Hardware Design for Motion Estimation

Nutan Satya Sai Raj Kondempudi
School of Electronics Engineering
Vellore Institute of Technology,
Vellore– 632014, Tamil Nadu, India
nutan.satyasairaj2022@vitstudent.ac.in

Akshay Vijay Dandekar
School of Electronics Engineering
Vellore Institute of Technology,
Vellore– 632014, Tamil Nadu, India
dandekar.akshay2022@vitstudent.ac.in

Abhinav Sai Gude
School of Electronics Engineering
Vellore Institute of Technology,
Vellore– 632014, Tamil Nadu, India
gudeabhinav.sai2022@vitsudent.ac.in

Prayline Rajabai C
School of Electronics Engineering
Vellore Institute of Technology,
Vellore– 632014, Tamil Nadu, India
prayline.c@vit.ac.in

***Abstract*— A hardware implementation of the motion estimation architecture for real-time video processing systems using the Sum of Absolute Differences (SAD) module is designed. The design also includes a pre-processing unit and a motion vector estimation unit that use a block-matching algorithm. The use of on-chip memory units in the architecture reduces data transfer overhead. Two block-search algorithms, namely the full search and the hexagonal search algorithm with adaptive search pattern, are realised. As the full search algorithm is exhaustive, a parallel processing architecture is implemented to improve its throughput and performance. The hardware design is implemented in Verilog HDL using ModelSim and synthesised on a Vertex-7 FPGA using Xilinx ISE 14.7. From the obtained results, hexagonal search gave good performance in comparison with full search.**

***Keywords*— *Motion estimation, Parallel Processing architecture, Block-matching algorithm,* adaptive search pattern.**

I. INTRODUCTION

Motion estimation is a crucial task in many video-processing applications, including video compression along with object tracing. Video compression aims to effectively decrease the duplication contained in the temporal domain of video data, which consists of a sequence of image frames. A key component of video encoding systems is Motion Estimation (ME) serves an essential role in reducing temporal redundancy.

The growing demand for multimedia applications, especially video, has created a significant requirement for efficient video compression techniques. The telecommunications services, entertainment industry, and even home applications rely heavily on fast and high-quality video compression methods. However, the challenge lies in the fact that digital video data rates are typically large, consuming substantial bandwidth for transmission, storage, and computing resources (Taubman and Marcellin, 2001; Iain and Richardson, 2003). Without effective compression, multimedia applications such as videoconferencing, teleconferencing, satellite communications, and digital movie storage would be impractical due to their extensive data requirements.

Despite the fact that a number of standards for video compression have been created and made accessible, continuous research strives to improve compression speed and quality through effective strategies. Notably, video image compression poses a computational burden that exceeds the capabilities of conventional sequential processing (Kung et al., 2008). This is primarily due to the time-consuming repetitive operations involved in motion estimation algorithms and the substantial volume of similar data that needs to be processed. Consequently, to enable the real-time transmission, storage, or display of video, it becomes imperative to employ efficient motion estimation techniques and leverage parallel processing for video compression [3] [9].

A key element of many video compression techniques is motion estimation. With the exception of differences brought on by moving objects inside the frames, there is often a high connection between succeeding pictures in a motion film [2]. Both inside a single picture frame, also known as intra frame as well as between successive frames (inter-frames), many strategies may be used to minimize video data. By removing irrelevant information from a frame, data compression may be accomplished while potentially lowering picture resolution. Video data may be compressed for a series of frames using techniques like difference coding, which is often used in well-liked standards of video compression such as H.264 [4][12].

In difference coding, only the pixel values that have altered when compared to the actual benchmark frame are coded after comparing a frame to it. By using this method, fewer pixel values must be encoded and delivered. The efficiency of encoding can be further enhanced by performing difference detection and encoding at the block level, using macroblocks instead of individual pixels. This method enables larger areas to be compared, and only blocks that exhibit significant differences are encoded.

The method of motion estimate is used to ascertain the movement of objects inside a video in order to get over the restrictions of difference coding. Motion estimation creates motion vectors that depict the estimated motion by analyzing how things move over a series of photographs. Data compression is then accomplished by using this information in motion compensation. Every block in a picture frame is compared to a predetermined region in a previous frame during motion estimation to choose the block that matches the other the best and has the least distortion. The vector of motion can be calculated by measuring the separation between the two blocks. The motion estimating technique produces vectors of motion for every block as well as the pixel values that vary between the coincided blocks in the frame of reference and the current frame. With the goals of lowering computing complexity, properly capturing motion, and minimizing bit-rate, a variety of approaches have been suggested for estimating movement in video compression. A popular but time-consuming technique for motion estimation is Full Search block matching, in which the blocks in one frame are matched to the blocks in subsequent frames in order to determine motion vectors. Motion estimation, often performed using specialized hardware, is an essential step in video compression, also known as inter-frame coding. While real-world video sequences with complicated patterns of rotation and translation make it difficult to predict motion accurately, translational motion may be simply calculated and has been effectively used in motion-compensated coding.

The Block Matching Algorithm and pixel-based motion estimation are two approaches to estimate the motion. Although the fact that the objective of pixel-based motion prediction is to determine the vectors of motion for each pixel, this technique is computationally complex and unsuitable for real-time applications [5][7]. On the other hand, a quicker option is block-based motion estimate, which is frequently utilized in video coding standards. The frame is divided into macroblocks via block-based motion estimation. A new frame may be anticipated by contrasting the blocks in the present frame with those in the reference frame. The encoder encodes the motion vector, which shows the corresponding matching block's location in the frame of reference, rather than the content of every block. Less bits are needed to encode this motion vector, which results in compression.

The motion vectors help the decoder locate the corresponding blocks in the frame of reference during decompression to ensure the current frame may be rebuilt. It is crucial to keep in mind that motion estimation only considers translational movement in the picture and may struggle with video zooms, pans, brightness changes, and rotating motion. In these situations, intra-frame coding methods are utilized to maintain accuracy and computing cost. Depending on the unique needs, motion estimate systems may change the block dimensions within a frame.

Utilizing a technique called inter-frame predictive coding, redundant temporal and spatial information is removed from video sequences. This method transmits and encodes the variation between the present frame and the expected frame. The transmitted rate of the data and error are directly impacted by the prediction's accuracy. A pixel from the same area of the object that is moving is thought to provide a better forecast for the present one whenever there is movement in the sequence.

Different matching standards are utilized for targeting and candidature blocks in video compression to assess how similar they are. The Pel Difference Classification (PDC), Sum of Absolute Difference (SAD) etc., are common matching standards for estimation of motion of the blocks [1]. These criteria make it possible to effectively estimate motion and measure the match's quality.

The SAD block matching criteria is often utilized in estimation of motion [10]. Comparing similar pixels from two different blocks and adding up their absolute differences is what it entails. At each search location, the calculation is carried out for each candidate block. The SAD criteria, which is defined by the equation below, calls for calculations of subtractions and additions with absolute values but not multiplications [8]. This makes it physically implementable and computationally effective.

$$\mathrm{SAD}_{A,B}(x,y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left| A(x,y) - B(x+m, y+n) \right|$$

It is possible to implement motion estimation techniques such the diamond search algorithm, hexagonal search algorithm, full search algorithm, etc. An easy way for estimating motion is the full search block comparison algorithm. To determine the position of the corresponding block, it analyses a block in the present picture frame to each block in the frames that follow. The area of search is often constrained to a smaller area centered on the selected block in the present frame. A candidate block may only vary from the current block's location in the frame by a certain number of pixels, which is determined by this constraint, referred as the maximum amount of displacement. The maximum displacement guarantees that object motions between frames stay within a controllable range and is essential for efficient coding.

The hexagonal search algorithm is a technique used to efficiently determine the vector of motion in video compression. With fewer search locations and more accurate motion estimates than the entire search method, it is an improvement. The beginning search point, often the center of the search region, is where the hexagonal search algorithm begins the search process. Concentric hexagons are created around the starting point by the placement of the nearby search points in a hexagonal arrangement. A subset of the search locations is examined at each step as the algorithm moves outward from the center in a spiral search pattern.

The method compares the block at the present search point with the equivalent block in the image frame of the reference at each search step. A matching criterion, such as the SAD, is used to assess how similar the blocks are to one another [6][11]. A point of search is modified if an improved match is discovered. The entire process shall be repeated until the optimal match for the search area has been determined.

When compared to the complete search method, the hexagonal search algorithm provides a considerable reduction in computing complexity. It more effectively covers the search region by using a spiral search and hexagonal pattern, which lowers the number of comparisons needed. This leads to more accurate motion vector determination with quicker motion estimates.

II. Methodology

There are various algorithms to perform motion estimation using SAD. Our work includes implementation of SAD architecture using full search algorithm and hexagonal search algorithm.

A. *Methodology for performing Full Search Algorithm using SAD:*

1) Establish the search range: Determine the maximum displacement allowed for the block matching algorithm, which defines how far the algorithm will search for a matching block. In our work, we have taken a reference and current image size of 96x96.

2) Divide the reference frame into blocks: Partition the reference frame or image into blocks of equal size. The block size can vary depending on the application's requirements. In our work, we have taken the block size of 16x16, so a total of 36 blocks are in the current frame and reference frame.

3) Choose an anchor block: Choose a block from the frame of reference that will serve as the search's starting point. We have loaded the complete anchor and candidate blocks into the FIFO.

4) Initialize the minimum SAD threshold value: Set the initial values for the minimum SAD threshold values (set to a large value initially) and the associated motion vector (initialized as zero).

5) Start your search: Start looking for the anchor block's best match inside the predetermined search range. This involves comparing each block in the searching region of the current frame to the anchor block.

6) Compute the SAD: To determine the SAD for each candidate block, add up the absolute differences in pixel values between candidate and anchor blocks.

7) Compare SAD values: Compare the calculated SAD value to the existing lowest SAD value. Modify the minimum SAD value and its corresponding motion vector if the calculated SAD value is lower than the current minimum.

8) Repeat the process: Iterate the search process by moving the anchor block to the next position within the search range. Repeat steps 6 and 7 for each position.

9) Determine the motion vector: After searching all positions within the search range, the vector of motion with the smallest value of SAD is the optimal match with respect to the anchor block.

10) Apply to every block: For every block in the reference frame, repeat steps 3 through 9. A motion-based vector field spanning the full frame will result from this.

Initially, we are loading the blocks of reference into the FIFOs in the following manner. The blocks we are taking are 16x16 sized, so there are 16 columns and 16 rows that are present in both reference and current blocks and we are considering each pixel as a byte width.

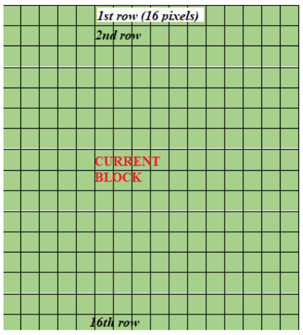

Fig .1. Current Block (CB)

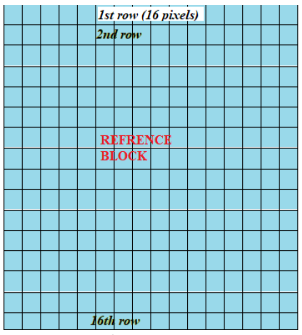

Fig .2. Reference Block (RB)

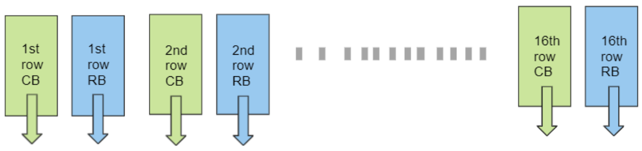

Fig.3. Loading of row pixels in respective FIFOs

There are 16 FIFOs for each reference and current block, and the size of each FIFO is 16x8 (which contains 16 rows, and each row is 8 bits wide). The row information belonging to the present and reference blocks are included in each FIFO. Each FIFO contains row data from either the current or reference blocks. When all FIFOs are full, then the go signal triggers in the FSM, and then we perform the SAD operation. For each clock cycle, we read each pixel of data from the FIFO.

After reading all the values, the FIFO gets empty, then the next block gets loaded, and the same operation continues. So, to load the pixel values into the FIFO, we need 16 clock cycles, and to perform the SAD value computation, we need 16 clock cycles. So, for a 96x96 block, there are 36 16x16 blocks, so to process the whole frame, we require a total clock pulse of 16 clocks per block x 36 blocks =576 clocks to process the whole frame. The controller FSM contains five states. The initial state is S0 (also called the reset state). When all the FIFOs have completed storing the data inside them, the data required for the operation is present on the chip, and the go signal gets triggered.

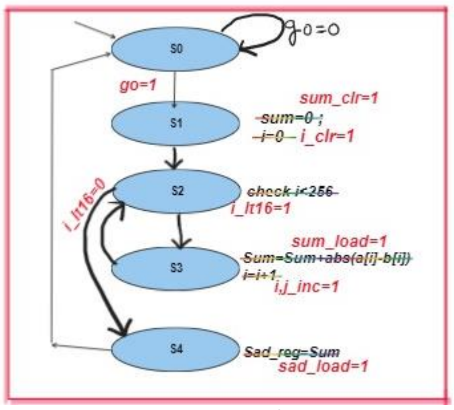

Fig. 4. Controller FSM for Full Search

Then, in the s1 state, all the counters and default values are initialised, and in the s2 state, it checks for the internal counter and then performs the full search because the hardware reuses the modules 16 times to complete the SAD calculation for the whole block. In s3 state, we will perform the accumulation of all the absolute differences and store them in sum reg. After all the interactions of the counter are complete, we can now go to the S4 state to give the final output.

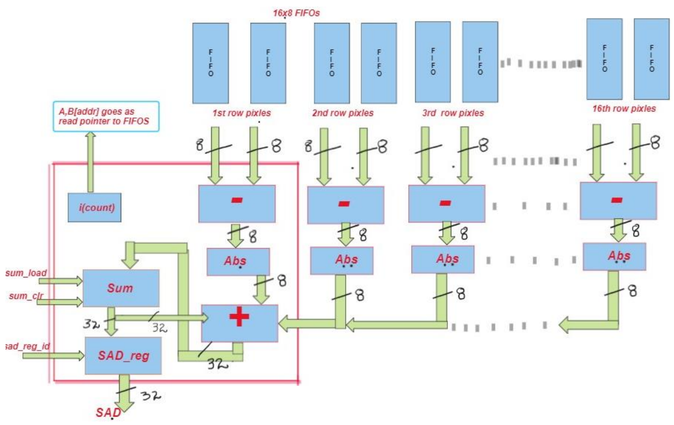

Fig.5.Parallel Processing SAD architecture for performing Full Search

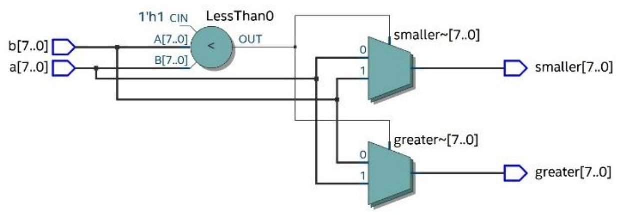

Fig. 6. RTL Schematic of Absolute module

B. *Methodology for performing Hexagonal Search Algorithm using SAD:*

The disadvantage with the Algorithm of Full Search is, it is an exhaustive search process in which we are supposed to calculate SAD value of each and every pixel in a frame. So, it requires a huge number of clock cycles to perform search operation.

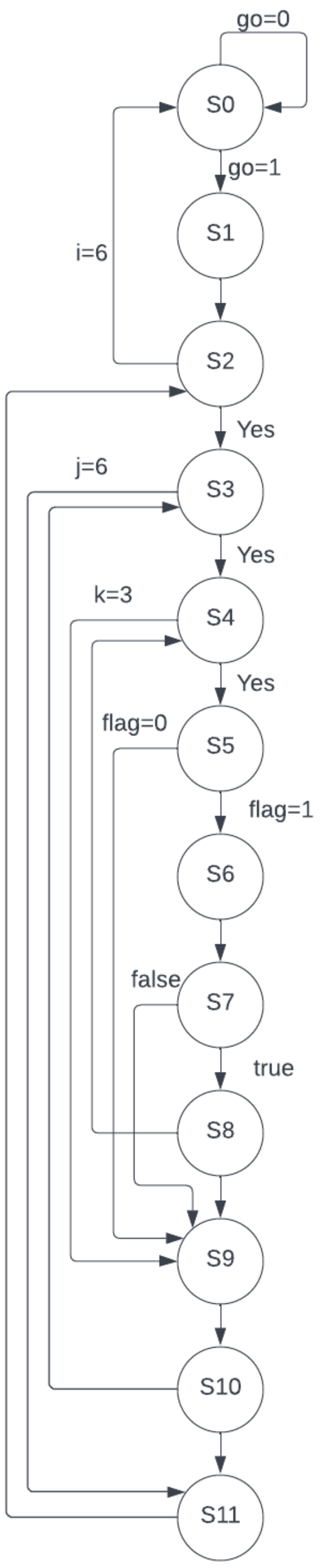

Fig. 6. Hexagonal Search FSM

S0: Initial state after reset When the frame current and reference frame data are stored in external memory, the "*go signal*" goes high, and so the transition happens.

S1: Initialization of all control signals used in the control path of the design is done in this state.

S2: The "i" variable denotes the number of rows present in the blocks present in the frame. In the proposed design, the frame size is 96x96, hence there are 6 rows. The state checks the "i" value, and if it's less than 6, a transition to *S3* takes place; otherwise, if i = 6, the frame is completely processed, and the state shifts to *S0*.

S3: "j" denotes the number of columns of blocks. In the proposed design, there are & blocks of 16x16 size here, and if j6, the FRM shifts to the *S4* state to perform the SAD operation on the block; otherwise, the FSM shifts to *S9***,** which denotes that all blocks of 16x16 size present in a row of the frame are processed.

S4: Here, k denotes the number of hexagon centre changes to perform to obtain the best match and minimum SAD value. If the value of K is less than 3, FSM shifts to *S4;* otherwise, it shifts to *S9*, which denotes that the processing on the current block of the image is finished.

S5: This state includes the centre initialization for the hexagons in the current reference block. This state also possesses a control signal to calculate all hexagon coordinates using the centre coordinates and provide them to external memory. If the calculated coordinates are going outside the frame, the flag goes zero and FSM shifts to the *S9* state; otherwise, it shifts to the *S6* state to process the block.

S6: The FSM sends a control signal to calculate the coordinate in the frame of reference with the least value of SAD. The FSM will then switch to S7.

S7: FSM shifts to S9 if the least value of SAD is obtained for the hexagon's centre coordinate. The FSM shifts to S8 if the minimal value of SAD becomes available for any hexagon vertex.

S8: In this state, FSM shifts its centre of the hexagon in the reference frame block to the vertex having the minimum SAD value. The k-counter is incremented, and from there, it shifts to *S4***.**

S9: This state indicates that the processing to identify the optimal match for a block is complete and that the centre of reference frame-blocks have been stored in the register to generate the motion vector.

S10: This state indicates that all blocks have been processed and the adjustment of each block in the present picture frame to locate the corresponding vector of motion.

S11: This state denotes that the processing of the entire block is done and the FSM shifts to the new row of blocks.

The hexagonal search using the FSM can be performed as shown above. Synthesizable hardware using Verilog was created.

In order to calculate the search points, center of the block is need to be fixed first, and then the search radius of the operation is fixed. The default search radius of three units is considered. This can also be user-defined in the design to create various search radius design and the hexagonal search points are taken as shown below. There are a total of seven points, including the center. The absolute differences between the pixels are calculated and the SAD operation is performed.

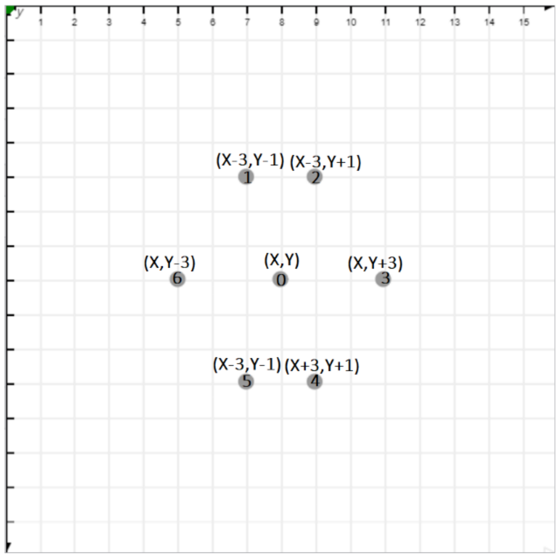

Fig. 7. Hexagonal Search Points

CENTER POINT	*(X , Y)*
1st point	*(X-3 , Y-1)*
2nd point	*(X-3 , Y+1)*
3rd point	*(X , Y+3)*
4th point	*(X+3 , Y+1)*
5th point	*(X-3 , Y-1)*
6th point	*(X , Y-3)*

Table 1. Hexagonal Search Points

We can also perform the diamond search operation with the same coordinates by taking the midpoint of the top two and bottom two (1st and 2nd, 4th and 5th) points.

III. Results and discussion

The results section deals with the simulation and synthesis results of the SAD module with full search-based and hexagonal-based algorithms. Simulation is performed using the ModelSim tool, and the synthesis is performed using the Xilinx Vivado tool. For simulation purposes, we have considered two images of size 96x96, and the reference block size of 16x16.

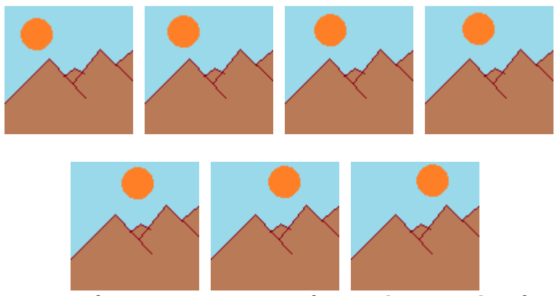

Fig. 8. Reference 96x96 images frames that are taken for simulation purpose.

A. *FULL SEARCH ALGORITHIM*

These are the reference frames of the data set, where the majority of the frame data is stable (mountains and sky), but only a few portions of the images are changing (sun). So, we are trying to detect the motion of the sun by performing the SAD operations between the two consecutive frames using Full Search Algorithm.

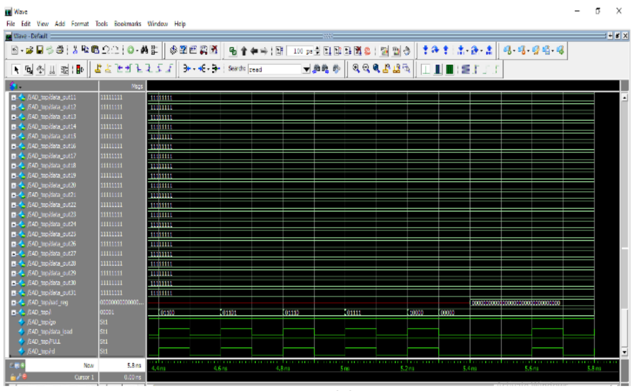

Fig. 9. Simulation result of full search based SAD

B. *HEXAGONAL SEARCH ALGORITHIM*

In the Hexagonal search for performing three shifts in centre of hexagon, the SAD calculation is as follows

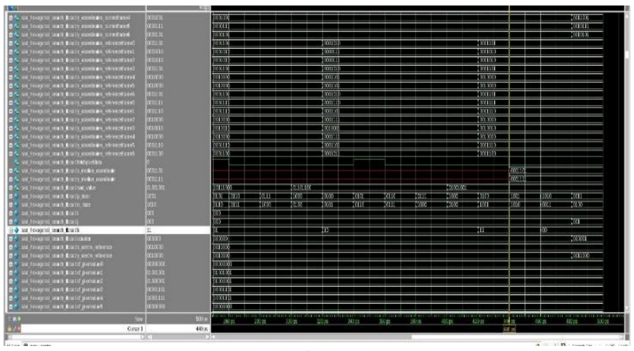

Fig. 10. Simulation result of Hexagonal search based SAD

From the above simulation of both hexagonal and full search simulations we can observe that, number of clock cycles to perform SAD for whole 96x96 frame (with 16x16 internal block) using Full Search are, 32 clocks/block x 36 blocks = 1152 clocks. Number of clock cycles to perform SAD for whole 96x96 frame (with 16x16 internal block) using Hexagonal Search is 19 clocks/block x 36 blocks = 684 clocks.

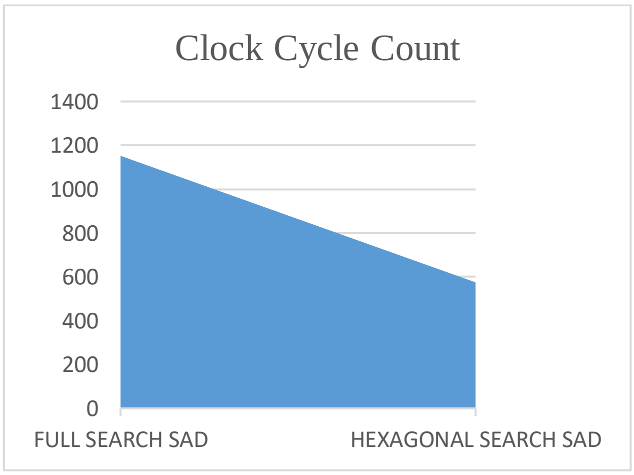

Fig. 11. Simulation result of full search based SAD

The below are the synthesis results of the both Hexagonal and Full search based SAD hardware. The synthesis is done on Xilinx tool using virtex7 hardware platform.

Algorithm used	***SLICES***	***LUT***	***Maximum Delay***	***Frequency of operation***
FULL SEARCH	378	1037	7.797 ns	128.36 MHz
HEXAGONAL SEARCH	146	1794	1.859 ns	537.86 MHz

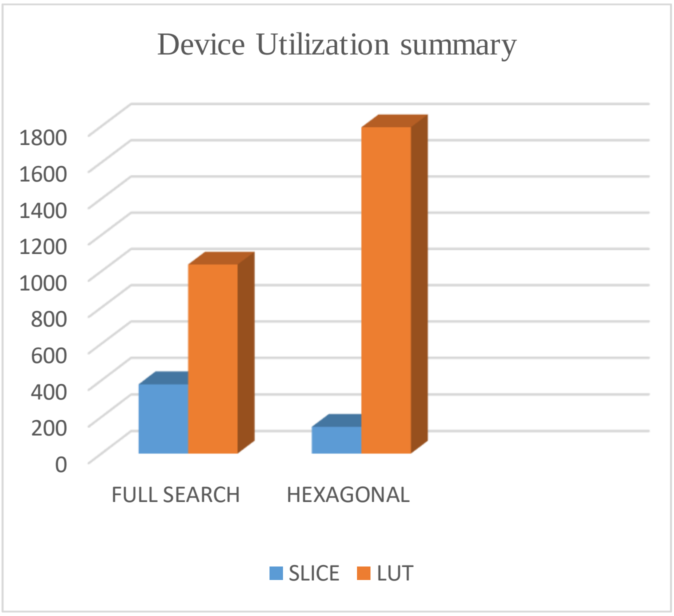

Fig. 12. Device utilization summary for both algorithms

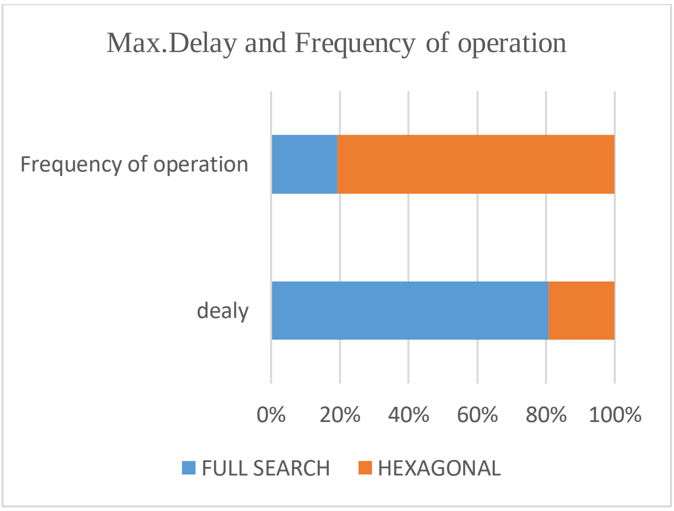

Fig. 13. Delay and Frequency for both algorithms

The above synthesis results suggest that the hexagonal-based search methodology is resulting in good performance when compared to full search. In full search, the critical path delay is so huge that we need a higher frequency of operation. But in a hexagonal path, it only requires a few pixels and iterations. So the overall delay becomes less, and a greater frequency of operation is achieved when compared to the full search.

IV. Conclusion

The full search and hexagonal search algorithms using the SAD hardware architecture were implemented, and synthesis was performed on the Virtex-7 FPGA board. From the simulation and synthesis results, we can infer that the hardware model of SAD that has been implemented can be used to perform both the full search and hexagonal search block matching techniques. The findings from implementing this methodology are that the hexagonal search is more efficient than the full search in terms of operating frequency and critical path delay. These algorithms can be used to perform operations on block sizes ranging from 8x8 to 128x128. By using some special adders, other parameters, such as area and delay, can be improved. Reducing the critical path delay and using higher operating frequencies can be solutions for processing higher-resolution videos.

References

[1] Silveira, B., G., Abreu, Paim B., M., Diniz, Grellert, C. M., da Costa, E. A. C., & Bampi, S. (2017). Power-efficient sum of absolute differences hardware architecture using adder compressors for integer motion estimation design. *IEEE Transactions on Circuits* 64(12), 3126-3137.

[2] Srinivas Rao, K., & Paramkusam, A. V. (2022). Block Matching Algorithms for the Estimation of Motion in Image Sequences: Analysis. *Pattern Recognition and Image Analysis*, *32*(1), 33-44.

[3] Paim, G., Santana, G. M., Abreu, B. A., Rocha, L. M., Grellert, M., da Costa, E. A., & Bampi, S. (2020). Exploring high-order adder compressors for power reduction in sum of absolute differences architectures for real-time UHD video encoding. *Journal of Real-Time Image Processing*, *17*(5), 1735-1754.

[4] Medhat, A., Shalaby, A., Sayed, M. S., Elsabrouty, M., & Mehdipour, F. (2014, November). A highly parallel SAD architecture for estimation of motion in HEVC encoder. *IEEE Asia Pacific conference (APCCAS)* (pp. 280-283). IEEE.

[5] Pandian, S. I. A., Bala, G. J., & Anitha, J. (2013). A pattern based PSO approach for block matching in motion estimation. *Engineering Applications of Artificial Intelligence*, *26*(8), 1811-1817

[6] Kumar, U. A., Guturu, S., & Ahmed, S. E. (2022). Design and exploration of low-power SAD architectures using approximate compressors for Integer Motion Estimation. *Microprocessors and Microsystems*, *94*, 104659.

[7] Gogoi, S., & Peesapati, R. (2022). Design and Implementation of Gray-Coded Bit-Plane Based Reconfigurable Motion Estimation Architecture Using Binary Content Addressable Memory for Video Encoder. *IEEE Transactions on Consumer Electronics*, *68*(1), 85-92.

[8] Koshta, J., Khare, K., & Gupta, M. K. (2019). Efficient absolute difference circuit for SAD computation on FPGA. *International Journal of VLSI Design (VLSICS) Vol, 10*.

[9] Fontanari, T. V., Paim, G., Rocha, L. M., Ücker, P., Costa, E., & Bampi, S. (2020, February). An efficient N-bit 8-2 adder compressor with a constant internal carry propagation delay. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)* (pp. 1-4). IEEE.

[10] Wong, S., Vassiliadis, S., & Cotofana, S. (2001, November). SAD implementation in FPGA hardware. In *Proceedings of the 12th Annual Workshop on Circuits, Systems, and Signal Processing (PRORISC2001)*.

[11] Selvan, S. K. (2021, November). Verilog-HDL Based Implementation of Sum of Absolute Difference Architecture using Adder Compressors. In *2021 IEEE 2nd International Conference on Applied Electromagnetics (AESPC)* (pp. 1-5). IEEE.

[12] He, G., Zhou, D., Li, Y., Chen, Z., Zhang, T., & Goto, S. (2015). High-throughput power-efficient VLSI architecture of fractional motion estimation for ultra-HD HEVC video encoding. *IEEE Transactions on VLSI Systems*, *23*(12), 3138-3142.