Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 105979, 20 pages
doi:10.1155/2009/105979

Research Article

A Prototyping Virtual Socket System-On-Platform Architecture with a Novel ACQPPS Motion Estimator for H.264 Video Encoding Applications

Yifeng Qiu and Wael Badawy

Department of Electrical and Computer Engineering, University of Calgary, Alberta, Canada T2N 1N4

Correspondence should be addressed to Yifeng Qiu, yiqiu@ucalgary.ca

Received 25 February 2009; Revised 27 May 2009; Accepted 27 July 2009

Recommended by Markus Rupp

H.264 delivers the streaming video in high quality for various applications. The coding tools involved in H.264, however, make its video codec implementation very complicated, raising the need for algorithm optimization, and hardware acceleration. In this paper, a novel adaptive crossed quarter polar pattern search (ACQPPS) algorithm is proposed to realize an enhanced inter prediction for H.264. Moreover, an efficient prototyping system-on-platform architecture is also presented, which can be utilized for a realization of H.264 baseline profile encoder with the support of integrated ACQPPS motion estimator and related video IP accelerators. The implementation results show that ACQPPS motion estimator can achieve very high estimated image quality comparable to that from the full search method, in terms of peak signal-to-noise ratio (PSNR), while keeping the complexity at an extremely low level. With the integrated IP accelerators and optimized techniques, the proposed system-on-platform architecture sufficiently supports the H.264 real-time encoding with the low cost.

1. Introduction

Digital video processing technology is to improve the coding validity and efficiency for digital video images [1]. It involves the video standards and relevant realizations. With the joint efforts of ITU-T VCEG and ISO/IEC MPEG, H.264/AVC (MPEG-4 Part 10) has been built up as the most advanced standard so far in the world, targeting to achieve very high data compression. H.264 is able to provide a good video quality at bit rates which are substantially lower than what previous standards need [2–4]. It can be applied to a wide variety of applications with various bit rates and video streaming resolutions, intending to cover practically almost all the aspects of audio and video coding processing within its framework [5–7].

H.264 includes many profiles, levels and feature definitions. There are seven sets of capabilities, referred to as profiles, targeting specific classes of applications: Baseline Profile (BP) for low-cost applications with limited computing resources, which is widely used in videoconferencing and mobile communications; Main Profile (MP) for broadcasting and storage applications; Extended Profile (XP) for streaming video with relatively high compression capability; High Profile (HiP) for high-definition television applications; High 10 Profile (Hi10P) going beyond present mainstream consumer product capabilities; High 4 : 4 : 2 Profile (Hi422P) targeting professional applications using interlaced video; High 4 : 4 : 4 Profile (Hi444P) supporting up to 12 bits per sample and efficient lossless region coding and an integer residual color transform for RGB video. The levels in H.264 are defined as Level 1 to 5, each of which is for specific bit, frame and macroblock (MB) rates to be realized in different profiles.

One of the primary issues with H.264 video applications lies on how to realize the profiles, levels, tools, and algorithms featured by H.264/AVC draft. Thanks to the rapid development of FPGA [8] techniques and embedded software system design and verification tools, the designers can utilize the hardware-software (HW/SW) codesign environment which is based on the reconfigurable and programmable FPGA infrastructure as a dedicated solution for H.264 video applications [9, 10].

The motion estimation (ME) scheme has a vital impact on H.264 video streaming applications, and is the main function of a video encoder to achieve image compression. The block-matching algorithm (BMA) is an important and widely used technique to estimate the motions of regular block, and generate the motion vector (MV), which is the critical information for temporal redundancy reduction in video encoding. Because of its simplicity and coding efficiency, BMA has been adopted as the standard motion estimation method in a variety of video standards, such as the MPEG-1, MPEG-2, MPEG-4, H.261, H.263, and H.264. Fast and accurate block-based search techniques and hardware acceleration are highly demanded to reduce the coding delay and maintain satisfied estimated video image quality. A novel adaptive crossed quarter polar pattern search (ACQPPS) algorithm and its hardware architecture are proposed in this paper to provide an advanced motion estimation search method with the high performance and low computational complexity.

Moreover, an integrated IP accelerated codesign system, which is constructed with an efficient hardware architecture, is also proposed. With integrations of H.264 IP accelerators into the system framework, a complete system-on-platform solution can be set up to realize the H.264 video encoding system. Through the codevelopment and co-verification for system-on-platform, the architecture and IP cores developed by designers can be easily reused and therefore transplanted from one platform to others without significant modification [11]. These factors make a system-on-platform solution outperform a pure software solution and more flexible than a fully dedicated hardware implementation for H.264 video codec realizations.

The rest of paper is organized as follows: in the next Section 2, H.264 baseline profile and its applications are briefly analyzed. In Section 3, the ACQPPS algorithm is proposed in details, while Section 4 describes the hardware architecture for the proposed ACQPPS motion estimator. Furthermore, a hardware architecture and host interface features of the proposed system-on-platform solution is elaborated in Section 5, and the related techniques for system optimizations are illustrated in Section 6. The complete experimental results are generated and analyzed in Section 7. The Section 8 concludes the paper.

2. H.264 Baseline Profile

2.1. General Overview. The profiles and levels specify the conformance points, which are designed to facilitate the interoperability between a variety of video applications of the H.264 standard that has similar functional requirements. A profile defines a set of coding tools or algorithms that can be utilized in generating a compliant bitstream, whereas a level places constraints on certain key parameters of the bitstream.

H.264 baseline profile was designed to minimize the computational complexity and provide high robustness and flexibility for utilization over a broad range of network environment and conditions. It is typically regarded as the simplest one in the standard, which includes all the H.264 tools with the exception of the following tools: B-slices, weighted prediction, field (interlaced) coding, picture/macroblock adaptive switching between the frame and field coding (MB-AFF), context adaptive binary arithmetic coding (CABAC), SP/SI slices and slice data partitioning. This profile normally targets the video applications with low computational complexity and low delay requirements.

For example, in the field of mobile communications, H.264 baseline profile will play an important role because the compression efficiency is doubled in comparison with the coding schemes currently specified by the H.263 Baseline, H.263+ and MPEG-4 Simple Profile.

2.2. Baseline Profile Bitstream. For mobile and videoconferencing applications, H.264 BP, MPEG-4 Visual Simple Profile (VSP), H.263 BP, and H.263 Conversational High Compression (CHC) are usually considered. Practically, H.264 outperforms all other considered encoders for video streaming encoding. H.264 BP allows an average bit rate saving of about 40% compared to H.263 BP, 29% to MPEG-4 VSP and 27% to H.263 CHC, respectively [12].

2.3. Hardware Codec Complexity. The implementation complexity of any video coding standard heavily depends on the characteristics of the platform, for example, FPGA, DSP, ASIC, SoC, on which it is mapped. The basic analysis with respect to the H.264 BP hardware codec implementation complexity can be found in [13, 14].

In general, the main bottleneck of H.264 video encoding is a combination of multiple reference frames and large search ranges.

Moreover, the H.264 video codec complexity ratio is in the order of 10 for basic configurations and can grow up to the 2 orders of magnitude for complex ones [15].

3. The Proposed ACQPPS Algorithm

3.1. Overview of the ME Methods. For motion estimation, the full search algorithm (FS) of BMA exhaustively checks all possible block pixels within the search window to find out the best matching block with minimal matching error (MME). It can usually produce a globally optimal solution to the motion estimation, but demand a very high computational complexity.

To reduce the required operations, many fast algorithms have been developed, including the 2D logarithmic search (LOGS) [16], the three-step search (TSS) [17], the new three-step search (NTSS) [18], the novel four-step search (NFSS) [19], the block-based gradient descent search (BBGDS) [20], the diamond search (DS) [21], the hexagonal search (HEX) [22], the unrestricted center-biased diamond search (UCBDS) [23], and so forth. The basic idea behind these multistep fast search algorithms is to check a few of block points at current step, and restrict the search in next step to the neighboring of points that minimizes the block distortion measure.

These algorithms, however, assume that the error surface of the minimum absolute difference increases monotonically as the search position moves away from the global minimum on the error surface [16]. This assumption would be reasonable in a small region near the global minimum, but not absolutely true for real video signals. To avoid trapped in undesirable local minimum, some adaptive search algorithms have been devised intending to achieve the global optimum or sub-optimum with adaptive search patterns. One of those algorithms is the adaptive rood pattern search (ARPS) [24].

Recently, a few of valuable algorithms have been developed to further improve the search performance, such as the Enhanced Predictive Zonal Search (EPZS) [25, 26] and Unsymmetrical-Cross Multi-Hexagon-grid Search (UMHexagonS) [27], which were even adopted by H.264 as the standard motion estimation algorithms. These schemes, however, are not especially suitable for the hardware implementation, as the search principle of these methods is complicated. If the hardware architecture is required for the realization of H.264 encoder, these algorithms are usually not regarded as the efficient solution.

To improve the search performance and reduce the computational complexity as well, an efficient and fast method, adaptive crossed quarter polar pattern search algorithm (ACQPPS), is therefore proposed in this paper.

3.2. Algorithm Design Considerations. It is known that a small search pattern with compactly spaced search points (SP) is more appropriate than a large search pattern containing sparsely spaced search points in detecting small motions [24]. On the contrary, the large search pattern has the advantage of quickly detecting large motions to avoid being trapped into local minimum along the search path and leads to unfavorable estimation, an issue that the small search pattern encounters. It is desirable to use different search patterns, that is, adaptive search patterns, in view of a variety of the estimated motion behaviors.

Three main aspects are considered to improve or speed up the matching procedure for adaptive search methods: (1) type of the motion prediction; (2) selection of the search pattern shape and direction; (3) adaptive length of search pattern. The first two aspects can reduce the number of search points, and the last one is to give more accurate searching result with a large motion.

For the proposed ACQPPS algorithm under H.264 encoding framework, a median type of the predicted motion vector, that is, median vector predictor (MVP) [28], is produced for determining the initial search range. The shape and direction of the search pattern is adaptively selected. The length (radius) of the search arm is adjusted to improve the search. Two main search steps are involved in the motion search: (1) initial search stage; (2) refined search stage. In the initial search stage, some initial search points are selected to obtain an initial MME point. For the refined search, a unit-sized square pattern is applied iteratively to obtain the final best motion vector.

3.3. Shape of the Search Pattern. To determine the following search step according to whether the current best matching point is positioned at the center of search range, a new search pattern is devised to detect the potentially optimal search points in the initial search stage. The basic concept is to pick up some initial points along with the polar (circular) search pattern. The center of the search circles is the current block position.

Under the assumption that the matching error surface has a property of monotonic increasing or decreasing, however, some redundant checking points may exist in the initial search stage. It is obvious that some redundant points are not necessary to be examined under the assumption of unimodal distortion surface. To reduce the number of initial checking points and keep the probability of getting optimal matching points as high as possible, a fractional or quarter polar search pattern is used accordingly.

Moreover, it is known that the accuracy of motion predictor is very important to the adaptive pattern search. To improve the performance of adaptive search, extra related motion predictors can be used other than the initial MVP. The extra motion predictors utilized by ACQPPS algorithm only require an extension and a contraction of the initial MVP that can be easily obtained. Therefore, at the crossing of quarter circle and motion predictors, the search method is equipped with the adaptive crossed quarter polar patterns for efficient motion search.

3.4. Adaptive Directions of the Search Pattern. The search direction, which is defined by the direction of a quarter circle contained in the pattern, comes from the MVP. Figure 1 shows the possible patterns designed, and Figure 2 depicts how to determine the direction of a search pattern. The patterns employ the directional information of a motion predictor to increase the possibility to get the best MME point for the refined search. To determine an adaptive direction of the search pattern, certain rules are obeyed.

(3.4.1) If the predicted MV (motion predictor) = 0, set up an initial square search pattern with a pattern size = 1, around the search center, as shown in Figure 2(a).

(3.4.2) If the predicted MV falls onto a coordinate axis, that is, $\text{PredMV}y = 0$ or $\text{PredMV}x = 0$, the pattern direction is chosen to be E, N, W, or S, as shown in Figures 1(a), 1(c), 1(e), 1(g). In this case, the point at the initial motion predictor is overlapped with an initial search point which is on the N, W, E, or S coordinate axis.

(3.4.3) If the predicted MV does not fall onto any coordinate axis, and $\text{Max}\{|\text{PredMV}y|, |\text{PredMV}x|\} > 2^*\text{Min}\{|\text{PredMV}y|, |\text{PredMV}x|\}$, the pattern direction is chosen to be E, N, W, or S, as shown in Figure 2(b).

(3.4.4) If the predicted MV does not fall onto any coordinate axis, and $\text{Max}\{|\text{PredMV}y|, |\text{PredMV}x|\} \leq 2^*\text{Min}\{|\text{PredMV}y|, |\text{PredMV}x|\}$, the pattern direction is chosen to be NE, NW, SW, or SE, as shown in Figure 2(c).

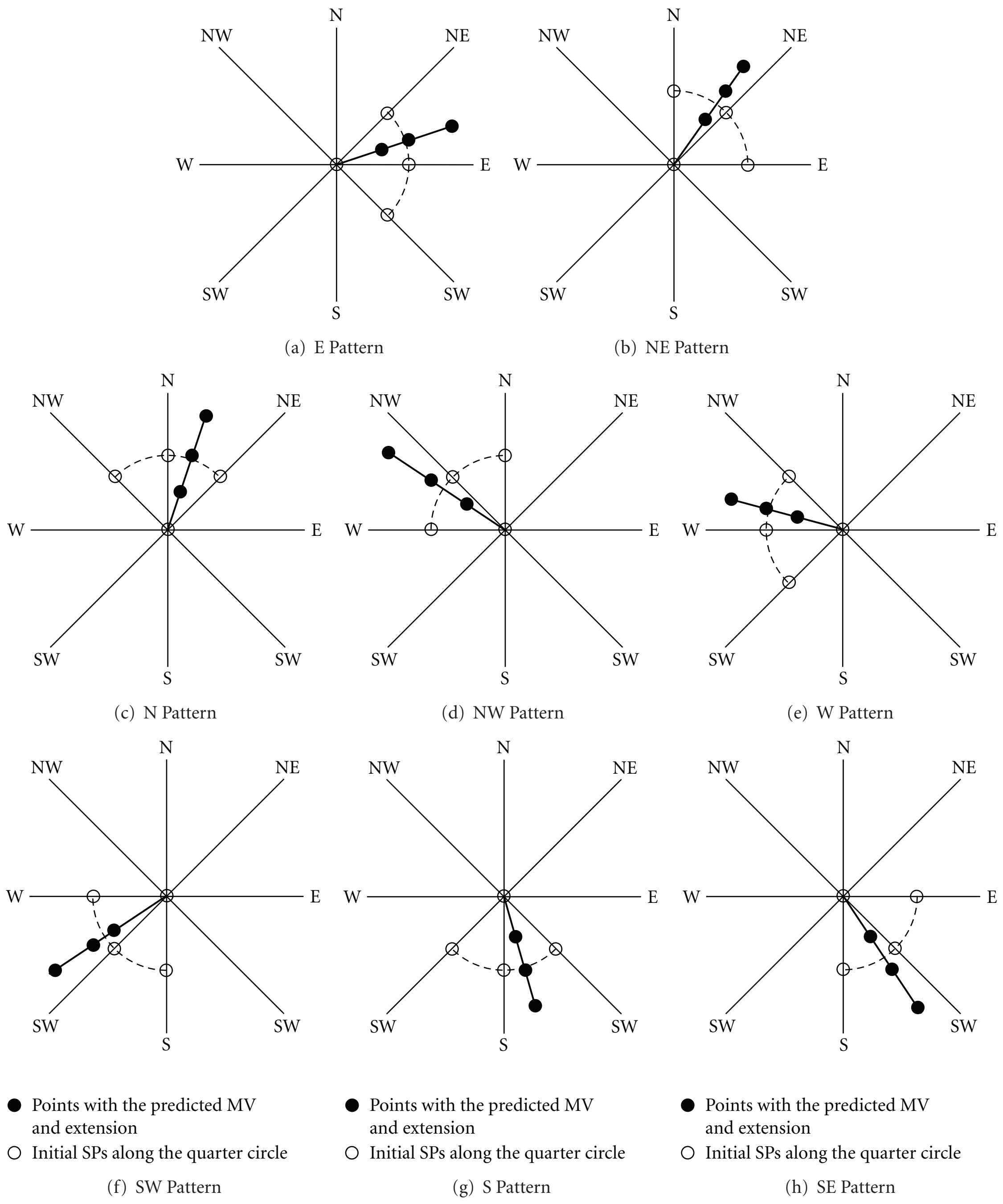


Figure 1: Possible adaptive search patterns designed.

3.5. Size of the Search Pattern. To simplify the selection of search pattern size, the horizontal and vertical components of motion predictor is still utilized. The size of search pattern, that is, the radius of a designed quarter polar search pattern, is simply defined as

$$R = \text{Max}\{|\text{PredMV}y|, |\text{PredMV}x|\}, \quad (1)$$

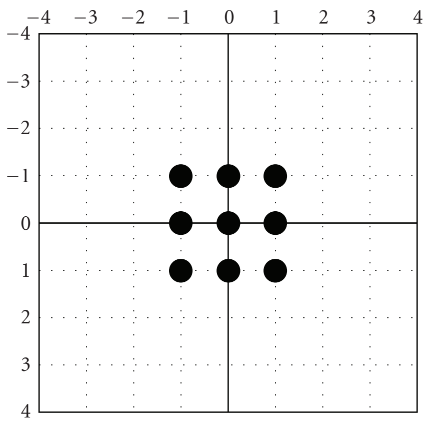
where R is the radius of quarter circle, PredMVy and PredMVx the vertical and horizontal components of the motion predictor, respectively.

3.6. Initial Search Points. After the direction and size of a search pattern are decided, some search points will be selected in the initial search stage. Each search point represents a block to be checked with intensity matching. The initial search points include (when MVP is not zero):

(1) *the predicted motion vector point;*

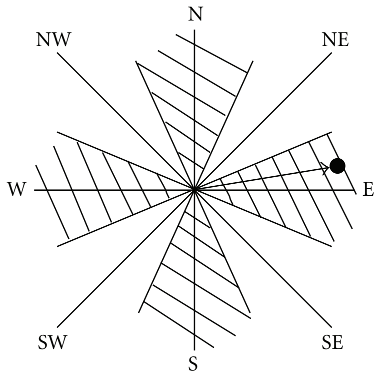
(2) *the center point of search pattern, which represents the candidate block in the current frame;*

(3) *some points on the directional axis;*



● Initial SPs with a square pattern when PredMV = 0

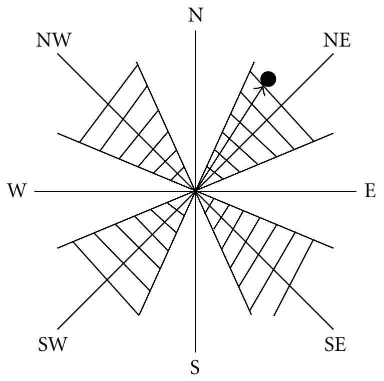
(a)



● Point with the predicted MV

▨ Max{|PredMVy|, |PredMVx|} > 2*Min{|PredMVy|, |PredMVx|} N/E/W/S pattern selected

(b)



● Point with the predicted MV

▨ Max{|PredMVy|, |PredMVx|} ≤ 2*Min{|PredMVy|, |PredMVx|} NW/NE/SW/SE pattern selected

(c)

Figure 2: (a) Square pattern size = 1, (b) N/W/E/S search pattern selected, (c) NW/NE/SW/SE search pattern selected.

Table 1: A look-up table for the definition of vertical and horizontal components of initial search points on NW/NE/SW/SE axis.

R	$\|SP_x\|$	$\|SP_y\|$	R	$\|SP_x\|$	$\|SP_y\|$
0	0	0	6	4	4
1	1	1	7	5	5
2	2	2	8	6	6
3	2	2	9	6	6
4	3	3	10	7	7
5	4	4	—	—	—

(4) *the extension predicted motion vector point (the point with prolonged length of motion predictor), and the contraction predicted motion vector point (the point with contracted length of motion predictor)*

Normally, if no overlapping exists, there will be totally seven search points selected in the initial search stage, in order to get a point with the MME, which can be used as a basis for the refined search stage thereafter.

If a search point is on the axis of NW, NE, SW, or SE, the corresponding decomposed coordinates of that point will satisfy,

$$R = \sqrt{(\mathrm{SP}_x)^2 + \left(\mathrm{SP}_y\right)^2}, \tag{2}$$

where SP_x and SP_y are the vertical and horizontal components of a search point on the axis of NW, NE, SW, or SE. Because $|\mathrm{SP}_x|$ is equal to $|\mathrm{SP}_y|$ in this case, then

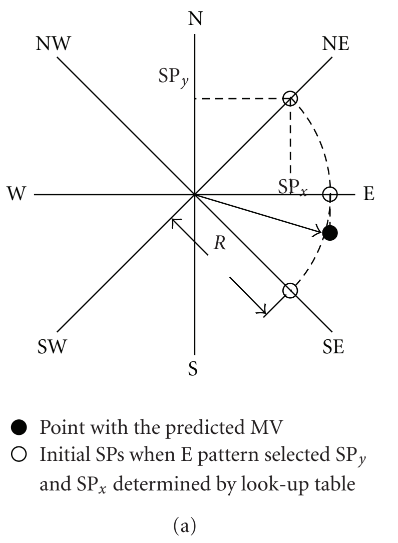
$$R = \sqrt{2} \cdot |\mathrm{SP}_x| = \sqrt{2} \cdot \left|\mathrm{SP}_y\right|. \tag{3}$$

Obviously, neither $|\mathrm{SP}_x|$ nor $|\mathrm{SP}_y|$ is an integer, as R is always an integer-based radius for block processing. To simplify and reduce the computational complexity of a search point definition on the axis of NW, NE, SW or SE, a look-up table (LUT) is employed, as listed in Table 1. The values of SP_x and SP_y are predefined according to the radius R, and now they are integers. Figure 3 illustrates some examples of defined initial search points with the look-up table.

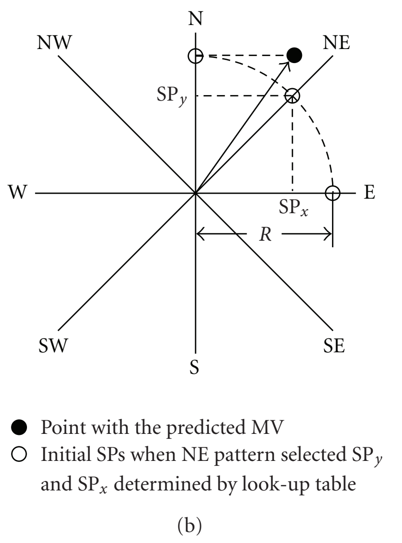
When the radius $R > 20$, the value of $|\mathrm{SP}_x|$ and $|\mathrm{SP}_y|$ can be determined by

$$|\mathrm{SP}_x| = \left|\mathrm{SP}_y\right| = \mathrm{Round}\left(\frac{\mathrm{R}}{\sqrt{2}}\right). \tag{4}$$

There are two initial search points related to the extended motion predictors. One is with a prolonged length of motion predictor (extension version), whereas the other is with a reduced length of motion predictor (contraction version). Two scaled factors are adaptively defined according to the radius R, for the lengths of those two initial search points can be easily derived from the original motion predictor, as shown in Table 2. The scaled factors are chosen so that the initial search points related to the extension and contraction of the motion predictor can be distributed reasonably around the motion predictor point to obtain the better motion predictor points.



N
NW
NE
W
E
SW
SE
S
SP$_y$
SP$_x$
R
● Point with the predicted MV
○ Initial SPs when E pattern selected SP$_y$ and SP$_x$ determined by look-up table
(a)



N
NW
NE
W
E
SW
SE
S
SP$_y$
SP$_x$
R
● Point with the predicted MV
○ Initial SPs when NE pattern selected SP$_y$ and SP$_x$ determined by look-up table
(b)

Figure 3: (a) An example of initial search points defined for E pattern using look-up table; (b) an example of initial search points defined for NE pattern using look-up table.

Table 2: Definition of scaled factors for initial search points related to motion predictor.

R	Scaled factor for extension (SF_E)	R	Scaled factor for contraction (SF_C)
0 ~ 2	3	0 ~ 10	0.5
3 ~ 5	2	>10	0.75
6 ~ 10	1.5		
>10	1.25		

Therefore, the initial search points related to the motion predictor can be identified as

$$\mathrm{EMVP} = \mathrm{SF}_E \cdot \mathrm{MVP}, \quad (5)$$

$$\mathrm{CMVP} = \mathrm{SF}_C \cdot \mathrm{MVP}, \quad (6)$$

where MVP is a point representing the median vector predictor. SF_E and SF_C are the scaled factors for the extension and contraction, respectively. EMVP and CMVP are the initial search points with the prolonged and contracted lengths of predicted motion vector, respectively. If the horizontal or vertical component of EMVP and CMVP is not an integer after the scaling, the component value will be truncated to the integer for video block processing.

3.7. Algorithm Procedure

Step 1. Get a predicted motion vector (MVP) for the candidate block in current frame for the initial search stage.

Step 2. Find the adaptive direction of a search pattern by rules (3.4.1)–(3.4.4), determine the pattern size "R" with the (1), choose initial SPs in the reference frame along the quarter circle and predicted MV using look-up table, (5) and (6).

Step 3. Check the initial search points with block pixel intensity measurement, and get an MME point which has a minimum SAD as the search center for the next search stage.

Step 4. Refine local search by applying unit-sized square pattern to the MME point (search center), and check its neighboring points with block pixel intensity measurement. If after search, the MME point is still the search center, then stop searching and obtain the final motion vector for the candidate block corresponding to the final best matching point identified in this step. Otherwise, set up the new MME point as the search center, and apply square pattern search to that MME point again, until the stop condition is satisfied.

3.8. Algorithm Complexity. As the ACQPPS is a predicted and adaptive multistep algorithm for motion search, the algorithm computational complexity exclusively depends on the object motions contained in the video sequences and scenarios for estimation processing. The main overhead of ACQPPS algorithm lies in the block SAD computations. Some other algorithm overhead, such as the selection of adaptive search pattern direction, the determination of search arm and initial search points, are merely consumed by a combination of if-condition judgments, and thus can be even ignored when compared with block SAD calculations.

If the large, quick, and complex object motions are included in video sequences, the number of search points (NSP) will be reasonably increased. On the contrary, if the small, slow and simple object motions are shown in the sequences, it only requires the ACQPPS algorithm a few of processing steps to finish the motion search, that is, the number of search points is correspondingly reduced.

Unlike the ME algorithms with fixed search ranges, for example, the full search algorithm, it is impractical to precisely identify the number of computational steps for ACQPPS. On an average, however, an approximation

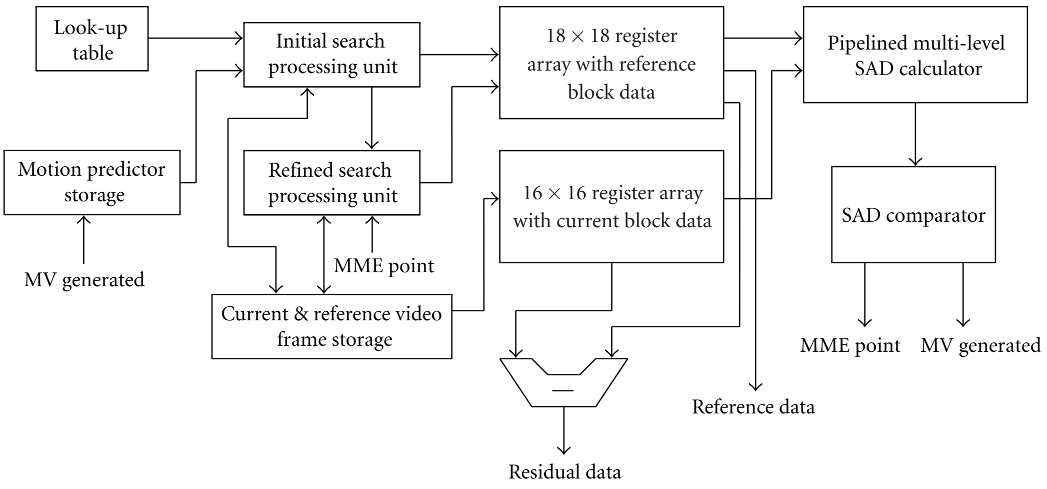

Look-up table
Initial search processing unit
18 × 18 register array with reference block data
Pipelined multi-level SAD calculator
Motion predictor storage
Refined search processing unit
16 × 16 register array with current block data
SAD comparator
MV generated
MME point
Current & reference video frame storage
MME point
MV generated
Reference data
Residual data

Figure 4: A hardware architecture for ACQPPS motion estimator.

equation can be utilized to represent the computational complexity for ACQPPS method. The worst case of motion search for a video sequence is to use the 4 × 4 block size, if the fixed block size is employed. In this case, the number of search points for ACQPPS motion estimation is usually around 12 ∼ 16, according to the practical motion search results. Therefore, the algorithm complexity can be simply identified as, in terms of image size and frame rate,

$$\begin{aligned} C \approx 16 &\times \text{Block SAD computations} \\ &\times \text{Number of blocks in a video frame} \times \text{Frame rate}, \end{aligned} \tag{7}$$

where the block size is 4 × 4 for the worst case of computations. For a standard software implementation, it actually requires 16 subtractions and 15 additions, that is, 31 arithmetic operations, for each 4 × 4 block SAD calculations. Accordingly, the complexity of ACQPPS is approximately 14 and 60 times less than the one required by full search algorithm with the $[-7, +7]$ and $[-15, +15]$ search range, respectively. In practice, the ACQPPS complexity is roughly at the same level as the simple DS algorithm.

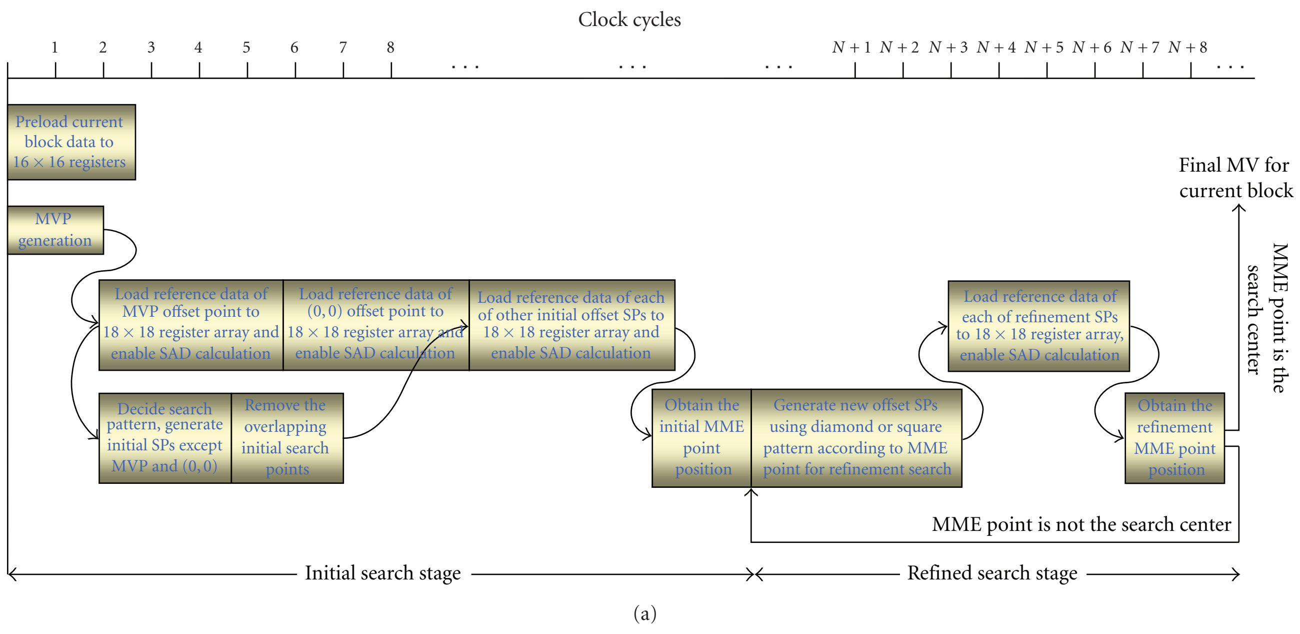
4. Hardware Architecture of ACQPPS Motion Estimator

The ACQPPS is designed with low complexity, which is appropriate to be implemented based on a hardware architecture. The hardware architecture takes advantage of the pipelining and parallel operations of the adaptive search patterns, and utilizes a fully pipelined multilevel SAD calculator to improve the computational efficiency and, therefore, reduce the clock frequency reasonably.

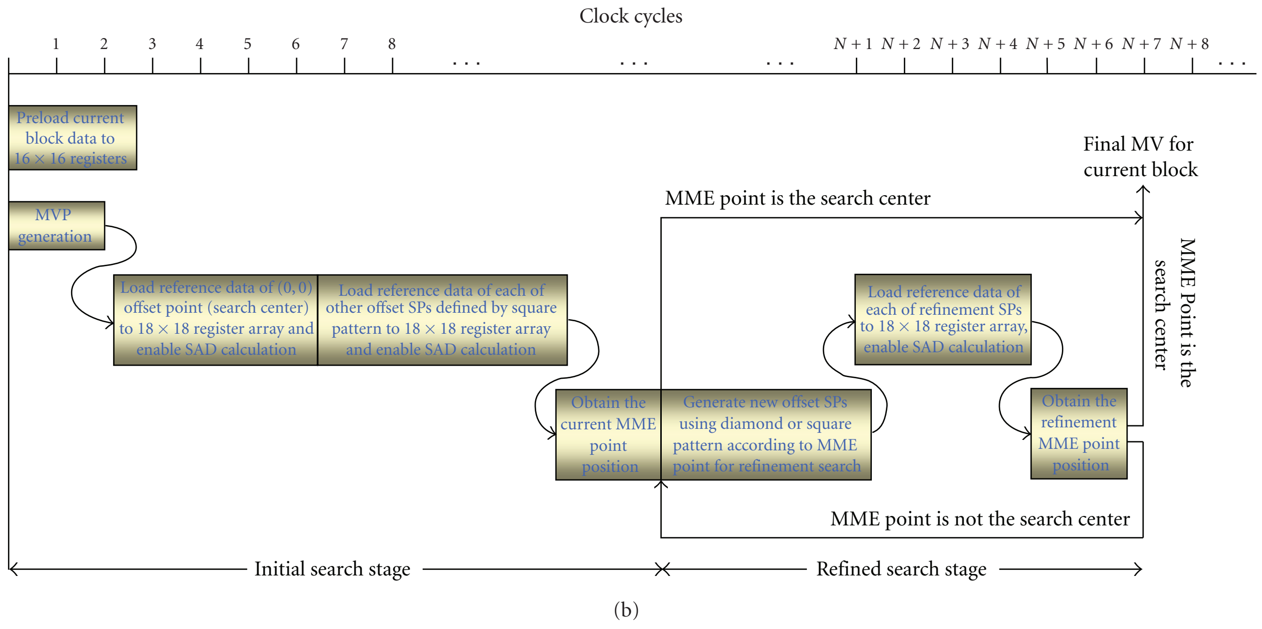
As mentioned above, the computation of motion vector for a smallest block shape, that is, 4 × 4 block, is the worst case for calculation. The worst case refers to the percentage usage of the memory bandwidth. It is necessary that the computational efficiency be as high as possible in the worst case. All of the other block shapes can be constructed from 4 × 4 blocks so that the computation of distortion in 4 × 4 partial solutions and result additions can solve all of the other block shapes.

4.1. ACQPPS Hardware Architecture. An architecture for the ACQPPS motion estimator is shown in Figure 4. There are two main stages for the motion vector search, including the initial and refined search, indicated by the hardware semaphore. In the initial search stage, the architecture utilizes the previously calculated motion vectors to produce an MVP for the current block. Some initial search points are generated utilizing the MVP and LUT to define the search range of adaptive patterns. After an MME point is found in this stage, the search refinement will take into effect applying square pattern around MME points iteratively to obtain a final best MME point, which indicates the final best MV for the current block. For motion estimation, the reference frames are stored in SRAM or DRAM, while the current frame and produced MVs are stored in dual-port memory (BRAM). Meanwhile, The LUT also uses the BRAM to facilitate the generation of initial search points.

Figure 5 illustrates a data search flow of the ACQPPS hardware IP with regard to each block motion search. The initial search processing unit (ISPU) is used to generate the initial search points and then perform the initial motion search. To generate the initial search points, previously calculated MVs and an LUT are employed. The LUT contains the vertical and horizontal components of the initial search points defined in Table 1. Both produced MVs and LUT values are stored in BRAM, for they can be accessed through two independent data ports in parallel to facilitate the processing. When the initial search stage is finished, the refined search processing unit (RSPU) is enabled to work. It employs the square pattern around the MME point derived in initial search stage to refine the local motion search. The local refined search steps might be iteratively performed a few of times, until the MME point is still at the search center


Clock cycles
1 2 3 4 5 6 7 8 ... ... N + 1 N + 2 N + 3 N + 4 N + 5 N + 6 N + 7 N + 8 ...
Preload current block data to 16 × 16 registers
MVP generation
Load reference data of MVP offset point to 18 × 18 register array and enable SAD calculation
Load reference data of (0, 0) offset point to 18 × 18 register array and enable SAD calculation
Load reference data of each of other initial offset SPs to 18 × 18 register array and enable SAD calculation
Decide search pattern, generate initial SPs except MVP and (0, 0)
Remove the overlapping initial search points
Obtain the initial MME point position
Generate new offset SPs using diamond or square pattern according to MME point for refinement search
Load reference data of each of refinement SPs to 18 × 18 register array, enable SAD calculation
Obtain the refinement MME point position
Final MV for current block
MME point is the search center
MME point is not the search center
Initial search stage
Refined search stage

(a)


Clock cycles
1 2 3 4 5 6 7 8 ... ... N + 1 N + 2 N + 3 N + 4 N + 5 N + 6 N + 7 N + 8 ...
Preload current block data to 16 × 16 registers
MVP generation
Load reference data of (0, 0) offset point (search center) to 18 × 18 register array and enable SAD calculation
Load reference data of each of other offset SPs defined by square pattern to 18 × 18 register array and enable SAD calculation
Obtain the current MME point position
Generate new offset SPs using diamond or square pattern according to MME point for refinement search
Load reference data of each of refinement SPs to 18 × 18 register array, enable SAD calculation
Obtain the refinement MME point position
Final MV for current block
MME point is the search center
MME Point is the search center
MME point is not the search center
Initial search stage
Refined search stage

(b)

Figure 5: (a) A data search flow for the individual block motion estimation when MVP is not zero; (b) a data search flow for the individual block motion estimation when MVP is zero. Note The clock cycles for each task are not on the exact timing scale, only for illustration purpose.

after certain refined steps. The search data flow of ACQPPS IP architecture conforms to the algorithm steps defined in Section 3.7, with further improvement and optimization of hardware parallel and pipelining features.

4.2. Fully Pipelined SAD Calculator. As main ME operations are related to SAD calculations that have a critical impact on the performance of hardware-based motion estimator, a fully pipelined SAD calculator is designed to speed up the SAD computations. Figure 6 displays a basic architecture of the pipelined SAD calculator, with the processing support of variable block sizes. According to the VBS indicated by block shape and enable signals, SAD calculator can employ appropriate parallel and pipelining adder operations to generate SAD result for a searched block. With the parallel calculations of basic processing unit (BPU), it can take 4 clock cycles to finish the 4 × 4 block SAD computations (BPU for 4 × 4 block SAD), and 8 clock cycles to produce a final SAD result for a 16 × 16 block.

To support the VBS feature, different block shapes might be processed based on the prototype of the BPU. In such case, a 16 × 16 macroblock is divided into 16 basic 4 × 4 blocks.

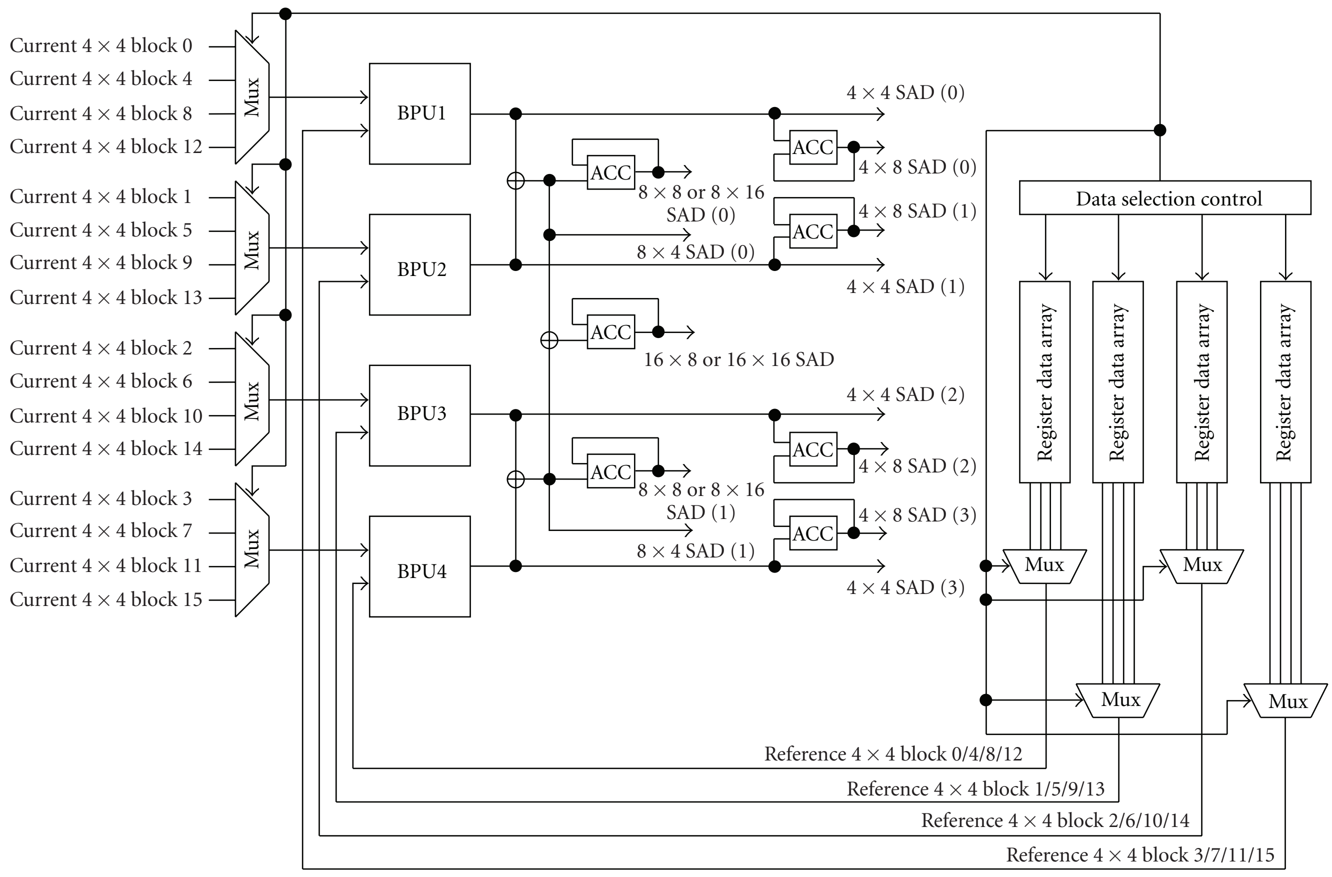
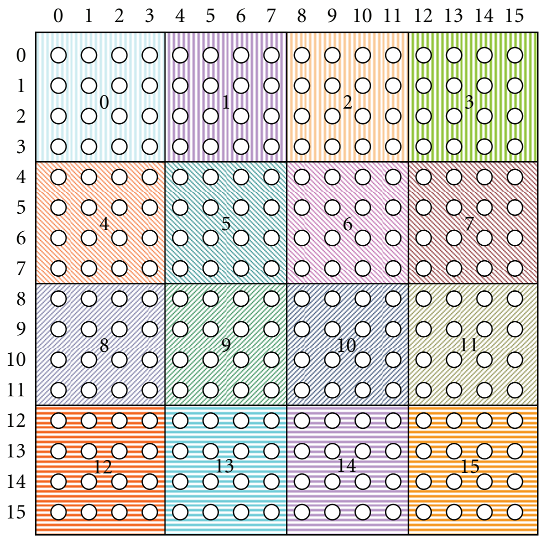


Figure 6: An architecture for pipelined multilevel SAD calculator.



Organization of VBS using 4 × 4 blocks

16 × 16:	{0, 1, . . . , 14, 15}
16 × 8:	{0, 1, . . . , 6, 7}, {8, 9, . . . , 14, 15}
8 × 16:	{0, 1, 4, 5, 8, 9, 12, 13}, {2, 3, 6, 7, 10, 11, 14, 15}
8 × 8:	{0, 1, 4, 5}, {2, 3, 6, 7}, {8, 9, 12, 13}, {10, 11, 14, 15}
8 × 4:	{0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9}, {10, 11}, {12, 13}, {14, 15}
4 × 8:	{0, 4}, {1, 5}, {2, 6}, {3, 7}, {8, 12}, {9, 13}, {10, 14}, {11, 15}
4 × 4:	{0}, {1}, . . . , {14}, {15}

Computing stages for VBS using 4 × 4 blocks

VBS	Stage 1	Stage 2	Stage 3	Stage 4
16 × 16	{0, 1, 2, 3}	{4, 5, 6, 7}	{8, 9, 10, 11}	{12, 13, 14, 15}
16 × 8	{0, 1, 2, 3}/{8, 9, 10, 11}	{4, 5, 6, 7}/{12, 13, 14, 15}	-	-
8 × 16	{0, 1}/{2, 3}	{4, 5}/{6, 7}	{8, 9}/{10, 11}	{12, 13}/{14, 15}
8 × 8	{0, 1}/{2, 3}/ {8, 9}/{10, 11}	{4, 5}/{6, 7}/ {12, 13}/{14, 15}	-	-
8 × 4	{0, 1}/{2, 3}/{4, 5}/{6, 7}/ {8, 9}/{10, 11}/{12, 13}/{14, 15}	-	-	-
4 × 8	{0}/{1}/{2}/{3}/ {8}/{9}/{10}/{11}	{4}/{5}/{6}/{7}/ {12}/{13}/{14}/{15}	-	-
4 × 4	{0}/{1}/. . ./{14}/{15}	-	-	-

Figure 7: Organization of Variable Block Size based on Basic 4 × 4 Blocks.

Other 6 block sizes in H.264, that is, 16 × 16, 16 × 8, 8 × 16, 8 × 8, 8 × 4, and 4 × 8, can be organized by the combination of basic 4 × 4 blocks, shown in Figure 7, which also describes computing stages for each variable-sized block constructed on the basic 4 × 4 blocks to obtain VBS SAD results.

For instance, for a largest 16 × 16 block, it will require 4 stages of the parallel data loadings from the register arrays to the SAD calculator to obtain a final block SAD result. In this case, the schedule of data loading will be {0, 1, 2, 3} → {4, 5, 6, 7} → {8, 9, 10, 11} → {12, 13, 14, 15}, where "{}"

indicates each parallel pixel data input with the current and reference block data.

4.3. Optimized Memory Structure. When a square pattern is used to refine the MV search results, the mapping of the memory architecture is important to speed up the performance. In our design, the memory architecture will be mapped onto a 2D register space for the refined stage. The maximum size of this space is 18×18 with pixel bit depth, that is, the mapped register memory can accommodate a largest 16×16 macroblock plus the edge redundancy for the rotated data shift and storage operations.

A simple combination of parallel register shifts and related data fetches from SRAM can reduce the memory bandwidth, and facilitate the refinement processing, as many of the pixel data for searching in this stage remain unchanged. For example, 87.89% and 93.75% of the pixel data will stay unchanged, when the $(1,-1)$ and $(1,0)$ offset searches for the 16×16 block are executed, respectively.

4.4. SAD Comparator. The SAD comparator is utilized to compare the previously generated block SAD results to obtain a final estimated MV which corresponds to the best MME point that has the minimum SAD with the lowest block pixel intensity. To select and compare the proper block SAD results as shown in Figure 6, the signals of different block shapes and computing stages are employed to determine the appropriate mode of minimum SAD to be utilized.

For example, if the 16×16 block size is used for motion estimation, the 16×16 block data will be loaded into the BPU for SAD calculations. Each 16×16 block requires 4 computing stages to obtain a final block SAD result. In this case, the result mode of "16×8 or 16×16 SAD" will be first selected. Meanwhile, the signal of computing stages is also used to indicate the valid input to the SAD comparator for retrieving proper SAD results from BPU, and thus obtain the MME point with a minimum SAD for this block size.

The best MME point position obtained by SAD comparator is further employed to produce the best matched reference block data and residual data which are important to other video encoding functions, such as mathematical transforms and motion compensation, and so forth.

5. Virtual Socket System-on-Platform Architecture

The bitstream and hardware complexity analysis derived in Section 2 helps guiding both the architecture design for prototyping IP accelerated system and the optimized implementation of an H.264 BP encoding system based on that architecture.

5.1. The Proposed System-On-Platform Architecture. A variety of options, switches, and modes required in video bitstream actually results in the increasing interactions between different video tasks or function-specific IP blocks. Consequently, the functional oriented and fully dedicated architectures will become inefficient, if high levels of the flexibility are not provided in the individual IP modules. To make the architectures remain efficient, the hardware blocks need optimization to deal with the increasing complexity for visual objects processing. Besides, the hardware must keep flexible enough to manage and allocate various resources, memories, computational video IP accelerators for different encoding tasks. In view of that the programmable solutions will be preferable for video codec applications with programmable and reconfigurable processing cores, the heterogeneous functionality and the algorithms can be executed on the same hardware platform, and upgraded flexibly by software manipulations.

To accelerate the performance on processing cores, parallelization will be demanded. The parallelization can take place at different levels, such as task, data, and instruction. Furthermore, the specific video processing algorithms performed by IP accelerators or processing cores can improve the execution efficiency significantly. Therefore, the requirements for H.264 video applications are so demanding that multiple acceleration techniques may be combined to meet the real-time conditions. The programmable, reconfigurable, heterogeneous processors are the preferable choice for an implementation of H.264 BP video encoder. Architectures with the support for concurrent performance and hardware video IP accelerators are well applicable for achieving the real-time requirement imposed by the H.264 standard.

Figure 8 shows the proposed extensible system-on-platform architecture. The architecture consists of a programmable and reconfigurable processing core which is built upon FPGA, and two extensible cores with RISC and DSP. The RISC can take charge of general sequences control and IP integration information, give mode selections for video coding, and configure basic operations, while DSP can be utilized to process the particular or flexible computational tasks.

The processing cores are connected through the heterogeneous integrated onplatform memory spaces for the exchange of control information. The PCI/PCMCIA standard bus provides a data transfer solution for the host connected to the platform framework, reconfigures and controls the platform in a flexible way. Desirable video IP accelerators will be integrated in the system platform architecture to improve the encoding performance for H.264 BP video applications.

5.2. Virtual Socket Management. The concept of virtual socket is thus introduced to the proposed system-on-platform architecture. Virtual socket is a solution for the host-platform interface, which can map a virtual memory space from the host environment to the physical storage on the architecture. It is an efficient mechanism for the management of virtual memory interface and heterogeneous memory spaces on the system framework. It enables a truly integrated, platform independent environment for the hardware-software codevelopment.

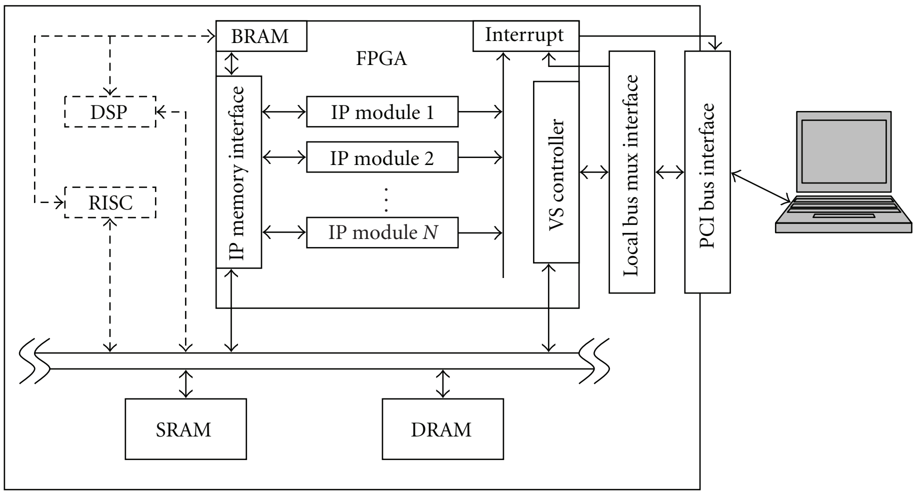


Figure 8: The proposed extensible system-on-platform hardware architecture.

Through the virtual socket interface, a few of virtual socket application programming interface (API) function calls can be employed to make the generic hardware functional IP accelerators automatically map the virtual memory addresses from the host system to different memory spaces on the hardware platform. Therefore, with the efficient virtual socket memory organization, the hardware abstraction layer will provide the system architecture with simplified memory access, interrupt based control and shielded interactions between the platform framework and the host system. Through the integration of IP accelerators to the hardware architecture, the system performance will be improved significantly.

The codesign virtual socket host-platform interface management and system-on-platform hardware architecture actually provide a useful embedded system approach for the realization of advanced and complicated H.264 video encoding system. Hence, the IP accelerators on FPGA, together with the extensible DSP and RISC, construct an efficient programmable embedded solution to perform the dedicated and real-time video processing tasks. Moreover, due to the various video configurations for H.264 encoding, the physically implemented virtual socket interface as well as APIs can easily enable the encoder configurations, data manipulations and communications between the host computer system and hardware architecture, in return facilitate the system development for H.264 video encoders.

5.3. Integration of IP Accelerators. The IP accelerator illustrated here can be any H.264 compliant hardware block which is defined to handle a computationally extensive task for video applications without a specific design for interaction controls between IP and the host. For encoding, the basic modules to be integrated include Motion Estimator, Discrete Cosine Transform and Quantization (DCT/Q), Deblocking Filter and Context Adaptive Variable Length Coding (CAVLC), while Inverse Discrete Cosine Transform and Inverse Quantization (IDCT/Q^{-1}), and Motion Compensation (MC) for decoding. An IP memory interface is provided by the architecture to achieve the integration. All IP modules are connected to the IP memory interface, which provides accelerators a straight way to exchange data between the host and memory spaces. Interrupt signals can be generated by accelerators when demanded. Moreover, to control the concurrent performance of accelerators, an IP bus arbitrator is designed and integrated in the IP memory interface, for the interface controller to allocate appropriate memory operation time for each IP module, and avoid the memory access conflicts possibly caused by heterogeneous IP operations.

IP interface signals are configured to connect the IP modules to the IP memory interface. It is likely that each accelerator has its own interface requirement for interaction between the platform and IP modules. To make the integration easy, it is required that certain common interface signals be defined to link IP blocks and memory interface together. With the IP interface signals, the accelerators will focus on their own computational tasks, and thus the architecture efficiency can be improved. Practically, the IP modules can be flexibly reused, extended, and migrated to other independent platforms very easily. Table 3 defines the necessary IP interface signals for the proposed architecture. IP modules only need to issue the memory requests and access parameters to the IP memory interface, and rest of the tasks are taken by platform controllers. This feature is especially useful when motion estimator is integrated in the system.

5.4. Host Interface and API Function Calls. The host interface provides the architecture with necessary data for video processing. It can also control video accelerators to operate in sequential or parallel mode, in accordance with the H.264 video codec specifications. The hardware-software partitioning is simplified so that the host interface can focus on the data communication as well as flow control for video tasks, while hardware accelerators deal with local memory

Table 3: IP interface signals.

Interface signals	Description
Clk, reset, start	Platform signals for IP
Input_Valid, Output_Valid	Valid strobes for IP memory access
Data_In, Data_Out	Input and output memory data for IP
Memory_Read	IP request for memory read
Mem_HW Accel, offset, count	IP number, offset, and data count provided by IP/Host for memory read
Mem_HW Accel1, offset1, count1	IP Number, offset, and data count provided by IP/Host for memory write
Mem_Read_Req Mem_Write_Req	IP bus request for memory access
Mem_Read_Release_Req Mem_Write_Release_Req	IP bus release request for memory access
Mem_Read_Ack Mem_Write_Ack	IP bus request grant for memory access
Mem_Read_Release_Ack Mem_Write_Release_Ack	IP bus release grant for memory access
Done	IP interrupt signal

accesses and video codec functions. Therefore, the software abstraction layer covers the feature of data exchange and video task flow control for hardware performance.

A set of related virtual socket API functions is defined to implement the host interface features. The virtual socket APIs are software function calls coded in C/C++, which perform data transfers and signal interactions between the host and hardware system-on-platform. The virtual socket API as a software infrastructure can be utilized by a variety of video applications to control the implementation of hardware feature defined. With virtual socket APIs, the manipulation of video data in local memories can be executed conveniently. Therefore, the efficiency of hardware and software interactions can be kept high.

6. System Optimizations

6.1. Memory Optimization. Due to the significant memory access requirement for video encoding tasks, a large amount of clock cycles is consumed by the processing core while waiting for the data fetch from local memory spaces. To reduce or avoid the overhead of memory data access, the memory storage of video frame data can be organized to utilize multiple independent memory spaces (SRAM and DRAM) and dual-port memory (BRAM), in order to enable the parallel and pipelined memory access during the video encoding. This optimized requirement can practically provide the system architecture with the multi-port memory storage to reduce the data access bandwidth for each of the individual memory space.

Furthermore, with the dual-port data access, DMA can be scheduled to transfer a large amount of video frame data through PCI bus and virtual socket interface in parallel with the operations of encoding tasks, so that the processing core will not suffer memory and encoding latency. In such case, the data control flow of video encoding will be managed to make the DMA transfer and IP accelerator operations in fully parallel and pipelined stages.

6.2. Architecture Optimization. As the main video encoding functions (such as ME, DCT/Q, IDCT/Q^{-1}, MC, Deblocking Filter, and CAVLC) can be accelerated by IP modules, the interconnection between those video processing accelerators has an important impact on the overall system performance. To make the IP accelerators execute main computational encoding routines in full parallel and pipelining mode, the IP integration architecture has to be optimized. A few of caches are inserted between the video IP accelerators to facilitate the encoding concurrent performance. The caches can be organized as parallel dual-port memory (BRAM) or pipelined memory (FIFO). The interconnection control of data streaming between IP modules will be defined using those caches targeting to eliminate the extra overhead of processing routines, for encoding functions can be operated in full parallel and pipelining stages.

6.3. Algorithm Optimization. The complexity of encoding algorithms can be modified when the IP accelerators are shaping. This optimization can be taken after choosing the most appropriate modes, options, and configurations for the H.264 BP applications. It is known that the motion estimator requires the major overhead for encoding computations. To reduce the complexity of motion estimation, a very efficient and fast ACQPPS algorithm and corresponding hardware architecture have been realized based on the reduction of spatio-temporal correlation redundancy. Some other algorithm optimizations can also be executed. For example, a simple algorithm optimization may be applied to mathematic transform and quantization. As many blocks tend to have minimal residual data after the motion compensation, the mathematic transform and quantization for motion-compensated blocks can be ignored, if SAD of such blocks is lower than a prescribed threshold, in order to facilitate the processing speed.

The application of memory, algorithm, and architecture optimizations combined in the system can meet the major challenges for the realization of video encoding system. The optimization techniques can be employed to reduce the encoding complexity and memory bandwidth, with the well-defined parallel and pipelining data streaming control flow, in order to implement a simplified H.264 BP encoder.

6.4. An IP Accelerated Model for Video Encoding. An optimized IP accelerated model is presented in Figure 9 for a realization of simplified H.264 BP video encoder. In this architecture, BRAM, SRAM, and DRAM are used as multi-port memories to facilitate video processing. The current video frame is transferred by DMA and stored in BRAM. Meanwhile, IP accelerators fetch the data from BRAM and

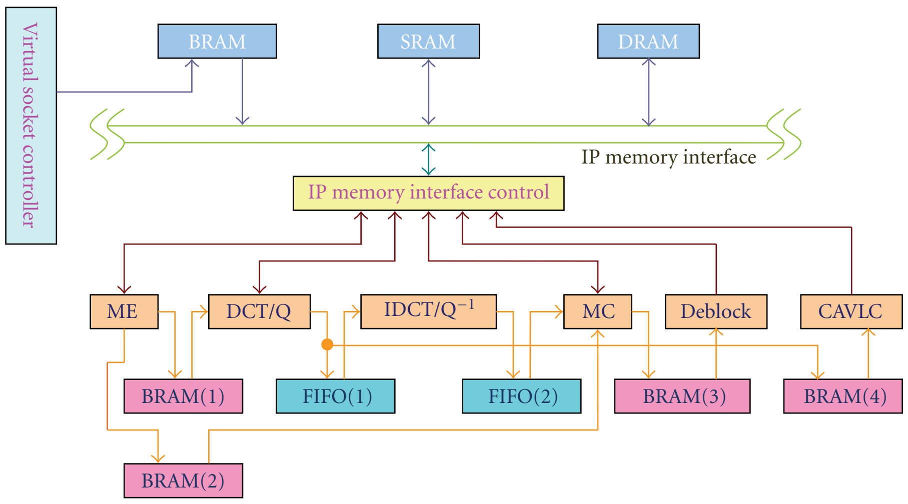


FIGURE 9: An optimized architecture for simplified H.264 BP video encoding system.

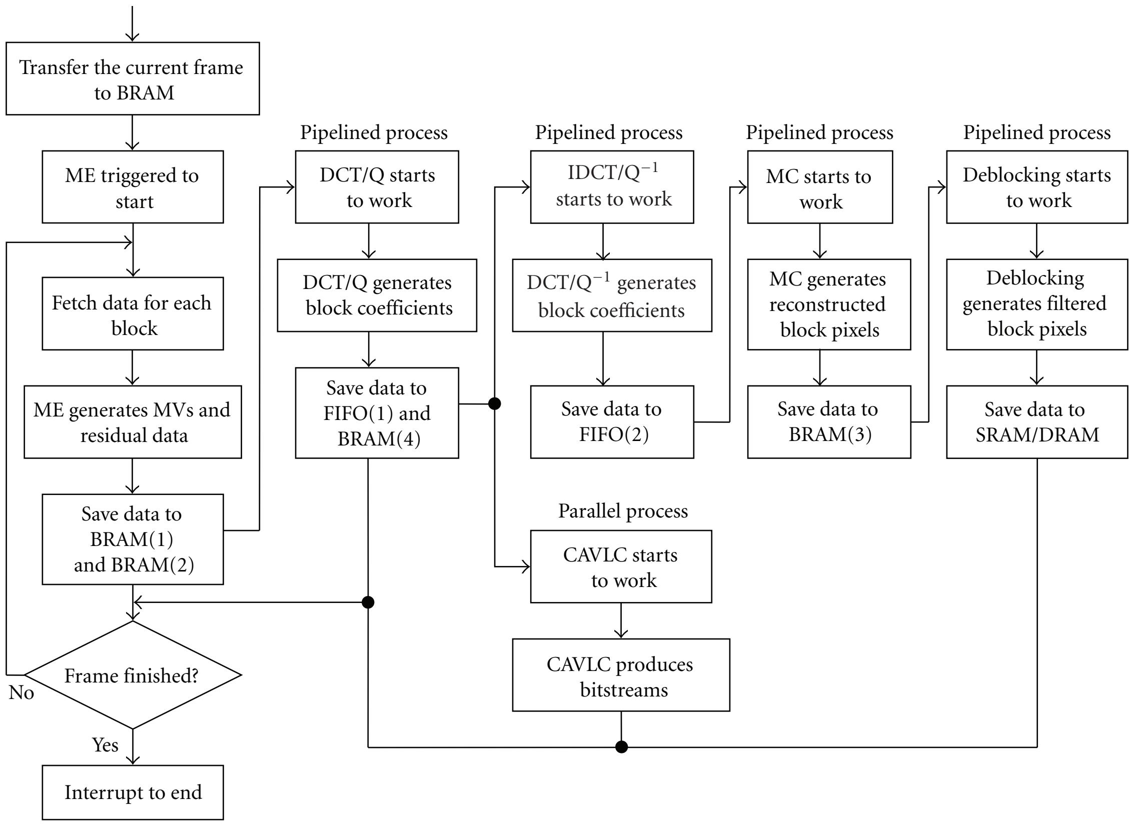

Transfer the current frame to BRAM
ME triggered to start
Fetch data for each block
ME generates MVs and residual data
Save data to BRAM(1) and BRAM(2)
Frame finished?
No
Yes
Interrupt to end
Pipelined process
DCT/Q starts to work
DCT/Q generates block coefficients
Save data to FIFO(1) and BRAM(4)
Pipelined process
IDCT/Q^{-1} starts to work
DCT/Q^{-1} generates block coefficients
Save data to FIFO(2)
Pipelined process
MC starts to work
MC generates reconstructed block pixels
Save data to BRAM(3)
Pipelined process
Deblocking starts to work
Deblocking generates filtered block pixels
Save data to SRAM/DRAM
Parallel process
CAVLC starts to work
CAVLC produces bitstreams

FIGURE 10: A video task partitioning and data control flow for the optimized system architecture.

start video encoding routines. As BRAM is a dual-port memory, the overhead of DMA transfer is eliminated by this dual-port cache.

This IP accelerated system model includes the memory, algorithm, and architecture optimization techniques to enable the reduction and elimination of the overhead resulted from the heterogeneous video encoding tasks. The video encoding model provided in this architecture is compliant with H.264 standard specifications.

A data control flow based on the video task partitioning is shown in Figure 10. According to the data streaming, it is obvious that the parallel and pipelining operations dominate in the whole part of encoding tasks, which are able to yield an efficient processing performance.

7. Implementations

The proposed ACQPPS algorithm is integrated and verified under H.264 JM Reference Software [28], while the hardware architectures, including the ACQPPS motion estimator and system-on-platform framework, are synthesized with Synplify Pro 8.6.2, implemented using Xilinx ISE 8.1i SP3 targeting Virtex-4 XC4VSX35FF668-10, based on the WILDCARD-4 [29].

The system hardware architecture can sufficiently process the QCIF/SIF/CIF video frames with the support of on-platform design resources. The Virtex-4 XC4VSX35 contains 3,456 Kb BRAM [30], 192 XtremeDSP (DSP48) slices [31], and 15,360 logic slices, which are equivalent to almost 1 million logic gates. Moreover, WILDCARD-4 integrates the large-sized 8 MB SRAM and 128 MB DRAM. With the sufficient design resources and memory support, the whole video frames of QCIF/SIF/CIF can be directly stored in the on-platform memories for the efficient hardware processing.

For example, if a CIF YUV (YCbCr) 4 : 2 : 0 video sequence is encoded with the optimized hardware architecture proposed in Figure 9, the total size of each current frame is 148.5 Kb. Therefore, each of the current CIF frame can be transferred from host system and directly stored in BRAM for motion estimation and video encoding, whereas the generated reference frames are stored in SRAM or DRAM. The SRAM and DRAM can accommodate a maximum of up to 55 and 882 CIF reference frames, respectively, which are more than enough for the practical video encoding process.

7.1. Performance of ACQPPS Algorithm. A variety of video sequences which contain different amount of motions, listed in Table 4, is examined to verify the algorithm performance for real-time encoding (30 fps). All sequences are in the format of YUV (YCbCr) 4 : 2 : 0 with luminance component to be processed for ME. The frame size of sequences varies from QCIF to SIF and CIF, which is the typical testing condition. The targeted bit rate is from 64 Kbps to 2 Mbps. SAD is used as the intensity matching criterion. The search window is $[-15, +15]$ for FS. EPZS uses extended diamond pattern and PMVFAST pattern [32] for its primary and secondary refined search stages. It also enables the window based, temporal, and spatial memory predictors to perform advanced motion search. UMHexagonS utilizes search range prediction and default scale factor optimized for different image sizes. Encoded frames are produced in a sequence of IPP,... PPP, as H.264 BP encoding is employed. For reconstructed video quality evaluation, the frame-based average peak signal-to-noise ratio (PSNR) and number of search points (NSP) per MB (16×16 pixels) are measured. Video encoding is configured with the support of full-pel motion accuracy, single reference frame and VBS. As VBS is a complicated feature defined in H.264, to make easy and practical the calculation of NSP regarding different block sizes, all search points for variable block estimation are normalized to the search points regarding the MB measurement, so that the NSP results can be evaluated reasonably.

The implementation results in Tables 6 and 7 show that the estimated image quality produced by ACQPPS, in terms of PSNR, is very close to that from FS, while the number of average search points is dramatically reduced. The PSNR difference between ACQPPS and FS is in the range of -0.13 dB ~ 0 dB. In most cases, PSNR degradation of ACQPPS is less than 0.06 dB, as compared to FS. In some cases, PSNR results of ACQPPS can be approximately equivalent or equal to those generated from FS. When compared with other fast search methods, that is, DS (small pattern), UCBDS, TSS, FSS and HEX, ACQPPS result is able to outperform their performance. ACQPPS can always yield higher PSNR than those fast algorithms. In this case, ACQPPS can obtain an average PSNR of +0.56 dB higher than those algorithms with evaluated video sequences.

Besides, ACQPPS performance is comparable to that of the complicated and advanced EPZS and UMHexagonS algorithms, as it can achieve an average PSNR in the range of -0.07 dB ~ +0.05 dB and -0.04 dB ~ +0.08 dB, as compared to EPZS and UMHexagonS, respectively.

In addition to the real-time video sequence encoding with 30 fps, many other application cases, such as the mobile scenario and videoconferencing, require video encoding under the low bit and frame rate environment with less

Table 4: Video sequences for experiment with real-time frame rate.

Sequence (bit rate Kbps)	Size/frame rate	No. of frames
Foreman (512)	QCIF/30 fps	300
Carphone (256)	QCIF/30 fps	382
News (128)	QCIF/30 fps	300
Miss_Am (64)	QCIF/30 fps	150
Suzie (256)	QCIF/30 fps	150
Highway (192)	QCIF/30 fps	2000
Football (2048)	SIF/30 fps	125
Table_Tennis (1024)	SIF/30 fps	112
Foreman (1024)	CIF/30 fps	300
Mother_Daughter (128)	CIF/30 fps	300
Stefan (2048)	CIF/30 fps	90
Highway (512)	CIF/30 fps	2000

Table 5: Video sequences for experiment with low bit and frames rates.

Sequence (bit rate Kbps)	Size/frame rate	No. of frames
Foreman (90)	QCIF/7.5 fps	75
Carphone (56)	QCIF/7.5 fps	95
News (64)	QCIF/15 fps	150
Miss_Am (32)	QCIF/15 fps	75
Suzie (90)	QCIF/15 fps	75
Highway (64)	QCIF/15 fps	1000
Football (256)	SIF/10 fps	40
Table_Tennis (150)	SIF/10 fps	35
Foreman (150)	CIF/10 fps	100
Mother_Daughter (64)	CIF/10 fps	100
Stefan (256)	CIF/10 fps	30
Highway (150)	CIF/10 fps	665

Table 6: Average PSNR performance for experiment with real-time and frame rate.

Sequence	FS	DS	UCBDS	TSS	FSS	HEX	EPZS	UMHexagonS	ACQPPS
Foreman (QCIF)	38.48	38.09	37.93	38.27	38.19	37.87	38.45	38.44	38.42
Carphone (QCIF)	36.43	36.23	36.16	36.30	36.24	36.04	36.42	36.37	36.37
News (QCIF)	37.44	37.26	37.35	37.28	37.29	37.25	37.43	37.35	37.43
Miss_Am (QCIF)	39.07	39.01	39.01	39.00	38.94	39.01	38.98	39.01	39.03
Suzie (QCIF)	38.65	38.46	38.47	38.59	38.54	38.45	38.61	38.58	38.60
Highway (QCIF)	38.23	37.99	38.13	38.11	38.09	38.06	38.18	38.17	38.13
Football (SIF)	31.37	31.23	31.23	31.22	31.24	31.20	31.40	31.37	31.36
Table_Tennis (SIF)	33.87	33.71	33.79	33.62	33.72	33.71	33.87	33.84	33.84
Foreman (CIF)	36.30	35.91	35.83	35.72	35.70	35.69	36.27	36.24	36.25
Mother_Daughter (CIF)	36.26	36.16	36.24	36.21	36.21	36.22	36.26	36.23	36.26
Stefan (CIF)	33.87	33.46	33.36	33.45	33.39	33.30	33.89	33.82	33.82
Highway (CIF)	37.96	37.70	37.83	37.79	37.77	37.76	37.89	37.87	37.83

Table 7: Average number of search points per MB for experiment with real-time and frame rate.

Sequence	FS	DS	UCBDS	TSS	FSS	HEX	EPZS	UMHexagonS	ACQPPS
Foreman (QCIF)	2066.73	60.64	109.70	124.85	122.37	109.26	119.02	125.95	55.63
Carphone (QCIF)	1872.04	46.82	91.54	108.44	106.52	94.82	114.83	121.91	54.02
News (QCIF)	1719.92	33.72	74.48	88.28	90.30	81.12	81.36	79.73	41.32
Miss_Am (QCIF)	1471.96	30.70	64.35	74.95	76.52	68.43	62.94	56.27	32.32
Suzie (QCIF)	1914.32	44.19	88.19	108.19	104.97	93.21	96.98	88.74	47.59
Highway (QCIF)	1791.86	40.27	85.14	101.49	100.94	90.27	85.04	84.24	46.12
Football (SIF)	2150.45	68.42	118.21	131.82	129.91	117.62	184.81	202.19	72.63
Table_Tennis (SIF)	2031.72	55.66	105.56	120.09	121.27	108.79	128.36	124.95	54.25
Foreman (CIF)	1960.07	76.83	124.56	128.85	125.76	117.21	122.22	124.26	67.20
Mother_Daughter (CIF)	1473.73	35.08	70.39	82.18	82.80	73.67	80.38	63.51	40.89
Stefan (CIF)	1954.21	69.32	116.72	118.23	122.37	113.50	137.59	149.80	58.91
Highway (CIF)	1730.90	45.63	90.81	104.90	103.98	93.22	78.82	75.98	47.57

than 30 fps. Accordingly, the satisfied settings for video encoding are usually 7.5 fps ∼ 15 fps for QCIF and 10 fps ∼ 15 fps for SIF/CIF with various low bit rates, for example, 90 Kbps for QCIF and 150 Kbps for SIF/CIF, to maximize the perceived video quality [40, 41]. In order to further evaluate the ME algorithms under low bit and frame rate cases, video sequences are provided in Table 5, and Tables 8 and 9 generate the corresponding performance results.

The experiments show that the PSNR difference between ACQPPS and FS is still small, which is in an acceptable range of −0.49 dB ∼ −0.02 dB. In most cases, there is only less than 0.2 dB PSNR discrepancy between them. Moreover, ACQPPS still sufficiently outperforms DS, UCBDS, TSS, FSS and HEX. For mobile scenarios, there are usually quick and considerable motion displacements existing, under the environment of low frame rate video encoding. In such case, ACQPPS is particularly much better than those fast algorithms, and a result of up to +2.42 dB for PSNR can be achieved with the tested sequences. When compared with EPZS and UMHexagonS, ACQPPS can yield an average PSNR in the range of −0.36 dB ∼ +0.06 dB and −0.15 dB ∼ +0.07 dB, respectively.

Normally, ACQPPS is useful to produce a favorable PSNR for the sequences not only with small object motions, but also large amount of motions. In particular, if a sequence includes large object motions or considerable amount of motions, the advantage of ACQPPS algorithm is obvious, as the ACQPPS can adaptively choose different shapes and sizes for the search pattern which is applicable to the efficient large motion search.

Such search advantage can be observed when ACQPPS is compared with DS. It is know that DS has a simple diamond pattern for a very low complexity based motion search. For video sequences with slow and small motions contained, for example, Miss_Am (QCIF) and Mother_Daguhter (CIF) at 30 fps, the PSNR performance of DS and ACQPPS is relatively close, which indicates that DS performs well in the case of simple motion search. When the complicated and large amount of motions included in video images, however, DS is unable to yield good PSNR, as its motion search will be easily trapped in undesirable local minimum. For example, the PSNR differences between DS and ACQPPS are 0.34 dB and 0.44 dB, when Foreman (CIF) is tested with 1 Mbps at 30 fps and 150 Kbps at 10 fps, respectively. Furthermore, ACQPPS can produce an average PSNR of +0.02 dB ∼ +0.36 dB higher than DS in the case of real-time video encoding, and +0.07 dB ∼ +1.94 dB in the case of low bit and frame rate environment.

Table 8: Average PSNR performance for experiment with low bit and frame rates.

Sequence	FS	DS	UCBDS	TSS	FSS	HEX	EPZS	UMHexagonS	ACQPPS
Foreman (QCIF)	34.88	34.44	34.19	34.40	34.42	33.99	34.85	34.80	34.80
Carphone (QCIF)	34.12	33.99	33.96	34.02	33.99	33.84	34.08	34.04	34.06
News (QCIF)	35.28	35.21	35.20	35.11	35.20	35.19	35.25	35.21	35.24
Miss_Am (QCIF)	38.36	38.23	38.33	38.25	38.23	38.31	38.28	38.27	38.34
Suzie (QCIF)	36.54	36.40	36.34	36.44	36.39	36.26	36.52	36.50	36.50
Highway (QCIF)	36.19	35.80	36.01	35.96	35.94	35.90	36.13	36.09	35.98
Football (SIF)	25.11	24.82	24.92	24.79	24.84	24.89	25.08	25.10	25.01
Table_Tennis (SIF)	27.57	26.65	26.95	26.85	26.84	26.86	27.57	27.60	27.45
Foreman (CIF)	31.95	31.29	31.32	31.16	31.25	31.06	31.90	31.79	31.73
Mother_Daughter (CIF)	36.07	35.84	35.95	35.90	35.92	35.91	36.07	36.02	36.05
Stefan (CIF)	27.02	24.59	24.92	24.11	24.12	24.96	26.89	26.67	26.53
Highway (CIF)	37.21	36.88	36.98	36.94	36.92	36.94	37.12	37.09	37.01

Table 9: Average number of search points per MB for experiment with low bit and frame rates.

Sequence	FS	DS	UCBDS	TSS	FSS	HEX	EPZS	UMHexagonS	ACQPPS
Foreman (QCIF)	2020.51	90.20	140.01	134.64	133.92	125.12	163.63	190.38	98.94
Carphone (QCIF)	1836.04	58.40	102.76	112.65	111.28	100.74	141.56	160.32	71.81
News (QCIF)	1680.68	34.74	74.11	87.22	88.92	79.64	96.40	102.30	52.61
Miss_Am (QCIF)	1406.26	32.60	64.56	75.39	75.08	67.74	68.05	63.20	44.22
Suzie (QCIF)	1823.23	52.96	94.39	110.43	106.30	95.11	115.96	112.17	64.30
Highway (QCIF)	1710.86	42.06	84.42	97.99	97.34	87.36	98.22	97.77	58.13
Football (SIF)	1914.43	80.13	132.67	123.20	125.01	119.62	192.88	246.76	92.51
Table_Tennis (SIF)	1731.44	50.10	98.45	97.71	100.73	93.82	159.45	182.19	64.39
Foreman (CIF)	1789.76	91.32	140.31	124.01	124.24	120.48	154.55	170.89	88.62
Mother_Daughter (CIF)	1467.56	42.21	78.32	87.45	87.75	78.42	90.40	78.14	52.36
Stefan (CIF)	1663.89	65.44	110.17	100.53	102.36	103.71	153.97	194.64	78.69
Highway (CIF)	1715.63	52.26	97.20	109.24	107.49	96.94	91.74	92.27	64.45

The number of search points for each method, which mainly represents the algorithm complexity, is also obtained to measure the search efficiency of different approaches. The NSP results show that the search efficiency of ACQPPS is higher than other algorithms, as ACQPPS can produce very good performance, in terms of PSNR, with reasonably possessed NSP. The NSP of ACQPPS is one of the least among all methods.

If ACQPPS is compared with DS, it is shown that ACQPPS has the similar NSP as DS. It is true that NSP of ACQPPS is usually a little bit increased in comparison with that of DS. However, the increasing of the NSP is limited and very reasonable, and is able to in turn bring ACQPPS much better PSNR for the encoded video quality. Furthermore, for the video sequences containing complex and quick object motions, for example, Foreman (CIF) and Stefan (CIF) at 30 fps, the NSP of ACQPPS can be even less than that of DS, which verifies that ACQPPS has a much satisfied search efficiency than DS, due to its highly adaptive search patterns.

In general, the complexity of ACQPPS is very low, and with high search performance, which makes it especially useful for the hardware architecture implementation.

7.2. Design Resources for ACQPPS Motion Estimator. As the complexity and search points of ACQPPS have been greatly reduced, design resources used by ACQPPS architecture can be kept at a very low level. The main part of design resources is for SAD calculator. Each BPU requires one 32-bit processing element (PE) to implement SAD calculations. Every PE has two 8-bit pixel data inputs, one from the current block and the other from reference block. Besides, every PE contains 16 subtractors, 8 three-input adders, 1 latch register, and does not require extra interim registers or accumulators. As a whole, a 32×4 PE array will be needed to implement the pipelined multilevel SAD calculator, which requires totally 64 subtractors, 32 three-input adders, and 4 latch registers. Other related design resources mainly include an 18×18 register array, a 16×16 register array, a few of accumulators, subtractors and comparators, which are used to generate the block SAD results, residual data and final estimated MVs. Moreover, some other multiplexers, registers, memory access, and data flow control logic gates are also needed in the architecture. A comparison of design resources between ACQPPS and other ME architectures [33–36] is presented in Table 10. The results show that proposed ACQPPS architecture can utilize greatly reduced design

Table 10: Performance comparison between proposed ACQPPS and other motion estimation hardware architectures.

	[33]	[34]	[35]	[36]	Proposed architecture
Type	ASIC	ASIC	ASIC	ASIC	FPGA + DSP
Algorithm	FS	FS	FS	FS	ACQPPS
Search range	[−16, +15]	[−32, +31]	[−16, +15]	[−16, +15]	Flexible
Gate count	103 K	154 K	67 K	108 K	35 K
Support block sizes	All	All	8 × 8 16 × 16 32 × 32	All	All
Freq. [MHz]	66.67	100	60	100	75
Max fps of CIF	102	60	30	56	120
Min Freq. [MHz]for CIF 30 fps	19.56	50	60	54	18.75

Table 11: Design resources for system-on-platform architecture.

Target FPGA	Critical Path	Gates	DFFs/Latches
XC4VSX35FG668-10	5 ns	279,774	3,388
Target FPGA	LUTs	CLB Slices	Resource
XC4VSX35FG668-10	3,161	3,885	25%

Table 12: DMA performance for video sequence transfer.

QCIF 4 : 2 : 0 YCrCb	DMA Write (ms)	DMA Read (ms)	DMA R/W (ms)
WildCard-4	0.556	0.491	0.515
CIF 4 :2 : 0 YCrCb	DMA Write (ms)	DMA Read (ms)	DMA R/W (ms)
WildCard-4	2.224	1.963	2.059

resources to realize a high-performance motion estimator for H.264 encoding.

7.3. Throughput of ACQPPS Motion Estimator. Unlike the FS which has a fixed search range, search points and search range of ACQPPS depend on video sequences. ACQPPS search points will be increased, if a video sequence contains considerable or quick motions. On the contrary, search points can be reduced, if a video sequence includes slow or small amount of motions.

The ME scheme with a fixed block size can be typically applied to the throughput analysis. In such case, the worst case will be the motion estimation using 4 × 4 blocks, which is the most time consuming in the case of fixed block size. Hence, the overall throughput result produced by ACQPPS architecture can be reasonably generalized and evaluated.

In general, if the clock frequency is 50 MHz and the memory (SRAM, BRAM and DRAM) structure is organized as DWORD (32-bit) for each data access, the ACQPPS hardware architecture will approximately need an average of 12.39 milliseconds for motion estimation in the worst case of using 4 × 4 blocks. For a real-hardware architecture implementation, the typical throughput in the worst case of 4×4 blocks can represent the overall motion search ability for this motion estimator architecture.

Therefore, the ACQPPS architecture can complete the motion estimation for more than 4 CIF (352 × 288) video sequences or equivalent 1 4 CIF (704 × 576) video sequence at 75 MHz clock frequency within each 33.33 milliseconds time slot (30 fps) to meet the real-time encoding requirement for a low design cost and low bit rate implementation. The throughput ability of ACQPPS architecture can be compared with those of a variety of other recently developed motion estimator hardware architectures, as illustrated in Table 10. The comparison results show that the proposed ACQPPS architecture can achieve higher throughput than other hardware architectures, with the reduced operational clock frequency. Generally, it will only require a very low clock frequency, that is, 18.75 MHz, to generate the motion estimation results for the CIF video sequences at 30 fps.

7.4. Realization of System Architecture. Table 11 lists the design resources utilized by system-on-platform framework. The implementation results indicate that the system architecture uses approximately 25% of the FPGA design resources when there is no hardware IP accelerator integrated in the platform system. If video functions are needed, there will be more design resources demanded, in order to integrate and accommodate necessary IP modules. Table 12 gives a performance result of the platform DMA video frame transfer feature.

Different DMA burst sizes will result in different DMA data transfer rates. In our case, the maximum DMA burst size is defined to accommodate a whole CIF 4 : 2 : 0 video frame, that is, 38,016 Dwords for each DMA data transfer buffer. Accordingly, the DMA transfer results verify that it only takes an average of approximately 2 milliseconds to transfer a whole CIF 4 : 2 : 0 video frame based on WildCard-4. This transfer performance can sufficiently support up to level 4 bitstream rate for the H.264 BP video encoding system.

7.5. Overall Encoding Performance. In view of the complexity analysis of H.264 video tasks described in Section 2, the most time consuming task is motion estimation. Other encoding tasks have much less overhead. Therefore, the video tasks can be scheduled to operate in parallel and pipelining stages as displayed in Figures 9 and 10 for the proposed architecture

Table 13: An overall performance comparison for H.264 BP video encoding systems.

Implementation	[37]	[38]	[39]	Proposed architecture
Architecture	ASIC	Codesign	Codesign	Codesign(Extensible multiple processing cores)
ME Algorithm	Full Search (FS)	Full Search (FS)	Hexagon (HEX)	ACQPPS
Freq. [MHz]	144	100	81	75
Max fps of CIF	272.73	5.125	18.6	120
Min Freq. [MHz] for CIF 30 fps	15.84	585	130.65	18.75
Core Voltage Supply	1.2 V	1.2 V	1.2 V	1.2 V
I/O Voltage Supply	1.8/2.5/3.3 V	1.8/2.5/3.3 V	2.5/3.3 V	2.5/3.3 V

model. In this case, the overall encoding time for a video sequence is approximately equal to the following

$$\begin{aligned}&\text{Encoding time}\\&\quad=\text{Total motion estimation time}\\&\qquad+\text{Processing time of DCT/Q for the last block}\\&\qquad+\text{Max}\Big\{\text{Processing time of IDCT/Q}^{-1}+\text{MC}\\&\qquad\qquad+\text{Deblocking for the last block},\\&\qquad\qquad\text{Processing time of CAVLC for the last block}\Big\}.\end{aligned}\tag{8}$$

The processing time of DCT/Q, IDCT/Q^{-1}, MC, Deblocking Filter, and CAVLC for a divided block directly depends on the architecture design for each of the module. On an average, it is normal that the overhead of those video tasks for encoding an individual block is much less than that of motion estimation. As a whole, the encoding time derived from those video tasks for the last one block can be even ignored, when it is compared to the total processing time of the motion estimator for a whole video sequence. Therefore, to simplify the overall encoding performance analysis for the proposed architecture model, the total encoding overhead derived from the system architecture for a video sequence can be approximately regarded as

$$\text{Encoding time}\approx\text{Total motion estimation time}.\tag{9}$$

This simplified system encoding performance analysis is valid as long as the video tasks are operated in concurrent and pipelined stages with the efficient optimization techniques. Accordingly, when the proposed ACQPPS motion estimator is integrated into the system architecture to perform the motion search, the overall encoding performance for the proposed architecture model is generalized.

A performance comparison can be presented in Table 13, where the proposed architecture is compared with some other recently developed H.264 BP video encoding systems [37–39] including both fully dedicated hardware and codesign architectures. The results indicate that this proposed system-on-platform architecture, when integrated with the IP accelerators, can yield a very good performance which is comparable or even better than other H.264 video encoding systems. Especially, if compared with other codesign architectures, the proposed system has much higher encoding throughput, which is about 30 and 6 times higher than the processing ability of the architectures presented in [38, 39], respectively. The generated high performance of proposed architecture is directly contributed from the efficient ACQPPS motion estimation architecture and the techniques employed for the system optimizations.

8. Conclusions

An integrated reconfigurable hardware-software codesign IP accelerated system-on-platform architecture is proposed in this paper. The efficient virtual socket interface and optimization approaches for hardware realization have been presented. The system architecture is flexible for the host interface control and extensible with multiple cores, which can actually construct a useful integrated and embedded system approach for the dedicated functions.

An advanced application for this proposed architecture is to facilitate the development of H.264 video encoding system. As the motion estimation is the most complicated and important task in video encoder, a block-based novel adaptive motion estimation search algorithm, ACQPPS, and its hardware architecture are developed for reducing the complexity to extremely low level, while keeping the encoding performance, in terms of PSNR and bit rate, as high as possible. It is beneficial to integrate video IP accelerators, especially ACQPPS motion estimator, into the architecture framework for improving the overall encoding performance. The proposed system architecture is mapped on an integrated FPGA device, WildCard-4, toward an implementation for a simplified H.264 BP video encoder.

In practice, with the proposed system architecture, the realization of multistandard video codec can be greatly facilitated and efficiently verified, other than the H.264 video applications. It can be expected that the advantages of the proposed architecture will become more desirable for prototyping the future video encoding systems, as new video standards are emerging continually, for example, the coming H.265 draft.

Acknowledgment

The authors would like to thank the support from Alberta Informatics Circle of Research Excellence (iCore), Xilinx Inc., Natural Science and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), and the Department of Electrical and Computer Engineering at the University of Calgary.

References

[1] M. Tekalp, *Digital Video Processing*, Signal Processing Series, Prentice Hall, Englewood Cliffs, NJ, USA, 1995.

[2] "Information technology—generic coding of moving pictures and associated audio information: video," ISO/IEC 13818-2, September 1995.

[3] "Video Coding for Low Bit Rate Communication," ITU-T Recommendation H.263, March 1996.

[4] "Coding of audio-visual objects—part 2: visual, amendment 1: visual extensions," ISO/IEC 14496-4/AMD 1, April 1999.

[5] Joint Video Team of ITU-T and ISO/IEC JTC 1, "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC)," JVT-G050r1, May 2003; JVT-K050r1 (non-integrated form) and JVT-K051r1 (integrated form), March 2004; Fidelity Range Extensions JVT-L047 (non-integrated form) and JVT-L050 (integrated form), July 2004.

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

[7] S. Wenger, "H.264/AVC over IP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 645–656, 2003.

[8] B. Zeidman, *Designing with FPGAs and CPLDs*, Publishers Group West, Berkeley, Calif, USA, 2002.

[9] S. Notebaert and J. D. Cock, *Hardware/Software Co-design of the H.264/AVC Standard*, Ghent University, White Paper, 2004.

[10] W. Staehler and A. Susin, *IP Core for an H.264 Decoder SoC*, Universidade Federal do Rio Grande do Sul (UFRGS), White Paper, October 2008.

[11] R. Chandra, *IP-Reuse and Platform Base Designs*, STMicroelectronics Inc., White Paper, February 2002.

[12] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 688–703, 2003.

[13] J. Ostermann, J. Bormans, P. List, et al., "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, 2004.

[14] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704–716, 2003.

[15] S. Saponara, C. Blanch, K. Denolf, and J. Bormans, "The JVT advanced video coding standard: complexity and performance analysis on a tool-by-tool basis," in *Proceedings of the Packet Video Workshop (PV '03)*, Nantes, France, April 2003.

[16] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, vol. 29, no. 12, pp. 1799–1808, 1981.

[17] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proceedings of the IEEE National Telecommunications Conference (NTC '81)*, vol. 4, pp. 1–9, November 1981.

[18] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, pp. 438–442, 1994.

[19] L.-M. Po and W.-C. 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.

[20] L.-K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 4, pp. 419–421, 1996.

[21] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," in *Proceedings of the International Conference on Information, Communications and Signal Processing (ICICS '97)*, vol. 1, pp. 292–296, Singapore, September 1997.

[22] 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.

[23] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp. 369–377, 1998.

[24] Y. Nie and K.-K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1442–1449, 2002.

[25] H. C. Tourapis and A. M. Tourapis, "Fast motion estimation within the H.264 codec," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '03)*, vol. 3, pp. 517–520, Baltimore, Md, USA, July 2003.

[26] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Visual Communications and Image Processing*, vol. 4671 of *Proceedings of SPIE*, pp. 1069–1079, January 2002.

[27] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Fast integer pel and fractional pel motion estimation for AVC," JVT-F016, December 2002.

[28] K. Sühring, "H.264 JM Reference Software v.15.0," September 2008, http://iphome.hhi.de/suehring/tml/download.

[29] Annapolis Micro Systems, "Wildcard^{TM}—4 Reference Manual," 12968-000 Revision 3.2, December 2005.

[30] Xilinx Inc., "Virtex-4 User Guide," UG070 (v2.3), August 2007.

[31] Xilinx Inc., "XtremeDSP for Virtex-4 FPGAs User Guide," UG073(v2.1), December 2005.

[32] A. M. Tourapis, O. C. Au, and M. L. Liou, "Predictive motion vector field adaptive search technique (PMVFAST)—enhanced block based motion estimation," in *Proceedings of the IEEE Visual Communications and Image Processing (VCIP '01)*, pp. 883–892, January 2001.

[33] Y.-W. Huang, T.-C. Wang, B.-Y. Hsieh, and L.-G. Chen, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '03)*, vol. 2, pp. 796–798, May 2003.

[34] M. Kim, I. Hwang, and S. Chae, "A fast VLSI architecture for full-search variable block size motion estimation in MPEG-4 AVC/H.264," in *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*, vol. 1, pp. 631–634, January 2005.

[35] J.-F. Shen, T.-C. Wang, and L.-G. Chen, "A novel low-power full-search block-matching motion-estimation design for H.263+," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 7, pp. 890–897, 2001.

[36] S. Y. Yap and J. V. McCanny, "A VLSI architecture for advanced video coding motion estimation," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '03)*, vol. 1, pp. 293–301, June 2003.

[37] S. Mochizuki, T. Shibayama, M. Hase, et al., "A 64 mW high picture quality H.264/MPEG-4 video codec IP for HD mobile applications in 90 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 11, pp. 2354–2362, 2008.

[38] R. R. Colenbrander, A. S. Damstra, C. W. Korevaar, C. A. Verhaar, and A. Molderink, "Co-design and implementation of the H.264/AVC motion estimation algorithm using co-simulation," in *Proceedings of the 11th IEEE EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08)*, pp. 210–215, September 2008.

[39] Z. Li, X. Zeng, Z. Yin, S. Hu, and L. Wang, "The design and optimization of H.264 encoder based on the nexperia platform," in *Proceedings of the 8th IEEE International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD '07)*, vol. 1, pp. 216–219, July 2007.

[40] S. Winkler and F. Dufaux, "Video quality evaluation for mobile applications," in *Visual Communications and Image Processing*, vol. 5150 of *Proceedings of SPIE*, pp. 593–603, Lugano, Switzerland, July 2003.

[41] M. Ries, O. Nemethova, and M. Rupp, "Motion based reference-free quality estimation for H.264/AVC video streaming," in *Proceedings of the 2nd International Symposium on Wireless Pervasive Computing (ISWPC '07)*, pp. 355–359, February 2007.