Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding

Marcelo Porto[1], André Silva[2], Sergio Almeida[2], Eduardo da Costa[2], Sergio Bampi[1]

[1]Informatic Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
[2]Polytechnique Institute, Catholic University of Pelotas, Pelotas, Brazil
email: msporto@inf.ufrgs.br

ABSTRACT

This paper presents real time HDTV (High Definition Television) architecture for Motion Estimation (ME) using efficient adder compressors. The architecture is based on the Quarter Sub-sampled Diamond Search algorithm (QSDS) with Dynamic Iteration Control (DIC) algorithm. The main characteristic of the proposed architecture is the large amount of Processing Units (PUs) that are used to calculate the SAD (Sum of Absolute Difference) metric. The internal structures of the PUs are composed by a large number of addition operations to calculate the SADs. In this paper, efficient 4-2 and 8-2 adder compressors are used in the PUs architecture to achieve the performance to work with HDTV (High Definition Television) videos in real time at 30 frames per second. These adder compressors enable the simultaneous addition of 4 and 8 operands respectively. The PUs, using adder compressors, were applied to the ME architecture. The implemented architecture was described in VHDL and synthesized to FPGA and, with Leonardo Spectrum tool, to the TSMC 0.18µm CMOS standard cell technology. Synthesis results indicate that the new QSDS-DIC architecture reach the best performance result and enable gains of 12% in terms of processing rate. The architecture can reach real time for full HDTV (1920x1080 pixels) in the worst case processing 65 frames per second, and it can process 269 HDTV frames per second in the average case.

Index Terms: Motion estimation, fast algorithm, HDTV video coding, adder compressors.

1. INTRODUCTION

This paper presents our high performance architecture for Motion Estimation (ME) using efficient 4-2 and 8-2 adder compressors to calculate the Quarter Sub-sampled Diamond Search (DS) block matching algorithm with Dynamic Iteration Control (QSDS-DIC) [1].

Motion estimation (ME) is the most important task in the current standards of video compression. Full Search (FS) is the most used block matching algorithm for hardware implementation of ME, due to its regularity, easy parallelization and good performance. However, hardware architecture for FS algorithm usually requires a large amount of gate resources to achieve real time, mainly for high resolution videos. On the other hand, the QSDS-DIC algorithm can significantly reduce the number of SAD (Sum of Absolute Differences) calculations, when compared to the FS algorithm, enabling significant reduction in the hardware resources that are necessary to achieve real time for high resolution videos. The proposed architecture is based on the QSDS-DIC algorithm, using a very large search area, generating results with quality that are close to the FS results [1].

The main characteristic of the proposed architecture is the large number of processing units (PUs) that are used to calculate the SAD metric. Therefore, the implementation of an efficient SAD calculation block is necessary in order to improve the performance of ME architecture. Since the PUs are composed by a large number of additions to calculate the SAD metric, we have used efficient adders to reduce the computational complexity and to increase the performance of the PUs. A combination of 4-2 and 8-2 adder compressors [2], which perform the simultaneous addition of 4 and 8 operands respectively are used in the new solutions for PU architectures.

This paper presents two new PU architectures using 4-2 adder compressors (the PU-42 architecture) and 8-2 adder compressors (the PU-82 architecture). These new PUs (PU-42 and PU-82) were applied to the QSDS-DIC architecture. The designed ME architecture was described in VHDL and synthesized to the TSMC 0.18µm CMOS standard cell with the Leonardo Spectrum CAD tool. This architecture can

reach real-time for HDTV 1080p videos with a low hardware cost.

The results show that the use of PU-42 and PU-82 can significantly improve the performance of the QSDS-DIC architecture. Moreover, the improvement in performance of the PUs enables gains of 12% in the QSDS-DIC architecture in terms of processing rate, when compared to the original architecture.

The rest of the paper is organized as follows: Section II presents a short review of some related works. Section III presents the main characteristics of the 4-2 and 8-2 adder compressors. Section IV presents the QSDS-DIC algorithm and a software evaluation with quality and computational cost results. In Section V the QSDS-DIC architecture is presented, including the original PU and the developed PU-42 and PU-82 architectures. Section VI discusses the synthesis results. Finally, Section VII presents the main conclusions of this work and some ideas for future work.

2. RELATED WORKS

Hardware implementations for motion estimation are widely used to speed up the generation of the motion vectors. The use of hardware solutions is still more important when real time processing is considered, mainly for high definition video coding. There are a large amount of published works with hardware architectures for the motion estimation process, including both FPGA and ASIC solutions. Most of the solutions are based on the traditional Full Search (FS) algorithm.

The FS algorithm was chosen by Mohammadzadeh [3], Gaedke [4], He [5] and Kao [6] to develop their ME architectures. However, only Mohammadzadeh [3] synthesizes his architecture to FPGA. These solutions, based on the use of FS algorithm, do not present much innovation. In fact, as the FS algorithm is very regular, with no data dependences, thus there is no much innovation to be explored when this algorithm is implemented in hardware. In general, when solutions aim high performance, several levels of parallelism have to be considered. It implies on a very high cost for area resources.

The architecture presented by Kao [6] is based on the use of a threshold value to speed-up the matching process. When the difference between the candidate block and the current block is lower than this threshold, then the search stops. The solution proposed by Larkin in [7] also uses the FS algorithm. However, he implements a 4:1 pixel sub-sampling, combined with an early termination approach by using threshold.

The hardware solutions for fast ME algorithms are not very common. It can be explained mainly by the characteristics required by the fast algorithms such as, data dependences, unlimited number of iteration to find the best match, and hard parallelization. One fast algorithm named Dynamically Variable Step Search (DVSS) was used by Tasdizen [8], where his architecture was implemented in a Spartan-3 FPGA, and used 256 processing units (PU). With this large number of PUs, his architecture is able to can process real time HDTV with an operational frequency of 130MHz. This solution also uses a threshold as parameter to stop the search, and its performance, and also the quality results, are directly dependent to this threshold value.

As some of the published work, our proposed solution aims the possibility of reducing the number of clock cycles, to achieve high performance by using a fast algorithm. However, neither of the published work has considered the techniques we have proposed in this work. Our approach is based on two main ideas: i) reducing dynamically the number of iterations for the calculation of the motion vector, and ii) using efficient hierarchical adder compressors to reduce the latency of the PU.

3. ADDER COMPRESSOR CIRCUITS

One of the main goals of this work is the efficient calculation of the large amount of the additions generated by the PUs that are present in the ME. For this task, adder compressors were used. These compressors enable the simultaneous addition of more than two operands at a time. In this work, namely 4-2 and 8-2 adder compressors are used for the efficient addition calculations of the PUs.

A. Basic Structure

A structure to add four bit inputs simultaneously was presented by Weinberger in [2]. This structure was named 4-2 Carry Save module and contains a combination of full adder cells in truncated connection in which a fast compression is possible. The 4-2 compressor has five inputs and three outputs, where the five inputs and the output Sum have the same weight. On the other hand, the outputs Cout and Carry have a greater weight. The Sum, Cout and Carry terms are calculated according to equations 1, 2 and 3 [2]. One important point to be emphasized in the equations is the independence of the output carry Cout on the input carry (Cin) in equation 2. This aspect enables higher performance addition implementation.

$$Sum = A \oplus B \oplus C \oplus D \oplus Cin \quad (1)$$

$$Cout = (A \cdot B) \cdot (A \cdot C) \cdot (B \cdot C) \quad (2)$$

$$Carry = [A \oplus B \oplus C] \cdot (D \cdot Cin) \cdot (D \cdot Cin) \quad (3)$$

The improvement in the structure of a 4-2 compressor, using multiplexer is shown in Figure 1. This structure proposed by Oklobdzija [9] has a more reduced critical path, where the maximum delay is given by three exclusive-OR gates. This critical path is shorter than the previous structure given by Weinberger [2] which presents four exclusive-OR gates in the critical path (equations 2, 3 and 4).

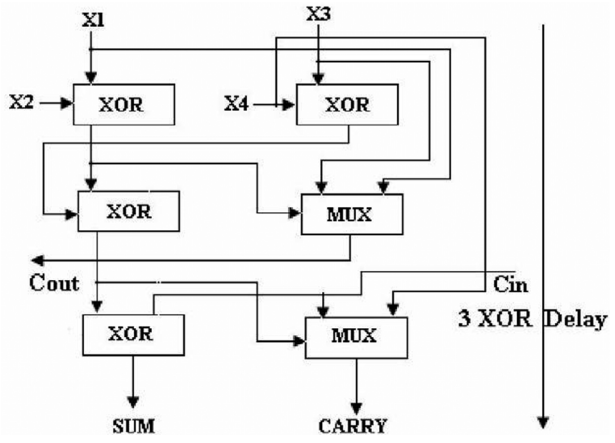

Figure 1. 4-2 Compressor structure proposed by [9]

The final sum result of the 4-2 compressor is given by a combination of Sum, Cout and Carry terms, where the terms Cout and Carry have greater weight than the Sum term, as can be seen in equation (4).

$$S = Sum + 2(Cout + Carry) \quad (4)$$

B. Hierarchical Structure

From the implementation of the 4-2 compressor it was possible to extend up the number of simultaneous additions by implementing an 8-2 compressor. For the 8-2 compressor architecture, a cascade of 4-2 compressors is used, where eight numbers can be added simultaneously. The architecture is composed by thirteen inputs (eight primary inputs and five carry inputs) and seven outputs (one Sum, one Carry and five Cout terms). The 8-2 compressor architecture can be seen in Figure 2. The final sum in the 8-2 compressor is obtained as in equation 4.

4. QSDS-DIC MOTION ESTIMATION ALGORITHM

This paper focuses in a high throughput solution for motion estimation in digital video coding, using the QSDS-DIC algorithm and adder compressors. QSDS-DIC is based on three main principles: (1) Diamond Search algorithm; (2) 4:1 Pixel Sub-sampling technique and (3) dynamic iteration control [1]. These main principles will be detailed on the next paragraphs.

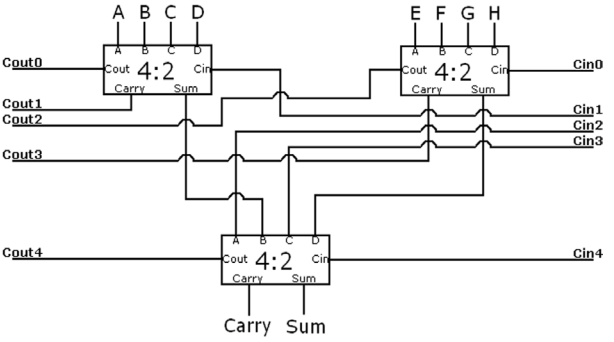

Figure 2. 8-2 adder compressor structure

The Diamond Search (DS) algorithm defines two diamond patterns, the Large Diamond Search Pattern (LDSP) and the Small Diamond Search Pattern (SDSP) [10]. The LDSP consists of nine comparisons around the center of the search area. The SDSP is used in the end of the search, when four candidate blocks are evaluated around the center of the LDSP, in order to refine the final result. Figure 3(a) shows the LDSP (white circles) and the SDSP (gray circles).

The search ends when the lower SAD is found at the center of the LDSP. The SDSP is then applied and the search is finished. When the best match is found in a vertex, five more blocks are evaluated to form a new LDSP, as shown in Figure 3(b). If the best match was obtained in an edge, three more blocks are evaluated, as can be seen in Figure 3(c).

The 4:1 Pixel Sub-sampling (PS) technique was investigated in our previous work with very promising results. The Pixel sub-sampling is applied to each evaluated candidate block. Thus, only half of pixels of each line of the original block will be considered. The number of lines is also sub-sampled by half.

The pixel sub-sample by half in terms of the number of lines, and in terms of the number of elements in each line, reduces the number of pixels in a candidate block by four.

Figure 4(a) shows a no sub-sampled 8x8 block, and Figure 4(b) shows a 4:1 pixel sub-sampled 8x8 block. Only the black pixels will be calculated, while the white ones are not taken into account.

Considering a candidate block with 16x16 samples, it becomes a sub-sampled block with 8x8 samples. Then, by using PS it is possible to improve by four the throughput in the block matching search.

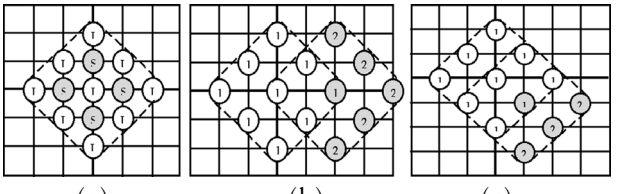

Figure 3. (a) LDSP and SDSP, (b) Search for a vertex, and (c) Search for an edge

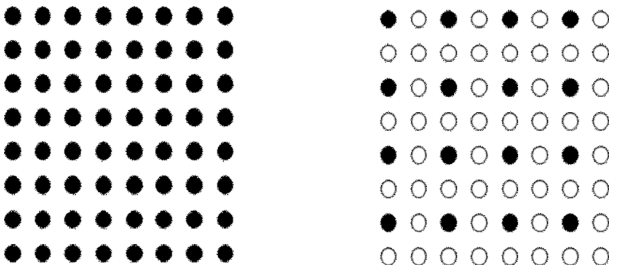

Figure 4. (a) Original 8x8 block, (b) 4:1 Pixel Sub-sampled 8x8 block

This aspect enables reductions in terms of hardware at the cost of a small loss in terms of quality.

Finally, the Dynamic Iteration Control (DIC) was designed focusing on the hardware design of ME. This feature allows a good tradeoff between hardware resources consumption, throughput, synchronization and quality. One important characteristic of the fast ME algorithms is the unrestricted number of iterations. Usually, the fast algorithms can repeat the used standard for search (diamond in this work) until a given condition for finishing the search is found. However, when a hardware implementation is considered, a restriction on the number of used iterations must be implemented. To solve this problem DIC was developed to dynamically control the iterations usage.

Through the software analysis of 10 video samples, it was possible to conclude that the DS algorithm uses an average number of 3.2 iterations while, in the worst case, DS uses 89 iterations. This analysis considered a search area with 208x208 samples and blocks with 16x16 samples. These results also indicated that DS rarely uses more than 14 iterations. Then, this characteristic was also explored in this paper through the design of DIC control hardware.

The restriction in the number of iterations is important to make easier the synchronization of this module with other video coder modules. This restriction is also important to allow for the best relationship between high throughput and low hardware cost. A static restriction could be implemented. However, it causes degradation in the results quality. Then, we proposed a dynamic (DIC) restriction. DIC allows the algorithm to use a variable number of iterations, which depends on the actual algorithm performance.

In the DIC implementation, a variable stores the number of iterations used for the generation of the 16 last motion vectors. Each motion vector has a limited number of 20 iterations to use (much more than the average of 3.2). When a motion vector uses less iterations than the maximum available for it, the saved iterations are accumulated and this extra slack is available to be used, if needed for better matching, in the estimation of the next motion vector. The great part of the motion vector will be generated with less iteration of the maximum available (20), even the average is equal to 3.2 iterations. Thus, when some vector needs to use 40 iterations to achieve a far distance, and find a best matching, it will be possible. Even using a variable number of cycles to calculate each motion vector, the performance can be defined by the static number of cycles available for the set of 16 motion vectors.

The use of DIC also becomes the search area variable. The size of the search area will depend on the choices of the algorithm. The search area will increase according to the use of iteration by the algorithm to find the best matches.

A. Algorithm Evaluation

This section presents the results of a software evaluation between Full Search (FS), FS with 4:1 Pixel Sub-sampling (PS), Diamond Search (DS) [10], QSDS, DS-DIC (with no sub-sampling) and QSDS-DIC [1]. FS block matching algorithm is the only one that presents the optimal results in terms of matching. All the others are faster algorithms which use different types of heuristics to reduce the computational complexity of the motion estimation process. The main results of the software analysis are shown in Table I.

The search algorithms were developed in C language and the results for quality and computational cost were generated. The search area used was 64x64 pixels (except for DIC based algorithms which has a dynamic search area) with block size of 16x16 pixels. The QSDS-DIC algorithm restricted to 320 the number of iterations used to generate 16 motion vectors. It means that, in the worse case, the QSDS-DIC algorithm can reach a very high search area (and rarely necessary) of 1300x1300 samples.

The DS-DIC version does not use pixel sub-sampling, and it was evaluated with an iteration restriction of 160. This hard restriction leads to a maximum search area of 660x660 pixels. The algorithms were applied to the first 100 frames of 10 well know SDTV video sequences with a resolution of 720x480 pixels.

The quality results were evaluated through the averages of PSNR and percentage of residual reduction (PRR), shown in Table I.

The PRR is measured comparing the results for generated residual by the ME process with the residual results generated by the direct subtraction between the reference and the current frames (differential coding). Table I also presents the average number of SAD oper-

Table I. Results of the software analysis.

Search Algorithm	PRR (%)	PSNR (dB)	# of SADs (x109)
FS	54.66	28.483	82.978
4:1 FS	50.20	27.244	20.745
DS	48.63	27.233	0.852
DS-DIC	48,27	27.207	0.850
QSDS	43.33	26.004	0.207
QSDS-DIC	43.64	26.030	0.209

ations used by each algorithm, when processing the same video sequences in the benchmark. Both FS and 4:1 sub-sampled FS require two orders of magnitude more SAD calculations than the DS variations.

By using 4:1 PS in the FS algorithm it is possible to reduce four times the number of SAD operations. The quality losses generated with the use of 4:1 PS are small. These losses are only of 4.46% in the error reduction and only of 1.239dB in the PSNR when comparing to the FS without sub-sampling. The results for DS and DS-DIC algorithms are very similar to 4:1 FS in terms of quality. However, the gain in terms of reduction of SAD operations is about 24 times higher than the FS with 4:1 PS solution.

QSDS with DIC, as expected, is able to improve the results of QSDS, with a small impact in terms of the number of SAD calculations. QSDS-DIC presents an error reduction 3.1% higher than the QSDS and 5% less than DS. The PSNR of QSDS-DIC is 0.026dB higher than the QSDS and 1.203dB lower than DS. The number of SAD calculation is a little bit higher in QSDS-DIC than in QSDS, since QSDS-DIC uses only 1% more operations. This small increase in SAD calculations is responsible for increasing the PSNR. When compared to the FS algorithm, QSDS-DIC presents a loss in error reduction of 11% and a loss in PSNR of 2.453dB. In the other hand, QSDS-DIC is able to reduce the number of SAD calculations by 397 times.

FS algorithm with PS is widely used for motion estimation hardware architectures due to the good quality results and the regularity of the algorithm. However, the use of hardware resources for real time applications in high resolution videos is very intensive.

There are many restrictions for the use of fast motion estimation algorithms for hardware design. In general, fast algorithms are based on an application of a search standard which can be repeated while a condition is not satisfied. For hardware implementation, a restriction for these repetitions must be implemented. It is impossible to determine the number of necessary clock cycles to generate a motion vector without this restriction. Thus, it is also impossible to calculate the performance of the architecture.

The QSDS-DIC algorithm is a fast algorithm that dynamically performs the restriction in the standard repetition. This algorithm was implemented in hardware, and it was chosen to be designed in this paper to allow a hardware design with high performance, and low hardware resources utilization. These characteristics are achieved while keeping the quality of the generated motion vectors.

5. QSDS-DIC ARCHITECTURE

The QSDS-DIC architecture explores the maximum parallelism available by the DS search engine.

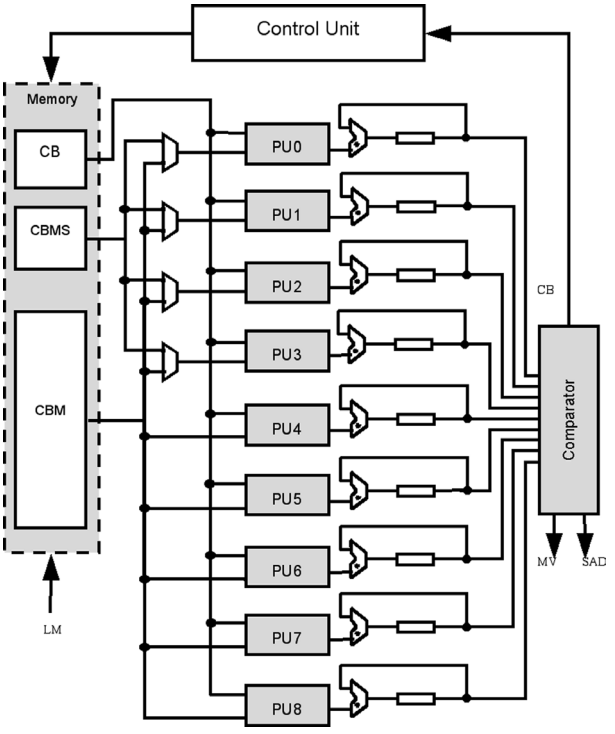

Figure 5. QSDS-DIC block diagram architecture

The designed architecture for the QSDS-DIC algorithm uses SAD as a distortion criterion. The architectural block diagram is shown in Figure 5. The architecture has nine processing unities (PU), to calculate the SAD between the reference and candidate block. A set of internal memories are used to store the current and candidates blocks. A special memory and a set of multiplexers are used to speed up the SDSP calculation.

The nine candidate blocks of the LDSP are calculated in parallel and the results are sent to the comparator. Each candidate block receives an index, to identify the position of this block in the LDSP. The comparator finds the lowest SAD and sends the index of the chosen candidate block to the control unit. The control unit analyses the block index and decides the next step of the algorithm. If the best result was found at the center of the LDSP, the control unit starts the final refinement with the SDSP. In the SDSP calculation, four more candidate blocks are evaluated. The lowest SAD is identified and the control unit generates the corresponding motion vector for this block. When the best result was not found at the center, a new iteration is applied.

A. Memory Organization

The internal memory is organized in 15 different local memories, as presented in Figure 6. The local memory (LM) stores the region with the nine candidate blocks of the first LDSP and all the possible blocks for the next step. Thus, when the control unit

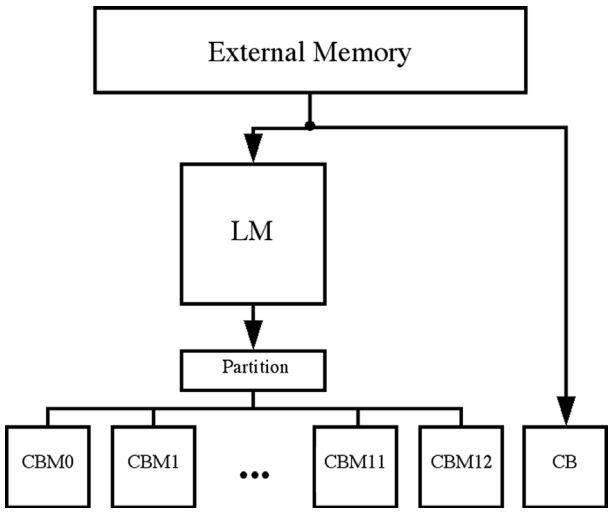

Figure 6. Memory organization

decides which blocks must be evaluated in the next step, the LM already has this data. LM is composed by 16 words of 128 bits wide. Another 13 small memories are used to store the candidates block (CBM) and one for the current block (CB). These 14 memories are composed by 8 words with 64 bits (8 samples of 8 bits) and they store one sub-sampled block with 8x8 samples.

There are many common data between the candidate blocks. For example, the second line of the LM provides data for the second line of the candidate block 0, and also for the candidate block 1 and 2 first lines. The LM second line also provides data for the first line of the candidate block 0 of the SDSP.

LM is read line by line and the data is stored in the CBMs. Nine CBMs are used to store the candidate blocks from the LDSP. Four CBMs are used to store the blocks of the SDSP. These blocks are stored at the same time as the LDSP blocks, and they are ready to be used by the PUs, if the control decides to start the SDSP. This solution speeds up the architecture and reduces the memory access latency. When the SDSP mode is active, the control unit selects the multiplex in Figure 5 and the PUs receive the data from these memories with no memory latency.

Each line from the LM has 128 bits. However, a partition unit (Partition block in Figure 6) cuts the line to 64 bits. This unit selects the correct part of the 128 bits which corresponds to the candidate block. When the first line of the LM is read, the Partition block selects the correct part of the word which corresponds to the line 0 of the candidate block 0, to be stored in the CBM0. When the second line is read, this module selects the correct part of the second line of the candidate block 0, to be stored in the line 1 of the CBM0, and the correct part for the first line of the candidate block 1, to be stored in the CBM1. This process finishes when all LM is read and all CBMs are full.

A local control was developed to control the memory access. The control unit in Figure 5 is not responsible for controlling the memories read/write process. When the datapath finishes the SAD calculation, the control unit informs the memory if the search should continue or if the SDSP should be applied. If the search continues, by an edge or a vertex, the CBMs are stored with the new candidate blocks and the LM memory is reloaded. In this case, there are 10 cycles of latency to write the first line in the nine CBMs. If SDSP is applied, the four candidate blocks are already stored and the datapath starts with no memory latency. Even using 15 different memories, the total memory consumption is small. The search area is loaded according to the algorithms necessity, such that no irrelevant data is stored. A total of 9.2 Kbits are used for all motion estimator internal memories.

B. Dynamic Iteration Control

As mentioned before, it is clear that a restriction in the number of iterations, must be implemented to ensure the desired throughput. Without an iteration restriction the architecture could use, in the worst case, a very large number of iterations, and consequently, a large number of clock cycles to generate the motion vector. As the hardware must be able to process the ME also in the worst case, the overall architecture must be overestimated to allow the desired throughput in the worst case, while in most of the time the hardware resources will be underutilized. It is also impossible to determine the architecture throughput without any restriction and it can also be difficult to synchronize it with the other blocks of the video coder.

The architecture used to implement DIC is shown in Figure 7. The number of iterations used is sent to a shift register (SR) with 16 positions. A set of add/accumulator (ACC) generates the total number of iterations used in the last 16 motion vectors. The DIC was developed allowing a maximum of 20 iteration per motion vector, thus the used iterations for the last 16 vectors are subtracted of 320 (maximum value for a set of 16 motion vectors), and the result is the number of available iterations for the next motion vector generation. This results in a dynamic search area. Considering the DIC with 20 iterations per motion vector, the QSDS architecture can use a maximum search area of 1300x1300 pixels.

C. Processing Unit

The ME architecture uses nine Processing Unit (PU), where each one of the PUs calculates the SAD for a candidate block. Each PU can process eight samples in parallel and this means that one line of the

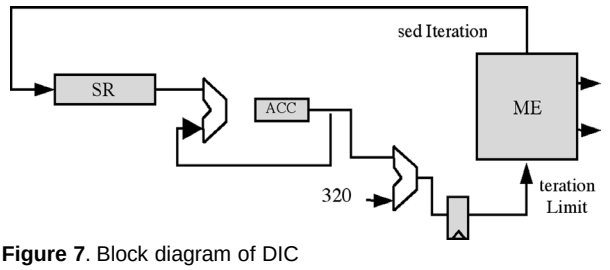

Figure 7. Block diagram of DIC

16x16 sub-sampled blocks is processed in parallel. Thus, eight accumulations must be used to generate the final SAD result of each block, one accumulation per line. The Processing Unit (PU) was designed as an adder tree, in a four-stage pipeline. Figure 8 shows the PU architecture.

The PUs receive data from the memories of the candidate blocks (CBM) and the actual block (CB). Each PU receives a word from the CBM and one word from the CB and calculates the difference between the eight samples that are present in each one of these words. First, the eight samples of the actual and the candidate blocks are subtracted. The absolute values from the difference between the samples are added and the final result of the accumulation of the differences, for all samples of the line, is stored in the output register.

The pipelined PU can calculate one line of candidate block per cycle, after the latency of four clock cycles. A set of adder/accumulator is used in the outputs of the PUs in order to store the intermediate results from each one of the candidate blocks.

In general, sixteen accumulations to generate the final results of the SAD for a candidate block (one accumulation for each line of the block) are needed.

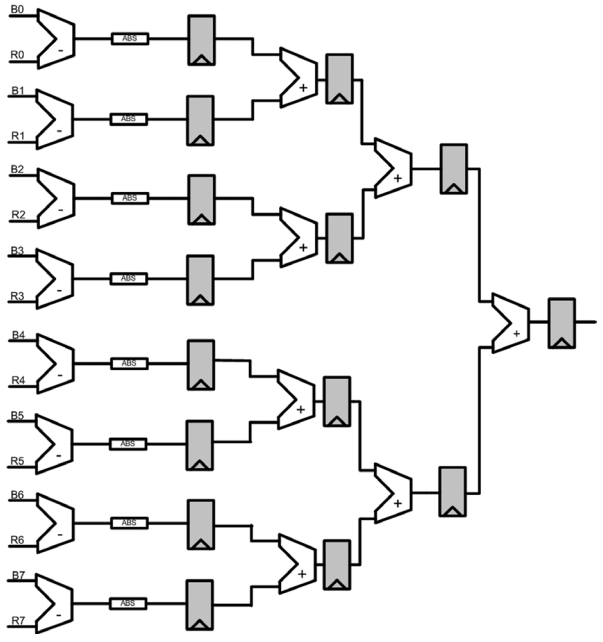

Figure 8. PU architecture

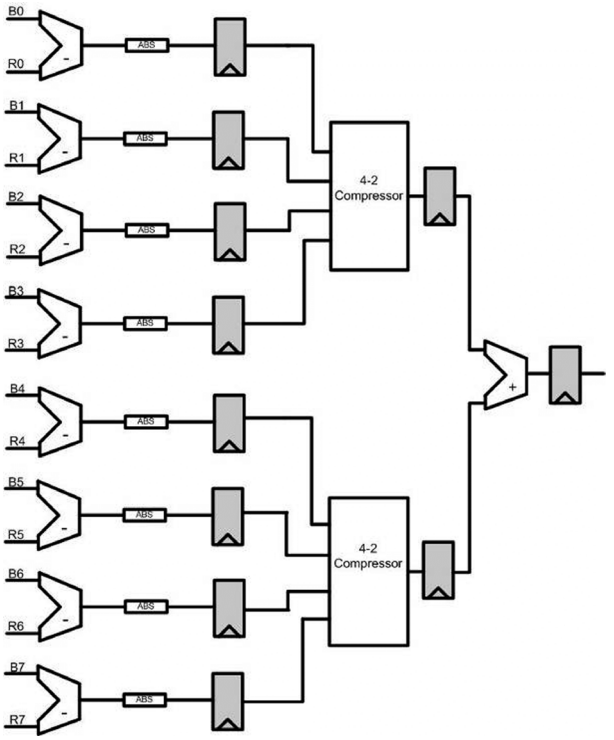

Figure 9. PU-42 Architecture

In this work we have proposed and tested the implementation of two versions of PU by using 4-2 and 8-2 adder compressors (PU-42 and PU-82 respectively). These adder compressors are used to perform efficiently the large number of additions in the PU.

As in the original architecture presented previously, our solution processes eight samples in parallel. The main difference between these PUs architectures is the type of adders that are used. The proposed architecture uses efficient adder compressors to perform the additions in the PU. Figure 9 shows the PU-42 architecture.

Two blocks of 4-2 adder compressors are used to perform the addition of eight operands that are available at the output of the first column of registers. The values added by the 4-2 adder compressors represent the module of the subtractions that are performed at each pair of the input operands. The final PU result is calculated by the addition of the results produced by the output of each block of the 4-2 adder compressors. Since four simultaneous additions are performed by the 4-2 adder compressor, it is possible the elimination of one column of adders, as can be compared in Figures 8 and 9. Thus, the proposed architectural solution saves one clock cycle when compared to the original architecture.

In order to improve the performance of the PU, we have proposed the use of 8-2 adder compressor (PU-82), as can be observed in Figure 10. In this architecture, eight samples can be added simultaneously by the 8-2 adder compressor enabling the elimination of two columns of adders. Thus, the PU with

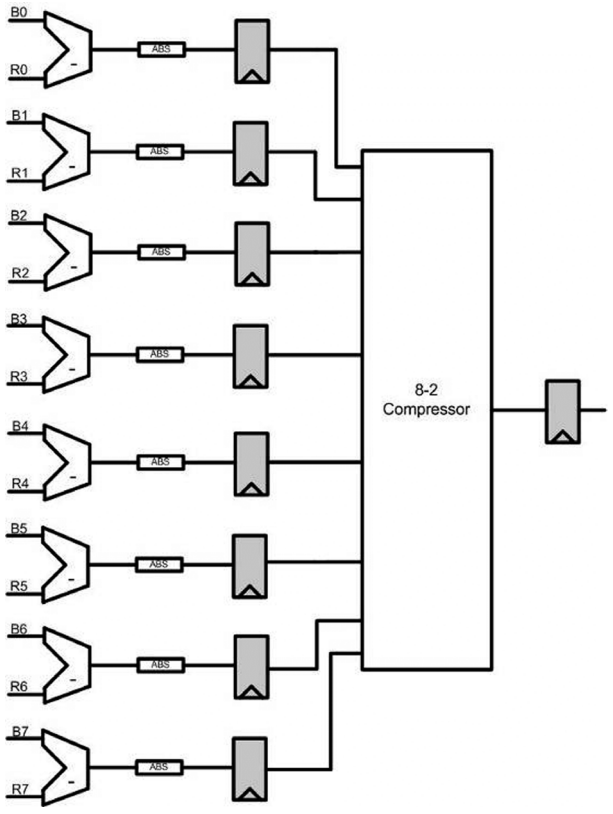

Figure 10. PU-82 Architecture

8-2 adder compressor can save two clock cycles, when compared with the original solution. As can be observed in Figure 10, after the first row of registers, only one 8-2 adder compressor is necessary to calculate the final result.

D. Performance evaluation

The original QSDS-DIC architecture and the two proposed versions (using PU-42 and PU-82) use 26 clock cycles to fill all the memories and to start the SAD calculation.

The original PU has a latency of four cycles while the PU-42 and PU-82 have a latency of three and two cycles, respectively. All versions need seven more cycles to calculate the SAD of one block.

The comparator uses five cycles to choose the lowest SAD. Thus, 42 clock cycles are necessary to process the first LDSP in the original QSDS-DIC architecture. The SDSP needs only 20 cycles to calculate the SAD for the four candidate blocks of the SDSP. As a result, in the best case, the original architecture can generate a motion vector in 62 cycles.

The developed architecture, with PU-42, uses 41 clock cycles to process the first LDSP, and 19 clock cycles to calculate the SDSP. Thus, in the best case, the QSDS-DIC architecture with PU-42 can generate a motion vector in 60 clock cycles. The second architecture, using PU-82, uses only 40 clock cycles to calculate the first LDSP and 18 cycles to process the SDSP. Then, in the best case, the QSDS-DIC architecture with PU-82 uses 58 clock cycle to generate a motion vector.

The average case was obtained through software analysis [1]. The QSDS-DIC algorithm uses an average of 3 iterations to find the best match. 10 clock cycles are needed for the memory latency. The original PU uses 11 additional clock cycles, while the PU-42 uses 10, and the PU-82 uses 9. Thus, each iteration in the original ME architecture uses 26 clock cycles, while the solutions with PU-42 and PU-82, use 25 and 24 cycles respectively.

The worst case is defined by the DIC mechanism. In this architecture, the DIC allows the use of 20 clock cycles for each motion vector generation. Thus, considering the worst case, the original architecture uses a maximum of 582 cycles to generate a motion vector. The QSDS-DIC architecture with PU-42 and PU-82 uses 560 and 538 cycles respectively.

6. RESULTS AND COMPARISONS

This section presents synthesis results from the original QSDS-DIC architecture and the new one using adder compressors. The proposed architecture was all described and validated in hardware description language - VHDL. Synthesis results for both FPGA and standard-cell are presented.

A. Synthesis Results from the Original Architecture

The proposed architecture was described in VHDL. ISE 8.1i tool was used to synthesize the architecture to the Virtex-4 XC4VLX15 device and the ModelSim 6.1 tool was used to simulate and to validate the architecture design. Table II presents the synthesis results.

The device hardware utilization is small, since only 3.5k look-up-tables (FPGA LUTs) are used. The synthesis results show the high frequency achieved by the QSDS-DIC architecture.

The performance results presented in Table II take into account HDTV (1920x1080 pixels) videos. Thus, the QSDS-DIC architecture can process 45 fps for HDTV 1080p video in the worst case. Considering the DIC with 20 iterations per vector, the archi-

Table II. QSDS-DIC Architecture Synthesis Results.

Parameter	QSDS Results
BRAMs	32
Slices	2007
Slice FF	2086
LUTs	3610
Frequency (MHz)	213.3
HDTV 1080p fps (worst case)	45
HDTV 1080p fps (average case)	188

tecture can use, in the worst case, 9312 clock cycles to generate 16 motion vectors, which represents an average of 582 cycles per motion vector.

The average case was obtained through the software implementation of the algorithm. The algorithm uses an average of three iterations in the second step of iteration. Then, the QSDS-DIC architecture needs 140 clock cycles to generate a motion vector in the average case. This implies on a processing rate of 188 HDTV 1080p frames per second in this case. The synthesis results presented in Table II show that the QSDS-DIC architecture has a throughput which is larger than that is necessary to process HDTV 1080p frames in real time.

Table III shows the minimum frequency operation for the QSDS-DIC architecture in several video standards, for the worst and average cases.

The minimum frequency operation for the QSDS-DIC architecture is very low for low resolution videos. Considering the worst case, the QSDS-DIC architecture needs only 1.7 MHz to process QCIF videos in real time. For the average case, the minimum frequency operation is still less, only 0.4 MHz. Even for the standard definition SDTV 480p the minimum frequency is low, about 23.5 MHz in the worst case. For HDTV 1080p frames, the minimum frequency is about 141.4 MHz in the worst case and 34.0 MHz for the average case.

The minimum frequency operation results show that, depending on the video resolution, the QSDS-DIC architecture can be synthesized to operate on a lower speed or it can be synthesized to an older FPGA technology. It is also interesting for low power application, as mobile phone for example, that usually process data for low resolution videos.

Table IV shows FPGA results, where our QSDS-DIC architecture is compared to the FS architecture presented by Mohammadzadeh in [3], FTS (Flexible Triangle Search) architecture presented by Rehan in [11], FS+PS4:1 and early termination architecture presented by Larkin in [7], and the DVSS (Dynamically Variable Step Search) presented by Tasdizen in [8]. The solutions presented by Mohammadzadeh in [3] and Rehan in [11] have a constant throughput. On the other hand, as our proposed QSDS-DIC architecture, the architecture presented by Larkin in [7] has a variable throughput (represented by average throughput in Table IV) due to the early termination. The result presented by Tasdizen [8] has considered a threshold value equal to 256 for block matching. This threshold can be changeable, and thus the performance result can be varied. While the search area used by Larkin [7] is 32x32 samples, the Tasdizen architecture [8] uses a search area of 48x24 samples. The architectures presented by Mohammadzadeh in [3] and Rehan in [11] do not present this information. The processing rate of HDTV frames (1920x1080) is also presented in Table IV. The number of CLBs used by the architecture presented in [7] by Larkin was not mentioned in that paper.

Table III. Minimum frequency operation for real time in several video standards.

Standard	Minimum Frequency Opera-tion Worst Case (MHz)	Average Case (MHz)
QCIF (176 x 144)	1.7	0.4
CIF (352 x 268)	6.9	1.6
VGA (640 x 480)	2.9	5.0
SDTV (720 x 480)	23.5	5.6
HDTV (1280x720)	83.8	20.1
HDTV (1920 x 1080)	141.4	34.0

Table IV. FPGA comparative results.

Solution	FPGA	CLBs	Frequency (MHz)	HDTV (fps)
Mohammadzadeh [3]	Spartan 2	939	109.8	1.66
Larkin [7]	Virtex 2	-	120.0	*24
Tasdizen [8]	Spartan 3	2282	130.0	34.3
Rehan [11]	Spartan 3	6142	74.0	45
QSDS-DIC	Virtex 4	492	213.3	*188

* Average throughput

As can be observed in Table IV, QSDS-DIC architecture is the fastest one among the presented designs. Due to the efficiency of the QSDS-DIC algorithm, the designed architecture uses less hardware resources than the other solutions, reaching the highest throughput among them.

B. Synthesis Results with Adder Compressors

Table V presents the synthesis results for the original QSDS-DIC architecture and the developed solutions with PU-42 and PU-82. Leonardo Spectrum tool was used to synthesize the architectures to standard-cell TSMC 0.18μm CMOS technology.

The latency, hardware area (expressed as the number of gates), frequency operation and the number of frames per second (fps) for HDTV, are presented for the original architecture [1] and the two new developed versions with adder-compressors, where both the worst case and the average case are considered. These new architectures are: QSDS-DIC with PU-42 (using 4-2 adder compressor) and QSDS-DIC with PU-82 (using 8-2 adder compressor). The average case for the calculation of the fps was obtained through the software implementation of the algorithm [1]. The algorithm uses an average of three iterations in the second step of iterations. The worst case is defined by the DIC, and all versions allow 20 iterations for each motion vector generation.

As can be observed in Table V, the QSDS-DIC with PU-42 and PU-82 versions are more efficient than the original one. Particularly, the QSDS-DIC with PU-42 is the one with the highest frequency

operation among the architectures considered herein, because the 4-2 adder compressor used in PU-42 has a shorter and faster critical path than that of the 8-2 adder compressor used in PU-82. As the PU-42 architecture has a latency of 60 clock cycles, thus it can process 63 frames per second for HDTV 1080p video in the worst case. On the other hand, in the average case, where generally three iterations to calculate a motion vector are needed, this architecture can process 240 frames per second.

The QSDS-DIC architecture with PU-82 has a lower frequency operation than the architecture with PU-42. However, as the QSDS-DIC architecture with PU-82 has two less clock cycle (58 against 60 of QSDS-DIC with PU-42), and the difference in frequency operation between the two versions is not significant, then the QSDS-DIC with PU-82 can reach the best results in terms of HDTV 1080p frames processed per second (65 frames per second in the worst case and 269 frames per second in average case).

Another important point to be observed in Table V is the higher number of gates presented by the architecture with PU-42. In fact, we have observed that the use of two separate blocks of 4-2 adder compressors contributes to the higher number of gates presented in the QSDS-DIC architecture with PU-42. On the other hand, even using more complex 8-2 adder compressor, the architecture with PU-82 uses one less column of adders and one less column of registers, all contributing to a slightly smaller number of gates required by the PU-82 architecture.

The developed architecture for the QSDS-DIC algorithm was compared with other published designs for ME. Table VI shows the comparisons of the original QSDS-DIC architecture, the architecture with PU-42 and PU-82 versions, and the designs presented by Li in [12], Gaedke in [4], He in [5] and Kao in [6].

The architectures presented by Gaedke in [4], He in [5] and Kao in [6] are based on the FS algorithm and they were designed targeting HDTV applications. QSDS-DIC (original [1] and with PU-42 and PU-82) architecture presents significant reduction in terms of area, when compared to the results presented by Gaedke [4] and He [5], where more than 30 times less gates are used, and when compared to the architecture presented by Kao [6], where more than 10 times less gates are used. However, the QSDS-DIC architecture presents slightly more gates than the architecture presented by Li in [12]. This occurs because that architecture uses the PVMFAST algorithm for a small search range of 32x32 samples, and does not achieve the required performance for real time HDTV coding.

Table V. Synthesis results for QSDS-DIC architecture.

Parameter	Original Architecture [1]	QSDS-DIC with PU-42	QSDS-DIC with PU-82
UP Latency	11	10	9
Area (gates)	28.087	29.702	28.858
Frequency (MHz)	272.6	287.3	283.0
HDTV 1080p fps (worst case)	58	63	65
HDTV 1080p fps (average case)	240	263	269

Table VI. Comparison of Standard-cells results.

Solution	Process (µm)	Area (k gates)	Freq. (Mhz)	HDTV (fps)
Gaedke [4]	0.09	1200	334	25
He [5]	0.25	1020	100	46
Kao [6]	0.18	321	154	30
Li [12]	0.18	17.5	200	23
QSDS-DIC Original [1] (worst case)	0.18	28.0	272	57
QSDS-DIC PU-42 (worst case)	0.18	29.7	287	63
QSDS-DIC PU-82 (worst case)	0.18	28.8	283	65

As the QSDS-DIC architecture is based on fast algorithm and considers very wide search areas, then this architecture is more efficient than those presented in [4], [5], [6] and [12]. The QSDS-DIC presents the highest throughput, even if its frequency operation is not the highest among all the standard-cells solutions. QSDS-DIC also presented the best tradeoff between hardware consumption and throughput among all works herein compared. Particularly, the QSDS-DIC with PU-82 presents the best results for processing rate, with a very small increase in terms of gates utilization, when compared to the original architecture [1].

7. CONCLUSIONS

This paper presented a real time HDTV motion estimation hardware architecture based on the Quarter Sub-sampled Diamond Search with Dynamic Iteration Control (QSDS-DIC) algorithm. This paper also presented two new architectural alternatives for the processing units (PU) used in the motion estimation architecture to the SAD calculation.

The designed ME architecture is able to run at 213.3 MHz in a Xilinx Virtex 4 FPGA and only 3610 LUTs were used to implement it. The QSDS-DIC architecture can operate videos in real time for HDTV 1080p (1920x1080 pixels) resolution. In the worst case, the QSDS-DIC architecture is able to process 45 HDTV frames per second. In the average case, the architecture is able to process 188 HDTV frames per second.

The QSDS-DIC architecture reaches the best results for maximum throughput and the lowest resources utilization among all based on FPGA and standard-cell solutions of the literature presented in this work. Another important feature is that QSDS-DIC architecture presents the best tradeoff between

throughput and hardware consumption among these presented solutions of literature.

The processing unit of the QSDS-DIC architecture was also developed using efficient 4-2 (PU-42) and 8-2 (PU-82) adder compressors, in order to improve the efficiency of the addition calculation in it.

The QSDS-DIC architecture with PU-42 is able to run at 287.3 MHz when synthesized to TSMC 0.18μm CMOS standard cell and 29.702 equivalent gates were used to implement it. This architecture can operate videos in real time for HDTV 1080p (1920x1080 pixels) resolution. In the worst case, this architecture is able to process 63 HDTV frames per second. In the average case, it can process 263 HDTV frames per second. This architecture also reached the highest frequency operation. However, it has not reached the best performance because this architecture uses more clock cycles than the QSDS-DIC architecture with PU-82.

The best performance was achieved by the QSDS-DIC architecture with PU-82, which uses 8-2 adder compressors. This architecture is able to run at 283.0 MHz and has the lowest latency among all presented solutions. This architecture can process 65 and 269 HDTV frames per second in the worst and the average case, respectively.

In comparison to the original architecture, the QSDS-DIC with PU-42 presented a gain of 8.6% in the processing rate, with a very small increase in the number of gates (5.7%). The QSDS-DIC with PU-82 presented the best results for processing rate. Gains of 12% and 3.7% were achievable by this architecture when compared with the original and with the PU-42 solutions respectively. This gain was obtained at the cost of a small 2.7% increase in terms of gates utilization when compared with the original solution.

As future work we intend to develop 4-2 and 8-2 subtracted compressors to use at the input of the Processing Units and investigate the power consumption of the implemented architectures.

REFERENCES

[1] M. Porto, L. Agostini, A. Susin, S. Bampi, "Architectural Design for the New QSDS with Dynamic Iteration Control Motion Estimation Algorithm Targeting HDTV," ACM Symposium on Integrated Circuits and System Design, Gramado, Brazil, 2008. pp. 135-138.

[2] A. Weinberger, "4-2 Carry-Save Adder Module," IBM Technical Disclosure Bulletin, 1981.

[3] M. Mohammadzadeh, M. Eshghi, and M. Azdfar, "An Optimized Systolic Array Architecture for Full-Search Block Matching Algorithm and its-Implementation on FPGA chips," The 3rd International IEEE-NEWCAS Conference, 2005, pp.174-177.

[4] K. Gaedke, M. Borsum, M. Georgi, A. Kluger, J. Le Glanic, P. Bernard, "Architecture and VLSI Implementation of a programmable HD Real-Time Motion Estimator," IEEE International Symposium on Circuits and Systems, New Orleans, USA, 2007. pp. 1609-1612.

[5] W. He and Z. Mao, "An Improved Frame-Level Pipelined Architecture for High Resolution Video Motion Estimation," IEEE Int. Symposium on Circuits and Systems, New Orleans, USA, 2007. pp. 1381-1384.

[6] C.-Y. Kao, C.-L. Wu, Y.-L. Lin, "A High-Performance Three-Engine Architecture for H.264/AVC Fractional Motion Estimation" IEEE Transactions on Very Large Scale Integration (VLSI) Systems. Volume PP, Forthcoming, 2009. pp. 1 - 5.

[7] D. Larkin, V. Muresan, N. O'Connor,"A Low Complexity Hardware Architecture for Motion Estimation," IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, 2006, pp. 2677-2688.

[8] O. Tasdizen, A. Akin, H. Kukner, I. Hamzaoglu, "Dynamically variable step search motion estimation algorithm and a dynamically reconfigurable hardware for its implementation" IEEE Transactions on Consumer Electronics, Volume 55, Issue 3, August 2009. pp. 1645 - 1653

[9] V. Oklobdzija, D. Villeger, S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers and Alghorimic Approach," IEEE Transaction on Computers, Vol.45, N_3, 1996.

[10] P. M. Kuhn, Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Springer, June 1999.

[11] M. Rehan, M. Watheq El-Kharashi, P. Agathoklis, and F. Gebali, "An FPGA Implementation of the Flexible TriangleSearch Algorithm for Block Based Motion Estimation," In: IEEE International Symposium on Circuits and Systems, Island of Kos, Greece, 2006. pp. 521-523.

[12] T. Li, S. Li, C. Shen, "A Novel Configurable Motion Estimation Architecture for High-Efficiency MPEG-4/H.264 Encoding," IEEE Asia and South Pacific Design Aut. Conf., Shanghai, China, 2005, pp. 1264-1267.