

# A SCALABLE ARCHITECTURE FOR VARIABLE BLOCK SIZE MOTION ESTIMATION ON FIELD-PROGRAMMABLE GATE ARRAYS

*Theepan Moorthy and Andy Ye*

Department of Electrical and Computer Engineering  
Ryerson University  
350 Victoria Street, Toronto, Ontario, Canada M5B 2K3  
[tmoorthy@ee.ryerson.ca](mailto:tmoorthy@ee.ryerson.ca), [aye@ee.ryerson.ca](mailto:aye@ee.ryerson.ca)

## ABSTRACT

The flexibility of Field-Programmable Gate Arrays (FPGAs) encourages design reuse and can greatly enhance the upgradability of digital systems. This flexibility is particularly useful in the design of highly flexible video encoding systems that can accommodate a multitude of existing standards as well as the rapid emergence of new standards. In this paper, we investigate the use of FPGAs in the design of a highly scalable Variable Block Size Motion Estimation (VBSME) architecture for the H.264/AVC video encoding standard. The scalability of the architecture allows one to incorporate the system into low cost single FPGA solutions for low resolution encoding applications as well as into high performance multi-FPGA solutions targeting high-resolution video encoding applications. To overcome the performance gap between FPGAs and Application Specific Integrated Circuits (ASICs), our algorithm intelligently increases its parallelism as the design scales while minimizing the use of memory bandwidth. The core computing unit of the architecture is implemented on FPGAs and its performance is reported in this paper. It is shown that the computing unit is able to achieve real-time 40 fps performance for 640x480 resolution VGA video while incurring only 4% device utilization on a Xilinx XC5VLX330 (Virtex-5) FPGA. With 8 computing units (at 36% device utilization), the architecture is able to achieve real-time 45 fps performance for encoding full 1920x1088 progressive HDTV video.

*Index Terms*— Variable Block Size Motion Estimation, H.264/AVC, Field-Programmable Gate Arrays

## 1. INTRODUCTION

The Variable Block Size Motion Estimation (VBSME) algorithm is an essential part of the H.264/AVC video-encoding standard. Relative to Fixed Block Size Motion Estimation algorithms, VBSME provides much higher compression ratios and picture quality. VBSME algorithms, however, are much more computationally expensive. In particular, the H.264/AVC standard calls for up to 41

motion vectors for each macroblock and its corresponding subblocks. Due to this high computing demand, many hardware architectures have been proposed to accelerate the computation of VBSME motion vectors for H.264/AVC [1]–[8]. All of the architectures, however, have been implemented in Application Specific Integrated Circuit (ASIC) technology. Except for limited commercial implementations [9]–[11], little information exists on how these algorithms would perform on reconfigurable technologies such as Field-Programmable Gate Arrays (FPGAs). In particular, the FPGA implementation presented in [12] specifically targets portable multimedia devices with CIF-level resolution and cannot be easily scaled. On the other hand, the FPGA implementation presented in [13] only reaches VGA-level resolution and 27 frames per second performance. It too cannot be scaled. In this work, we propose a scalable hardware VBSME architecture based on the Propagate Partial SAD architecture [8] and measure its performance on FPGAs as the design scales.

The use of FPGAs encourages design reuse and can greatly enhance the upgradability of digital systems. The programmability of FPGAs is particularly useful for highly flexible encoding systems that can accommodate a multitude of existing standards as well as the emergence of new standards. In particular, our design can be incorporated into single FPGA solutions targeting low cost low-resolution applications as well as into multiple FPGA designs for high performance high-resolution applications.

The proposed architecture is based on one of the three widely used VBSME architectures — the Propagate Partial SAD [1] [8], SAD Tree [7], and the Parallel Sub-Tree [6]. The Propagate Partial SAD architecture was selected due to its unique blend of efficiency and scalability. While the SAD Tree architecture has the highest performance amongst the three [7], it, however, requires the support of a complex array of shifting registers that must have the capability of shifting in both horizontal and vertical directions. This array, while efficient to implement in ASICs, consumes a large amount of FPGA resources. On the other hand the Parallel Sub-Tree architecture is the most compact design amongst the three. The architecture, however, inherently does not scale well for high performance applications [6].

As proposed in [1] and [8], the Propagate Partial SAD architecture processes a single group of 16 reference blocks at a time. Our design enhances the original design by allowing it to be scaled to process several groups of 16 reference blocks simultaneously. These groups are selected in a manner that shares a large amount of their reference pixels. This sharing minimizes the increase in memory bandwidth as the design scales and makes high performance FPGA-based design feasible.

The remainder of this paper is organized as follows: Section 2 introduces the general Motion Estimation algorithm and the Propagate Partial SAD architecture, Section 3 presents the scalable VBSME architecture, Section 4 evaluates its performance, and section 5 concludes.

## 2. HARDWARE MOTION ESTIMATION

Video encoding algorithms typically process one 16x16 block of pixels (called a macroblock) at a time. The frame that contains the macroblocks currently being processed is referred to as the current frame. During the encoding process, the goal of Motion Estimation (ME) is to find the best match for a macroblock from a set of reference pixels (where this set is called a search window, and the frame that contains this search window is called a reference frame). To this end all ME algorithms accomplish this goal through three distinct stages of computation. First the macroblock is mapped onto a 16x16 block of pixels (called a reference block) in the search window, and the absolute difference values between the macroblock pixels and the corresponding reference block pixels are calculated. Second, the Sum of the Absolute Differences (SAD) value is calculated for the reference block by summing absolute difference values over the entire block. This process repeats until a SAD value is calculated for each of the reference blocks in the search window. Thirdly, the minimum of all the SAD values in the search window is computed and the corresponding reference block is used by the encoder to calculate the best-match Motion Vector (MV) for the macroblock currently being processed.

Equation 1 and 2 show the arithmetic for calculating the SAD value of a reference block such that pixel (x, y) in the macroblock is mapped to pixel (rx + x, ry + y) in the search window.

$$SAD(rx, ry) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} |R(rx + x, ry + y) - C(x, y)| \quad (1)$$

$$rx \in [0, RW - 16], ry \in [0, RH - 16] \quad (2)$$

In the equations, W and H represent the width and height of the macroblock. RW and RH represent the width and height of the search window. C(x, y) represents the value of pixel (x, y) in the macroblock while R(rx + x, ry + y)

represents the value of pixel (rx + x, ry + y) in the search window. Note that surpassing (vertically) the search window size set in [8], this paper assumes a search range of +/- 16 pixels in both the horizontal and vertical directions (i.e. RW=48 and RH=48).

Instead of just calculating one SAD per macroblock/reference-block pair, VBSME algorithms subdivide a 16x16 macroblock into a set of subblocks. Correspondingly, the reference block is also divided into subblocks and SAD values are then calculated for each of the subblocks in addition to the macroblock. In particular, as shown in Figure 1, the H.264/AVC standard subdivides a macroblock into 40 subblocks of size 16x8, 8x16, 8x8, 8x4, 4x4, and 4x4. Consequently, for a macroblock, 41 SAD values are needed per reference block.

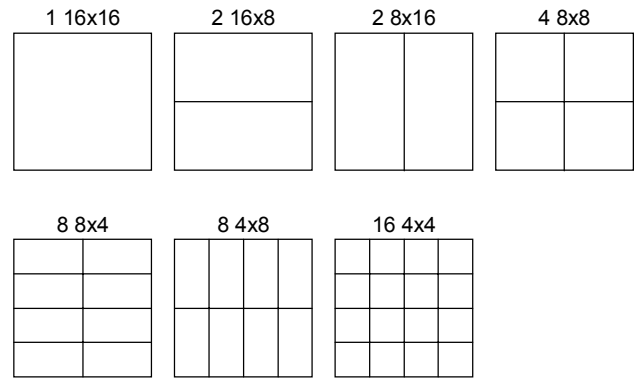


Figure 1: Macroblock and Subblocks in VBSME

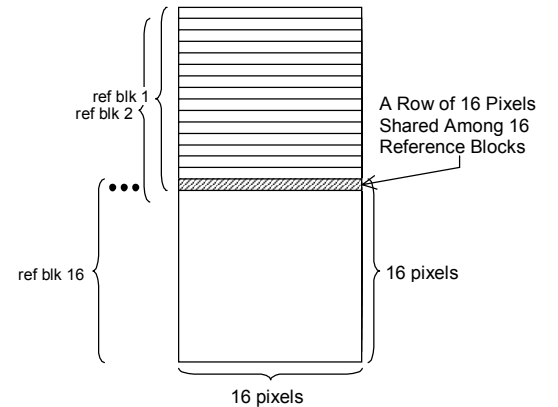


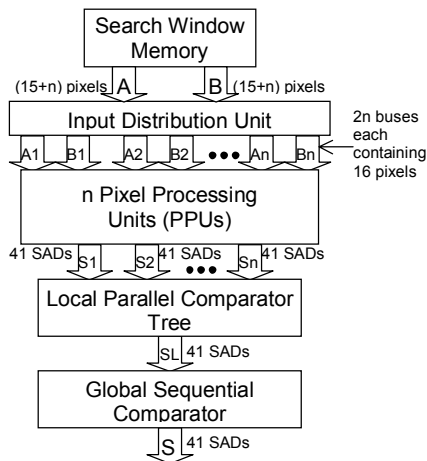
Figure 2: A Group of 16 Reference Blocks

The Propagate Partial SAD architecture speeds up the computing process of VBSME algorithms by simultaneously calculating SAD values for 16 reference blocks at a time. In particular, the architecture takes advantage of the fact that, in a search window, every vertical group of 16 reference blocks share a common row of 16-pixels (as shown in Figure 2). In the Propagate Partial SAD architecture, this common row is then used to simultaneously calculate 16 absolute difference values for each of the 16 reference blocks. A specialized pipeline

structure is then used to accumulate these absolute difference values to produce the 41 SAD values per reference block at every clock cycle.

### 3. SYSTEM ARCHITECTURE

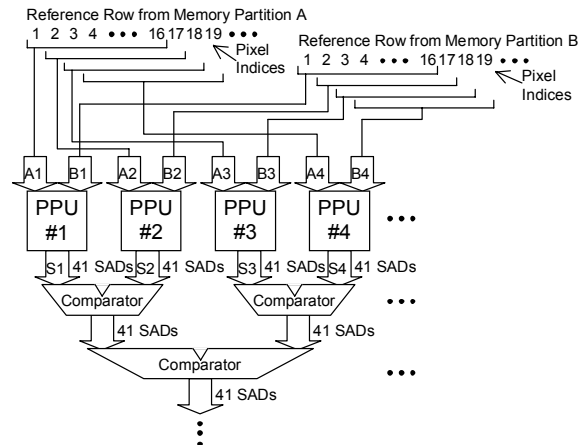
The overall structure of the scalable VBSME architecture is shown in Figure 3. It consists of a bank of memory that stores the search window, an input distribution unit,  $n$  Pixel Processing Units (PPUs), and two sets of comparators. As in [8], the memory storing the search window is divided into two partitions. Each partition contains an output of  $15+n$  pixels. These outputs are expanded into  $2n$  buses by the input distribution unit, where each bus contains 16 pixels. The  $2n$  buses are then fed into  $n$  PPUs, which have been initialized with a macroblock's pixel values. The PPUs are then used to produce  $n \times 41$  SAD values at each clock cycle. These  $n \times 41$  SAD values are then used to compute the minimum SAD values of the search window in two steps. First the  $n \times 41$  SADs are fed into the local parallel comparator tree. This tree computes 41 minimum SAD values from its  $n \times 41$  inputs. These local minimum SAD values are then forwarded to the global sequential comparator, which determines the 41 minimum SAD values for the entire search window. Note that the global comparator is of a conventional less-than comparator design [8] and the scaling of the VBSME architecture does not affect its complexity.



**Figure 3: The Scalable VBSME Architecture**

The detailed design of the input distribution unit, the PPUs, and the local parallel comparator tree is shown in Figure 4. As shown, the core of the scalable VBSME architecture is the PPUs, which are based on the Propagate Partial SAD architecture. As discussed in Section 2, each PPU produces 41 SAD values (corresponding to an entire set of SADs for a single reference block) at every clock cycle. The number of PPUs utilized in the scalable architecture, therefore, corresponds directly to the number of reference blocks that can be processed in a clock cycle

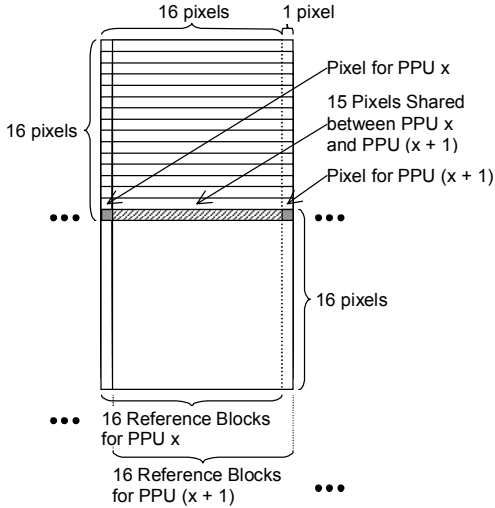
and the overall performance of the system. However, as the number of PPUs increases, the output bandwidth required for the search window memory increases as well. In particular, in order to keep a PPU fully utilized during motion estimation, one would require two rows of 16-pixels to be forwarded from the search window memory to the PPU at every clock cycle (one row from each of the search window memory partitions) [8]. Typically a byte is used to encode a pixel, therefore one needs to transport 32 bytes from the search window to a PPU in every clock cycle.



**Figure 4: Input Distribution Unit, PPUs and Local Comparators**

A naive approach would be to simply increase the output of the search window memory by 32 bytes for every additional PPU. However, this can quickly exhaust the internal memory bandwidth of an FPGA (if the search window is stored on the same chip as the PPUs) or the IO pin limit of even the largest modern FPGAs (if the search window is stored off chip). For example, the Xilinx XC5VLX330 is the largest device that Xilinx currently offers. It contains 1200 available IO pins. Assume that the search window is stored off chip. Implementing a single PPU on the XC5VLX330 would require 256 input pins. Implementing four PPU copies would require 1024 pins (over 85% of the available IOs on the XC5VLX330) – leaving an insufficient number of IOs for output and control signals.

More importantly, the above approach does not take into account the large number of pixels that are shared among the reference blocks. For example, Figure 5 shows 32 reference blocks in a search window. These blocks are divided into two groups where each group contains 16 reference blocks. Within a group, the reference blocks are organized as in Figure 2, where all blocks are contained within a single 16-pixel wide column and one block is offset from the next by a single row of pixels.



**Figure 5: Sharing of Pixels among PPUs**

As in Figure 2, 16 reference blocks from the same group share a row of 16 common pixels. Furthermore, since one group is offset from another by a single column of pixels, all 32 blocks in figure 5 share 15 common pixels.

To increase performance, these two groups can be simultaneously processed by two PPUs (shown as PPU  $x$  and PPU  $(x+1)$  in the figure). Since 15 pixels are shared between the groups, one would require 17 pixels (instead of 32) to be read from the search window at a time. In particular, if pixels  $(a, y)$ ,  $(a+1, y)$ , ...,  $(a+15, y)$  of the search window are being processed by PPU  $x$ , pixels  $(a+1, y)$ ,  $(a+2, y)$ , ...,  $(a+16, y)$  should be simultaneously processed by PPU  $(x+1)$ .

In general, to fully utilize  $n$  PPUs, where  $n \leq 33$ , in a  $48 \times 48$  search window, one would require  $(15 + n)$  pixels to be read from each partition of the search window memory for every clock cycle. These signals should then be distributed using the topology shown in Figure 4.

At its output, each PPU shown in Figure 4 produces 41 SAD values at every clock cycle. These SAD values amount to 573 bits of data. To keep the output width constant as the number of PPUs increases, the local parallel comparator tree can be implemented on the same FPGA as the PPUs. Note that the number of comparator tree stages is equal to  $\lceil \log_2(n) \rceil$  where  $n$  is the number of PPUs that the architecture contains. We observe that by registering the values produced at each stage of the comparator tree one can ensure that the comparator tree does not become the critical path of the system. Consequently the overall system performance does not degrade significantly when an increasing number of PPUs are used. Note that, as shown in Table 1 there is a drop in clock speed of 1 to 2 MHz when the number of PPUs is increased from 1 to 4 units. This degradation is due to a slight increase in routing delay as the size of the comparator tree grows and is not due to any increase in logic delay.

When targeting an FPGA with a moderate number of user-available IO pins, the scalable system shown in Figure 4 may still become an IO bottleneck. Consider the case of a system scaled to 8 PPUs. With each pixel encoded using a single byte, the input pixels will amount to 46 Bytes (16 bytes from each partition of the search window memory for the initial PPU followed by 1 extra byte from each partition for the 7 additional PPUs). The output will consist of 41 SAD values (irrelevant to the number of PPUs used) and would consume around 72 bytes of IO. When control signals are considered, the total IO requirement of the circuit shown in Figure 4 becomes 119 bytes.

On devices where such a number of IOs is not available, the on-chip RAM blocks available on most modern FPGA devices can be utilized to buffer the search window. For a search window of  $48 \times 48$  pixels, this translates to 2304 bytes of data per search window. Using double buffering, while the current search window is being processed, another 2304 bytes of on-chip memory can be utilized to receive the next search window, hence greatly reducing the overall number of required IOs.

#### 4. EXPERIMENTAL RESULTS

To evaluate the performance and area efficiency of the scalable VBSME architecture, we implemented five variations of the design shown in Figure 4 on a Xilinx XC5VLX330 FPGA. Each design contains 1, 2, 4, 8, or 16 PPUs. As the design scales, the target resolution scales as well from VGA ( $640 \times 480$ ) to High-Definition (HD) Video ( $1920 \times 1088$ ).

These designs are implemented in Verilog and synthesized using the Xilinx Synthesis Tool (XST) in the Xilinx Integrated Software Environment (ISE). The synthesis constraints are set to maximize speed. All designs meet the IO constraint of XC5VLX330 with 70%, 71%, 74%, 79%, and 90% IO utilization, respectively. The performance and area of each implementation is summarized in Table 1.

**Table 1: Area and Performance Results**

# of PPUs	Area				Performance		
	LUTs		DFFs		Target Resolution	Freq. (MHz)	fps
	# (K)	%	# (K)	%			
1	9	4	4	1	640x480 (VGA)	200.6	40
2	19	8	6	2	800x608 (SVGA)	199.0	50
4	38	18	10	4	1024x768 (XVGA)	198.3	61
8	77	36	18	8	1920x1088 (HD Video)	198.3	45
16	154	74	35	16	1920x1088 (HD Video)	198.3	91

Column 1 of the table lists the number of PPUs in the design. Columns 2 and 3 lists the number of LUTs required for the design and the number of LUTs required as a percentage of the total number of LUTs in the FPGA, respectively. The same values are summarized in column 4 and 5 for DFFs. Finally column 6 lists the target resolution of each design. The maximum operating frequencies of the

circuits are shown in column 7 and their corresponding frame-per-second (fps) performances are shown in column 8.

The fact that the circuit performance remains consistently near 200 MHz as the design scales from 1 to 16 PPU's offers much promise for FPGA-based H.264 motion estimation especially as future resolutions are scaled beyond HD Video. Table 1 shows that real time motion estimation performances can be achieved with 1, 2, 4, and 8 PPU's for the resolutions of VGA, SVGA, XVGA, and HD Video, respectively. It also shows that with 16 PPU's and beyond one can achieve real time motion estimation performance for resolutions that are beyond HD Video.

Note that the frame-per-second performances in column 8 of Table 1 are calculated based on the following formula:

$$\frac{(frequency \times n)}{(c\_per\_Frame \times refFrames)} \quad (3)$$

where frequency is the maximum operating frequency of a circuit, n is the number of PPU's used in the circuit, c\_per\_Frame is the total number of cycles it takes to process all of the macroblocks in a current frame, and refFrames is the number of reference frames that a macroblock must be compared to. In this work refFrames is always set to 4, as is required for full H.264 compatibility.

In particular, c\_per\_Frame is defined as:

$$\sum_1^3 (n\_type\_MBs \times c\_per\_type\_MB) \quad (4)$$

where n\_type\_MB's is the number of macroblocks that exist in the current frame resolution for a specific type of macroblock (out of three types), and c\_per\_type\_MB is the number of cycles it takes to process that specific type of macroblock.

For our fps calculations, a macroblock is classified as one of three types of macroblocks according to the region(s) it occupies within a current frame. These three types are full search, border search, and corner search macroblocks. The position of a macroblock within a current frame limits the size of its search window, and the search window size determines the number of reference blocks that need to be compared to a macroblock.

For example, a macroblock located in the top-right corner of a current frame will only have an available search range of 16 pixels to its left and to its bottom in the reference frame (17x17 reference blocks). Such a macroblock is classified as a corner search macroblock. This is in contrast to macroblocks located in the centre of a current frame which will have a full search range of +/- 16 pixels in both the vertical and horizontal directions (33x33 reference blocks). Such macroblocks are classified as full search macroblocks. Similarly macroblocks running along the vertical and horizontal edges of a current frame (and not

being one of the 4 corner macroblocks) will have their search window constricted on one side by one of the current frame's four borders (33x17 reference blocks). These macroblocks are classified as border search macroblocks.

The expressions used to calculate the n\_type\_MB's values for each of the three macroblock classifications, given a specific frame resolution, are provided below.

*Total macroblocks in a current frame :*

$$(fc \times fr) \div 256$$

*corner search macroblocks :*

$$4 \text{ (constant)}$$

*full search macroblocks :*

$$(fc - 32) \times (fr - 32) \div 256$$

*border search macroblocks :*

$$\text{Total macroblocks} -$$

$$(\text{full search macroblocks} + 4 \text{ corner macroblocks})$$

where fc and fr are the number of columns of pixels and the number of rows of pixels of a current frame, respectively, and 256 is the number of pixels contained in a macroblock. For example, a frame of full 1088p HD resolution video corresponds to a frame size of 1920 (fc) columns of pixels by 1088 (fr) rows of pixels. Therefore, the frame contains 8160 macroblocks in total with 4 corner search, 7788 full search, and 368 border search macroblocks.

Recall (from Section 2) that the calculations for a single reference block are completed by a PPU in every clock cycle. Thus the c\_per\_type\_MB values have a direct one-to-one relation to the number of reference blocks that need to be compared for each of the three types of macroblocks as explained above. Therefore the c\_per\_type\_MB values are 1089 (33x33), 561 (33x17), and 289 (17x17) for the full search, border search, and corner search macroblocks respectively.

## 5. CONCLUSIONS

Based on a survey of present FPGA-based H.264 VBSME architectures ([12]-[13]), the proposed architecture is the first to reach HD-level real time performances. We found that the architecture is able to perform real time (45 fps) H.264 Motion Estimation on 1920x1088 progressive HD video and is capable of being scaled for higher resolutions. The performance is measured with four reference frames, and a search window size of 48x48 pixels. When scaled for HD-level performance, the architecture utilizes 77 K LUTs and 18 K DFFs (with 8 processing units), and has a maximum clock frequency of 198 MHz when implemented on a Xilinx XC5VLX330 (Virtex-5) FPGA. Furthermore, the scalability of the architecture makes it suitable for FPGA-based applications where the upgradeability and flexibility of the video encoder are essential requirements.

## 6. REFERENCES

- [1] Y. Huang et. al, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proceedings of the 2003 International Symposium on Circuits and Systems*, Vol. 2, pp. 25-28, May 2003.
- [2] S. Yap and J. V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation," *IEEE Transactions on Circuits and Systems II*, Vol. 51, No. 7, pp. 384-389, July 2004.
- [3] 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," *Proceedings of the 2005 conference on Asia South Pacific design automation*, pp. 631-634, 2005.
- [4] Y. Song et. al, "Scalable VLSI Architecture for Variable Block Size Integer Motion Estimation in H.264/AVC," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 4, pp. 979-988, April 2006.
- [5] Y. Song et. al, "VLSI Architecture for Variable Block Size Motion Estimation in H.264/AVC with Low Cost Memory Organization," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 12, pp. 3594-3601, December 2006.
- [6] Z. Liu et. al, "A Fine-Grain Scalable and Low Memory Cost Variable Block Size Motion Estimation Architecture for H.264/AVC," *IEICE Transactions on Electronics*, Vol. E89-C, No. 12, pp. 1928-1936, December 2006.
- [7] T. Chen, et. al, "Analysis and Architecture Design of an HDTV720p 30 Frames/s H.264/AVC Encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, No. 6, pp. 673-688, June 2006.
- [8] Z. Liu et. al, "Hardware-Efficient Propagate Partial SAD Architecture for Variable Block Size Motion Estimation in H.264/AVC," *Proceedings of the 17th Great Lakes Symposium on VLSI*, pp. 160-163, 2007.
- [9] W. Chung, "Implementing the H.264/AVC Video Coding Standards on FPGAs," *Xilinx Broadcast Solution Guide*, pp. 18-21, September 2005.
- [10] "Faraday H.264 Baseline Video Encoder & Decoder IPs: FTMCP210/FTMCP220," *Faraday Technology Corporation Product Documentation*, 2005.
- [11] "H.264 Motion Estimation Engine (DO-DI-H264-ME)," *Xilinx Corporation Product Documentation*, October 2007.
- [12] S. Lopez, et. al, "Low Cost Efficient Architecture for H.264 Motion Estimation," *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 412-415, May 2005.
- [13] S. Yalcin et. al, "A High Performance Hardware Architecture for an SAD Reuse Based Hierarchical Motion Estimation Algorithm for H.264 Video Coding," *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*, pp. 509-514, August 2005.