

Design Considerations of SOPC-Based H.264/AVC Systems

Wei Zhao, Charles Castello, Jeffrey Fan

Department of Electrical and Computer Engineering, Florida International University, Miami, Florida, USA

Abstract — The primary advantage of the H.264 video compression technique is the ability to provide good video quality at substantially lower bit rates compared with other video compression methods. This is due to the use of variable block sizes, multiple reference frames, and the consideration of Rate-Distortion Optimization (RDO). However, these features come with the price of added complexity due to the introduction of Motion Estimation (ME) and Mode Decision (MD) in the design of H.264. This paper aims to compile a collection of research to aid designers in applying the H.264 technology to a variety of different applications. These research topics include complexity reduction, vector edge detection, and added efficiency in H.264 architectures. Complexity is reduced by a cost-effective coding algorithm for removing ME redundancy in System-on-a-Chip (SOPC)-based embedded systems. Improvement in vector edge detection is accomplished by a cost-effective method which stores all vector information within a frame and places said information in the Laplacian of Gaussian Operator. Lastly, an efficient architectural design is reviewed in analyzing the bandwidth of each component in an H.264 design. The entire system bandwidth is decomposed into several modules with predictable coefficients, thereby aiding designers in understanding the real cost of each hardware component.

I. INTRODUCTION

For today's media needs, video data is being digitally compressed due to storage and transmitting bandwidth limitations [1]. Therefore, improved methods of video compression are needed, particularly *H.264/Advanced Video Coding (AVC)* [2], which is capable of achieving improved video quality under certain bit rate constraints. This is possible due to three key features employed to make *H.264* more efficient in terms of performance in comparison to older standards such as *Motion Pictures Experts Group (MPEG)-1/2/4*, and *H.263*. These features include the adoption of variable block sizes, multiple reference frames, and the consideration of *RDO* as a non-normative tool within the codec for *ME* schemes [3][4][5][6].

These new features however add a considerable increase in terms of encoder complexity, mainly with regards to *ME* and *MD*. Constructing an efficient *ME* engine to support variable block sizes and multiple reference frames has become an essential issue that breaks the key bottleneck of coder design for *H.264* compliant circuits. Academic and industrial research and development has been carried out for the problem of *ME* complexity reduction resulting in novel algorithmic and

architectural solutions that focus on the *ME* tasks of implementing context-aware co-processors in real-time for low-power embedded systems [7]. A *Field-Programmable Gate Array (FPGA)*-based *3-Dimensional ME (3D-ME)* architecture significantly reduces the number of search candidates thereby producing a gate count suitable for *System-on-a-Chip (SOC)* designs utilizing *Digital Signal Processing (DSP)* processors [8]. Lastly, a *Sum of Absolute Differences (SAD)* reuse method is based on a three level hierarchical pyramid *ME* algorithm for high performance hardware architectures utilizing *FPGA* implementations [9][10].

Another consideration in addition to complexity reduction is vector edge detection techniques, which are utilized to give simple estimations for search windows in *ME* [3][4][5][6]. These methods are used to intuitively select the search window. The edge of the vector can easily show the edge of an object if moving in a certain direction when all objects in the video are moving simultaneously. There are some issues however with existing techniques, such as *Luminous Edge Detection* [11][12]. Edge recognition in this technique suffers from high levels of light or shade. Methods with better edge recognition quality are needed to ensure more accurate estimations.

The last aspect is efficient *H.264* architecture models for *SoC* designs. As mentioned previously, limitations in storage and transmitting bandwidth create a need for video compression techniques, which become more complicated as the compression ratio increases. Sophisticated video standards cause high complexity of the hardware design making the entire system (i.e. *SoC* design) more unpredictable. According to the instruction profiling with the *HDTV1024P* (2048x1024, 30fps) specification, *H.264/AVC* decoding process requires 83 Giga-Instructions Per Second (GIPS) computation and 70 Giga-Bytes Per Second (GBPS) memory access. In *H.264/AVC* encoder, up to 3,600 GIPS and 5,570 GBPS are required for the *HDTV720P* (1280x720, 30fps) specification [13].

Although many excellent works on *H.264* integer motion estimation schemes have been proposed [3][4][5][6], there still exists a wide gap for hardware to match with the efficiency. The software solution is always considered the better solution in terms of cost, when ideally, there are no computing time limitations. Realistically however, there exist applications with time constraints (i.e. real-time) where hardware solutions are favoured. The *H.264* hardware-software co-existing solution is a hot topic in recent research where undoubtedly, the implementation utilizes *SoC* designs. This is shown through increased popularity of the resource demanding *High Definition (HD)* standard.

With the three considerations described above, previous research techniques are reviewed to comply with complexity reduction, vector edge detection, and efficient *H.264* architectures. Complexity reduction is accomplished by utilizing a *SAD* reuse principle which is a cost-effective approach for removing *ME* redundancy and capable of realization with a loosely coupled accelerator's topology in *SOPC*-based embedded systems. Vector edge detection is achieved through the use of two data vectors instead of one, which is the case in the *Luminous Edge Detection* technique. Two separate matrices are used for better quality in edge recognition by convoluting both matrices by the *Laplacian of Gaussian Operator*. Efficient *H.264* architectural designs are actualized by calculating the bandwidths of each individual component for the *SoC* chip model which improves the efficiency of the overall system design cycle.

This paper starts by first discussing the background of the *H.264* followed by the complexity reduction method in [14]. The vector edge detection technique from [15] is reviewed and next, the modular bandwidth estimation from [16]. Lastly, the paper is concluded. It is with the hope of this paper to inform the reader on three important points that greatly affect the overall efficiency and robustness of *H.264* designs.

II. H.264/AVC BACKGROUND

Figure 1 is a simple demonstration of the *H.264* based *SoC* architecture. It contains a *Central Processing Unit (CPU)* (i.e. *ARM9*), *BUS* system (i.e. *AMBA bus*), and accessories allocated to the system. This basic illustration reveals a fundamental understanding of the overall *SoC* bandwidth construction where the different modules shown in Figure 1 are:

- *C* stands for the *CPU* software cost of the application. Usually, companies should have an estimate of *C* costs for specific applications (i.e. *MP3* detectors). Thus, these costs will not be listed in the prediction table.

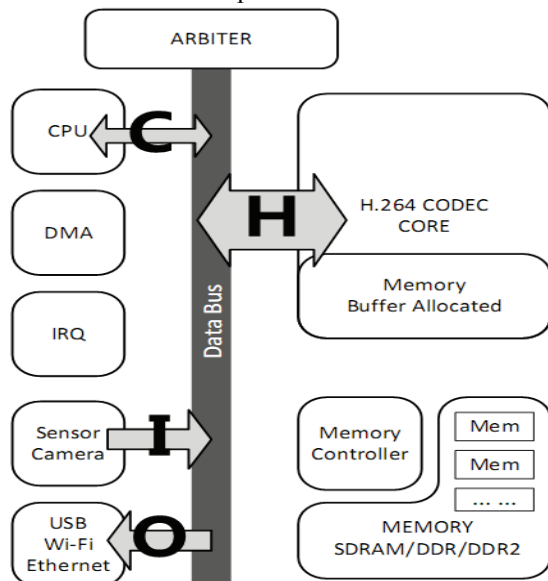


Figure 1 - H.264 Based SoC Architecture [16]

- *I* and *O* stand for the Input (i.e. sensor) and Output (i.e. *USB* controller, Ethernet).
- *H* is the *H.264* I/O bandwidth. Different algorithms have different bandwidth requirements where the designer may have to consume more build-in logic or memory resources in order to satisfy bandwidth requirements of certain applications.

III. COMPLEXITY REDUCTION

The complexity reduction methodologies will now be discussed from [14] where a cost-effective algorithm is used to remove *H.264* *ME* redundancy in *SOPC*-based embedded systems.

A. Complexity Reducing Algorithms

Variable block size *ME* allows a 16x16 Macro-Block (*MB*) to be partitioned into smaller sub-blocks. From these sub-blocks, there are as many as 41 *MV*'s to be determined in the *MD* process [10]. The selection of efficient algorithms is essential in reducing the computational cost for variable block sizes in *ME*. Therefore, an *SAD* reuse algorithm is presented below.

1) Motion Estimation and Mode Decision Algorithms

Based on an *RDO* framework to find the optimal *MV* among all sub-block modes, the reference software for *H.264* is presented in [19]. The best selection of the mode can be determined by finding the minimal *Lagrangian Cost* [3][9], which is calculated for both *ME* and *MD* as:

$$J_m(d) = SAD_M(d) + \lambda_m \cdot R(d - p), \quad (1)$$

where,

- *M* is a *MB* mode of size $m \times n$, $(m, n) \in \{(4, 4), (4, 8), (8, 4), (8, 8), (16, 8), (8, 16), (16, 16)\}$.
- SAD_M is the *SAD* for block type *M*.
- λ_m is the *Lagrangian Multiplier* for *ME*.
- $d = (d_x, d_y)$ is the current *MV* being considered.
- $p = (p_x, p_y)$ is the *MV* prediction used by *H.264* during the coding process.
- $R(d-p)$ is the bit rate spent for coding the *MV* difference information.

and *SAD* is computed as:

$$SAD_M(d) = \sum_{x=0, y=0}^{m-1, n-1} |c(x, y) - r(x + d_x, y + d_y)|, \quad (2)$$

where $c(\cdot)$ and $r(\cdot)$ represent the video data of current and reference frames respectively.

2) Complexity Reducing Principles

The *SAD Reuse Algorithms* will now be discussed. First, let (i, j) be a 2-D index for a 4x4 block mode where the sub-block partition is constructed from the 16x16 block mode in *H.264*. This limits the range of i and j from zero to three where the *SAD* of the 4x4 block located at the (i, j) position is denoted as $SAD_{4 \times 4}^{(i, j)}(d)$ and shown in Figure 2. The *SAD* of the 4x4 block mode is equal to:

$$SAD_{4 \times 4}^{(i,j)}(d) = \sum_{y=4i}^{4i+3} \left(\sum_{x=4j}^{4j+3} |c(x,y) - r(x+d_x, y+d_y)| \right), \quad (3)$$

$(0 \leq i < 4, 0 \leq j < 4)$

According to the definition, the *SAD* reuse mechanism can be derived from Equation (2) by using the common data of $SAD_{4 \times 4}^{(i,j)}(d)$.

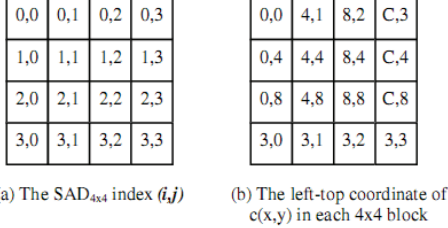


Figure 2 - The $SAD_{4 \times 4}$ Index and the Corresponding Coordinate [14]

The *SAD Reuse Algorithm* reuse terms must first be determined. For an *H.264 MB* mode of block size $m \times n$, the candidate *MV* d is located within the search range $[-R, R]$. There exists a $(m/4) \times (n/4)$ number of corresponding *SAD* reuse terms that are composed with a block mode of size 4×4 where each value is based on the basic 4×4 *SAD* computation. The *SAD* reuse function is calculated by:

$$SAD_M(d) = SAD_{m \times n}(d) = \sum_{i=0}^{\frac{m-1}{4}} \sum_{j=0}^{\frac{n-1}{4}} (SAD_{4 \times 4}^{(i,j)}(d)), \quad (4)$$

where $(m/4)$ and $(n/4)$ represent the number of row blocks and column blocks for size 4×4 sub-blocks that can be reused, respectively. The total number of *SAD* reuse terms for a given candidate *MV* can be simply calculated as $(m/4) \times (n/4)$. For example, given mode block size 4×8 , $SAD_{4 \times 8}(d)$ is calculated as:

$$\begin{aligned} SAD_M(d) &= SAD_{4 \times 8}(d) = \sum_{i=0}^0 \sum_{j=0}^1 (SAD_{4 \times 4}^{(i,j)}(d)) \\ &= SAD_{4 \times 4}^{(0,0)}(d) + SAD_{4 \times 4}^{(0,1)}(d), \end{aligned} \quad (5)$$

B. SOPC-Based Complexity Reducing Coder

There are two types of speedup mechanisms that can be used in *SOPC*-based embedded systems [20], namely custom instruction and custom accelerator. The custom instruction is a type of tightly coupled topology and the custom accelerator is a type of loosely coupled topology. Due to the outstanding efficiency, the custom accelerator topology is the preferred consideration for the implementation of a *H.264* video coder. The platform-based reference design with *Soft-CPU* cores and standard peripherals construct a fundamental paradigm of the *SOPC*-based embedded system. The design accommodates a memory resource consumption property, therefore requiring the use of external *Synchronous Dynamic Random Access Memory (SDRAM)*. However, the data bandwidth requirement is another bottleneck for the implementation of most *ME* algorithms. For smoothing the data pumping rate, the interfacing architecture of a distributed buffer with multiple-ports and memory controller is introduced.

To design a cost-effective *SAD* reuse architecture, the search range and reuse methodology for *ME* must first be considered. A *MB* with the search range $[-p, p]$ will contain $(2p+1)^2$ search locations. Hence, the basic *SAD* of block size 4×4 has a $[-4, 4]$ search range having $(2*4+1)^2=81$ search locations. According to Equation (5), if a row-wise data transfer is used, the $SAD_{1 \times 4}$ can be computed during every inward data shift. Four sets of said mechanism can be computed in parallel simultaneously to process four rows of data. The results of four corresponding rows of $SAD_{1 \times 4}$ can be summed together resulting in a $SAD_{4 \times 4}$ which is stored for further use. The *Processing Element (PE)* used for such computations can be designed with a behaviour of computing $SAD_{4 \times 4}$ results, the combination of *SAD* for *MD*, the determination of *MV*, etc.

IV. VECTOR EDGE DETECTION

Vector edge detection techniques for the *H.264* implementation will now be discussed from [15]. A cost effective method is used to calculate the vector edge for the current frame. All of the vector information is stored within the frame itself and put into the *Laplacian of Gaussian Operator* to detect edges by utilizing two separate matrices of angle and length.

The original format of a vector value which has been calculated by the *ME* module is formatted as X by Y pixels. The current *MB* is from the matched reference *MB*. In order to classify a *MV*, X and Y would have to be quantized to angle and length values of the vector which are calculated by:

$$Angle = \tan^{-1} \left(\frac{Y}{X} \right) \quad \& \quad Length = \sqrt{\left(\frac{Y}{16} \right)^2 + \left(\frac{X}{16} \right)^2} \quad (6)$$

The *tangent* or *square-root* functions however, are unacceptability complex in hardware design. Thus, two assumptions below are used to make both calculations fit into the hardware implementation. First, an angle equal to a 16^{th} of a circle is needed. There are two methods to accomplish this: dividing evenly or making an approximation. An approximation can be easily calculated in the hardware due to the approximation function. The second assumption is stated as:

$$Length = \frac{Y}{\cos(Angle)} \quad (7)$$

where $(y/16)$ can be used to determine the *Angle* value and border of the two objects that have the same angle of movement.

The *Edge Detection Function* [21] has been used in the *Digital Signal Processing* field, where edge detection can be classified into two categories: first-derivative and second-derivative edge detect operators. *Roberts*, *Prewitt*, and *Sobel* [21] are three popular first-derivative edge detection operators while the *Laplacian of Gaussian Operator* is a typical second-derivative operator.

1) Laplacian Operator

A second-derivative operator can detect the edge horizontally and vertically whereas the first-derivative operator can only detect one, but not both. The *Laplacian* of a 2-D function [22] $f(x, y)$ is defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (8)$$

Two digital approximations can be generated from the *Laplacian* equations, which are most useful in practice and shown as:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8) \quad (9)$$

and

$$\nabla^2 f = 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_6 + z_7 + z_8 + z_9) \quad (10)$$

2) Laplacian of Gaussian Operator

A *Laplacian of Gaussian Operator* may detect edges as well as noise (isolation of out-of-range image). Therefore, it may be desirable to smooth the image first by convolution with the *Gaussian* kernel of width σ where:

$$G_\sigma = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (11)$$

So, the *Laplacian of Gaussian* is equal to:

$$\nabla^2 G_\sigma = \frac{\partial^2 G_\sigma}{\partial x^2} + \frac{\partial^2 G_\sigma}{\partial y^2} \quad (12)$$

Let's assume variables x and y are equal in the equation where x is first solved by:

$$\frac{\partial^2 G_\sigma}{\partial x^2} = \frac{x^2 - \sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (13)$$

Now, let $x^2 + y^2 = r^2$ to get:

$$\nabla^2 G_\sigma = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{r^2 - 2\sigma^2}{\sigma^4} e^{-\frac{r^2}{2\sigma^2}} \quad (14)$$

Finally, with the selection of σ , the vector edge detection can now begin. Since the *Laplacian of Gaussian Operator* construct is a hardware implementation, the digital approximation of the equation must be calculated, which is:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (15)$$

The *Laplacian of Gaussian Operator* is a reliable operator that is easily built and quickly computable where the object's vector is very coherent. Therefore, the process of vector edge detection is not as complex as image edge detection. Also, the convolution is less complex on a hardware level rather than software. All that is needed are a group of shift-registers and the vector table.

V. MODULAR BANDWIDTH ESTIMATION

The modular bandwidth estimation scheme for the *H.264 SoC* design will now be discussed. An efficient architectural design is used to analyse the bandwidth of each component in *H.264*, which is shown in Figure 1. All of the *H.264* system bandwidths are decomposed into several modules with predictable coefficients to help designers understand the real cost of each hardware component, thus improving the efficiency of the overall system.

A. System-on-a-Chip Modeling

Based on the *H.264* system application, shown in Figure 1, the system bus bandwidth cost function can be assumed as:

$$B_{sys} = C + I + O + H \quad (16)$$

Since C is the control line from the *CPU*, it exists in every system. As previously mentioned, every company has different C models based on various applications where a universal C value cannot be easily estimated. Therefore, the C value is placed in Equation (16) for an appropriate time where C can be properly ascertained. However, the I , H , and O values are all connected to the aspect of memory. Hence, the bandwidth of memory is calculated as $B_{mem} = I + O + H$.

Specifically, the I value is the input source of the system. In a real-time specific *H.264* application, I stands for the input source of video data from either a sensor or camera. On the other hand, the O component is the compressed output data of the encoder. Generally speaking, O is the result of I compressed with the *H.264* algorithm where O is much smaller than I , in the order of (1/50) or (1/100). As the bandwidth goes extremely high in the *H.264* codec, O can be neglected. Hence, H is the dominant term which affects the bandwidth of the *SoC*. The H bandwidth is described in the following section.

B. H.264/AVC Modeling

Since H is the total bandwidth of the *H.264* module, it is possible to split H into several parts in order to measure the entire bandwidth. It is shown that the input is the sum of the inter-estimate unit inputs which consists of the *ME* and intra-estimate units of the current frame. The output includes two parts: the *Network Abstraction Layer (NAL)* output derived from the *H.264* compression and the reference frame output. Therefore, the total H bandwidth is calculated by:

$$H = H_{in} + H_{inter} + H_{intra} + H_{out} + H_{ref} \quad (17)$$

The H_{in} and H_{out} terms of the equation are the general input and output of the *H.264* algorithm. I and O may not be equal to H_{in} and H_{out} respectively because of the buffer effects in memory. However, bandwidth costs in H_{in} , H_{out} , I , and O are equivalent. The H_{ref} term is used for reference building where the *H.264* codec sends the reference frame to memory. The generation of reference frames is equivalent to the current frame inside the module except in the opposite direction. Hence, the H_{ref} is equal to H_{in} .

1) H_{inter} Bandwidth Estimation

In the *H.264 ME* algorithm, every frame is processed by *MBs* which contain 16x16 pixels. Furthermore, with partitioning an entire frame using *MBs*, the current *MB* can be found which is most likely to be in the reference frames. However, it is very difficult to search for an exact *MB* in an entire frame. Usually, a search window is used which contains many candidate *MBs* in the reference frame. Such search methods fall into two categories: the *Full Search Block Matching Algorithm (FSBMA)* and the *Variable Block Matching Algorithm (VSBMA)*. The *FSBMA* searches for the exact same 16x16 pixel block of the current *MB* in the full search window. Almost all hardware solutions are based on *FSBMA* because of high quality and regular computation. For *FSBMA*, there are several searching methods [7][8][9][10][11][12] which are significant because all said techniques shed light on the weights of hardware cost in order for added efficiency. However, in terms of bandwidth using *FSBMA*, equal bandwidth is intended outside *ME*.

For the *Cache-Min algorithm*, Δ is calculated as:

$$\Delta = 16_{\text{pixel}} \times \frac{1}{256_{\text{MB/pixel}}} = \frac{1}{16} \text{MacroBlock} \quad (31)$$

Under the r reference frames situation, only a $(r/16)MB$ bandwidth is needed. This means that if there were 16 reference frames, only $1X_{\text{spd}}$ of bandwidth would be needed. Since the cache size of the current MB is counted in the inter-mode, the need of the current MB becomes $3*16=48$ pixels.

For the *Bandwidth-Min algorithm*, extra bandwidth is not needed for the intra-prediction. The current MB will be the left MB in the next procedure where all candidate pixels that are needed have already been in the cache. As well, the cache size is relatively small, equal to one line of pixel storage: $1,920+16=1,936$ pixels, assuming a 1080p resolution, with an *SRAM* memory size of less than $4KB$.

VI. CONCLUSION

This paper has reviewed three very important research topics pertaining to the *H.264* coder that include: complexity reduction [14], vector edge detection [15], and efficient architectures [16]. The complexity reduction is accomplished by a complexity reducing algorithm with an *SAD* reuse architecture contributed to the *H.264* coder that has the potential to employ an efficient video encoder in *SOPC*-based embedded systems.

For vector edge detection, a cost-effective method is utilized to calculate the vector edge of the current frame. The result of this technique is very good but several parameters however need adjustments. Within differing frames, the edge power is not equal, meaning adjustments are needed between edges and tolerances where static or dynamic adjustment could be possible in future hardware.

As the screen increases in size, considerations should be taken to increase the MB size. This however, will cause a large architectural change, whether in a software or hardware implementation. However, if the MB size does not change, the equal movement of the same video in lower and higher resolutions will cause a different compression ratio. In order to enlarge the search window, it may be very costly in the hardware implementation. With a large search window, features may be added by vector recognition without confirming changes in the hardware architecture. On the other hand, a larger resolution will have a smoother vector set. This means that the vector can be easily detected.

Lastly, efficient architectures are researched by decomposing the *H.264* bandwidth model into several components based on *H.264* algorithms where two optimization methods are implemented for bandwidth and cache cost analysis. An analytical method is provided for every design to meet application requirements where precise predictions are unavailable. With the approaches described in section V, models can be generated and evaluated simply in the early prototyping stage. In the case of critical bandwidth analysis, the discussed methodologies will help designers and architects to avoid unnecessary cost from potential re-design due to prototyping errors.

REFERENCES

- [1] I.E.G. Richardson, "H.264 and mpeg-4 video compression," pp. 27–28, Aug. 2003.
- [2] 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 14496-40 AVC)," *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, Document JVT-G050, Dec. 2003.
- [3] H.-Y.C. Tourapis and A.M. Tourapis, "Fast motion estimation within the H.264 codec," *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'03)*, vol. 3, no. 3, pp. 517-520, Jul. 2003.
- [4] H. Zhihai and S.K. Mitra, "A unified rate-distortion analysis framework for transform coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1221-1236, Dec. 2001.
- [5] 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 Circuit and Systems for Video Technology*, vol. 13, no. 7, pp. 688-703, Jul. 2003.
- [6] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Transactions on Circuits and Systems I*, vol. 53, pp. 578-593, Mar. 2006.
- [7] M. Melani, L. Fanned, and S. Saponara, "Algorithmic/architectural design for H.264/MPEG-4 AVC low-power video codec," *PhD Research in Microelectronics and Electronics*, vol. 1, pp. 209-212, Jul. 2005.
- [8] G.G. Lee, K.A. Vissers, and B.-D. Liu, "On a 3D recursive motion estimation algorithm and architecture for digital video SoC," *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'04)*, vol. 2, no. 2, pp. 449-451, Jul. 2004.
- [9] H.F. Ates and Y. Altunbasak, "SAD reuse in hierarchical motion estimation for the H.264 encoder," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, no. 2, pp. 905-908, Mar. 2005.
- [10] S. Yalcin, H.F. Ates, and I. Hamzaoglu, "A high performance hardware architecture for an SAD reuse based hierarchical motion estimation algorithm for H.264 video coding," *IEEE International Conference on Field Programmable Logic and Applications*, pp. 509-514, Aug. 2005.
- [11] G. Scognamiglio, A. Rizzi, L. Albani, and G. Ramponi, "Picture enhancement in video and block-coded image sequences," *IEEE Transactions on Consumer Electronics*, vol. 45, no. 3, pp. 680-689, Aug. 1999.
- [12] K. Ishihara, S. Ishihara, and M. Nagamachi, "Object recognition system with perceiving subjective contours," *IEEE International Conference on Neural Networks*, Perth, Australia, pp. 2620-2625, Nov. 27-Dec. 21, 1995.
- [13] T.-C. Chen, C.-J. Lian, and L.-G. Chen, "Hardware architecture design of an h.264/avc video codec," *IEEE 2006*, Aug. 2006.
- [14] R.-X. Chen and J. Fan, "Complexity reduction for SOPC-based H.264/AVC coder via sum of absolute difference," *IEEE/CIE 7th International Conference on ASIC (ASICON'07)*, Guilin, China, pp. 1277-1280, Oct. 26-29, 2007.
- [15] W. Zhao, Z. Luo, J. Fan, and S. Tan, "Vector edge detection in H.264 implementation," *Proceedings of the IEEE ICESSE Symposium on Hardware/Software Optimization for Embedded Systems (ISHSO'08)*, Chengdu, China, Jul. 29-31, 2008.
- [16] R.-X. Chen, W. Zhao, Q. Liu, and J. Fan, "Efficient H.264 architecture using modular bandwidth estimation," *Proceedings of the IEEE 5th International Conference on Embedded Software and Systems (ICESSE'08)*, Chengdu, China, Jul. 29-31, 2008.
- [17] International Telecommunication Union (ITU). (2008) *Recommendation H.264*. [Online]. Available: <http://www.itu.int/rec/T-REC-H.262>
- [18] C. Gomila and P. Yin, "New features and applications of the H.264 video coding standard," *Proceedings of the IEEE International Conference on Information Technology: Research and Education (ITRE2003)*, pp. 6-10, Aug. 2003.
- [19] K. Sühring. (2008) *JVT Reference Software Coordination*. [Online]. Available: <http://iphome.hhi.de/suehring/ttml/>
- [20] Altera Corporation. (2008) *Quartus II Handbook Volume 5: Embedded Peripherals*. [Online]. Available: <http://www.altera.com/literature/lit-nio2.jsp>
- [21] R.C. Gonzalez and R.E. Woods, "Digital image processing," 3rd Edition, Prentice Hall, 2008.
- [22] H.S. Neoh and A. Hazanchuk, "Adaptive edge detection for real-time video processing using FPGAs," vol. 7, no. 3, pp. 2-3, 2004.