Scalable and Low Cost Design Approach for Variable Block Size Motion Estimation (VBSME)

H. Parandeh-Afshar, P. Brisk, and P. Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
Email: {hadi.parandehafshar, philip.brisk, paolo.ienne}@epfl.ch

ABSTRACT

Variable block size motion estimation (VBSME) in state-of-the-art video coding standards is one of the key features which improves the coding efficiency significantly compared to the previous standards. VBSME hardware design is a challenging task due to its complexity. The processing power requirement for VBSME depends on many factors such as frame size, frame rate and search area. In video coding standards these features are allowed to vary, depending on the requirements of the application. In this paper, a scalable and low cost approach is proposed for designing the VBSME which allows us to tailor the architecture for different applications requirements and implementation targets efficiently. This approach can be used in redesigning of current VBSME architectures to improve their scalability and reduce their design costs. Moreover, as this technique is not block size dependent, it can be employed in designing future coding standards with different block sizes.

INTRODUCTION

H.264/AVC [11] is a standard capable of providing good video quality at substantially lower bit rates than previous standards with the expense of more complex hardware. *Variable Block Size Motion Estimation (VBSME)* is one of the key features of H.264 standard which improves video quality and coding efficiency. To perform *Motion Estimation (ME)*, each frame is partitioned into blocks of pixels namely *Macro Block (MB)* and each MB is predicted from different corresponding MBs in the reference frame. The reference MB can be varied within a window in the reference frame. The reference MB which minimizes the *Sum of Absolute Differences (SAD)* of pixels is selected for the predication.

For a given sub-block, let *c(i, j)* be the pixel at location *i, j* in the current frame, and *r(i, j)* be the same pixel in the reference frame. For a *P*×*Q* sub-block, the *SAD* value is:

$$SAD_{P,Q}(m,n) = \sum_{i=0}^{P-1}\sum_{j=0}^{Q-1}\left|c(i,j) - r(i+m, j+n)\right| \qquad (1)$$

(m, n) denotes the motion vector. A *Motion Vector (MV)* specifies the position of the selected reference MB compared to the position of the current MB. Previous standards typically use *Fixed Block Size Motion Estimation (FBSME)*, while H.264 gives the ability to dynamically choose what block size will be used to represent the motion. When coding a video, the use of larger blocks can reduce the number of bits needed to represent the MVs, while use of smaller blocks can result in a smaller amount of prediction residual information to encode. In VBSME, each MB is divided into several overlapping blocks. For i.e. H.264 MB which is 16x16 pixel block is divided into seven sub-block types: *4×4*, *4×8*, *8×4*, *8×8*, *8×16*, *16×8*, and *16×16*, as shown in Fig. 1.

Due to the regular data dependency of motion estimation, systolic arrays are generally used for efficient implementation of FBSME and VBSME. A systolic array is composed of regular data processing elements (PE). Each PE shares data with its neighbors and more parallelism is obtained with fixed memory bandwidth. For low-power portable devices, 1-dimensional (1D) systolic array implementations of VBSME have been proposed in the past [6, 7, 12]; 2D arrays have also been proposed for high-end application domains, such as HDTV [1, 3], where the power budget is less of a concern; this paper focuses on the former.

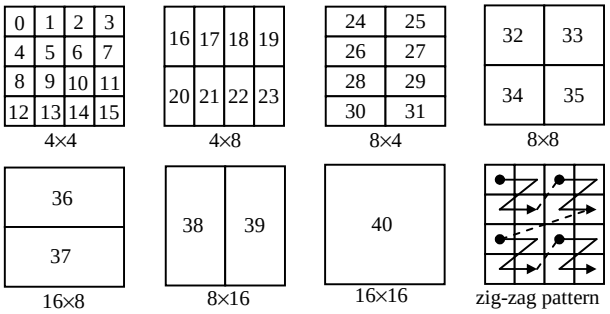


Fig. 1. Different sub-blocks of a 16×16 MB for H.264/AVC.

Generally FBSME architectures are extended to support VBSME. A FBSME engine is used to compute the SADs of primitive (smallest) sub-blocks within each MB. Primitive SADs can be combined to compute the non-primitive (larger) SADs. Thus, the key to an effective implementation of VBSME is to maximize the re-use of pre-computed SADs; however, doing so efficiently is a challenging task at the forefront of research on hardware implementations of VBSME.

Motivation and Contribution

We classify FBSME-based VBSME architectures into two distinct groups. In the first class, different PEs in parallel compute the different primitive SADs within a MB for a specific reference MB and the resulting SADs are stored in an external unit. These SADs are computed based on Eq. 1 and by a circuit which is called *FB* in this paper. Non primitive SADs are computed by an adder tree which re-uses the pre-computed primitive SADs. The intermediate registers and the adder tree is called *SAD Merge Tree (SMT)*. This architecture is shown in Fig. 2(a). The SMT is connected to a central comparator for computing the minimum SAD for each sub-block size. The comparator unit keeps the minimum SAD of each sub-block size which has been computed since that time.

In the second class, as shown in Fig. 2(b), each PE in the array is augmented with a *reuse unit (RU)* that contains additional registers for storing previously computed SADs, and adders for computing non-primitive SADs. A *SAD Bus Network (SBN)* transmits SADs to a centralized comparator in order to compute the minimum SAD (SAD_{MIN}) for each sub-block size. The RU also contains a complicated controller unit that handles resource sharing, scheduling, and bus allocation. A precise and complex schedule is required to efficiently use the SBN and the comparators. In contrast to the first method, here, each PE exclusively computes the whole SADs of the MB with a specific reference position. Therefore, different PEs simultaneously compute the SADs of each MB considering different

reference positions. The SAD of each sub-block is sent to the comparator unit whenever it is ready. As at each time slot, each PE finishes the SAD computation of a different sub-block, there should be enough bandwidth to send different SADs computed by different PEs to the comparator at the same time.

Neither of these approaches is scalable. If more PEs are added to the design, the whole SMT or SBN must be redesigned, including the control mechanism. This can easily become unmanageable when the number of PEs increases substantially. On the other hand, adding more PEs imposes a significant hardware overhead due to the need to redesign either the SMT or SBN. Prior VBSME architectures have been optimized for a fixed set of features and implementation targets and are not generally scalable. To increase the processing power, a complete redesign of the architecture is required which will be very costly in several aspects.

To address these concerns, this paper describes a systematic approach to design scalable VBSME architectures. The proposed approach is very cost efficient and can be used in the design of [5, 6, 7, 12] VBSME engines. As a case study, this approach is used to design a one-dimensional (1-D) systolic array VBSME architecture for H.264/AVC. Compared to prior 1-D architectures, the architecture proposed here has a higher clock frequency, reduced *macro block processing time (MBPT)*, and significantly reduced silicon area and power consumption.

RELATED WORK

The vast majority of architectures for VBSME in H.264/AVC are based on 1-D and 2-D systolic arrays. In this work, we focus on 1-D arrays. 2-D arrays, such as the one by Kim et al. [5] are beyond the scope of this work.

One of the first 1-D VBSME architectures for H.264/AVC was presented by Yap and McCanny [12], and later improved upon by Song et al. [7]. Both of these architectures reuse the results of smaller sub-block computations with an irregular workflow inside each PE. Moreover, both architectures produce multiple SADs in the same clock cycle. As such, SAD bus networks are necessary to send these SADs to a centralized comparator.

An improved architecture by Song et al. [6] employs a 1-D array and reuses SADs computed for smaller sub-blocks in the computation of SADs for larger ones. Each PE group consists of 16 PEs and process 16 pixels per cycle. Consequently, this architecture has a high memory bandwidth requirement. Moreover, the PE group structure is complex and requires a complex controller; once again, a SAD bus network is required.

Chen et al. [1] proposed several 2-D array architectures for VBSME. Both architectures use an SAD merge tree to compute larger SADs by reusing SADs computed by smaller 4×4 sub-blocks; this replaces the SAD bus network used by previous designs, but does not sidestep its general overhead. This circuit has since been fabricated for use in high-end HDTV applications [4]. A smaller SAD merging scheme was proposed by Cho et al. [2], but it still has considerable hardware overhead; moreover, it has an irregular structure and is not scalable.

To date, we are aware of three FPGA implementations [4-5, 10] of VBSME. Wei and Gang [10] built a 1D systolic array VBSME architecture, similar to those described above, with 16 PEs, a 4-stage adder tree, and two flexible (16×16 bit) 8-bit register arrays. López et al. [5] designed a similar 1D array using SRAMs to implement the RUs shown in Fig. 2(b). Both of these architectures employ an SBN, and suffer from the affects outlined above.

Li and Leong [4] proposed a bit-serial architecture for FPGA-based VBSME. Their architecture achieves high throughput and a high clock frequency; however, it uses an SMT, and therefore has a complicated controller and is not easily scalable; the controller must be redesigned in order to add extra PEs.

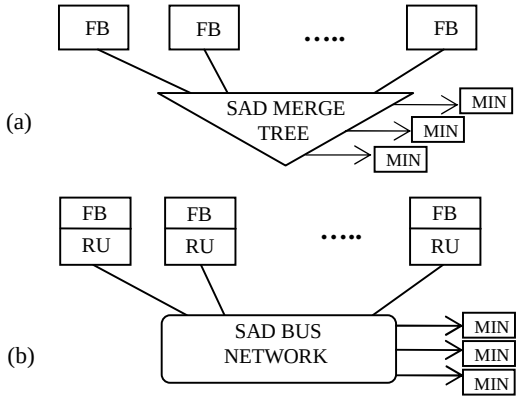
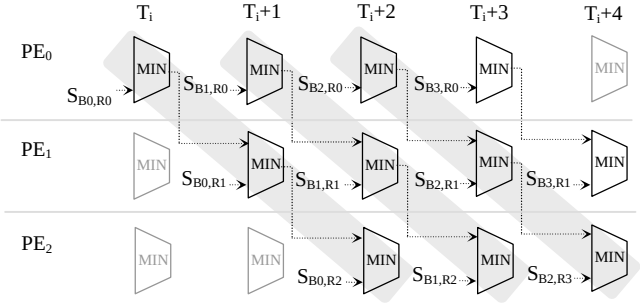


Fig. 2. Classification of 1-D VBSMEs.



T_i : Time slot i **$S_{Bi,Rj}$**: SAD of Block i w.r.t. Reference j

Fig. 3. Propagation of $SADs_{MIN}$ through PEs. Each PEs chain (highlighted) propagates the SAD_{MIN} of a different block.

PROPOSED DESIGN SCHEME

Overview

Here, we propose a new architecture for VBMSE that does not require a SAD merge tree or a SAD bus network. We retain the RU from Fig. 2 (b), but place a *comparison unit (CU)* in each PE, along with the FB and RU. The CU includes a set of parallel comparators, followed by a set of registers. The CU finds minimum SADs generated by different PEs in a distributed manner, as shown in Fig.3. The basic idea is to propagate SADs computed by one PE to the next PE in the array. Similar to the second method discussed earlier in Fig.2 (b), each PE exclusively computes the SADs of all sub-blocks within a MB considering a specific reference MB. Therefore, different PEs in parallel compute the complete SADs within the same MB for different reference MBs. As each PE is one clock cycle prior to its neighbor PE, in consecutive cycles, SADs for a specific block considering different reference MBs are computed by the PEs in the array. Through careful scheduling, the SAD computed by the previous PE in the previous clock cycle can be compared with the SAD computed by the FB block in the current PE in the current clock cycle. If done properly, this eliminates the need for either a SAD merge tree or a SAD bus network.

Smaller SADs, resulting from the comparisons, are propagated to the next PE in the array and the larger ones are thrown away. Using this strategy, each PE receives the SAD_{MIN} computed thus far, and compares it against its own generate SAD. This process repeats for each sub-block size within each MB; the result is the final SAD_{MIN}.

Processing Element (PE) Design

Fig. 4 (a) illustrates the new PE architecture. The FB unit, in the upper left, uses absolute value (ABS) units to compute the $|C(k, l)$ –

$R(i + k, j + l)|$ values of primitive blocks in parallel, which are then accumulated by an adder tree, producing a SAD. The ABS units, which compute the quantity $|A - B|$, are implemented using a circuit described by Vassiliadis et al. [9]. The adder tree is implemented using a compressor tree [3, 10].

The RU, in the lower left, consists of a register file (RF) and an adder. The RF stores previously computed SADs for re-use purposes; SADs for smaller sub-blocks are then added together to produce SADs for larger sub-blocks. The CU, shown on the right, compare SADs stored in the current PE's RF with SADs propagated from the previous PE. The smaller SAD is then stored into a register and propagated to the CU of the next PE during the next clock cycle.

Fig. 4 (b) shows the 1-D systolic array organization of each PE. The *Min SAD Register File (MSRF)* at the end of the array stores the SAD_{MIN} values for each sub-block computed by the array.

Under this scheme, there is no need for a SAD bus network or a SAD merge tree. The comparators are distributed inside the PEs, and SAD_{MIN} values for each sub-block can be read back into the array. When a SAD enters the chain from the MSRF, it passes through the PEs in consecutive cycles. Since there are generally more search regions than PEs in the array, for each MB, each PE must compute the SAD for multiple search regions. Consequently, it is necessary to compare a SAD for a search region computed previously with a SAD for the currently computed search region. This process repeats until SADs for all search regions have been computed and SAD_{MIN} has been found.

This 1-D array architecture is also scalable. Additional PEs can be added to increase performance without any additional delay. The only area overhead is the area of the PE itself. In contrast, the size of a SAD merge tree would increase, and a SAD bus network might become wider and require a more complicated controller.

A smart design of the CU improves the performance and reduces the number of registers in the RU. Let B be the number of different block sizes inside of each MB and T be the number of cycles required to compute each SAD. We define the *transfer rate*, Tr as follows:

$$Tr = \left\lceil \frac{B}{T} \right\rceil \qquad (2)$$

Tr specifies the number of parallel comparators in each PE. Two cases are discussed separately, as follows:

T < B: To achieve 100% utilization, each PE should process each MB within a search region in T cycles. Since $B > T$, Tr comparators are necessary in order to compare and transfer the B SADs to the next PE within T cycles. A larger RF in the RU is also necessary in order to store these SADs for re-use.

T >= B: As long as each SAD is computed in a distinct clock cycle, one comparator suffices for the CU, and at most one SAD is transferred to the next PE per cycle. The size of the RF varies on the regularity of the dataflow of SAD generation, but the RF size is generally smaller than the case above.

In both cases, if B SADs are generated uniformly across the T cycles, a relatively smaller RF size can be achieved in both the RU and CU. Moreover, regular dataflow can help simplify the controller. In general, if a SAD is stored in the register file for re-use, it is beneficial to generate the SADs with which it will be combined as quickly as possible, in order to free up the register to hold future SADs.

SAD Computation Scheduling

As discussed previous, the SAD computation of the primitive blocks is performed by the FB subsystem of the PE. Then, these computed SADs are used by the RU for computing the non-primitive SADs. Therefore, several registers are required for storing the

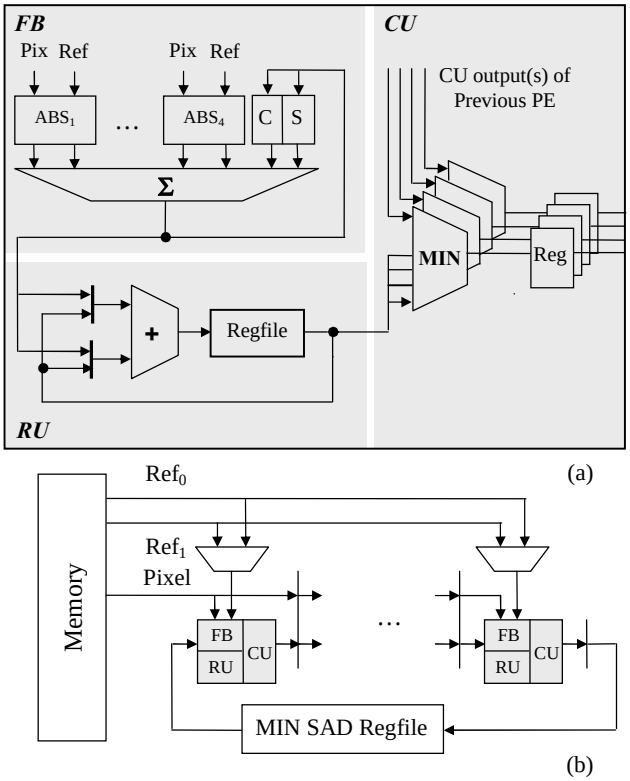


Fig. 4. Proposed PE architecture (a) and array organization (b).

primitive SADs. The RU unit can start to compute SADs of non-primitive blocks as soon as the required primitive SADs comprising the non-primitive ones are ready. This will free registers for other primitive SADs that are computed. A proper scheduling of the SAD computation of primitives will lead to significant reduction in the number of registers. Here we propose a zig-zag pattern for this scheduling. This pattern is shown in Fig. 1. The proposed zig-zag sequence also helps to evenly distribute the SAD computation for larger sub-blocks among the whole cycles required for all SADs to be computed. This achieves a highly regular workflow, which, in turn, simplifies the design of the controller. The zigzag sequence is also used for the processing of non-primitive blocks which comprise larger blocks.

CASE STUDY: H.264 VBSME ARCHITECTURE

The proposed approach can be generalized to any type of VBSME algorithm. For the experiments, we implement VBSME of H.264/AVC based on the aforementioned approach. In the case of H.264/AVC, $B = 41$, and the architecture designed here computes each SAD in $T = 64$ cycles.

The PE architecture is similar in principle to Fig. 4 (a). Four PEs are placed in the array. Each PE concurrently processes four pixels of each primitive block. Four cycles are required to compute the SAD of each primitive block, and 64 cycles are required to process all primitive blocks. The computation of SADs for the largest sub-blocks in the current MB is interleaved with the computation of the SADs of the first primitive blocks in the subsequent MB. The register-file contains 11 registers.

The zig-zag pattern also helps to evenly distribute the SAD computation for larger sub-blocks among the 64 cycles. This achieves a highly regular workflow, which, in turn, simplifies the design of the controller. The 64-cycle makespan can be divided into 4 identical sub-spans of 16 cycles. As such, the controller is implemented as a 6-bit saturating sequential counter.

VLSI Implementation

The VBSME architecture was modeled in VHDL and synthesized using Synopsys Design Compiler and Artisan Memory Compiler. The design was realized using 0.13µm and 0.18µm CMOS standard cell libraries to compare to prior work that has been published using these technologies. The 0.13µm design contains 18K gates and runs at 400MHz; the 0.18µm design contains 12K gates and runs at 285 MHz.

Let *SR* be the search range, *C* be the number of cycles required per MB, T_{CLK} be the clock period, and *N* be the number of PEs in the array. Then *MB Processing Time (MBPT)* is:

$$MBPT = \frac{SR \times C \times T_{CLK}}{N}, \qquad (3)$$

under the assumption of 100% PE utilization, which is satisfied by this architecture.

Table 1 compares our VBSME architecture to comparable 1-D systolic array architectures that have been published in the past; the numbers for our competitors are taken from the references. With four PEs, compared to 16 used by others, our VBSME architecture has a lower *MacroBlock Processing Time (MBPT)*, operates at a higher frequency, and has a smaller gate count. The reason for this success is the elimination of the SAD bus network and the ability to use fewer PEs. Yap and McCanny's design [12] consumed approximately 3× more power than ours. The papers by Song et al. [6, 7] did not report power consumption.

FPGA Implementation

In this section, we compare our VBSME architecture with several other FPGA designs that fully support H.264/AVC.

The architectures of Li and Leong [4] and López et al. [5] were synthesized on *Virtex 2* series FPGAs. Table 2 shows a comparison between our architecture, also synthesized on a Virtex 2, and both of these. Among these architectures, ours was the smallest in terms of LUT count, and it offers the highest throughput-per LUT. In terms of throughput and clock frequency, however, the bit-serial architecture of Li and Leong [4] achieves a higher clock frequency and throughput. Also Table 2 compares our architecture to that of Wei and Gang [10], which was synthesized on the *APEX* series FPGA. Their architecture achieves greater throughput and runs at a higher clock frequency; our architecture, in contrast, is much smaller and our throughput-per-LUT is approximately 2.5× greater than theirs. Both [4, 10] have a big SMT in their designs which makes them non-scalable and it is very costly to add extra PEs to their architectures.

Table 3 shows the scalability study of the architecture on the Stratix II, increasing the number of PEs from 4 to 8. For each PE that was added, we observed frequency degradation from 1 to 6 MHz, which is to be expected for FPGAs; due to omnipresent routing delays in FPGAs, frequency degradation is unavoidable.

Table 1. Comparison among different 1-D systolic array VLSI implementations of VBSME in H.264/AVC.

	0.13 µm		0.18 µm		
	[12]	Ours	[6]	[7]	Ours
PEs	16	4	16	16	4
MBPT(µs)	13.9	10.2	17.4	17.9	14.3
Freq. (MHz)	294	400	266	228	285
K-Gate Count	61	18	51.7	21	12
Power (mw)	24	7.7	-	-	-

Table 2. Comparison among different 1-D systolic array FPGA implementations of VBSME in H.264/AVC.

Architecture	Virtex-II			APEX	
	[4]	[5]	Ours	[10]	Ours
Area (LUTs)	3345	19576	1431	7381	1878
Freq. (MHz)	340	51.49	154	120	75
Throughput (MB[16x16]/Sec)	18674	3036	9615	7324	4695
Throughput/LUT	5.59	0.155	6.7	0.99	2.5

Table 3. Stratix II scalability study.

PE Count	Frequency (MHz)	Area (ALMs)
4	255	752
5	253	929
6	252	1102
7	251	1283
8	245	1465

CONCLUSION

In this paper, a scalable design scheme for variable block size motion estimation (VBSME) was proposed. Using the proposed design scheme we showed that the SAD bus network and SAD merge tree which are used in the conventional VBSME architectures can be eliminated. This reduces the design costs and increases the scalability of VBSME architectures; the processing power of such architectures can be improved by adding more PEs with minimum delay and area overhead and also without having to redesign the whole architecture. Using the proposed methodology, we designed the H.264/VAC VBSME. In the presented circuit, due to the elimination of the SAD bus network, a significant reduction in area and power consumption was obtained and the performance was improved by localizing the communication between the comparators within each PE.

REFERENCES

[1] C-Y. Chen, et al., "Analysis and architecture design of variable block size motion estimation for H.264/AVC", *IEEE Trans. Circuits and Systems—I: Regular Papers, Vol. 53, No. 3*, Feb., 2006, 578-593.

[2] C.Y. Cho, S.Y Huang, J.S. Wong, "An Embedded Merging Scheme for H.264/AVC Motion estimation," *IEEE Int. Conf on Image Processing*, vol. 3, pp. 1016–1019, Sept, 2005.

[3] Y-W. Huang, et al. "A 1.3TOPS H.264/AVC Single-Chip Encoder for HDTV Applications", *IEEE International Solid State Circuits Conference*, San Francisco, CA, USA, February 6-10, 2005, 128-129, and 588.

[4] B. M.H. Li, and P. H.W. Leong, "FPGA-based MSB-first bit-serial variable block size motion estimation processor," *Proc. FPT,* pp. 165-172, Dec. 2006.

[5] S. López, F. Tobajas, A. Villar, V. de Armas, J. F. Lopez, and R. Sarmiento, "Low cost efficient architecture for H.264 motion estimation," in *Proc. of the IEEE Int. Symp. Circuits and Systems*, vol. 1, May, 2005, pp. 412-415.

[6] Y. Song, Z. Liu, S. Goto, and T. Ikenaga, "Scalable VLSI Architecture for Variable Block Size Integer Motion Estimation in H.264/AVC", *IEICE Trans. Fundamentals, Vol. E89, No. 4*, April, 2006, 979-988.

[7] Y. Song, Z. Liu, T. Ikenaga, and S. Goto, "A VLSI Architecture for Variable Block Size Motion Estimation in H.264/AVC with Low Cost Memory Organization", *IEICE Trans. Fundamentals, Vol. E89, No. 12*, December, 2006, 3594-3601.

[8] S. Vassiliadis, E. A. Hakkennes, J. S. S. M. Wong, and G. G. Pechanek, "The Sum-Absolute-Difference Motion Estimation Accelerator", *24th Conference on EUROMICRO, Vol. 2*, Vesteras, Sweden, August 25-27, 1998, 20559-20566.

[9] C. S. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Trans. Electronic Computers, Vol. 13*, 1964, 14-17.

[10] C. Wei, M.Z. Gang, "A Novel SAD Computing Hardware Architecture for Variable-size Block Motion Estimation and Its Implementation with FPGA," *on Proc. 5th Intl. Conf. On ASIC*, vol.2, pp. 950-953, Oct. 2003.

[11] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE Trans. Circuits and Systems for Video Technology, Vol. 13, No. 7,* July, 2003, 560-576.

[12] S. Y. Yap, and J. V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation," *IEEE Trans. On Circuits and Systems—II: Express Briefs, Vol. 51, No. 7*, July, 2004, 384-389.