**J Real-Time Image Proc manuscript No.**
(will be inserted by the editor)

**Estefania Alcocer · Roberto Gutierrez · Otoniel Lopez-Granado · Manuel P. Malumbres**

# Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder

Received: 2015 / Revised: 2015

**Abstract** High Efficiency Video Coding (HEVC) was developed to improve its predecessor standard H264/AVC by doubling its compression efficiency. As in previous standards, Motion Estimation (ME) is one of the encoder critical blocks to achieve significant compression gains. However, it demands an overwhelming complexity cost to accurately remove video temporal redundancy, especially when encoding very high resolution video sequences. In order to reduce the overall video encoding time, we propose the implementation of the HEVC ME block in hardware. The proposed architecture is based on (a) a new memory scan order, and (b) a new adder tree structure, which supports asymmetric partitioning modes in a fast and efficient way. The proposed system has been designed in VHDL (VHSIC Hardware Description Language), synthesized and implemented by means of the Xilinx FPGA, Virtex-7 XC7VX550T-3FFG1158. Our design achieves encoding frame rates up to 116 fps and 30 fps at 2K and 4K video formats, respectively.

**Keywords** HEVC · FPGA · Integer Motion Estimation · Inter prediction · SAD architecture

This research was supported by the Spanish Ministry of Education and Science under grant TIN2011-27543-C03-03, and by the GVA government under grant ACOMP2015016.

Estefania Alcocer
Physics and Computer Architecture department, Miguel Hernandez University of Elche, Spain
Tel.: +34-96-6658389
E-mail: ealcocer@umh.es

Roberto Gutierrez
Communication Engineering department , Miguel Hernandez University of Elche, Spain

Otoniel Lopez-Granado · Manuel P. Malumbres
Physics and Computer Architecture department, Miguel Hernandez University of Elche, Spain

## 1 Introduction

HEVC, the High Efficiency Video Coding standard, is the most recent joint video project of the ITU-T VCEG and ISO/IEC MPEG standardization organizations, working together in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) [1]. Previous video coding standards are currently used for many applications such as broadcast of High Definition (HD) TV, video content acquisition, Internet and mobile video streaming, and real time conversational applications. However, new video services with Ultra High Definition (UHD) formats are emerging, which need higher coding efficiency than previous standards. HEVC has been designed to deal with these demands, working with higher video resolutions and adapting its design to allow the use of parallel processing techniques. It can compress video about twice as much as its predecessor H264/AVC without sacrificing quality, providing video delivery with higher resolutions and frame rates, higher dynamic range, and a wider color gamut. Furthermore, HEVC contains new key features that are friendly with the use of parallel processing techniques [2].

As in previous video standards, Motion Estimation (ME) is one of the most critical modules in the video encoding process since it is able to efficiently remove the temporal redundancy between successive frames. However, the ME module is by far the most complex task of the encoder, requiring more than 90% of the encoding time [3].

In HEVC, the complexity is even more critical due to several issues such as (a) a large set of Coding Tree Unit (CTU) partitioning modes, (b) the presence of multiple reference frames, and (c) the varying size of Coding Units (CU) in comparison with its predecessor H264/AVC. In addition, HEVC adopts Variable Block Size Motion Estimation (VBSME) to obtain advanced coding efficiency, which comes at the expense of a huge increase of computational complexity.

For these reasons, several hardware architectures have been proposed to speed up the HEVC ME module, reducing the overall encoder complexity as much as possible. The Integer-pel Motion Estimation (IME) block is in charge of motion estimation and it is composed of (a) an integer and/or fractional motion search algorithm, and (b) a Rate/Distortion (R/D) optimization procedure that optimally reduces the temporal redundancy found at the video sequence. In most of the works found in the literature, the proposed IME hardware architectures are only focused on the motion search algorithm since it takes most of the computational complexity of the IME block. There are a lot of motion search algorithms that can be used to find the motion in video sequences. The most popular in hardware implementations is the Full Search (FS) algorithm. It follows a greedy behavior by searching motion in all points of the established search area of a reference frame, and, as a consequence, it is able to provide an optimal result (i.e., a motion vector that minimizes the residual error of the actual CTU).

The architecture proposals in [3, 4, 6, 7, 9] present an IME hardware block using FS strategy. In [3], a Sum of Absolute Differences (SAD) unit on Field-Programmable Gate Array (FPGA) is proposed that is able to test all partition modes of a CTU except the set of asymmetric partition modes. Authors fixed a search area size lower than the one established by the standard, being able to run as fast as 30 fps at 2k video resolutions. The work presented in [4] proposes a SAD unit that computes all CTU partitions, achieving the same frame rates than previous work at 4k video formats. In their proposal, the search area has the same size as the maximum CTU, being implemented on an Application Specific Integrated Circuits (ASIC). In [6], the maximum CTU size is reduced to 32x32 with a search area size of $\pm$ 23 pixels. This architecture is implemented on an FPGA device and achieves 30 fps at 1080p video resolutions. Different configurable search areas are studied in [7], achieving a maximum frame rate of 57 fps for a 720p video resolution. Several SAD units implemented on FPGA are described in [9], with different levels of parallelization, but no data about search area size, memory management, or how they obtain the minimum SAD are included.

On the other hand, [5, 8, 15] have proposed architectures which increase the throughput by limiting the number of searches in the reference frame. In [15], a motion estimation system for the HEVC encoder is presented. This design includes both integer-pel and fractional-pel motion estimation, achieving video encoding speeds of 1080p@60fps and 2160p@30fps when implemented in FPGA and ASIC technologies, respectively. The process in [15] is interrupted when the number of motion searches arrives at a limit fixed for a given resolution.

In addition, in [5] and [8], different implementations of suboptimal motion search strategies called fast ME algorithms such as new Diamond Search (DS) or new Three Step Search (TSS) are shown. Similar hardware ME architectures have also been studied for the previous standard H264/AVC in [10, 11, 12, 13, 14], which are of interest for our work due to the high similarity of the IME block architecture in both standards.

Therefore, our purpose is to design a new hardware architecture that may perform IME computation in a fast and accurate way in order to significantly reduce the computation cost of the overall encoder. We will use FPGA technology, since it 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 ones [12].

Regarding the novelty of the proposed architecture, we present both innovative techniques: (a) a new SAD adder tree structure, and (b) a new memory scan order.

Firstly, we designed a new SAD adder tree structure, in order to perform the additions at the first level of the tree, starting from the maximum size of CTU, and halving the amount of additions at the next tree levels. This approach is different from the rest of state-of-the-art works, which divide a CTU into smaller blocks for consecutive accumulations, keeping the same additions in each step and thus requiring a higher number of steps to acquire all SADs. With our proposal, we took advantage of the resources provided by the FPGA, obtaining the minimum possible latency when calculating SADs of all levels and partitions for a CTU. In this way, SADs corresponding to asymmetric partitions are obtained in a fast and efficient way.

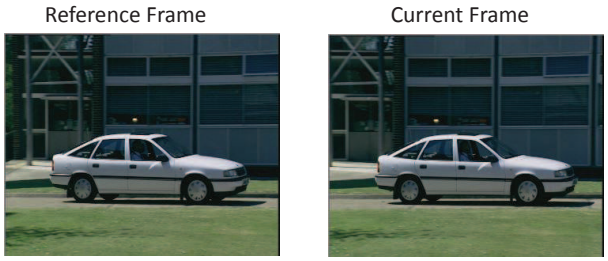
Secondly, regarding the new memory scan order, a series of reconfigurable shift registers and processing elements are responsible for storing the necessary pixels of both reference and current frames, keeping them always available for calculating the SADs and MVs of a CTU. With our system, we avoid external memory accesses since available data are highly reused by reconfiguring the displacement in a more efficiently way.

The rest of the paper is organized as follows. Section 2 describes the HEVC ME module. Section 3 presents the architecture design of the proposed ME system while in Section 4, implementation results are provided in terms of hardware resources, time encoding, and R/D performance. Finally, in Section 5 some conclusions are drawn.

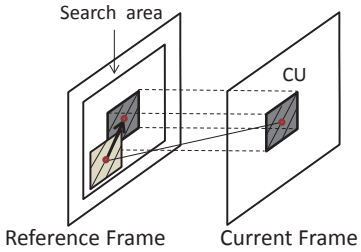
## 2 HEVC Motion Estimation

The motion estimation technique is based on the similarity between adjacent video frames, predicting the current frame based on a previous or subsequent reference frame in order of appearance.

The Motion Vector (MV) represents the translational movement of a picture area in the current frame compared to its position in the reference frame. This movement is found inside a defined search area in order to




(a) Current and reference frame




(b) Obtaining motion vector

**Fig. 1** Motion Estimation

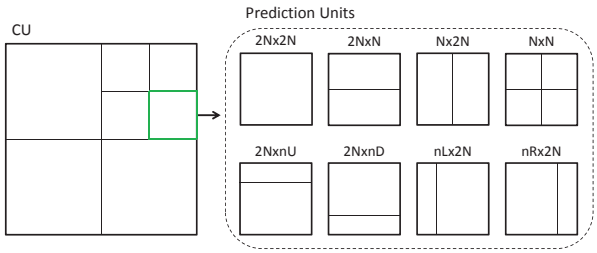



**Fig. 2** Predictions units within a CU

bound the overall motion search complexity, as shown in Fig. 1.

In the ME process, each video frame is subdivided and partitioned into basic coding units called CTUs. The coding structure in HEVC consists of CUs with a maximum size of 64x64 pixels, as large as that of CTUs, which can be recursively divided in picture squares until achieving a block size of 8x8 pixels. Each CU consists of prediction units (Intra or Inter) and its size can vary from the maximum size of the CU to 4x4 pixels for Intra prediction, or to 4x8 or 8x4 for Inter prediction, supporting 8 partitioning modes as shown in Fig. 2. Prediction units of sizes 2Nx2N and NxN are called square motion partitions (Square), 2NxN and Nx2N as Symmetric Motion Partitions (SMP) and 2NxnU, 2NxnD, nLx2N and nRx2N as Asymmetric Motion Partitions (AMP). The total number of different partitions for a 64x64 CTU is more than 600, and for each of these partitions, the HEVC encoder performs one ME process in order to determine the best CTU partitions in terms of bitrate and video quality.

There are many kinds of algorithms for block-based IME. The most accurate strategy is the FS algorithm, which exhaustively finds motion for all prediction unit blocks at every single point of the established search area. Due to computational regularity and excellent video quality, FS motion estimation is commonly employed in Very Large Scale Integration (VLSI) implementations [16]. Therefore, we will focus our work towards the design and hardware implementation of an FS algorithm that is able to significantly speed up the motion estimation process of the HEVC encoder without losing R/D performance.

## 3 Hardware Architecture

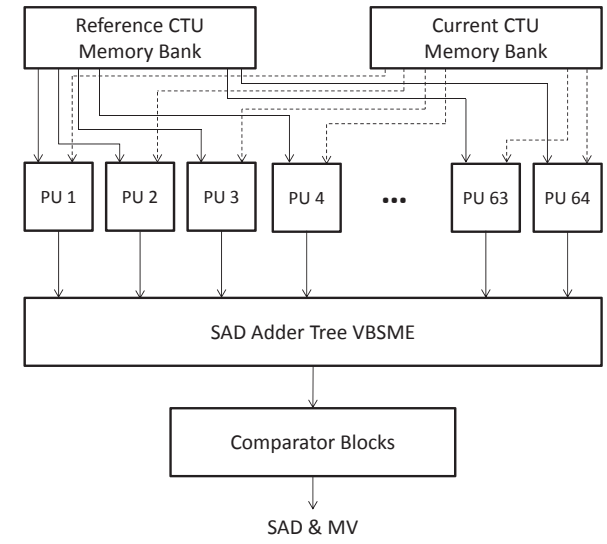
In this section, we present a high-performance IME hardware unit in HEVC that provides the minimum SADs and associated MVs of all possible partitions from a 64x64 CTU for inter prediction, exploiting parallelism in an efficient way. The system is composed of memory areas for current CU and reference search area pixels, 64 Processing Units (PU), one SAD Adder Tree Block (SATB), and one comparison block that saves the minimum SAD values and their corresponding MVs for all CU partitions. Fig. 3(a) shows the proposed hardware architecture.

As shown in Fig. 3(b), one PU consists of 64 Processing Elements (PEs), where each PE computes the difference of both the current and the reference pixel (see Fig. 3(c)). So each PU calculates the distortion values of a column of 64 pixels. At each clock cycle, current and reference pixel columns are delivered to the 64 PUs, being able to compute the pixel distortion values of a 64x64 block (maximum CU size) just in one clock cycle. That is, all distortions needed in order to obtain the SAD of a 64x64 CU in a particular position of the search area are calculated in just one clock cycle. The next block in our system, SATB, computes the SADs for all the possible prediction units (more than 600) by properly grouping the 64x64 pixel distortions obtained before.

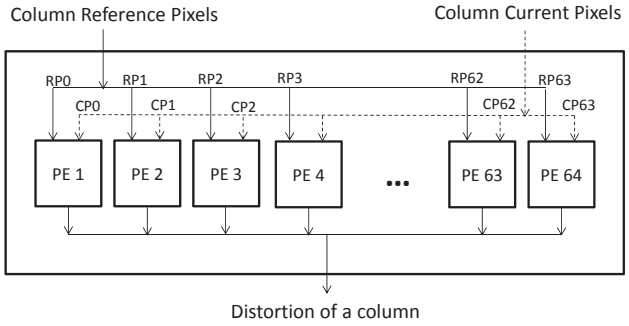
The process described above is performed for each of the positions of the search area, delivering the SADs to the comparison block, which is in charge of storing the minimum SADs with their corresponding MVs for each prediction unit of current CU. Table 1 lists the total number of different SAD partitions for a 64x64 CU.

### 3.1 Memory Read Controller Block

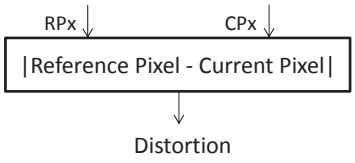
The memory read controller block is composed of a Block-RAM (BRAM) memory and a set of shift registers. A BRAM consist on an embedded memory block within the FPGA. Pixels belonging to the search area of the reference frame are stored in the BRAM and current CU pixels are saved in each PE. The reference pixels are spread from BRAM to the set of shift registers that are responsible for feeding PEs in order to calculate the distortion of the current CU in a particular search area position.




(a) Hardware block architecture




(b) Processing Unit (PU)




(c) Processing Element (PE)

**Fig. 3** Proposed IME architecture

The search area is just centered on the location of current CU and the default search window spans ±64 pixels from the current CU position, which defines a 128x128 search area as shown in Fig. 4. That is, the current CU will be matched in 128x128 different pixel positions, being necessary to load on BRAM memory the pixels belonging to a reference frame area of 191x191 pixels.

In order to provide high data reuse, a snake scan order and a reconfigurable data path with 64 propagation registers are adopted. The snake scan order visits all positions of the search area following a Hamiltonian path composed by consecutive vertical scans with alternating

**Table 1** Total number of SADs for each partition in a 64x64 CU

| Block size | No. of SADs | Block size | No. of SADs |
|---|---|---|---|
| 64x64(2Nx2N) | 1 | 32x32(2NxnU) | 8 |
| 64x64(2NxN) | 2 | 32x32(2NxnD) | 8 |
| 64x64(Nx2N) | 2 | 16x16(2Nx2N) | 16 |
| 64x64(NxN) | 4 | 16x16(2NxN) | 32 |
| 64x64(nLx2N) | 2 | 16x16(Nx2N) | 32 |
| 64x64(nRx2N) | 2 | 16x16(NxN) | 64 |
| 64x64(2NxnU) | 2 | 16x16(nLx2N) | 32 |
| 64x64(2NxnD) | 2 | 16x16(nRx2N) | 32 |
| 32x32(2Nx2N) | 4 | 16x16(2NxnU) | 32 |
| 32x32(2NxN) | 8 | 16x16(2NxnD) | 32 |
| 32x32(Nx2N) | 8 | 8x8(2Nx2N) | 64 |
| 32x32(NxN) | 16 | 8x8(2NxN) | 128 |
| 32x32(nLx2N) | 8 | 8x8(Nx2N) | 128 |
| 32x32(nRx2N) | 8 | **TOTAL** | **677** |

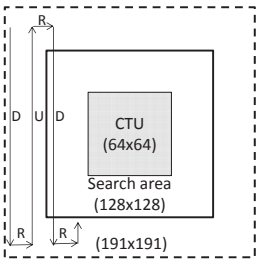



**Fig. 4** Scan order of search area

directions (the first vertical scan begins from top to bottom, then moves one pixel to the right and starts the next vertical scan in a bottom to up direction, and so on) as illustrated in Fig. 4. So, there are three scanning directions U (upward), D (downward), and R (rightward).

The current 64x64 CU pixels are stored in the PEs just only once (at the beginning). The reference pixels will also be loaded to the PEs but instead of loading from BRAM will be loaded from the shift registers, since they will help us to perform the snake scan order and as a consequence a huge reduction of memory load operations will be achieved.

So, the memory controller will be the one that manages the shift registers set by loading rows of 64 pixels from the reference frame area (BRAM) and performing the shift register operations to cope with the snake scan order.

In Fig. 5, a diagram with the shift register set and the loading and shifting operations is shown. At the beginning, the register set is empty, so we have to perform several (64) load and shift operations before calculating the first SAD. As can be seen in Fig. 5, the first 64 clock cycles are dedicated to load the first 64-pixel rows starting from the left most upper position of the search area, following a downward (D) scan direction. In this

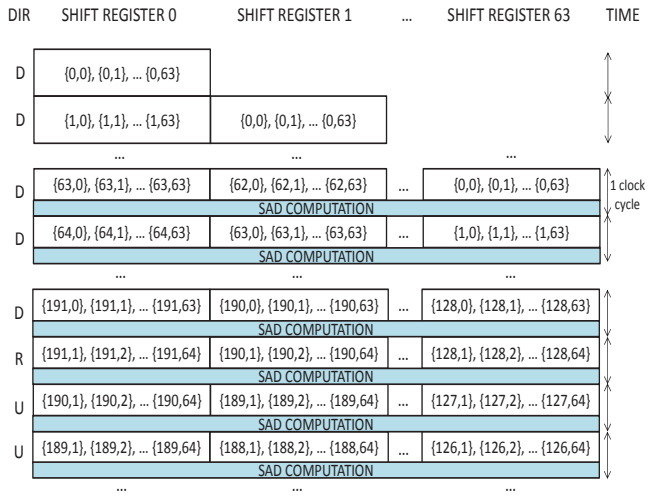



**Fig. 5** Shift registers set: Loading and shifting operations

figure, each 64-pixel row is labelled with the (x,y) pixel locations of the reference frame area. After loading the 64x64 reference frame block, all the pixels are sent to the PEs to compute the SAD in just one cycle (remember that the actual 64x64 CU pixels are already stored in the PEs waiting for this operation). At this point, the first SAD is computed. After that, we proceed in the D scan direction to compute the SAD of the next search area position. For this purpose, we only need to load an additional 64-pixel row in the D scan direction. So, in one clock cycle, (a) a right-shift operation is done, discarding the first pixel row stored in the shift register 63, and (b) the new pixel row is loaded from BRAM in shift register 0. Then, the 64x64 pixels stored in the shift registers are sent to the PEs to compute a new SAD.

After computing the last SAD in the downward scanning direction, we have to change the scan direction from D to R, following the snake pattern described before. Moving the search area position one pixel to the right could be easy if we simply shift to the left one pixel in all shift registers (see Fig. 5 at R scan direction). So, shift registers will contain the 64x64 search area block corresponding to the new position, and ready for the corresponding SAD computation.

After computing this SAD, we change again the scan direction from R to U, so we need to load a new 64-pixel row from BRAM, but now the loading is performed in the last shift register (63) and the register shift operation will be set to the left, discarding the contents of the first shift register (0).

The new SAD may now be computed, and as the scan direction is upwards, loading and shifting operations will be performed in the same way until a new change in scan direction is found.

This procedure will iterate until all searching area positions have been processed, providing one SAD at every clock cycle to the next module in the proposed architecture, the SATB.

### 3.2 SAD Adder Tree Block

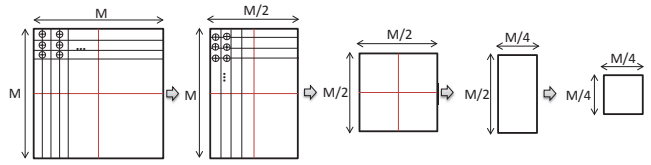
The SATB block is in charge of computing the SAD values for all partitions of each 64x64 CTU at every clock cycle. For Inter prediction, the HEVC standard proposes a partition size that ranges from 64x64 (maximum CU size) to 4x8/8x4 with different shapes - square, symmetric, and asymmetric partitions. After receiving the 64x64 distortions associated to the current search area position, a succession of aggregation stages are performed in this block to compute the corresponding SAD values for all the CTU partitions (a total number of 677), as shown in Fig. 6.

At the first stage, Fig. 6(a), all pairs of consecutive distortion columns/rows of the input 64x64 SAD block (M=64) are added, reducing to half the width/height of the resulting partition, until the block size of these added distortions is reduced to 16x16, from which the first SADs are obtained.

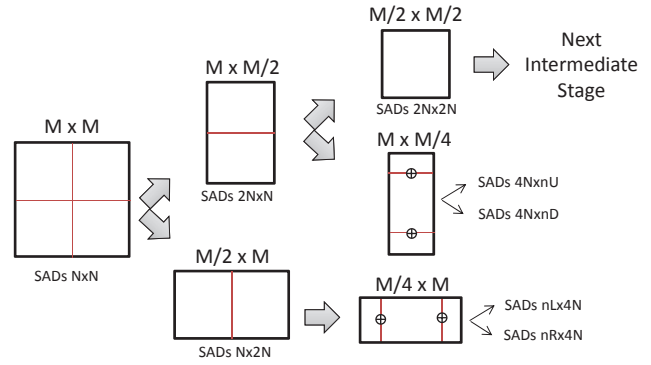
At the next three intermediate stages, a similar process to the one described above is followed. The successive sums of different configurations (row-column, column-column, row-row) are performed in order to get the SADs of all partitions of a 64x64 CTU. For instance, in the first intermediate stage, starting with a 16x16 block of intermediate values (M=16), all pairs of consecutive values for columns/rows are added as shown in Fig. 6(b). So, both the 16x8 (MxM/2) and the 8x16 (M/2xM) intermediate blocks, each one with 128 SADs correspond to 2NxN and Nx2N symmetric partitions of all possible 8x8 CUs contained in the current 64x64 CTU. This SAD aggregation process is followed until the last partition size is reached (1x1), i.e. the SAD in the last stage corresponding to the 2Nx2N partition of 64x64 CU (see Fig. 6(c)).

A particular case is the way asymmetric partitions are obtained from SADs corresponding to symmetric partitions. The idea is to repeat the same type of aggregation as the last one performed. If the start block has been obtained by the sum of consecutive columns, then the resulting consecutive columns are added again. The obtained values are SADs corresponding to asymmetric partitioning (left, right, up, and down) of the next size of CUs. For instance, in the last intermediate stage (M=4, N=16), after a sum of consecutive columns, we start with a 4x2 block of 8 SADs values corresponding to the 2NxN symmetric partition of the 4 32x32 CUs contained in the current 64x64 CTU. Then, all pairs of consecutive columns are added again as shown in Fig. 6(b). Thus, a 4x1 block of SAD values are obtained corresponding to 2NxnU and 2NxnD asymmetric partitions of the current 64x64 CTU.

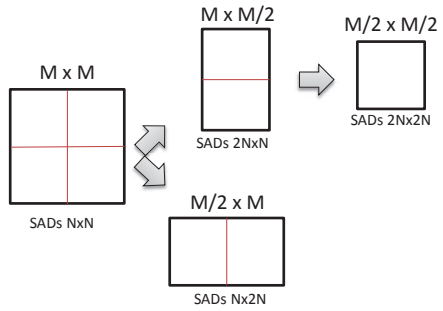
So, in the proposed architecture, the SATB module delivers 677 SADs of the current CTU block every single clock cycle to the next module, the comparison block.




(a) First stage: M=64




(b) Three intermediate stages: M=16 and N=4; M=8 and N=8; M=4 and N=16




(c) Last stage: M=2 and N=32

**Fig. 6** Structure of SAD Adder Tree Block

### 3.3 Comparison Block

The comparison block should keep the minimum SAD values for each CU partition with their corresponding motion vectors (search area positions). So, it will compare all incoming SADs from the SATB with the minimum SADs previously found. In a clock cycle, the comparison block receives 677 SADs corresponding to all partitions of all CUs contained in the current CTU, which is located in a particular position of the search area. So, in the next cycle, this module again receives 677 SADs corresponding to the next position of the search area. Therefore, this block compares SADs partition by partition, keeping the minimum SADs and the positions of the search area corresponding to those minimums. After comparing the SADs from the last search area location, the minimum SADs for each partition and the associated motion vectors are obtained.

## 4 Implementation Results

The proposed architecture is designed as a pipeline process shown in Fig. 7. The memory reading process and shift registers propagation require only one clock cycle. The PUs use one cycle, the SATB requires twelve additional clock cycles, and the comparison block needs one additional clock cycle. So, the proposed architecture requires 63 clock cycles to perform the initial load of the shift registers, 15 clock cycles to load the pipeline, and then as many clock cycles as positions the search area has.

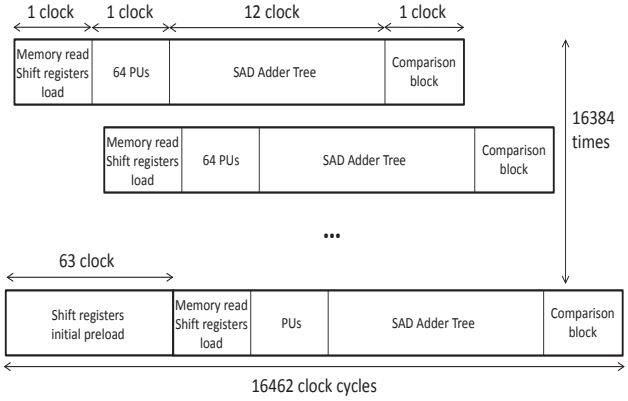



**Fig. 7** Pipeline process of the proposed architecture

Our proposal has been modelled in VHDL, and it has been synthesized, simulated, and implemented on the Xilinx FPGA, Virtex-7 XC7VX550T-3FFG1158. The correctness of our design was tested and verified with the HEVC HM 14 refrence model [17].

To evaluate the performance and efficiency of our design, we have parametrized our IME architecture to allow different configurations, such as (a) the maximum CTU size with values of 64x64 and 32x32, and (b) the size of the search area of the reference frame with values defined as the double size of the CTU, 80% of the double size of the CTU, and the same size as a CTU.

We proceed to first test our proposal with the Virtex 7 FPGA technology. In Table 2, we show (a) the resulting operating frequency (Clock), (b) the number of clock cycles for each CTU (latency), and (c) the system throughput in terms of the maximum frame rate under different video formats (1080p, 2K y 4K), for different configurations of CTUs and search area sizes. Our design can operate at the frequency of 247 MHz and 318 MHz for a 64x64 CTU and a 32x32 CTU, respectively. It enables the encoder to carry out the IME process with a 64x64 CTU size and a search area of 128x128 pixels (as the HM14 reference model [17] establishes), obtaining a throughput of 30 frames per second at 2K video formats (2K@30fps). Our proposal is able to process video in real time for both 1080p and 2K resolutions in all tested con-

**Table 2** Throughput for different configurations in Virtex-7

| CTU size | 64x64 | 64x64 | 64x64 | 32x32 | 32x32 | 32x32 |
|---|---|---|---|---|---|---|
| Search area | 128x128 | 104x104 | 64x64 | 64x64 | 52x52 | 32x32 |
| Clock (MHz) | 247 | 247 | 247 | 318 | 318 | 318 |
| Latency | 16462 | 10894 | 4174 | 4142 | 2750 | 1070 |
| Fps at 1080p | 32 | 48 | 124 | 39 | 59 | 151 |
| Fps at 2K | 30 | 45 | 116 | 37 | 55 | 141 |
| Fps at 4K | 8 | 12 | 30 | 10 | 15 | 37 |

figurations, and also with 4K video formats if the search area size is the same as the CTU size, as can be shown in 2.

In tables 3 and 4, we show the resources used to implement our proposal for maximum CTU sizes of 64x64 and 32x32, respectively, on a Virtex-7 FPGA. In both tables, we show the resource usage of each block of the proposed architecture, as a resource usage profile. As can be seen, the slice area required by flip-flops and LUTs increases ($\approx$ x4) linearly with the increase of the maximum CTU size, as expected. In terms of flip-flops, the SATB is the block that uses the most amount of them (around 40% of the total) in both configurations. This is due to the 12-stage pipeline design of the SATB. Moreover, calculating the distortion among pixels needs 50% of the LUTs, due to the amount of subtractions in absolute value required in this process, being 1024 operations done at each clock cycle. Regarding the required memory for storing the search area reference pixels, 36kB and 9 kB memories are used in the case of a 64x64 CTU and a 32x32 CTU, respectively. On a Virtex-7, a BRAM has a capacity of 36kbits. So, the slice area demanded by the used BRAMs also increases ($\approx$ x4) when going from 32x32 to the 64x64 maximum CU size.

An interesting analysis of our design can be observed at 2 when comparing the results of 64x64 search area size with both CTU sizes. As can be seen the latency is nearly the same but the throughput of the 64x64 CTU size is more than triple than the one obtained with 32x32 CTU size. In terms of resource usage the 64x64 CTU size requires near four times more resources, as shown at Tables 3 and 4. This implies the use of more resources in the design provides higher throughputs in a 4:3 relationship.

### 4.1 Systems evaluation

In Table 5, we compare our proposal with previous state-of-art architectures implemented on different FPGA platforms for both the 64x64 CTU and the 32x32 CTU, and different search area sizes. We have chosen those works whose architectures are comparable to our proposal (i.e perform the same functionality) and were implemented under FPGA technology. In order to make the comparison as fair as possible, we have obtained the performance results of our proposal with the same technologies, CTU sizes, and search area sizes than the ones used by the selected candidates. We will consider the system throughput as the main performance result of every proposal under comparison.

Regarding results for the 64x64 CTU size, Medhat et al. [3] present a parallel SAD block for the HEVC integer-pel full search architecture without supporting AMP modes with a search area of 104x104 pixels. They used the Virtex-7 technology, and their design can operate at the frequency of 458.7 MHz. The operating frequency of our proposal with the same technology an configurations is almost two times lower. However, our architecture is capable of processing 45 fps at 2K video formats instead of 30 fps as in [3]. Therefore, the proposed architecture is 1.5x as fast as the one proposed in [3] using the same search area size and considering all the AMP partition modes. This is due to the fact that our design takes advantage of the minimal latency to perform the same operations as we have an efficient pipeline design. Therefore, our system achieves higher throughput, reaching real-time processing for 2K video resolutions at 45 fps, and being on the way to accomplishing the same goal for 4K video formats, where 12 fps were obtained.

On the other hand, D'huys et al. [7] propose a reconfigurable design for HEVC motion estimation which can operate at the frequency of 150 MHz. They architecture is compared with our proposal, setting a common search area size to 64x64 pixels and the Virtex-5 technology. The operation frequency of our proposal is 159 MHz, achieving system througputs of 20 fps at 4K and 75 fps at 2K video formats. Our design significantly improves the performance presented in [7], which is able to process a lower resolution video (720p) at 57 fps. If the video resolution is set to 720p, our architecture is capable of processing 173 fps. So, our architecture presents good balance among the maximum frecuency and pipeline processing design, taking advantage of the low latency by leveraging all available resources.

Regarding results for 32x32 CTU size, in Table 5 is also shown the comparison results between our proposal (implemented on a Virtex-6 FPGA) and the integer motion estimation design found at [6], both with a search area size of 48x48 pixels. The most significant feature, worthy of attention, is that our proposal can provide a higher operation frecuency, achieving throughputs of 43 fps at 1080p and 40 fps at 2K. Meanwhile the ar-

**Table 3** Utilization resources for 64x64 CTU implementation in Virtex-7

| Resources | Flip-flops | LUTs | Memory (kB) |
|---|---|---|---|
| Memory Read Controller Block | 36657 (25.40%) | 36413 (19.30%) | 36 (100%) |
| PUs (Distortion computation) | 32768 (22.71%) | 94208 (49.93%) | - |
| SAD Adder Tree Block (SATB) | 58727 (40.70%) | 47063 (24.95%) | - |
| Comparison Block | 16150 (11.19%) | 10980 (5.82%) | - |
| TOTAL | 144302 | 188664 | 36 |

**Table 4** Utilization resources for 32x32 CTU implementation in Virtex-7

| Resources | Flip-flops | LUTs | Memory (kB) |
|---|---|---|---|
| Memory Read Controller Block | 10155 (27.55%) | 9812 (20.22%) | 9 (100%) |
| PUs (Distortion computation) | 8192 (22.22%) | 24541 (50.57%) | - |
| SAD Adder Tree Block (SATB) | 14580 (39.55%) | 11445 (23.58%) | - |
| Comparison Block | 3937 (10.68%) | 2733 (5.63%) | - |
| TOTAL | 36864 | 48531 | 9 |

**Table 5** Comparison of the proposed architecture with state-of-the-art works

| Design | Medhat [3] | Proposal 1 | D'huys [7] | Proposal 2 | Yuan [6] | Proposal 3 |
|---|---|---|---|---|---|---|
| CTU size | 64x64 | 64x64 | 64x64 | 64x64 | 32x32 | 32x32 |
| Search area | 104x104 | 104x104 | 64x64 | 64x64 | 48x48 | 48x48 |
| Technology | Virtex-7 | Virtex-7 | Virtex-5 | Virtex-5 | Virtex-6 | Virtex-6 |
| Clock (MHz) | 458.7 | 247 | 150 | 159 | 110 | 200 |
| AMP | No | Yes | No | Yes | Yes | Yes |
| Throughput | 2K@30fps | 2K@45fps | 720p@57fps | 720p@173fps | 1080p@30fps | 1080p@43fps |
| Flip-Flops | 39901 | 144302 | 199682 | 178620 | 19744 | 43531 |
| LUTs | 24957 | 188664 | 210158 | 184288 | 55346 | 45752 |
| Memory (kB) | 44 | 36 | 1229 | 36 | 148 | 9 |

chitecture presented in [6] is able to achieve 30 fps at 1080p video formats, using a similar amount of FPGA resources.

Considering the results, our architecture shows an efficient implementation of available resources in FPGA, overcoming the performance of previous state-of-the-art architectures.

### 4.2 HEVC R/D performance and time profiling

In order to better understand the capabilities of IME hardware devices, we have performed a set of tests to analyze the benefits of including an IME FPGA-based accelerator, like the one proposed here, in the HEVC reference software in terms of speedup and observe how both parameters, the CTU size, and the search area size impact on the R/D performance of the HEVC encoder. To perform these tests, we have used the HEVC HM 14 reference model [17] working with the main profile and low-delay configuration mode. The HEVC reference software is compiled with Visual Studio 2010 with the default compiler options and run over a PC platform with an Intel Core i7-3770 CPU 3.40GHz with 16GB RAM. Three video sequences from the HEVC common conditions video set were selected: (s1) Racehorses at 832x480 resolution (30 fps), (s2) BasketballDrive at 1920x1080 (50 fps), and (s3) PeopleOnStreet at 2560x1600 (30 fps).

The experiments were performed using different search area sizes (128x128, 104x104, 64x64, 52x52, and 32x32) and CTU sizes (64x64 and 32x32).

Tables 6 and 7 show all data gathered for CTU sizes of 64x64 and 32x32, respectively. The first row shows the total time (in seconds) required to encode each video sequence (10 frames). The second row shows the percentage of the total time needed by the IME software module using a full search algorithm (% IME Time SW). These percentages vary from 62% to 96% depending on the video sequence, the search area size, and the CTU size. As was expected, the time required by the IME software module decreases as both the search area size and the CTU size do. However, this time is not independent from the video sequence, and this is due to the early termination mode included in the HEVC reference software. Rows three and four show the number of CTUs per second that can be computed by software (CTU/s SW) and hardware (CTU/s HW) versions of IME module. As can be seen, these values also depend on the search area

**Table 6** Time profile of the IME HEVC for a 64x64 CTU

| Search area | 128x128 | | | 104x104 | | | 64x64 | | |
|---|---|---|---|---|---|---|---|---|---|
| Video sequences | s1 | s2 | s3 | s1 | s2 | s3 | s1 | s2 | s3 |
| Encoding Time SW (s) | 13670 | 61602 | 135970 | 9392 | 42050 | 92863 | 4117 | 17881 | 39933 |
| % IME Time SW | 95 | 96 | 95 | 90 | 94 | 93 | 83 | 86 | 85 |
| CTU/s SW | 0.24 | 0.26 | 0.23 | 0.38 | 0.39 | 0.35 | 0.91 | 1.00 | 0.89 |
| CTU/s HW | 14993 | 14993 | 14993 | 22625 | 22625 | 22625 | 59172 | 59172 | 59172 |
| HW gain | 62260 | 57767 | 64800 | 59621 | 58312 | 65384 | 64856 | 59115 | 66477 |

**Table 7** Time profile of the IME HEVC for a 32x32 CTU

| Search area | 64x64 | | | 52x52 | | | 32x32 | | |
|---|---|---|---|---|---|---|---|---|---|
| Video sequences | s1 | s2 | s3 | s1 | s2 | s3 | s1 | s2 | s3 |
| Encoding Time SW (s) | 3652 | 15892 | 35295 | 2634 | 11303 | 25293 | 1378 | 5748 | 12911 |
| % IME Time SW | 85 | 87 | 86 | 79 | 81 | 81 | 60 | 63 | 62 |
| CTU/s SW | 4 | 5 | 4 | 6 | 7 | 6 | 14 | 17 | 15 |
| CTU/s HW | 76923 | 76923 | 76923 | 115607 | 115607 | 115607 | 297619 | 297619 | 297619 |
| HW gain | 20383 | 17293 | 19457 | 20654 | 17384 | 19675 | 21096 | 17664 | 19912 |

size and maximum CTU size, and in the case of the IME software module, also depends on the video sequence because of the early termination process available in the HEVC reference software.

So by looking at the information provided in tables 6 and 7, we could assess that the IME module is a bottleneck in the HEVC reference software. Therefore, if the IME software module is replaced by our FPGA-based device, the overall encoding time will be significantly reduced. For example, for a high resolution video sequence like PeopleOnStreet (s3) and setting the CTU size to 64x64 and the search area size to 128x128 (default values in the HEVC reference software), the total encoding time (10 frames) will be reduced from 38 hours to 2 seconds, since the motion estimation module takes around 95% of the overall encoding time.

In order to reduce the hardware complexity, allowing faster versions with reduced power consumption, the CTU size and the search area must be reduced as much as possible. However, this may cause performance degradation in the encoding process, decreasing the overall video quality and/or reducing the compression rate. To evaluate this question, we will analyze the impact of these parameters on the R/D (rate/distortion) performance. In Fig. 8, we show the video quality of the test video sequence RaceHorses (s1) for each CTU and search area sizes at different compression levels (QP values). As can be seen, there are slight differences between the CTU size, being greater the difference as the compression rate increases. Differences between search areas are negligible. Although R/D differences may depend on the video content, similar results were obtained for the other two video sequences tested.

Finally, we also have performed a profile of the IME HEVC with another motion search algorithm which is available in the HEVC reference software (diamond-like search). This algorithm is used by default in the reference software and it is about 90 times as fast as the full search algorithm, with the disadvantage that it does not guarantee estimating optimal MVs, and as consequence video quality could be affected. As can be seen in Table 8, the inclusion of our IME hardware module will speed up the IME computation 230 times and 700 times for 32x32 and 64x64 CTU sizes, respectively.

After performing the whole analysis, a trade-off should be taken in order to determine which configuration bet-

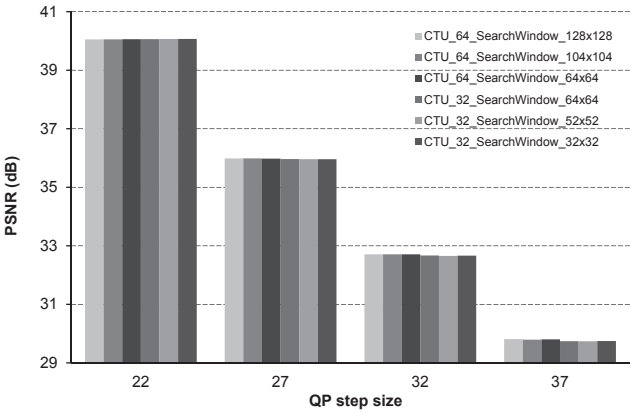



**Fig. 8** R/D performance for different CTU and search area sizes for the RaceHorses sequence

**Table 8** Average CTU IME time with 2xCTU search area size

| CTU size | Full search SW | Diamond search SW | Full search HW |
|---|---|---|---|
| 64x64 | 4.11 s | 4.65E-02 s | 6.67E-05 s |
| 32x32 | 2.48E-01 s | 3.05E-03 s | 1.30E-05 s |

ter adapts to the application requirements (low power consumption, encoding time, compresed video quality). The use of hardware accelerators designed in FPGA platforms like the one proposed here are mandatory when real-time UHD video encoding is the objective.

## 5 Conclusion

In this work we present a fast and efficient IME hardware unit for the HEVC video encoder which (a) supports AMP modes, (b) both CTU and search area sizes are configurable, and (c) is implemented on a Virtex-7 FPGA. The suitability of using FPGAs for implementing the HEVC IME module has been demonstrated in this paper, proposing a design that improves the previous results of other IME hardware systems.

Our FPGA-based design is able to process 2K video formats at 116 frames per second and 4K video sequences at 30 fps, which represents a huge complexity reduction of the HEVC video encoding process, achieving real-time encoding for high definition video contents and beyond.

We have also analyzed the impact of the maximum CTU and the search area sizes over the encoder complexity and the resulting video quality, showing that the encoder complexity decreases as both the maximum CTU size and the search area do. Furthermore, the maximum CTU size has a minimum impact over the R/D, being more noticeable at high compression rates. In the test video sequences analyzed, the impact over the quality of the search are size is negligible.

For future work, we are working to include our IME hardware module in the HEVC reference software and perform a complete test over an evaluation platform such as ZYNQ of Xilinx. In addition, we intend to expand the hardware module to perform the fractional-pel motion estimation, or even the SAD unit for Intra mode.

## References


1. Sullivan G.J. and Ohm J.R., Han W.J., Wiegand T. (2012) Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Trans. Circuits Syst. Video Technol. Vol:22, pp. 1649–1668
2. Sze V., Budagavi M., Sullivan G.J. (2014) High Efficiency Video Coding (HEVC) Algorithms and Architectures. Springer, Switzerland
3. Medhat A., Shalaby A., Sayed M.S., Elsabrouty M. (2014) A Highly Parallel SAD Architecture for Motion Estimation in HEVC Encoder. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS). Ishigaki, nov, pp. 280–283
4. Byun J., Jung Y., Kim J. (2013) Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD. Electronics Letters. Vol:49, No:18, pp. 1142–1143
5. Vidyalekshmi V.G., Yagain D., Ganesh Rao K. (2014) Motion Estimation Block for HEVC Encoder on FPGA. IEEE Int. Conf. Recent Advances and Innovations in Engineering (ICRAIE). Jaipur, may, pp. 1–5
6. Yuan X., Jinsong L., Liwei G., Zhi Z., Teng R.K.F. (2013) A High Performance VLSI Architecture for Integer Motion Estimation in HEVC. IEEE 10th Int. Conf. ASIC (ASICON). Shenzhen, oct, pp. 1–4
7. D'huys T. (2014) Reconfigurable data flow engine for HEVC motion estimation. IEEE Int. Conf. Image Processing (ICIP). Paris, oct, pp. 1223–1227
8. Davis P., Sangeetha M. (2014) Implementation of Motion Estimation Algorithm for H.265/HEVC. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering. Vol:3, No:3, pp. 122–126
9. Nalluri P., Alves L.N., Navarro A. (2014) High Speed SAD Architectures for Variable Block Size Motion Estimation in HEVC Video Coding. IEEE Int. Conf. Image Processing (ICIP). Paris, oct, pp. 1233–1237
10. Chen C.Y., Chien S.Y., Huang Y.W., Chen T.C., Wang T.C., Chen L.G. (2006) Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC. IEEE Trans. Circuits Syst. I: Regular Papers. Vol:53, No:3, pp. 578–593
11. Elhamzi W., Dubois J., Miteran J. (2014) An efficient low-cost FPGA implementation of a configurable motion estiation for H.264 video coding. Springer Journal of Real-Time Processing. Vol:9, No:1, pp. 19–30
12. Moorthy T., Ye A. (2008) A scalable architecture for variable block size motion estimation on field-programmable gate arrays. IEEE Canadian Conf. Electrical and Computer Engineering (CCECE). Niagara Falls, may, pp. 1303–1308
13. Kthiri M., Kadionik P., Levi H., Loukil H., Atitallah B., Masmoudi N. (2010) An FPGA Implementation of Motion Estimation Algorithm for H.264/AVC. IEEE 5th Int. Symp. I/V Communications and Mobile Network (ISVC). Rabat, sep, pp. 1–4
14. Pastuszak G., Jakubowski M. (2013) Adaptive Computationally Scalable Motion Estimation for the Hardware H.264/AVC Encoder. IEEE Trans. Circuits Syst. Video Technol. Vol:23, No:5, pp. 802–812
15. Pastuszak G., Trochimiuk M. (2015) Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder. J. Real Time Image Process. Online first articles
16. Lin Y.L.S., Kao C.Y., Kuo H.C., Hen J.W. (2010) VLSI Design for Video Coding - H.264/AVC Encoding from Standard Specification to Chip. Springer, New York
17. HEVC software repository HM–14.0 reference model. https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-14.0 (2014). Accessed 2 May 2014