if the minimum SAD point is the designated point in the first column itself. It is apparent that from second step only three new search points will be added to the system to calculate and find out the minimum SAD position. A mod-6 up counter has been used to generate required numbers shown as movements in Table I to add with the base address to generate the next six points sequentially as shown in the Table.

We can use only two adders with specified input lines as in Table II to generate the first six addresses of the corner points of the initial hexagon from the base address (Fig. 8). The outputs (*X, Y*) will generate the addresses of the six corner points of initial hexagon from base address (*x, y*) with each count value at every clock pulse. The values to be added to the base address are derived from a binary counter output ai's as shown in Table II.

SAD values are calculated for all seven points of the initial hexagon. When the centre of the hexagon does not correspond to the minimum SAD point, we have to generate the next three addresses for searching as shown in the third column of Table I. After calculating and comparing the new SAD values with the previous minimum SAD value, the minimum SAD position among these four search points is sorted out and the procedure repeats till the minimum SAD point is the same as the previous one. For every stage, we have to store current address as well as the count value for which SAD is minimum. The stored count value is decremented and incremented by one using a decrementer and an incrementer. The same address generation unit as of Fig. 8 is used to generate the three new addresses. The new control values will be count value stored corresponding to minimum SAD and just its previous and next count values. We proceed in the same manner, until the minimum SAD address corresponds to the same as that of the previous stage and then switch to DS immediately. The SAD values for the new four search points are calculated and compared to the previous stage minimum SAD value to find out the best matched position in the reference frame. To calculate the motion vector, the minimum SAD address is subtracted from the original base address.

A, *B*, *C* (in Table II and Fig. 8) are defined as:

$$A = a2 + (a1 \oplus a0)$$

$$B = a0 + \overline{a2} \cdot \overline{a1} \qquad (3)$$

$$\text{and} \quad C = \overline{a2} \cdot \overline{a1} + \overline{a1} \cdot a0 = \overline{a1} \cdot B$$

Fig. 9 shows the complete block diagram of the proposed architecture for block-based motion estimation using the HEXDS algorithm to generate the base address for the search pattern. This architecture iterates between the first two steps of the HEXBS algorithm and if at any step the minimum SAD point does not change, the search pattern switches to the one of the DS algorithm. The two mod-16 counters are used for address generation; one for column and one for row address generation, which are concatenated to generate an 8-bit address and then added with the base address to generate the reference address. When these mod-16 counters complete counting for one full count sequence, the binary counter in the base address generation unit increments to the next count value.

Now to generate the address required for the diamond search process, we have followed an approach similar to the one responsible for the generation of the hexagon edges during the first algorithm stage. We need a two bit counter to generate the addresses for DS as shown in Table III. The structural block accountable for the generation of the addresses in the DS pattern is shown in Fig. 10, which is identical to the architecture that shown in Fig. 8 except for a minor modification in some of the input lines.

D and *E* (in Table III and Fig. 10) are defined as

$$D = a_1 \cdot \overline{a}_0$$

$$\text{and} \quad E = \overline{a}_1 \cdot a_0 \qquad (4)$$

TABLE I. MOVEMENTS AND NEXT SEARCH POINTS IN HEXBS

Minimum SAD Point	Movements (*x, y*)	Next Search Points
0	(0, 0)	Smaller Hexagon
1	(–2, 0)	6, 1, 2
2	(–1, +2)	1, 2, 3
3	(+1, +2)	2, 3, 4
4	(+2, 0)	3, 4, 5
5	(+1, –2)	4, 5, 6
6	(–1, –2)	5, 6, 1

TABLE II. MOVEMENTS RELATED TO A BINARY COUNTER OUTPUTS

Counter Outputs			Movements (*x, y*)	Movement (in 2's complement)	
a_2	a_1	a_0		*x*	*y*
0	0	0	(–2, 0)	1 1 0	0 0 0
0	0	1	(–1, +2)	1 1 1	0 1 0
0	1	0	(+1, +2)	0 0 1	0 1 0
0	1	1	(+2, 0)	0 1 0	0 0 0
1	0	0	(+1, –2)	0 0 1	1 1 0
1	0	1	(–1, –2)	1 1 1	1 1 0
So to generate required addresses from base address we have to add →				*C B A*	a_2 *A* 0

TABLE III. MOVEMENTS FOR SMALLER HEXAGON

Binary Counter		Movement (*x, y*)	Movement (in 2's complement)	
a_1	a_0		*x*	*y*
0	0	(+1, 0)	0 0 1	0 0 0
0	1	(0, –1)	0 0 0	1 1 1
1	0	(–1, 0)	1 1 1	0 0 0
1	1	(0, +1)	0 0 0	0 0 1
So to generate required addresses from base address we have to add →			*D D* $\overline{a_0}$	*E E* a_0

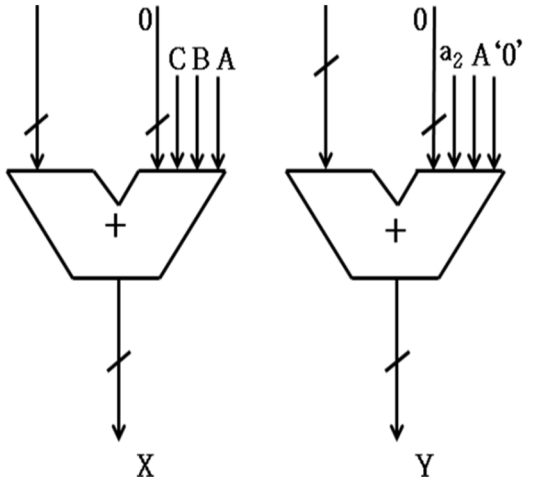

Fig. 8. Address generation unit for the initial hexagon

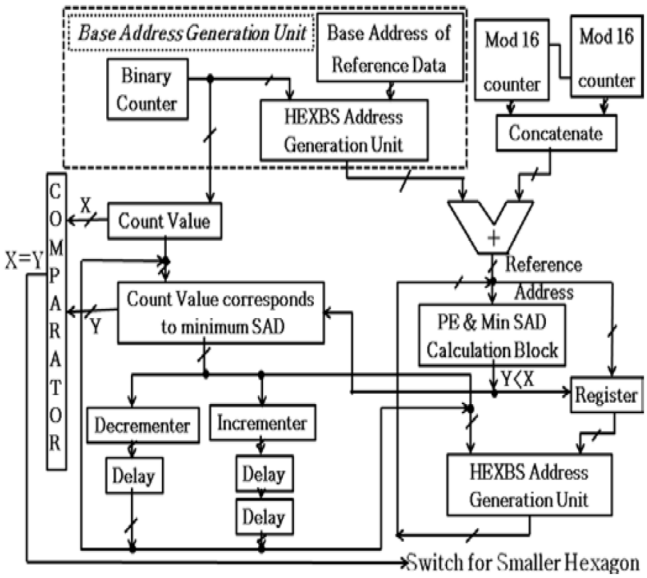

Fig. 9. Architecture of motion estimation block incorporating HEXDS

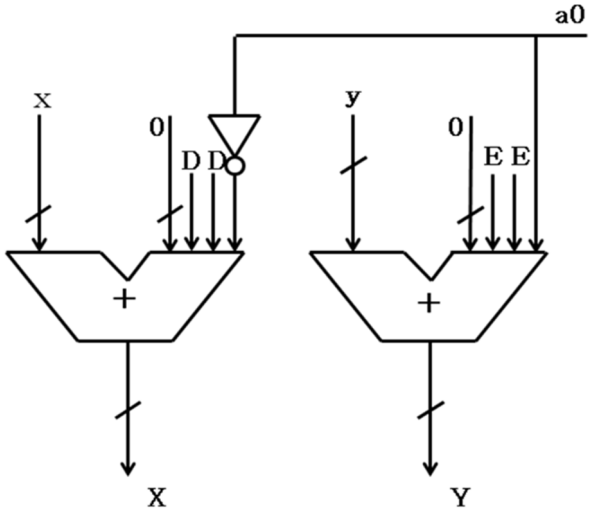

Fig. 10. Address generation unit for smaller hexagon (diamond search)

IV. Experimental Results and Discussions

The HEXDS architecture is implemented in MATLAB and performance analysis is done to compare with most popular and widely used fast ME algorithms like TSS, NTSS, 4SS, DS and HEXBS in terms of the PSNR values, average number of search points and average SAD values. For simulation and comparison purpose, we have chosen five standard video sequences, viz., "Football" (720 × 480, 59 frames), "Claire" (360 × 288, 165 frames), "Salesman" (352 × 288, 300 frames), "Coastguard" (352 × 240, 300 frames) and "Garden" (720 × 480, 98 frames). These video sequences vary in their motion contents as well as in frame sizes. All the simulations for the coded algorithms are consistently performed with ten reference frames with block size of 16 × 16 pixels and search window size of 7 × 7. The simulation results are shown in Table IV. The PSNR values are calculated taking the FS algorithm as the reference and it gives a measure of the search quality for different search algorithms. The table shows that the HEXDS has the best average peak signal-to-noise ratio and hence better improvement compared with all the other algorithms. The number of average search points per frame depicts the search speed for different algorithms. The table also shows that HEXDS has smallest number of average checking points per frame compared to the other techniques with just a marginal increase in average SAD values and hence proved to be most efficient in terms of search quality and speed. In many cases the performances of HEXBS and HEXDS are comparable and our proposed architecture can withstand both the techniques with minimal change in actual hardware.

We have also implemented our proposed architecture using Verilog HDL and finally mapped to Virtex-4 FPGA board. The evaluation shows the critical delay for the proposed architecture as 5 ns. Table V tabulates the cost comparison in terms of speed and hardware with some previous architectures [13-15] for HEXBS algorithms. No direct implementation of VLSI architecture for HEXDS algorithm has been reported in the literature as per our best knowledge. The area values in the table are represented in terms of gate counts which is calculated as number of equivalent NAND2 gates required. The table shows that the proposed architecture for implementation of HEXDS algorithm requires comparatively lesser gate count and is also superior to the other existing architectural structures of HEXBS. Moreover from Table IV, it is also clear that HEXDS outperforms its predecessor HEXBS in many attributes.

V. Conclusion

In this paper, we have proposed a VLSI architecture for motion estimation using HEXDS algorithm. The architecture can be made reconfigurable by modifying the architecture for address generation unit. The elementary motion estimation unit based on simple processing elements and motion vector calculation are common for all the algorithms. Among several BMAs, HEXDS is very efficient and speedy one for real time video processing as it needs fewer search points to generate the motion vector. It has been seen that the number of search points saved by the HEXDS algorithm increases with the size of the motion vector. The hardware implementation and simulation results also show that HEXDS is better in terms of search quality, and hardware cost than any other existing structure. The proposed architecture can also be reconfigured to implement other fast algorithms like DS and HEXBS or its variations without much scarification in the performance.

TABLE IV. COMPARISON OF PERFORMANCE FOR VARIOUS BMAS

Sequence	BMA	PSNR	Avg. no. of search points per frame	Average SAD values
Football	TSS	74.33	22	2135
	NTSS	74.43	22.7	2138
	4SS	73.40	21	2175
	DS	76.89	15.1	2170
	HEXBS	77.45	13.1	2199
	HEXDS	78.03	12.9	2206
Claire	TSS	83.14	22	445
	NTSS	83.67	22.2	455
	4SS	82.91	19.2	462
	DS	85.84	14.1	460
	HEXBS	86.27	13	466
	HEXDS	86.95	11.8	470
Salesman	TSS	80.48	22	690
	NTSS	80.67	22.8	706
	4SS	80.10	18	720
	DS	83.24	13.6	716
	HEXBS	85.87	12.3	725
	HEXDS	86.05	10.9	735
Coastguard	TSS	72.55	22	1290
	NTSS	72.82	22.5	1300
	4SS	71.93	18	1315
	DS	74.12	13.2	1305
	HEXBS	75.87	10.5	1325
	HEXDS	75.44	10.8	1320
Garden	TSS	76.23	22	1661
	NTSS	76.55	23	1666
	4SS	76.01	24.2	1716
	DS	78.98	17	1723
	HEXBS	79.07	15.7	1752
	HEXDS	79.89	14.2	1790

TABLE V. AREA AND SPEED OF THE PROPOSED HEXDS ARCHITECTURE COMPARED WITH THE EXISTING ONES

Architectures	Supported BMA	Frequency (in *MHz*)	Area (in *kgates*)
Tasdizen [13]	HEXBS	144	13.3
Vanne [14]	HEXBS	200	14.2
Muzammil [15]	HEXBS	127.27	12.5
Proposed	HEXDS	200	12.6

REFERENCES

[1] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," Proc. National Telecommunications Conference, pp. C9.6.1–9.6.5, New Orleans, La, USA, November 1981.

[2] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 313–317, 1996.

[3] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," IEEE Transactions on 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 Transactions on Circuits and Systems for Video Technology, vol. 12, no. 5, pp. 349-355, 2002.

[5] P. Goncalves, M. Porto, B. Zatt, L. Agostini, and G. Correa, "Octagonal-Axis Raster Pattern for Improved Test Zone Search Motion Estimation," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, pp. 1763-1767, April 2018.

[6] S. S. S. Ranjit, K. S. Sim, R. Besar, S. I. Md Salim, and S. K. Subramaniam, "Estimation of motion vector parameter using hexagon-diamond search algorithm," Journal of Real-Time Image Processing 6, no. 4, pp. 225-234, 2011.

[7] R. Priyadarshi and V. Nath, "A novel diamond–hexagon search algorithm for motion estimation," Microsystem Technologies, vol. 25, no.12, pp. 4587-4591, 2019.

[8] R. Mukherjee, I. G. Vinod, I. Chakrabarti, P. K. Dutta, and A. K. Ray, "Hexagon Based Compressed Diamond Algorithm for motion estimation and its dedicated VLSI system for HD videos," Expert Systems with Applications, vol. 141, no. 112919, 2020.

[9] N. Raj and S. Shabeer, "Hexagonal Search Based Compression Noise Estimation and Reduction in Video," IEEE 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, pp. 1061-1064, April 2018.

[10] S. Gogoi and R. Peesapati, "A Hybrid Motion Estimation Search Algorithm for HEVC/H. 265," IEEE International Symposium on Smart Electronic Systems (iSES), Rourkela, December 2019.

[11] S. H. Francis, P. Rajesh, and M. R. Raja, "An Efficient VLSI Architecture for Fast Motion Estimation Exploiting Zero Motion Prejudgment Technique and a New Quadrant-Based Search Algorithm in HEVC," Circuits, Systems, and Signal Processing, September 2021 (DOI: 10.1007/s00034-021-01850-2).

[12] M. Ahmed and L. Singh, "Verilog based Implementation of Real Time Digital Video Stabilization," 10th IEEE International Conference on Communication Systems and Network Technologies, Bhopal, June 2021.

[13] O. Tasdizen, A. Akin, H. Kukner, I. Hamzaoglu, and H. F. Ugurdag, "High Performance Hardware Architectures for a Hexagon-Based Motion Estimation Algorithm," IEEE / IFIP International Conference on VLSI - SoC, Rhodes, Greece, October 2008.

[14] J. Vanne, E. Aho, K. Kuusilinna, and T. D. Hamalainen, "A reconfigurable motion estimation architecture for block-matching algorithms," IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 4, pp. 466-477, 2009.

[15] M. Muzammil, I. Ali, M. Sharif, and K. A. Khalil, "An efficient FPGA architecture for hardware realization of hexagonal based motion estimation algorithm," IEEE International Conference on Consumer Electronics, Taiwan, pp. 422-423, June 2015.