Circuits Syst Signal Process (2018) 37:4548–4567
https://doi.org/10.1007/s00034-018-0778-8

# A High-Speed VLSI Architecture for Motion Estimation Using Modified Adaptive Rood Pattern Search Algorithm

**Baishik Biswas[1] · Rohan Mukherjee[1] · Indrajit Chakrabarti[1] · Pranab Kumar Dutta[2] · Ajoy Kumar Ray[1]**

Received: 27 April 2017 / Revised: 31 January 2018 / Accepted: 1 February 2018 /
Published online: 15 February 2018


**Abstract** The paper presents an efficient VLSI architecture for fast Motion Estimation in video codec using modified Adaptive Rood Pattern Search Algorithm. The proposed architecture uses an interleaved memory arrangement and an early check technique to compute the Sum of Absolute Differences. The proposed design can process High Definition (1080p) video frames in real time while optimizing the hardware area. The architecture has been implemented in verilog HDL and mapped to 45 nm FPGA. It uses only 6.8K gates for the implementation of the datapath and the controller. It achieves a maximum frequency of 120 MHz. However, working at 100 MHz, it is able to process 60 HD ($1920 \times 1080$) frames per second while consuming 39 mW of power. The proposed architecture achieves premium speed with an optimum power and area requirements and can be suitably incorporated in light-weight video-intensive devices like smart-phones, tablet computers.

✉ Baishik Biswas
boishik@gmail.com

Rohan Mukherjee
mukherjee.rohan666@gmail.com

Indrajit Chakrabarti
indrajit@ece.iitkgp.ernet.in

Pranab Kumar Dutta
pkd@ee.iitkgp.ernet.in

Ajoy Kumar Ray
akray@ece.iitkgp.ernet.in

1 Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal 721302, India

2 Department of Electrical Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal 721302, India

**Keywords** VLSI architecture · FPGA · Motion Estimation · Modified Adaptive Rood Pattern Search · Interleaved memory

## 1 Introduction

Recently, there has been a huge demand for high-quality video on mobile devices that operate within stringent power budget and area constraints. Video codecs are now required to perform highly computational tasks satisfying real-time demands. In a video coding system, Motion Estimation (ME) is usually identified as the most computationally intensive operation. ME is a widely used compression technique wherein temporal redundancies are exploited to achieve high efficiency in video coding. However, the computationally demanding Motion Estimation requires the use of dedicated VLSI architectures to achieve optimum speed, area and power requirement [16].

Block matching is a simple and an efficient method of motion estimation. The Sum of Absolute Differences (SAD) is usually adopted as the matching criteria. With a view to reduce the computational complexity, a number of fast search algorithms have been proposed such as Three Step Search (TSS) [9], Diamond Search (DS) [22], Hexagonal-Based Search (HEXBS) [21], Adaptive Rood Pattern Search (ARPS) [14] and the modified Adaptive Rood Pattern Search (ARPS2) [11]. It has been shown in [14] and [11] that the ARPS and ARPS2 are considerably faster than DS and HEXBS without compromising the visual quality. The speed and PSNR performance of ARPS and ARPS2 algorithms have made them valuable for the implementation in video codecs. However, unlike full search (FS), TSS, DS or HEXBS, the irregular search pattern in ARPS and ARPS2 does not make them attractive for hardware implementation.

Use of systolic array has been a popular choice for mapping of fast Block Matching Algorithms (BMA) to hardware. Processing Elements (PE), having basic hardware modules are arranged in array or mesh formation. As discussed in [10], systolic array architectures can be categorized by the dimension, the number of PEs and the data-flow involved. Any systolic array architecture requires the regular flow of data in and out of the PEs. Data are sent to all the PEs with an aim to parallelize the operation. The video frame is usually stored in Random Access Memory (RAM) which permits the desired data flow. The primary challenge encountered in conventional architectures is to maintain a regular data flow and ensure maximum PE utilization. Some architecture proposes reconfigurable PE array to support blocks of various sizes. Use of bigger PE generally leads to improved frame rate but at the cost of increased hardware area, high memory bandwidth requirement and more power consumption. Moreover, array architectures are usually suitable for the implementation of regular ME algorithms like FS, TSS, HEX, and DS that have a fixed initial stage of search with a predefined set of points. In contrast, the initial pattern in ARPS2 is highly adaptive with variable number of points. Not only does the adaptive pattern make mapping to PE array complex, also the hardware remains significantly unutilized. This paper proposes a VLSI architecture that avoids the use of traditional PE arrays for the implementation of fast search algorithm ARPS2. The proposed architecture avoids the use of multiple PEs to optimize the hardware area while leveraging the speed of ARPS2 algorithm.

Array architectures can also be classified according to the dataflow involved. The array architectures proposed in the literature can be broadly categorized into inter- and intra-classes of architectures which primarily differ in the data-flow. Each processing element (PE) in the inter-level architecture is responsible for the computation of the Sum of Absolute Difference (SAD) for one candidate block. On the other hand, the PEs in the intra-level architecture compute the distortion for each pixel in the candidate blocks. The architectures proposed in [1,3–6,8,10,13,15,17–20] are based on inter/intra-level schemes with PEs organized in either 1D or 2D arrays. The architecture proposed by Jehng et al. [7] for 3 step hierarchical search (3SHS) is a seminal work on 2D systolic mesh architectures. It is an example of intra -level scheme wherein the PEs are arranged in the form of $4 \times 4$ 2D array. Each PE is loaded with a pixel value of a $4 \times 4$ current frame macroblock. A tree-like SAD adder arrangement has been used to accumulate the difference values computed by the PEs. Although suitable for 3SHS, such a dataflow would not be able to capture the variability of ARPS2. The architecture proposed by Ndili et al. [13] for DS uses on-chip buffer memories that are loaded with the search passes. Although such a memory arrangement could be adapted for ARPS2, the buffer memory could remain largely unutilized owing to the variable number of search points. Recent work by Akin et al. [1] modified the traditional intra-level architecture to adapt to bilateral Motion Estimation. The architecture by Tsai et al. [17] takes in consideration an effective algorithm which is capable of reducing the number of search points and has offered a 2-D systolic array-based design that works at 200MHz with 191K gate count. Rehan et al. [15] have proposed FPGA-based implementation of Flexible Triangular Search (FTS) which lowers the area by adaptive selection of certain defined locations within the search window. A parallel architecture for TSS has been presented by Tseng et al. [18] with enhanced throughput and low area.

### 1.1 Motivation and Contribution of the Present Work

Certain hindrances exist while adopting the array-based designs and fine grained data reuse scheme for adaptive algorithms like ARPS and ARPS2. The irregular and unpredictable data flow in ARPS2 algorithm leads to a complex memory access pattern which further increases the intricacies of realizing intra and inter architectures. Moreover, the PE arrays and buffer memory can remain largely unutilized.

Previous VLSI designs for iterative and adaptive algorithms like hexagonal search and ARPS [2,12] have certain limitations. The designs use single PE and follow sequential execution. As a result, they can process frames of CIF format.

The architecture presented in this paper for ARPS2 explores a flexible computation methodology which does not involve PE arrays. Since ARPS2 is a considerably fast search algorithm, the proposed design aims to reduce the hardware area significantly without sacrificing real-time speed. The proposed architecture also introduces an efficient pattern generation scheme with an early SAD check technique to save the clock cycles. Another contributory feature of the architecture is the interleaved memory arrangement which improves the throughput. The design also introduces an organized sharing strategy of the search window pixels which enables the process-

ing of large video frames. The proposed architecture is able to process 60 Full HD frames ($1920 \times 1080$) per second while taking up only 49.50K gates. It overcomes the limitation of previous design of ARPS in [2] that can process frames of CIF format. The design methodology adopted in this work can also be used for implementation of other adaptive ME algorithms. The area and clock cycles performance of the proposed architecture has been compared with other existing fast search architectures and the present work is demonstrated to be ahead of others.

The remainder of the paper is organized as follows. Section 2 describes the modified ARPS algorithm; whereas Sect. 3 discusses the proposed architecture. Section 4 shows the results and finally, Sect. 5 draws the conclusion.

## 2 The Modified ARPS Algorithm

Speed of fast ME algorithms is usually indicated by how few points are needed to be searched to get the best match. The modified ARPS algorithm which is proposed by Ma et al. [11] is one such fast algorithm that has shown remarkable speed. Although based on the popular ARPS search technique proposed by Nie and Ma [14], ARPS2 has even fewer search points with very small degradation in the PSNR achieved. Table 1 compares the PSNR achieved by the existing search algorithms with that of ARPS2. Table 2 demonstrates the speed of ARPS2 algorithm tested on standard CIF sequences.

As seen from Table 2, ARPS2 search has the smallest number of search points as compared to the standard existing algorithms. The following paragraph discusses the ARPS2 algorithm in detail.

The ARPS2 algorithm consists of two distinct stages of search: an initial adaptive stage and compact unity rood pattern search stage. As shown in Fig. 1, the modified ARPS algorithm [11] is very similar to the ARP search algorithm [14] with the exception of the initial search pattern. Like ARPS, the search points of ARPS2 are located on the vertices of a symmetrical rood (cross) shaped pattern.

However, the center of the rood pattern is displaced from the origin as shown in Fig. 1b by a distance equal to the predicted Motion Vector (MV). Figure 1c shows the Unity Rood Pattern (URP) search points which are equivalent to the small diamond search described in [22]. In order to define the rood pattern parameters, the ARPS algorithm defines a set of co-located macroblocks called the Region of Support (ROS) shown in Fig. 2. The $X$ and $Y$ arm lengths, denoted by $T_x$ and $T_y$, respectively, are computed from the Motion Vectors (MVs) of the ROS blocks by using equations (1-2). Unlike ARPS, ARPS2 does not have any predicted point as the center of the rood pattern is itself placed on the predicted point. The predicted point is also derived from the Motion Vectors of the ROS. For simplicity, the Motion Vector of the left-most neighboring block (i.e., block A) has been chosen as the predicted Motion Vector and its motion vector is adopted as predicted motion vector (MVP).

$$T_x = \frac{1}{2}(\max(\mathrm{MV}_x) - \min(\mathrm{MV}_x)) \tag{1}$$

$$T_y = \frac{1}{2}(\max(\mathrm{MV}_y) - \min(\mathrm{MV}_y)) \tag{2}$$

**Table 1** PSNR of few existing search algorithms (unit of PSNR is dB)

| Sequence | FS | TSS [9] | DS [22] | HEX [21] | ARPS [14] | ARPS2 [11] |
|---|---|---|---|---|---|---|
| Foreman | 36.47 | 35.84 | 36.17 | 35.59 | 36.04 | 35.99 |
| Coastguard | 33.95 | 33.71 | 34.15 | 33.60 | 33.90 | 33.89 |
| Tennis | 29.81 | 29.15 | 28.10 | 29.18 | 29.28 | 29.20 |
| News | 40.11 | 39.95 | 39.47 | 39.84 | 39.97 | 39.97 |
| Mother and daughter | 42.90 | 42.77 | 42.06 | 42.65 | 42.80 | 42.79 |
| Mobile | 29.77 | 29.73 | 29.66 | 29.67 | 29.75 | 29.73 |
| Highway | 38.57 | 37.82 | 37.24 | 37.42 | 37.49 | 37.61 |
| Soccer | 22.69 | 22.22 | 22.13 | 22.13 | 22.62 | 22.53 |

**Table 2** Average number of search points of standard ME algorithms

| Sequence | FS | TSS [9] | DS [22] | HEX [21] | ARPS [14] | ARPS2 [11] |
|---|---|---|---|---|---|---|
| Foreman | 256 | 27 | 17.13 | 13.28 | 9.74 | 8.11 |
| Coastguard | 256 | 27 | 17.74 | 13.78 | 9.17 | 7.14 |
| Tennis | 256 | 27 | 17.90 | 13.71 | 9.89 | 8.95 |
| News | 256 | 27 | 13.45 | 11.26 | 6.04 | 6.00 |
| Mother and daughter | 256 | 27 | 14.67 | 11.84 | 7.43 | 7.33 |
| Mobile | 256 | 27 | 13.89 | 11.33 | 7.38 | 6.23 |
| Highway | 256 | 27 | 16.11 | 12.77 | 8.77 | 8.04 |
| Soccer | 256 | 27 | 24.88 | 17.41 | 14.01 | 13.37 |

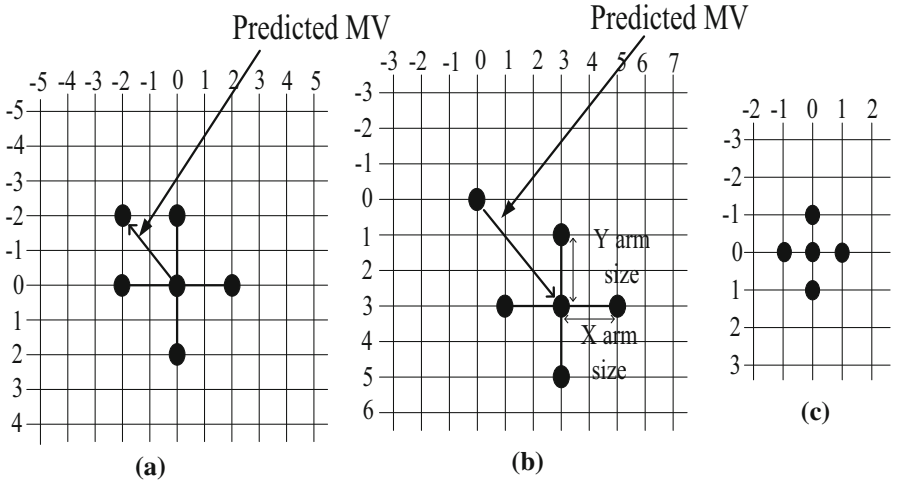


**Fig. 1** The search patterns in ARPS and ARPS2. The dots indicate the location of a 16 × 16 macroblock. **a** The initial adaptive pattern of ARPS. **b** The Modified Adaptive Rood Pattern. **c** The unity rood pattern

$MV_x$ and $MV_y$ refer to the $x$ and $y$ components of all the Motion Vectors in the ROS. The logical operators 'max' and 'min' compute the maximum and the minimum values, respectively, of all the arguments.

The initial rood pattern is followed by a compact Unity Rood Pattern (URP) for refined local search. The URP is applied iteratively as long as the new search center does not coincide with the previous search center. It consists of a rood-shaped pattern with an arm length of 1 as shown in Fig. 1c. The Sum of Absolute Differences (SAD) has been adopted as the distortion criterion. SAD for a block-size of $N \times M$ pixels is defined as

$$\mathrm{SAD} = \sum_{1 \le i \le N} \sum_{i \le j \le M} |\mathrm{ref}(x_p + i, y_p + j) - \mathrm{curr}(x_0 + i, y_0 + j)| \tag{3}$$

where $(x_p, y_p)$ and $(x_0, y_0)$ refer to the candidate block and the current block, respectively.

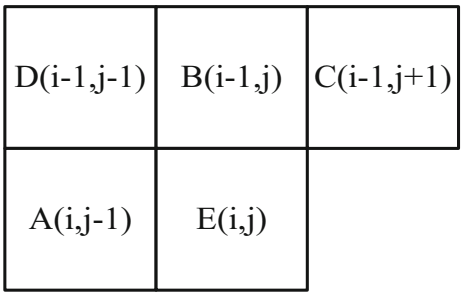


**Fig. 2** Region of support (ROS) for the prediction of roof pattern in ARPS2 algorithm. Blocks A, B, C, D are the four neighboring blocks of the current block E

## 3 Proposed Architecture

This section covers the details of the proposed design. The Sum of Absolute Differences (SAD) is computed for every point in a sequence and the minimum SAD point is obtained thereafter. Figure 3 shows the top-level diagram of the system required to implement the algorithm. The logical block marked as P is the pipeline registers. The Search Pattern Generator and the SAD computation blocks form a computational pipeline. The diagram also presents the architecture in the form of a point-to-point network of logical block, where the New Position Generator is the 'master' unit initialing all transactions. Unlike conventional bus-based architecture, the master unit has point to point links to all of the hardware units namely Motion Vector Storage, Register Block, SAD computation unit. The latter comprises of the reference and current frame buffers along with the SAD computation logic. The registers are not memory mapped. There are point to connection (register connection entails load enable, data lines) to each individual registers.

The architecture makes use of small buffer memory to store the search window. The buffer memory is periodically loaded with image data from an external memory. The following subsections discuss the implementation of the various modules of the architecture shown in Fig. 3.

### 3.1 New Position and Offset Generator

This module generates the offsets of the search points in the ARPS2 algorithm. The offsets are generated in the sequence they are numbered in Fig. 4a, b. The Sum of Absolute Difference is computed between a reference block and the current frame macroblock. In the proposed design, SAD computation is performed as long as the SAD is less than existing minimum SAD. Such a termination procedure helps save clock cycles. Ideally, searching the points sorted in the increasing order of SAD would lead to the maximum savings in clock cycle. However, a significant saving in clock cycle could be achieved by starting the search with the predicted point, i.e., point '0' of Fig. 4a. Although the predicted point may not be the global minimum, it lies close to the global minimum and it is expected to have comparatively low SAD value. Such early termination leads to a speedup of close 30%. The results section further discusses the extent of overall reduction in number of clock cycles.

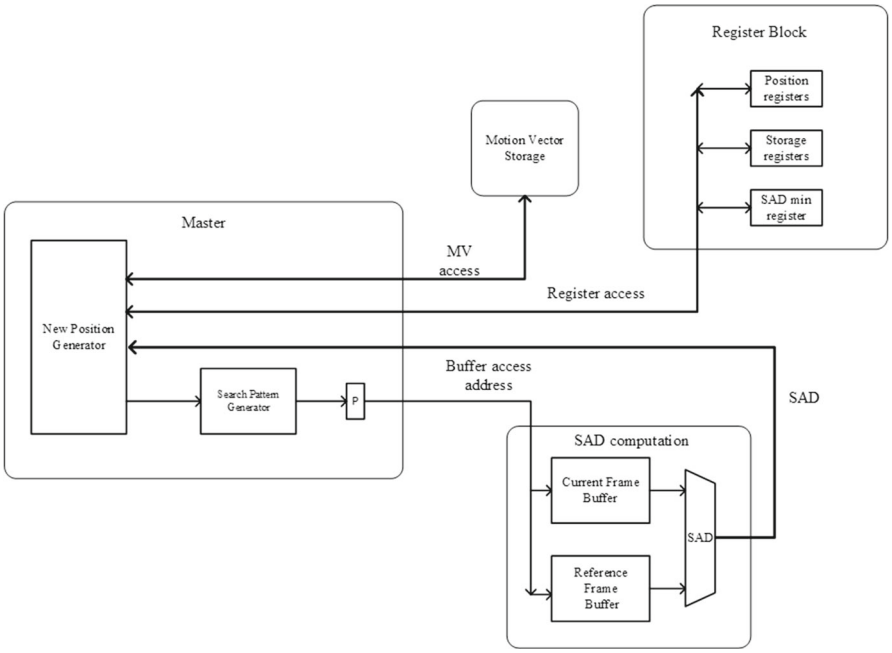


**Fig. 3** The top-level system diagram for the implementation of Modified Adaptive Rood Pattern Search Algorithm. The dotted lines represent control signals

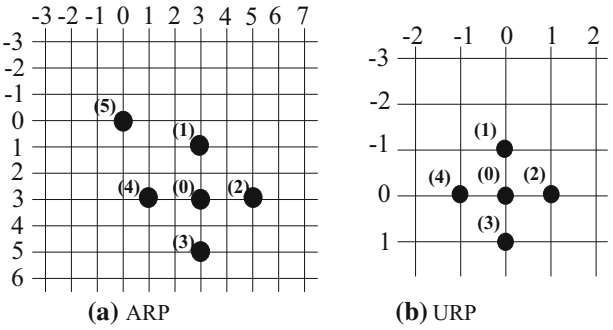


**Fig. 4** The search sequence for ARPS2 algorithm. Points are searched in the sequence they are marked

The high speed achieved by ARPS2 search is a consequence of the adaptive search pattern. Although the initial Adaptive Rood Pattern (ARP) consists of 6 points (see Fig. 4a), there could be a varying degree of overlap among the points. For example, if predicted arm lengths $T_x$ and $T_y$ are 0 and the predicted Motion vector ($MVP_x$, $MVP_y$) is not a null vector, search points 1–4 would merge with point 0, leading to only 2 points in ARP. Similarly, when magnitude of $MVP_x$ equals $T_x$, either point 2 or point 4 merges with the point 5. Therefore, it is evident that the ARP could contain all 6 possibilities, viz from having 1 point to having all the 6 points in the pattern. Table 3 further illustrates this statement. 'URP, ARP-min = pos 5' in Table 3 implies

**Table 3** Search sequences for various conditions

| Stage of search | Condition | Search sequence |
|---|---|---|
| ARP | $T_x = 0$, $T_y = 0$, $\mathrm{MV}_{\mathrm{predicted}} = \mathrm{null}$ | 0 |
| ARP | $T_x = 0$, $T_y = 0$, $\mathrm{MV}_{\mathrm{predicted}} \neq \mathrm{null}$ | 0, 5 |
| ARP | $T_x = 0$, $T_y \neq 0$, $\mathrm{MV}_{\mathrm{predicted}} = \mathrm{null}$ | 0, 1, 3 |
| ARP | $T_y = \lvert\mathrm{MVP}_y\rvert$, $\mathrm{MVP}_x = 0$ | 0, 1, 3, 4 |
| URP, ARP-min = pos 5 | $\lvert\mathrm{MVP}_x\rvert = T_y = 1$, $\lvert\mathrm{MVP}_y\rvert \neq T_x$ | 1, 3, 4 or 1, 2, 3 |
| URP, ARP-min = pos 0 | $T_x = 1$, $T_y \neq 1$ | 1, 3 |
| URP, ARP-min = pos 0 | $T_x = 1$, $T_y = 1$ | No points |

that the current search stage is the URP following the initial ARP search where the min SAD point is obtained at position 5 of Fig. 4a.

In order to generate the offsets of the search pattern, the 5 state of the search, i.e., parameters such as $T_x$, $T_y$, $\mathrm{MVP}_x$ and $\mathrm{MVP}_y$ are saved and the positions required to be skipped are found. A 6 bit loadable register is maintained to associate a '*skip*' status to every position. If *skip* is high, the current position is skipped and the next position is considered. A 3-bit register and an incrementer are maintained to generate the positions of the search pattern. The incrementer has provisions for incrementing by 2, 3, 5 or 6 places apart from the default increment by 1. Thus, using the skip logic, the increments of the counter are determined and the new position is generated at every positive edge of the clock. The 3 bit position register is loaded with an initial value of 0 to start the search from position 0. Figure 5 shows the hardware for position generation.

Equations (4)–(7) describe the Boolean equations for the generation of increment signals for the position counter.

$$\mathrm{inc}\,2 = \mathrm{skip}\,1 \tag{4}$$

$$\mathrm{inc}\,3 = \mathrm{skip}\,1\ \&\ \mathrm{skip}\,2 \tag{5}$$

$$\mathrm{inc}\,5 = \mathrm{skip}\,1\ \&\ \mathrm{skip}\,2\ \&\ \mathrm{skip}\,3\ \&\ \mathrm{skip}\,4 \tag{6}$$

$$\mathrm{inc}\,6 = \mathrm{skip}\,1\ \&\ \mathrm{skip}\,2\ \&\ \mathrm{skip}\,3\ \&\ \mathrm{skip}\,4\ \&\ \mathrm{skip}\,5 \tag{7}$$

The *skip* values are obtained from the knowledge of the search state. Although Table 3 is not exhaustive, it serves as the template to generate the *skip* logic for individual positions. The first location of the skip register (*skip 0*) of Fig. 5 belongs to the position being considered currently. For example, *skip 1* high implies that the next position has to be skipped, whereas *skip 2* high along with *skip 1* implies that next 2 positions are to be skipped. The skip register of Fig. 5 is initially loaded with the *skip* values at the beginning of a new search pattern (ARP or URP) using the external data input of the multiplexer. Therefore, in order to keep track of the current position, the skip register is itself left shifted by the '*increment value*' as shown in Fig. 5. The shifter is implemented as a barrel shifter with 3 bit (0 to 7 places) shift capability.

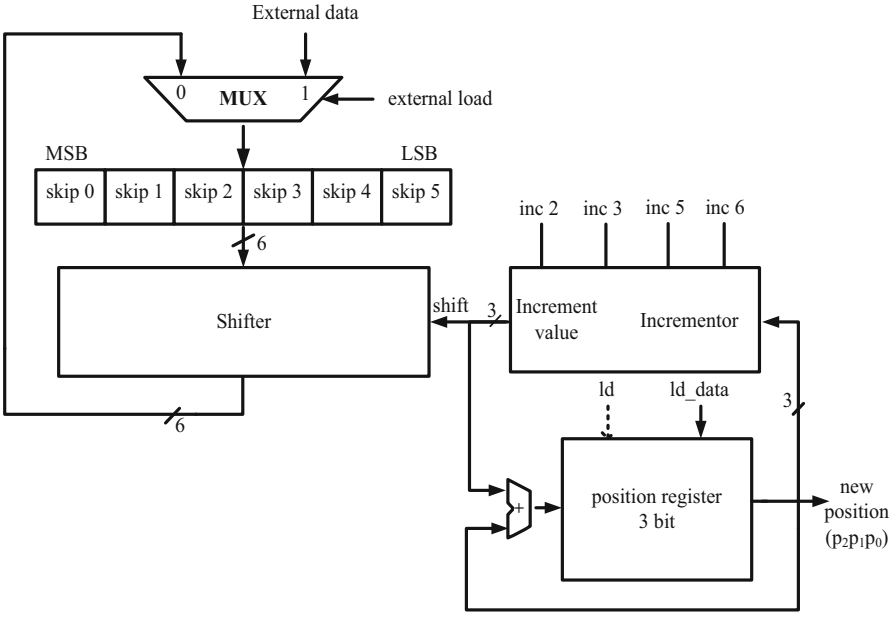


**Fig. 5** Hardware for the generation of new search position

**Table 4** Generation of offset from position

| Position | 5-bit signed offset (row, column) ARP | 5-bit signed offset (row, column) URP |
|---|---|---|
| 0 | (0, 0) | (0, 0) |
| 1 | $(-T_y, 0)$ | $(-1, 0)$ |
| 2 | $(0, T_x)$ | (1, 0) |
| 3 | $(T_y, 0)$ | (0, 1) |
| 4 | $(0, -T_x)$ | $(0, -1)$ |
| 5 | $(-\mathrm{MV}_x, -\mathrm{MV}_y)$ | Don't care |

Offsets are generated with respect to position 0, i.e., the predicted motion vector. Since motion vector is searched in the range − 8 to 7, the offsets are assumed to be 4-bit signed values. Offsets are derived from the position value as shown in Table 4.

### 3.2 Memory Organization

Two buffers each of size 32 × 32 bytes have been used to store the search window. As the search range is [− 8 to 7], 32 × 32 block of reference frame pixels is sufficient to store the possible search space for a 16 × 16 current frame macroblock. The buffers implemented as FPGA block RAM, work in ping pong style, i.e., while one is written to, the other is read from. The current frame buffer, on the other hand, contains 16 × 16 block of pixels. Both the current and the reference frame buffers are implemented as

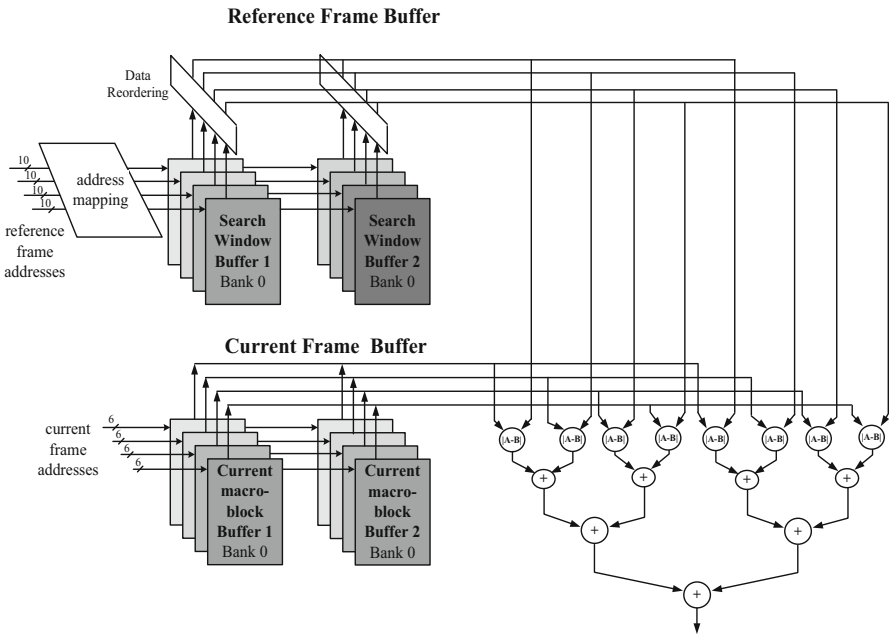


**Fig. 6** The current and the reference frame buffer arrangement

dual-port memories interleaved 4 ways with a view to increase the memory throughput. Since the proposed block matching architecture requires only contiguous memory accesses, it is possible to store the columns of the video frames in interleaved memory. The buffer sizes have been chosen as $32 \times 32$, a power of 2 so that memory address is 10 bits long and the upper 5 bits correspond to row index whereas the lower 5 bits correspond to the column index. Therefore, given any two-dimensional (row, column) pixel address, its location can be translated to the linear memory address without incurring any costs at all. The buffer is written from an external memory as shown in top-level diagram in Fig. 3. Eight bytes, i.e., 64 bits of data are read from the external memory every clock cycle during the writing phase of operation.

Figure 6 shows the arrangement of the interleaved memory blocks along with the tree-like adder arrangement. Since reference frame addresses are generated independent of the interleaved memory arrangement, there is a need to assign the correct address to the memory bank. The address mapping and the data reordering blocks shown in Fig. 6 have been used for this purpose. The address mapping logic assigns the correct addresses to the individual memory banks following simple shifting logic. The data re-ordering block is similar logic entity which ensures that the data values are synchronized with the input address. The current frame block, however, does not involve address mapping logic since the current frame addresses are always synchronized with the arrangement of the memory banks. As shown in Fig. 6, each buffer is composed of four dual-port memory banks in order to enhance the memory throughput. There is a 2:1 multiplexer to choose between the banks.

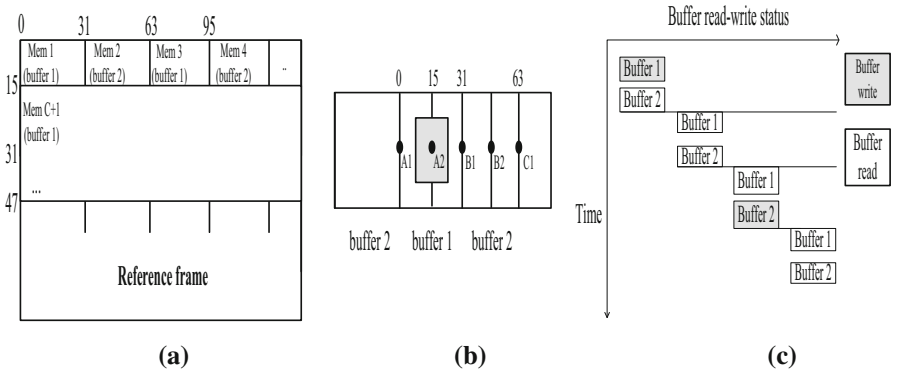


**Fig. 7** Mapping search window to search buffers. **a** The arrangement of reference frame pixels into chunks of 32 × 32 blocks. **b** Search order of macroblocks of size 16 × 16. **c** The buffer read-write sequence

Every macroblock has a unique 32 × 32 search window. However, there is significant overlap between two consecutive search windows. For 16 × 16 search blocks having search window size of 32 × 32, there would be an overlap of 32 × 16 pixels between two consecutive search windows in the $x$ (column) direction. In the proposed memory organization, the reference frame buffer is loaded with the consecutive columns of reference frame values thereby avoiding the need to re-load the common pixels.

However, overlapping pixels among neighboring search windows in the $y$ (row) direction could not be reused, since the macroblocks are searched in a raster scan order. The search window is loaded from the external memory at the beginning of a new row. The address generation module views the reference frame buffers together as 32 × 64 memory space. Therefore, a 5 bit row and 6 bit column address is generated to address the buffer memory space. The most significant bit (MSB) of the row address chooses the search buffer. The generated 11 bit search window address belongs to search buffer 1 if the row MSB is logical low otherwise it belongs to search buffer 2.

In order to process large video frames, the reference frame is divided into sequences of 32 × 32 memory blocks marked as Mem 1, Mem 2 etc shown in Fig. 7a. These memory blocks are alternatively loaded into the search window buffers, for, e.g., Mem 1 is written to buffer 1, while Mem 2 is mapped to buffer 2, Mem 3 is written to buffer 1 again and so on. Each such memory block is associated with the search window of a 16 × 16 macroblock. Points marked as A1, A2, B1, B2 etc shown in Fig. 7b represent a (16 × 16) macroblock centered on the dots. It can be observed that only the 32 × 32 search window corresponding to macroblocks lying in the middle such as A2, B2 lie entirely in a single buffer memory, i.e, buffer 1 or buffer 2.

Interestingly, for macroblocks on the boundary such as A1, B1 and C1, data from both the buffers are required as the search window is shared between the two buffer memories. Therefore, buffer 2 is written while the macroblock A2 is being processed such that both the buffers are available when B1 is being searched. Figure 7c further elucidates the buffer read-write sequences.

A higher search window translates to a larger buffer. Doubling the search range in both $x$ and $y$ directions typically quadruples the size of the buffer memories required

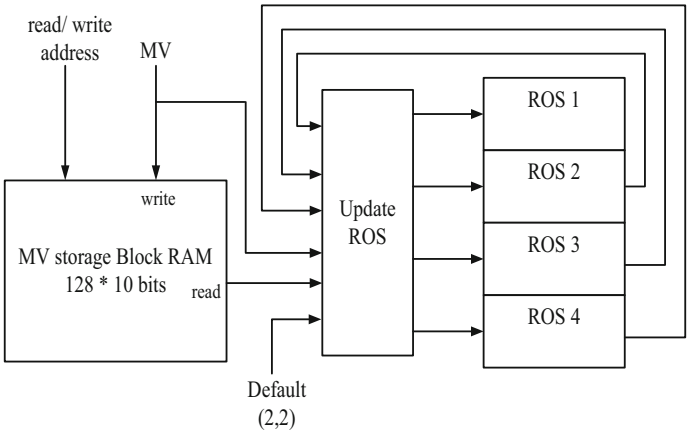


**Fig. 8** Motion Vector Storage

to hold the search window. Since the proposed architecture uses a ping-pong buffer arrangement, a larger buffer implies more time to load buffer from external memory. The ARPS2 search algorithm, on the other hand, typically converges quickly within 2 to 3 iterations, where each iteration refers the searching of a full rood pattern points. As it is always efficient to keep the ME engine busy all the time, this imposes higher bandwidth requirements on the external DRAM to on-chip buffer interfaces. Ideally, it is desired that the ME engine becomes the bottleneck in achieving higher throughput. Use of the larger buffers not only demands larger bandwidth from external memory, but it also contributes to increased hardware area and dynamic power consumption. A search window of $[-8, 7]$ was chosen to demonstrate the effectiveness of the proposed hardware without imposing high bandwidth constraints on external memory.

### 3.3 Motion Vector Storage

The computation of the rood arm lengths requires the motion vector values of the four Region of Support (ROS) blocks. The motion vectors are stored in on-chip block RAM having 128 locations each 10 bits wide. The ROS for two consecutive macroblocks have three MVs in common. Block E becomes the new A whereas block B becomes the new block D and C becomes the new B for the ROS of next macroblock whose best match is to be found. Therefore, only one memory access is required to fetch the MV values of the new C block. There are 128 locations in the Block RAM to accommodate for 128 MV locations required for processing of frames of size $1980 \times 1020$ or higher. Figure 8 shows the hardware arrangement for storing the MVs. The 'update ROS' logic block performs a shifting operation to assign the appropriate values of the MVs to the ROS registers.

### 3.4 Pixel Address Generation

The architecture addresses 8 pixels in chunks of $2 \times 4$ blocks per clock cycle. Therefore, to address a $16 \times 16$ block, two bit counter for the column index and three bit counter for the row index are used.

**Table 5** Equivalent gate count

| Components | Equivalent gates |
|---|---|
| NOR2, NAND2 | 1.0 |
| 2:1 MUX (1 bit) | 3.0 |
| Full adder (1 bit) | 11.0 |
| DFF (1 bit) | 2.0 |
| Dual port RAM (1 bit) | 2.0 |

**Table 6** FPGA resource usage

| FPGA resource | Used | Available | Utilization % |
|---|---|---|---|
| No of slice registers | 93 | 93,120 | 0.10 |
| No of slice LUTs | 299 | 46,560 | 0.64 |
| No of fully used LUT-FF pair used | 72 | 320 | 22.50 |
| No of bonded input output buffers (IOBs) | 97 | 240 | 40.42 |
| No of block RAM | 9 | 156 | 5.78 |

## 4 Results

The architecture has been implemented in verilog HDL and mapped to 45 nm FPGA technology. Simulation results have been presented in this section to compute the throughput of the system and the gate count of the architecture.

### 4.1 Hardware Evaluation

This subsection discusses the area requirement of the proposed design. In order to compute the number of gates required by the proposed architecture, the methodology followed by [2,12] is adopted wherein a two input NAND gate is considered one equivalent gate. Considering the accepted methodology of [2,12], an equivalent gate count is computed for some of the basic hardware modules. Table 5 presents the gate equivalent of some of the basic architectural units in CMOS technology.

The synthesis results from the HDL compiler yields the FPGA resource usage and the macroblock statistics. Table 6 shows the uses of the target FPGA resources whereas Table 7 shows the gate requirements for the various modules of the proposed architecture.

### 4.2 Frame Throughput

The critical path of the architecture obtained after post-placement and routing simulation is found to be 7.990 ns. The minimum input arrival time before clock is found to be 1.709 ns whereas the maximum output time after positive edge of clock is found to be 7.923 ns. Therefore, the architecture can be operated at a maximum frequency

**Table 7** Hardware requirements of the architecture

| Components | Equivalent gates |
|---|---|
| *Datapath and controller* | |
| Adders/subtractors | 3870 |
| Decoder | 15 |
| Comparators | 790 |
| Multiplexer | 1152 |
| Registers | 1034 |
| Equivalent gate count of datapath and controller | 6861 |
| *Memory units* | |
| Frame storage buffer RAM (32 × 32 bytes) × 2 | 32.8 K |
| Current frame storage RAM (16 × 16 bytes) × 2 | 8.2 K |
| MV storage RAM (128 bytes) | 1.64 K |
| Equivalent gate count of the memory units | 42.64 K |
| Total gate count | 49.50 K |

**Table 8** Clock cycles requirements

| Sequences | Cycles per MB |
|---|---|
| Foreman | 177 |
| Coastguard | 147 |
| Tennis | 217 |
| News | 88 |
| Mother and daughter | 159 |
| Mobile | 130 |
| Highway | 204 |
| Soccer | 358 |
| Stefan | 191 |

of 120 MHz. The average number of search points comes out to be 8.1. However, the more significant parameter is the number of clock cycles required to process a frame. Table 8 shows the average number of clock cycles required to process a macroblock. It takes an average of 186 clock cycles to find the best match.

Without the SAD check, it would have taken $8.1 \times (16 \times 16)/8 = 236.8$ cycles theoretically. Therefore, the proposed architecture is able to save around 29% of clock cycles on an average. The architecture operating at 120 MHz frequency is able to process the 645K number of $16 \times 16$ macroblocks per second on an average. Therefore, a throughput of 79 full HD ($1920 \times 1080$) frames per second can be achieved. The FPGA post-layout power consumption is calculated using a gate-level netlist and switching activity file generated from the verilog test bench. The dynamic power consumption is reported to be 39 mW at a frequency of 100 MHz which is sufficient to process 60 full HD frames per second.

The proposed design is not fundamentally limited by architectural units to process full HD resolutions of $1920 \times 1080$. From an architectural standpoint, it has the ability to support full HD resolution of $1920 \times 1080$ as shown in Fig. 7. The architecture uses two on-chip buffers which work in a ping-pong style, i.e, one is read from while the other is written to. This allows the mapping of the entire reference frame into buffers. However, from an algorithmic standpoint, the proposed design uses a search window $[-8, 7]$ which achieves PSNR close to that obtained from Full Search usage. The paper aims to leverage the speed of ARPS2 algorithm to arrive at a compact, lightweight hardware while achieving significantly high throughput at the same time. However, the architecture is quite flexible and the search window can be scaled to higher values, for example $[-32, 31]$. As discussed earlier, using a larger buffer might mean that the frame loading from external memory will become the critical path as opposed to the Motion Estimation engine pipeline.

### 4.3 Comparison

Table 9 compares the hardware performance metrics of the proposed architecture with the existing architectures of some of the standard ME algorithms. Comparisons are done with VLSI architectures executing a varied range of ME algorithms including the well-known FS, TSS and DS and their variants. A parameter called the AT metric [2,12] which is the product of Hardware Area (A) and Clock cycles (T) required for a macroblock has been used for comparison. To demonstrate the hardware efficiency, another metric S/A has been defined which denotes the number of macroblocks processed per second per gate. Table 10 on the other hand compares the performance of the proposed design with existing FPGA-based architectures. Similarly, an efficiency metric S/L has been defined which quantifies the number of macroblocks that can be processed per second by a single LUT.

It is evident from Tables 9 and 10 that the architecture presented in this paper delivers a better performance. By avoiding the use of intra/inter-level architectures, the hardware area has been reduced significantly. The proposed architecture stands ahead of others in terms of the AT metric. Also, it exhibits higher S/A value as compared to the existing architectures. A higher S/A metric is an indication of efficient hardware utilization. Apart from speed and area, our architecture also demonstrates power efficiency as the act of interleaving the memory allows the operating frequency to be scaled down for a given target speed. The FPGA post-route power analyzer indicated that the architecture consumes 15 mW dynamic power while operating at 40 MHz which is sufficient to process 30 720p HD ($1280 \times 720$) frames per second. Hence, the proposed architecture can also be used in low power portable applications as well.

The architecture proposed by Ndili et al. [13], however, exhibits better throughput but it takes up large hardware area (268K) and dissipates higher power as compared to the proposed design. Although the architecture implemented by Tsai [17] covers a large search range ($48 \times 32$), it demonstrates a lesser hardware efficiency as compared to the proposed architecture on account of using 256 PE-based architecture. Chat2 [4] demonstrates an inter-level architecture using 4 PEs, but the hardware requirement is significantly higher as compared to the proposed design, primarily due to

**Table 9** Performance comparison with existing ME architectures

| Architecture | Algorithm | Cycles per MB(T) | Area (A) | Throughput (S) | AT metric | S/A metric |
|---|---|---|---|---|---|---|
| Proposed | ARPS2 | 186 | 49.50 K | 645 K | 9.21 K | 13.03 |
| Ndili [13] | Hardware modified DS | 270 | 268 K | 913 K | 72.36 K | 3.39 |
| Tseng [18] | Variable block TSS | 774 | 50.8 K | 486 K | 39.32 K | 9.57 |
| Kim [8] | Variable block full search | 4111 | 39 K | 101 K | 127.44 K | 2.59 |
| Wei [19] | Variable block size ME | 1129 | 160 K | 177 K | 180.64 K | 1.106 |
| Ding [5] | Diamond search | 235 | 48 K | < 567 K | 11.28 K | 11.81 |
| Yin [20] | Multi-resolution search | 384 | 382 K | 248 K | 146.69 K | 0.65 |
| Tsai [17] | Dedicated algorithm | 1614 | 191 K | 124 K | 308.27 K | 0.65 |
| Chat2 [4] | Fast 2 stage search | 350 | 457.5 K | 368 K | 160.13 K | 0.80 |

**Table 10** Comparison with FPGA architectures

| Architecture | Algorithm | Number of slices | Number of LUT (L) | Number of BRAMs | Throughput (S) Macroblocks/s | Macroblocks/s per LUT (S/L) |
|---|---|---|---|---|---|---|
| Proposed | ARPS2 | – | 299 | 9 | 645 K | 2.15 K |
| Ndili [13] | Modified DS | 11,400 | 18,700 | 129 | 908 K | 0.05 K |
| Akin [1] | Dedicated | 4320 | 14,067 | 16 | 2.25 M | 0.16 K |
| Mukherjee [12] | HEXBS | – | 1929 | 128 | 135 K | 0.7 K |
| Chat1 [3] | DS | 731 | 1103 | – | 964 K | 0.87 K |
| Rehan [15] | FTS | 1995 | 1558 | 4 | 540 K | 0.35 K |
| El-Ashry [6] | FTS | 2412 | 1777 | 4 | 1.5 M | 0.84 K |

large on-chip memory requirements. Table 10 compares the performance of the proposed architecture with some of the existing architectures implemented on FPGA. The proposed architecture achieves the highest throughput per LUT as compared to the existing FPGA architectures. Akin [1] demonstrates a better throughput but the FPGA resource usage (14067 LUTs) is more than that of the proposed design on account of using 256 PE-based intra-array architecture. Moreover, as a result of using highly parallel architecture, the dynamic power consumption is considerable higher (165 mW) in [1] as compared to the proposed design which dissipates 39 mW for the processing of full HD frames (1980 × 1080). The proposed design is able to process a significantly higher number of macroblocks per second as compared to Mukherjee et al. [12] because of the use of interleaved memory architecture. Chat1 [3] demonstrates high throughput in FPGA by employing fast 1 bit transformed ME. It consumes 59 mW of power whereas the proposed architecture executing ARPS2 consumes only 15 mw of power for processing of SDTV frames (1280 × 720). Therefore, the proposed design demonstrates a good trade-off between speed, area and power.

## 5 Conclusions

The proposed architecture in this paper implements a highly adaptive Motion Estimation algorithm to achieve high speed and optimized area. The design presented here addresses all the challenges of implementing an adaptive algorithm like ARPS2 in hardware. The design methodology can also be adopted for implementing other adaptive ME algorithms. The proposed design is able to compute the ME in the just 186 clock cycles in the average case. Owing to the high speed of the adaptive search, frame throughput of 60 SHD (1920 × 1080) frames is achieved at the cost of using small hardware area. The paper also presents a novel interleaved memory arrangement to speed up the computation and an efficient offset generation mechanism for the adaptive search patterns. The proposed design can be suitably incorporated in high-quality video coding applications like HDTV, video conferencing, smart-phones and tablet computers.

## References

1. A. Akin, M. Cetin, Z. Ozcan, B. Erbagci, I. Hamzaoglu, An adaptive bilateral Motion Estimation algorithm and its hardware architecture. IEEE Trans. Consum. Electron. **58**(2), 712–720 (2012)
2. B. Biswas, R. Mukherjee, I. Chakrabarti, Efficient architecture of adaptive rood pattern search technique for fast motion estimation. Microprocess. Microsyst. **39**(3), 200–209 (2015)
3. S.K. Chatterjee, I. Chakrabarti, Low power VLSI architecture for 1-bit transformation based fast motion estimation. IEEE Trans. Consum. Electron. **56**(4), 2652–2660 (2010)
4. S.K. Chatterjee, I. Chakrabarti, Power efficient Motion Estimation algorithm and architecture based on pixel truncation. IEEE Trans. Consum. Electron. **57**(4), 1782–1790 (2011)
5. Y. Ding, X.L. Yan, Parallel architecture of motion estimation for video format conversion with center biased diamond search, in *International Conference on Information Engineering and Computer Science, 2009. ICIECS 2009*, pp. 1–4 (2009)
6. R. El-Ashry, M. Rehan, H. El-Kamchouchi, F. Gebali, Performance-optimized FPGA implementation for the flexible triangle search block-based motion estimation algorithm, in *Proceeding of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'11)*, pp. 640–643 (2011)

7. Y.S. Jehng, L.G. Chen, T.D. Chiueh, An efficient and simple VLSI tree architecture for motion estimation algorithms. IEEE Trans. Signal Process. **41**(2), 889–900 (1993)
8. J. Kim, T. Park, A novel VLSI architecture for full-search variable block-size motion estimation. IEEE Trans. Consum. Electron. **55**(2), 728–733 (2009)
9. T. Koga, K. Linuma, A. Hirano, T. Ishiguro, Motion-compensated inter frame coding for video conferencing, in *Proceedings of National Telecommunications Conference (NTC'81)*, pp. 3–5 (1981)
10. T. Komarek, P. Pirsch, Array architectures for block motion algorithms. IEEE Trans. Circuits Syst. **6**(10), 1301–1308 (1989)
11. K.K. Ma, G. Qiu, An improved adaptive rood pattern search for fast block-matching motion estimation in JVT/H.26L, in *Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS'03)*, pp. 708–711 (2003)
12. R. Mukherjee, B. Biswas, I. Chakrabarti, P.K. Dutta, S. Sengupta, A.K. Ray, Speed-area optimized VLSI architecture of hexagonal search algorithm for Motion Estimation of $512 \times 512$ frames. Circuits Syst. Signal Process. **36**(2), 640–657 (2017)
13. O. Ndili, T. Ogunfunmi, Algorithm and architecture co-design of hardware-oriented, modified diamond search for fast Motion Estimation in H. 264/AVC. IEEE Trans. Circuits Syst. Video Technol. **21**(9), 1214–1227 (2011)
14. Y. Nie, K.K. Ma, Adaptive rood pattern search for fast block-matching motion estimation. IEEE Trans. Image Process. **11**(12), 1442–1449 (2002)
15. M. Rehan, M. El-Kharashi, P. Agathoklis, F. Gebali, An fpga implementation of the flexible triangle search algorithm for block based motion estimation, in *Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS'06)*, pp. 521–524 (2006)
16. I.E.G. Richardson, *Video codec design: developing image and video compression systems* (John Wiley & Sons, 2002)
17. A.C. Tsai, K. Bharanitharan, J.F. Wang, K.I. Lee, Effective search point reduction algorithm and its vlsi design for HDTV H.264/AVC variable block size Motion Estimation. IEEE Trans. Circuits Syst. Video Technol. **22**(7), 1214–1227 (2012)
18. C. Tseng, Y.T. Lai, M.J. Lee, A VLSI architecture for three-step search with variable block size motion vector, in *Proceedings of IEEE 1st Global Conference on Consumer Electronics (GCCE'12)*, pp. 628–631 (2012)
19. C. Wei, H. Hui, T. Jiarong, L. Jinmei, M. Hao, A high-performance reconfigurable vlsi architecture for vbsme in H.264. IEEE Trans. Consum. Electron. **54**(3), 1338–1345 (2008)
20. H. Yin, D.S. Park, X.Y. Zhang, Buffer structure optimized VLSI architecture for efficient hierarchical integer pixel Motion Estimation. J. Real Time Image Process. **11**(3), 507–525 (2016)
21. C. Zhu, X. Lin, L. Chau, Hexagon-based search pattern for fast block Motion Estimation. IEEE Trans. Circuits Syst. Video Technol. **12**(5), 349–355 (2002)
22. S. Zhu, K.K. Ma, A new diamond search algorithm for fast block-matching Motion Estimation. IEEE Trans. Image Process. **9**(2), 287–290 (2000)