# A HIGHLY DATA REUSABLE AND STANDARD-COMPLIANT MOTION ESTIMATION HARDWARE ARCHITECTURE

*Xing Wen, Oscar C. Au, Jiang Xu, Lu Fang, Run Cha, Jiali Li*

Department of Electronic and Computer Engineering
The Hong Kong University of Science and Technology
Email: {wxxab, eeau, eexu, fanglu, charun, jiali}@ust.hk

## ABSTRACT

Motion Estimation (ME) is the most computationally intensive part in the whole video compression process. The ME algorithms can be divided into full search ME (FS) and fast ME (FME). The FS is not suitable for high definition (HD) frame size videos because its relevant high computation load and hard to deal with complex motions in limited search range. A lot of FME algorithms have been proposed which can significantly reduce the computation load compared to FS. Though many kinds of hardware implementations of ME have been proposed, almost all of them fail to consider about the motion vector field (MVF) coherence and rate-distortion (RD) cost which have significant impact to the coding efficiency. In this paper, we propose a hardware friendly ME algorithm and corresponding highly data reusable hardware architecture. Simulation results show that the proposed ME algorithm performs better RD performance than conventional FME algorithm. The proposed reconfigurable ME hardware is implemented in VHDL and mapped to a low cost Xilinx XC3S1500 FPGA. It works at 100MHz and is capable to process $1920 \times 1080$ of 30fps video format in real time and have very high data reuse ratio.

***Keywords*—** motion estimation, hardware implementation, VHDL

## 1. INTRODUCTION

Video sequences are consisted of series of highly correlated images, which are called frames. The RAW data of the video sequences have a lot of redundance, both in the spacial and temporal domains. The temporal domain redundance is more significant based on the fact that each frame is only take a very short time (like 1/30 or 1/60 second) and adjacent frames are highly correlated. For efficient storage and transmission, a lot of video coding standard have been developed, for example: MPEG-1/2/4 and H.26x. Almost every video coding standards have adopted the ME process which can successfully eliminate most of the temporal redundance based on the fact that only the difference (residue) between the current frame and the prediction frame is need to be coded and transmitted.

Most of the existing ME methods are based on Block Match (BM) algorithms. Its basic idea is to partition the current frame into non-overlapping squares and try to find the best matching blocks (prediction blocks) in a certain range (search window) of the reference frame based on the sum of absolute difference (SAD) criteria. The displacement between the current block and the prediction block is called motion vector (MV).

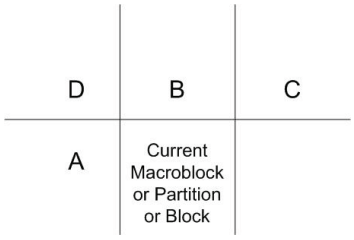



**Fig. 1**. Neighboring successfully coded blocks

Besides FS, a lot of FME algorithm have been developed to deal with the high computation load of the FS. The FME can be divided into two categorizes. The first kind can be named as "Zero biased" which is based on the gradient decent method and adopts hierarchical search scheme with different search patterns, for examples, NTSS [1], Diamond search [2], FTS [3] and Cross search [4]. Thought this kind of methods can significantly reduce the number of searching location, it could be easily trapped into a local minimum which results in fake motion and bad RD performance. The second kind method is motion vector predictor (MVP) biasd which is adopted by state of art video coding standard MPEG-4 and H.264. The basic idea of this kind methods is that rather than zero biased, it adopts the MV information from neighboring successfully encoded blocks to generate several MVPs as the candidates of the search window center. Further more, not only the SAD but also the bits needed to code the MVs are also been included into the criteria which is called RD cost function. For example: PMVFAST [5], UMHexagonS [6]. This kind of method can achieve almost same or even better RD performance than the FS. However, it is hard to efficiently implement this kind of methods because the complex algorithm can easily break the pipe line and increase the times of memory access.

Though many different architectures have been proposed to implement FME algorithm, for example [7, 8], but almost none of them have adopted the RD cost function as criteria which can

978-1-4244-7493-6/10/$26.00 ©2010 IEEE 

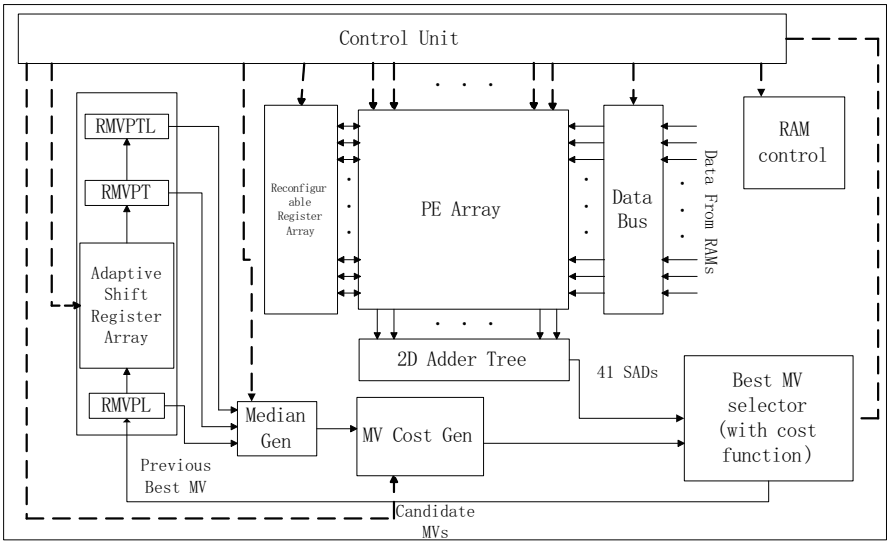



**Fig. 2**. Top-level block diagram of proposed architecture

achieve better RD performance than only use SAD. Although [9] adopts a approximated lambda cost function, it has not considered about the floating point multiplication and corresponding coding loss to the original video coding standard. So in this paper, at first we propose an hardware friendly MVP biased FS adopted an hardware friendly RD cost function according to the H.264 video coding standard [10], and then a highly data reusable hardware architecture is developed to implement the proposed algorithm. This paper is organized as follows: in section 2, we proposed the hardware friendly fast motion algorithm; the corresponding hardware architecture is described in section 3; implementation results and comparison are shown in section 4; at last, section 5 concludes this work.

## 2. PROPOSED MOTION ESTIMATION ALGORITHM

The state of art FME methods always adopt multiple MVPs to avoid being trapped by local minimum search locations. Many MVP candidates are available during the coding process, for example, the MVs of neighboring coded blocks and collocated blocks, etc. But it is hard to implement this kind of methods by hardware based on the fact that multiple MVPs may lower the hardware efficiency and easily increase the times of memory access and bandwidth.

The proposed ME method is a single MVP (SMVP) biased fast full search method. As shown in Fig.1, we generate the SMVP from MVs of the top, top-left and right neighboring blocks, which is:

$$SMVP = Median(MV_{left}, MV_{top}, MV_{top\_left}) \quad (1)$$

We choose this median MVP based on two reasons. First, as described in [5], this median MVP may have best correlation to the real MVs. Second, the tree components can be easily reused based on their spacial locations.

Rather than only compare the SAD between the current and reference blocks, the proposed ME method also applies an adjust RD cost function in which includes the rate spent on coding the candidate MV. Because for every video coding standard, the coded MVs may take a significant part in the coded streams and also have strong impact to RD performance. Similar to the H.264/AVC video coding standard, we use the RD cost function as follows:

$$Cost(MV_{cand}) = SAD + \lambda_{SMVP} R(\mid SMVP - MV_{cand} \mid) \quad (2)$$

where $\lambda_{SMVP}$ is the Lambda Multiplier (lambda), $R()$ is to calculate how many bits are needed to code the MV difference (residue) between candidate MV and selected MVP. In H.264, the lambda is defined as follows:

$$\lambda_{mode,I,P} = 0.85 \times 2^{(QP-12)/3} \quad (3)$$

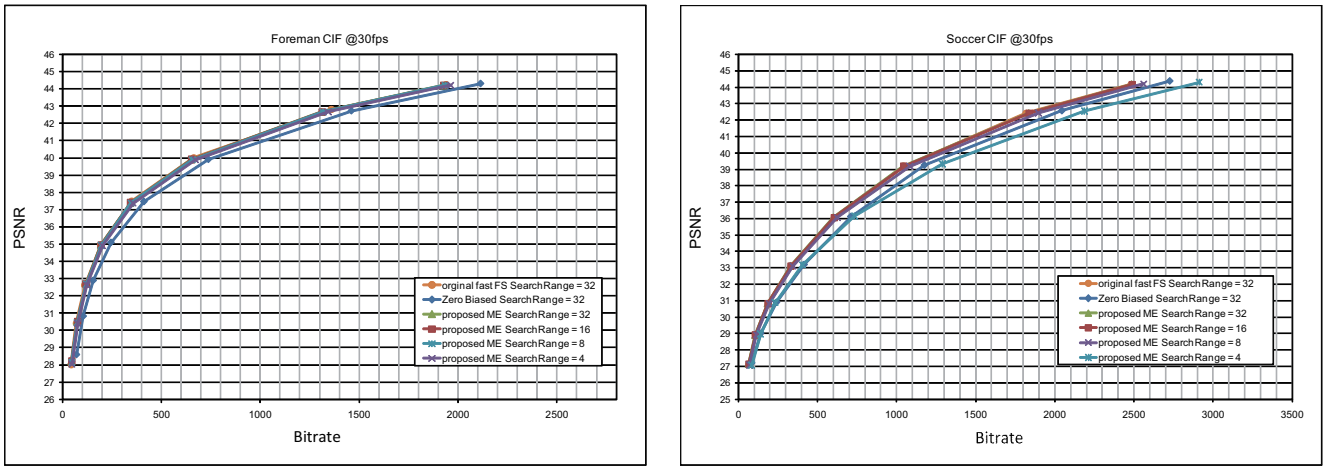
$$\lambda_{mode,B} = max(2, min(4, \frac{QP-12}{6})) \times \lambda_{mode,I,P} \quad (4)$$

$$\lambda_{ME} = \sqrt{\lambda_{mode}} \quad (5)$$

where the QP standards for the quantization parameter, $\lambda_{mode,I,P}$ is used for I and P frames and $\lambda_{mode,B}$ for B frames. However, the $\lambda_{ME}$ would result in floating point computation which is very timing and resource consuming, so we adjust it as follows:

$$\lambda_{SMVP} = floor\lfloor \frac{\ln \lambda_{ME}}{\ln 2} \rfloor \quad (6)$$

which makes the floating point multiplication become simple left and right shift operation and can be easily implemented by hardware. The RD performance of proposed method is shown in Fig. 3, the RD curves of zero biased FS with SAD only criterion and RD curves of original fast FS in JM12.4 with various search ranges are also shown as comparisons. In Table 2, the RD performance of state-of-the-art multiple MVP ME method




(a) (b)

**Fig. 3**. (a): RD performance of RDOMFS for Foreman CIF@30fps (b): RD performance of Soccer CIF@30fps

UMHexgonS [6] with various search ranges are shown as comparisons. We have tested many sequences with different resolutions and motion level, and two CIF sequences of them are Foreman (CIF@30fps), Soccer (CIF@30fps); three 720P sequences: Crew(720P@30fps), Raven (720P@30fps) and Jets (720P@30fps).

In Fig. 3, we could observe that the proposed methods can approach almost same coding efficiency as the original fast FS in JM12.4, even with much smaller search range (like $[-4, 4)$, which is corresponding to "SearchRange = 8"). And compared with zero biased FS method with SAD only criterion, our method can achieve about $0.4 - 1$dB BD-PSNR gain [11, 12] over various QPs. For the video sequences contained fast and complex motion, like Stefan and Soccer, more gain of proposed method can be achieved. This is because the fast motion may cause object to move outside the search window, if without MVP, ME process can only result in bad prediction. The MVP is provide an estimation of the true motion and can actually increase the search range, even with a limited search window size.

Table 2 shows the RD performance of the UMHexgonS which is state-of-the-art fast ME method with multiple MVP and can achieve much better coding efficiency than conventional fast ME methods like NTSS [1], Diamond search [2], FTS [3] and Cross search [4]. We could observe that the proposed method could achieve significant BD-PSNR gain and BD-Bitrate [11, 12] reduction.

## 3. PROPOSED RECONFIGURABLE ME HARDWARE

### 3.1. System Overview

Top-level block diagram of the proposed architecture is shown in Fig. 2. Data reuse is achieved by adding an Reconfigurable Register Array (RRA) and applying "Snake" scan order which is described in next section. The processor element (PE) array consists of 16 sub-PE arrays, each one stands for certain $4 \times 4$ current blocks. After load current $16 \times 16$ block pixels, reference pixels is propagated into the PE array and each clock cycle it computes 1 SAD for certain location in the search windows and pass the SAD to 2D Adder Tree (2DAT). The 2DAT takes 2 clock cycles to get sixteen $4 \times 4$ SADs; 3 clock cycles to get eight $8 \times 16$ SADs and eight $16 \times 8$ SADs; 4 clock cycles to get four $8 \times 8$ SADs; 5 clock cycles for two $16 \times 8$ and two $8 \times 16$ SADs and total 6 clock cycles to get the final $16 \times 16$ SAD. Then we have the 41 SADs for certain search location and pass them to Best MV Selector (BMVS). The best MV (BMV) is chosen after combination the SADs with corresponding MV coding cost.

Rather than write every BMVs into certain RAM and load back again, we choose to propagate the previous BMV to a Adaptive Shift Register Array (ASRA) which can fit different frame sizes. The Best MV is reusable based on the fact that the BMV of left block would become top and top-left BMV after the search of first line blocks. The BMV propagation and median computation can be done simultaneously when load the new current block pixels and do not take extra clock cycles.

### 3.2. Snake Scan Order

Most of the existing ME hardware architectures employ the raster scan order when doing the search and can achieve relevant high data reuse ratio. However this scan method can only reuse data in same line and redundant load may be caused by this. After adding an RRA to PE array, we propose a "Snake" scan order with which the propagated reference pixels can have better reuse ratio than raster scan.

The Fig. 4 shows the proposed "Snake" scan order which consist of four basic steps from A to D. The first step, A, fetches $N$ reference pixels (block size $N \times N$) in a column for each clock, at the same time, PE array would propagate the loaded reference pixel data to the RRA. The $M$ is the data reuse parameter which stand for how many columns can be reused and

**Table 1**. Performance of RDOMFS and UMHexgonS with Various Search Range

(a) RDOMFS SearchRange = 32

| Sequence | QP | UMHexgonS SearchRange = 64 Bitrate (kbit/s) | UMHexgonS SearchRange = 64 PSNR (dB) | RDOMFS SearchRange = 32 Bitrate (kbit/s) | RDOMFS SearchRange = 32 PSNR (dB) | BD-PSNR (dB) | BD-Bitrate (%) |
|---|---|---|---|---|---|---|---|
| Jets (720P @ 30fps) | 20 | 5878.38 | 42.80 | 5646.02 | 42.97 | 0.27 | -19.83 |
| | 24 | 1095.55 | 41.07 | 1142.65 | 41.47 | | |
| | 28 | 420.29 | 39.53 | 450.53 | 40.12 | | |
| | 32 | 244.46 | 37.82 | 279.89 | 38.41 | | |
| Raven (720P @ 30fps) | 20 | 7988.62 | 44.30 | 7722.81 | 44.55 | 0.38 | -12.09 |
| | 24 | 3172.36 | 42.01 | 3121.13 | 42.32 | | |
| | 28 | 1449.22 | 39.83 | 1403.87 | 40.14 | | |
| | 32 | 738.63 | 37.18 | 734.02 | 37.61 | | |
| Crew (720P @ 30fps) | 20 | 13940.43 | 43.92 | 13748.74 | 43.92 | 0.15 | -5.99 |
| | 24 | 5208.97 | 41.26 | 4930.55 | 41.22 | | |
| | 28 | 2105.70 | 39.32 | 1862.11 | 39.28 | | |
| | 32 | 1046.22 | 37.46 | 896.46 | 37.40 | | |
| Averaged | | | | | | 0.27 | -12.64 |

(b) RDOMFS SearchRange = 16

| Sequence | QP | UMHexgonS SearchRange = 64 Bitrate (kbit/s) | UMHexgonS SearchRange = 64 PSNR (dB) | RDOMFS SearchRange = 16 Bitrate (kbit/s) | RDOMFS SearchRange = 16 PSNR (dB) | BD-PSNR (dB) | BD-Bitrate (%) |
|---|---|---|---|---|---|---|---|
| Jets (720P @ 30fps) | 20 | 5878.38 | 42.80 | 5692.01 | 42.96 | 0.25 | -16.73 |
| | 24 | 1095.55 | 41.07 | 1168.38 | 41.45 | | |
| | 28 | 420.29 | 39.53 | 472.03 | 40.11 | | |
| | 32 | 244.46 | 37.82 | 279.28 | 38.39 | | |
| Raven (720P @ 30fps) | 20 | 7988.62 | 44.30 | 7764.14 | 44.55 | 0.37 | -11.70 |
| | 24 | 3172.36 | 42.01 | 3134.19 | 42.32 | | |
| | 28 | 1449.22 | 39.83 | 1410.45 | 40.14 | | |
| | 32 | 738.63 | 37.18 | 738.08 | 37.62 | | |
| Crew (720P @ 30fps) | 20 | 13940.43 | 43.92 | 14004.43 | 43.94 | 0.09 | -3.44 |
| | 24 | 5208.97 | 41.26 | 5078.56 | 41.23 | | |
| | 28 | 2105.70 | 39.32 | 1931.65 | 39.28 | | |
| | 32 | 1046.22 | 37.46 | 930.42 | 37.42 | | |
| Averaged | | | | | | 0.24 | -10.62 |

different $M$ may have different impact to data reuse ratio and hardware cost. After step A, we load another row of reference data, that is, step B. In step C, we should shift our search location leftward by $M$ pixel, rather than load $MN$ reference pixels, we only need to propagate the loaded data from RRA and load only the missing row ($M$ pixels) in $M$ clock cycles. So the bandwidth in step D is decreased from $N$ pixel per clock (p/c) to 1 p/c and the ratio of memory access saving is $(N-1)/N$. The last step is same as step B. When reaching the search window bottom along the "Snake" scan order, we can just shift to rightward by 1 pixel and do an inverse direction of A, B, C and D. This procedure will iterate until all searching candidates in the search window have been calculated.

By applying proposed scan order and RRA, the data reuse during the search process can be maximized, and can make hardware utilization approximate to $100\%$.

### 3.3. PE design

In this section, we use a sub-PE array with a RRA ($M = 2$), as shown in Fig. 5, to illustrate the PE design. Each sub-PE array is consisted of 16 single PEs, and each single PE stand for certain pixel in the current block. The red arrows stand for the leftward direction reference data path, blue arrows for rightward direction , green arrows for upward direction and black solid arrows for downward direction. So each single PE should have a MUX to select reference data from four possible directions. The

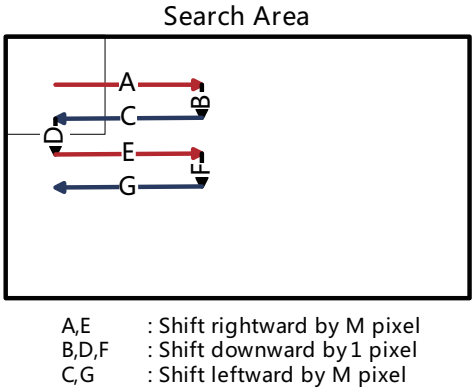



**Fig. 4**. Snake scan order

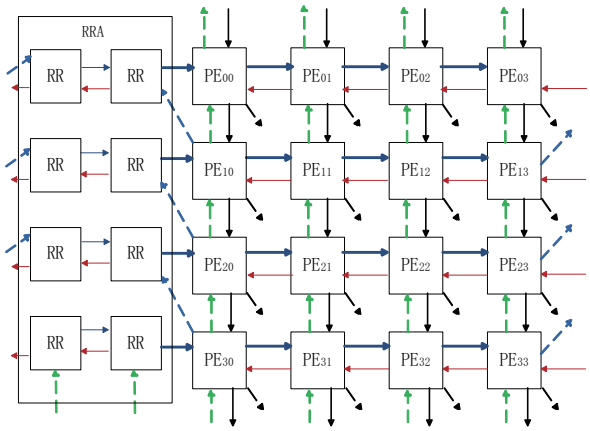



**Fig. 5**. Sub-PE array and RRA of M = 2

blue dashed arrows stand for the reference data propagation to RRA and the black dashed arrows stand for every pixel position SADs propagate to 2DAT.

The proposed architecture can load one row/column pixels (up to 16) in one clock cycle, so it take three clock cycles to load the 16 pixels and then the reference pixels are propagated to the PE array and take another three clock cycles, then we start the search with "Snake" scan. The step A may take 2 clock cycles ($M = 2$); at the same time, PE array load one column each clock from right ports and each PE propagates the reference data to the left direction, then the $PE_{10}, PE_{20}$ and $PE_{30}$ may pass the previous reference data to RRA. And one row pixel data is loaded from bottom ports in one clock. During the step C, only two pixel data should be loaded to RRA rather than eight pixels in two clock cycles, and the RRA would propagate the stored reference data to right direction.

## 4. IMPLEMENTATION RESULTS

The proposed ME hardware architecture is implemented in VHDL, verified by simulation using Modelsim XE 6.3c, synthesized and mapped to a low cost Xilinx XC3S1500-5 FPGA with Xilinx ISE 10.1. The proposed hardware works at 100MHz and consumes 8923 slices. The reconfigurable systolic PE array with the adder tree consumes 7413 slices.

The number of clock cycles per MB required by the proposed hardware depends on the search window size. Starting an iteration has a start-up cost of 15 clock cycles to load the current block pixels, which is called iteration latency, and three more clock cycles to load first location's reference data. At the same time, MVP is generated by the left, top and top-left best MVs which do not consume extra clock cycles. With the help of the RRA and "Snake" order, we could obtain 41 SADs every clock and do not have latency when searching in the search window. So if we set the search window with size $m \times n$, the total number of clock cycles per MB required to complete a search pattern is only relevant to total search locations $mn$. The performance of proposed ME hardware for several search patterns are calculated based on this equation and shown in Table 2 and we use the settings that $M = 4$ and $m, n = 16$ for an example.

As shown in the Table 2, in the same FPGA and for the same MB size, [7] implements a enlarged NTSS [1], uses almost same hardware resources working at 130MHz in which consumes 9083 slices and 7510 slices for PE array and AT; [8] implements FTS [3] and uses 6142 CLBs and can work at 74MHz. Compared with these two designs, our architecture can achieve equal or better data throughput and can achieve very high data reuse ratio which limits the bandwidth about $40\%$ lower. And the most important point is that, rather than implement some zero bias fast ME method, the proposed hardware implement an MVP biased ME method, which can be easily cooperated with existing H.264 video coding standard and achieve better RD performance and shown in Fig. 3.

## 5. CONCLUSION

In this paper, a hardware friendly MVP biased fast full search algorithm and corresponding hardware implementation architecture have been proposed. The simulation results show that the proposed ME method can achieve almost same RD performance with H.264 default fast full search method. The proposed architecture can efficiently implement the proposed ME method, and also achieve good data reuse ratio. The proposed ME hardware is implemented on a low cost Xilinx FPGA and it is capable of processing HD high frame rate video formats in real time.

## 6. REFERENCES

[1] Reoxiang Li, Bing Zeng, and M.L. Liou, "A new three-step search algorithm for block motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 4, pp. 438–442, Aug 1994.

[2] Shan Zhu and Kai-Kuang Ma, "A new diamond search algorithm for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 9, no. 2, pp. 287–290, Feb 2000.

[3] M. Rehan, P. Agathoklis, and A. Antoniou, "Block-based motion estimation using an enhanced flexible tri-

**Table 2**. Performance of proposed architecture

| Architecture | ME Category | Required clock (cycles/MB) | Slices/(CLBs) for PEs | Slices/(CLBs) for Design | Bitwidth (bits/cycle) | Frame size & rate |
|---|---|---|---|---|---|---|
| Proposed (M=4) | MVP biased | 287 | 7413/(N/A) | 8923/(N/A) | 74.6 | 1080P@38fps |
| [7](parttern A) | Zero biased | 633 | 7510/(N/A) | 9083/(2271) | 128 | 1080P@25fps |
| [8] | Zero biased | 202 | N/A/(N/A) | N/A/(2725) | 128 | 1080P@45fps |


angle search algorithm," in *Electrical and Computer Engineering, 2005. Canadian Conference on*, May 2005, pp. 269–272.

[4] M. Ghanbari, "The cross-search algorithm for motion estimation [image coding]," *Communications, IEEE Transactions on*, vol. 38, no. 7, pp. 950–953, Jul 1990.

[5] A.M. Tourapis, O.C. Au, M.L. Liou, et al., "Predictive motion vector field adaptive search technique (PMVFAST)-enhancing block based motion estimation," in *proceedings of Visual Communications and Image processing*. Citeseer, 2001, vol. 2001.

[6] Z. Chen, J. Xu, Y. He, and J. Zheng, "Fast integer-pel and fractional-pel motion estimation for H. 264/AVC," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 264–290, 2006.

[7] O. Tasdizen, H. Kukner, A. Akin, and I. Hamzaoglu, "A high performance reconfigurable motion estimation hardware architecture," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, April 2009, pp. 882–885.

[8] M. Rehan, M. Watheq El-Kharashi, P. Agathoklis, and F. Gebali, "An fpga implementation of the flexible triangle search algorithm for block based motion estimation," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, 0-0 2006, pp. 4 pp.–.

[9] Yu-Wen Huang, Tu-Chih Wang, Bing-Yu Hsieh, and Liang-Gee Chen, "Hardware architecture design for variable block size motion estimation in mpeg-4 avc/jvt/itu-t h.264," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, May 2003, vol. 2, pp. II–796–II–799 vol.2.

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

[11] Gisle Bjontegaard, "Calculation of average PSNR differences between RD-Curves," *ITU-T SG16 Q.6 Document*, vol. VCEG-M33, Austin, April 2001.

[12] Gisle Bjontegaard, "Improvements of the BD-PSNR model," *ITU-T SG16 Q.6 Document*, vol. VCEG-AI11, Berlin, July 2001.