

# Real-time motion estimation diamond search algorithm for the new high efficiency video coding on FPGA

Randa Khemiri<sup>1</sup> · Hassan Kibeya<sup>2</sup> · Hassen Loukil<sup>2</sup> · Fatma Ezahra Sayadi<sup>1</sup> · Mohamed Atri<sup>1</sup> · Nouri Masmoudi<sup>2</sup>

Received: 5 August 2017 / Revised: 31 October 2017 / Accepted: 6 November 2017 / Published online: 10 November 2017  
© Springer Science+Business Media, LLC, part of Springer Nature 2017

**Abstract** High efficiency video coding (HEVC) is the latest video coding standard aimed to replace the H.264/AVC standard according to its high coding performance, which allows it to be mostly suitable for application in high definition videos. However, this performance is accompanied by a high computational complexity due principally to the motion estimation (ME) algorithm. As in H.264/AVC, the ME in HEVC is a highly computational demanding part that takes the largest part of the whole encoding time. Hence, many fast algorithms have been proposed in order to reduce computation, but, the majority, do not study how they can be effectively implemented by hardware. In this paper, two hardware architectures of the diamond pattern search algorithm for HEVC video coding with sequential and parallel techniques, are proposed. These architectures are based on parallel processing techniques. The sequential and parallel VHDL codes have been verified and can achieve at a high frequency on a Virtex-7 field-programmable gate-array design (FPGA) circuit. Compared to other designs, our parallel design provides better efficient implementation of available resources on FPGA. Our architecture can meet the real-time processing of the FHD @ 30 frames per second.

**Keywords** HEVC · Motion estimation · Diamond search pattern · Hardware implementation · VHDL · FPGA

## 1 Introduction

With the rapid development of multimedia technologies and network communication, video contents have recently been presented to be of a high definition (HD) and an ultra HD. When compared with the standard definition videos, the resolution of the HD videos is larger and the visual quality is better [1]. Extreme demands are put forward to the video processing technologies. To follow these great demands, the International ITU-T Video Coding Express Group (VCEG) and the joint collaborative team on video coding (ISO/IEC) moving picture express group (MPEG) have developed the new generation of video coding standard, called high efficiency video coding (HEVC) [2].

Compared to its predecessor H.264/AVC, HEVC offered around 50% higher compression efficiency (bit rate saving) at the same video quality [3, 4]. The major loads at video coding are Motion estimation (ME) and Motion compensation (MC), whose consumption exceeds 90% of the inter prediction time [5]. ME is more challenging and time-consuming than MC. The HEVC ME consumes about 70% of the inter prediction time [6]. This is due to the new block coding hierarchy.

This new concept called Coding Tree Units (CTUs) is similar to the macroblocks in H.264/AVC. Each picture frame is split into square blocks with a maximum size of  $64 \times 64$ , called Coding Units (CUs) [7], and recursively subdivided into  $8 \times 8$  square blocks. Each CU contains some prediction and transform blocks, named Prediction Units (PUs) and Transform Units (TUs), where a PU is the main unit in the motion estimation algorithm. Whence the

✉ Randa Khemiri  
randa.khemiri@gmail.com

<sup>1</sup> Electronics and Microelectronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Environment Street, 5019 Monastir, Tunisia

<sup>2</sup> Electronics and Information Technology Laboratory, National Engineering School of Sfax, University of Sfax, Sfax, Tunisia

name quad-tree process of CU, PU and TU, as represented in Fig. 1.

ME technique is founded on the similarity between the current and the reference frame. This process finds the best matched block in the future frames for every block in the current video frame. The ME is defined by the translation position of the picture area in the current frame compared to the reference frame. The main goal of the ME process is to find the motion vector (MV) (Fig. 2), which represents the displacement of the best-matched block, in one search window, defined in the configuration file of the HEVC code. The motion vector is computed when the best-matched block is selected.

The matching algorithm is based on computing an error cost function between the current and the reference frame, known by SAD value, called Sum of Absolute Differences. When RB and CB represent the reference pixels and the current block, the SAD is, then, defined as:

$$SAD = \sum_{i=1}^M \sum_{j=1}^N |CB(i,j) - RB(i,j)| \quad (1)$$

There are many well-known sequential fast-search algorithms such as the diamond search (DS) algorithm [8], the cross-diamond search (CDS) [9], the horizontal diamond search algorithm (HDS) [10], and the line diamond parallel search algorithm (LDPS) [11]. To reduce the ME computational complexity, the time required for the search algorithm should be reduced. The parallelization of one diamond pattern reduces the computational complexity of all these algorithms, which decreases the ME in terms of execution time, and this, in turn, leads to the decrease of the total HEVC execution time.

This paper proposes two FPGA-based hardware architectures for diamond algorithm, with and without

parallelism, in order to get less computing time with minimum area.

The rest of this paper is organized as follows: Sect. 2 details some related works on HEVC ME algorithm to reduce the encoding time. In Sect. 3, an overview of the diamond pattern in the ME is given and followed by our proposed architectures detailed in Sect. 4. Then, experimental results and their comparison and analysis are presented in Sect. 5. Finally, the conclusion is given in Sect. 6.

## 2 Related works

The complexity in the HEVC encoder is a critical topic. Since the motion estimation part takes the lion’s share of the encoding time (70% of the inter prediction time [6]), some works revolve around the reduction of the motion estimation time which leads to the reduction in the computational complexity of the total HEVC. This time reduction is done through the software and the hardware methods.

For the software, three types of search: Horizontal diamond search (HDS), Large diamond pattern search (LDPS) and small diamond pattern search (SDPS) are proposed in [12, 13] to replace the TZSearch algorithm for the HEVC encoder. Using a Windows 7 with an Intel® core TM i7-3770 @ 3.4 GHz processor and 12 GB RAM, the proposed algorithms allow saving around 49% of ME computational time with a little decrease in bit-rate and PSNR compared to the original HM8.0 test model. To reduce the HEVC encoding time, the authors in [1] propose a fast inter CU decision based on SAD algorithm. The simulations were achieved on an Intel @ 2.30 GHz CPU, 32 GB RAM.

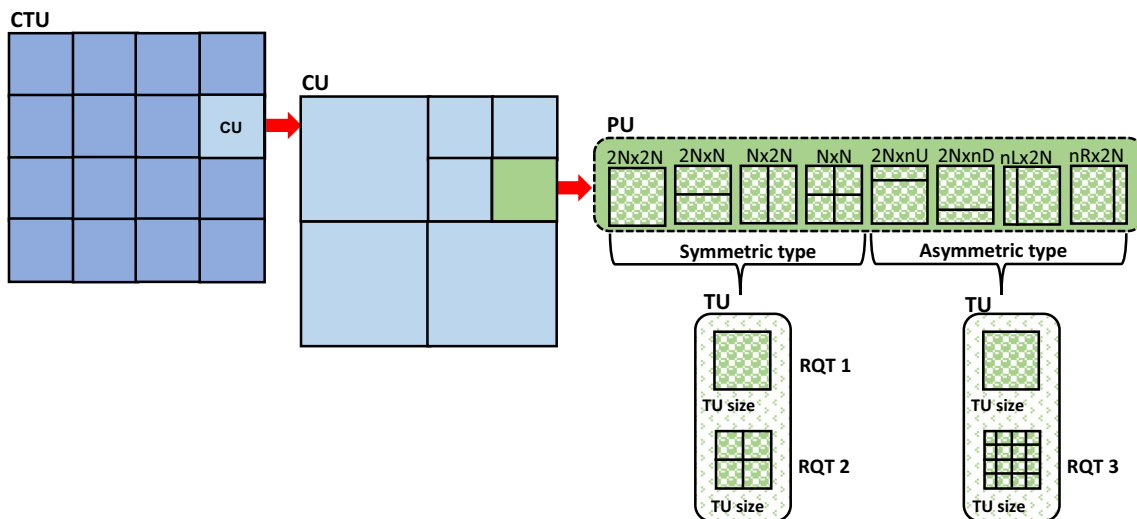
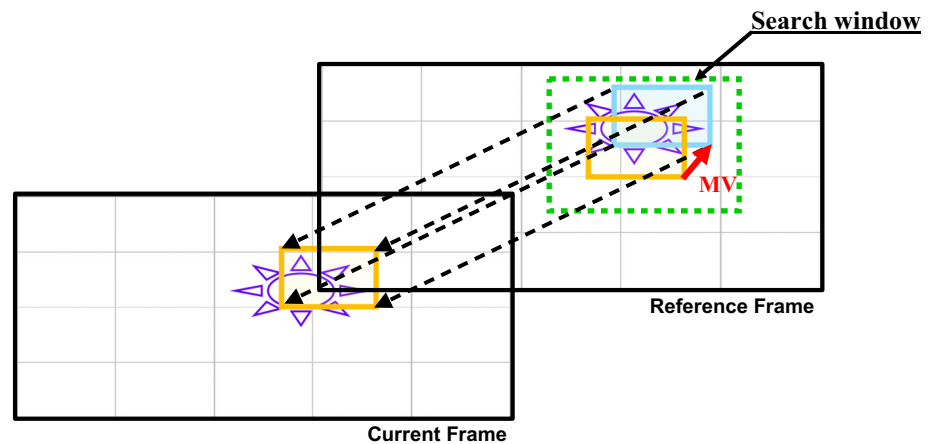


Fig. 1 Partitioning modes of coding unit

**Fig. 2** Motion estimation process



Nalluri et al. [14] replace the 8-diamond-point and 8-Square-point by a 6-Hexagon-point in the TZS algorithm, which leads to a reduction in the encoding time by almost 50% without any degradation in the bit-rate and the video quality. All simulations were carried out on Windows 7 with an Intel® I7@2.93 GHz processor and 4 GB RAM. In the work published by Khemiri et al. [15] one fast ME of HEVC video encoder based on diamond pattern search and three fast mode decisions; the coded block flag (CBF) fast method (CFM), the early CU termination (ECU) and the early skip detection (ESD) modes is presented. The results, obtained on Windows 8 with an Intel® core TM i7-3770 @3.4 GHz processor and 12 GB RAM, prove that the HEVC complexity is saved by 56.75%, with a little degradation in the bit-rate and the video quality, compared to the HM.16.2 test model. In [16], the authors propose an implementation of the Sum of Absolute Differences and the Sum of Square Differences (SSD) algorithms on a NVIDIA GeForce GTX 480 with CUDA language to optimize the ME time. For class E, the time saving obtained is around 32.14% accompanied by a non-significant degradation in the bit-rate and the PSNR, compared to the ME in the HM16.2 test model. Dongkyu et al. [17] propose a parallel HEVC integer-pel ME (IME) algorithm, where concurrent parallel reductions (CPR) are introduced. This is done using a parallel processing architecture, performed on Graphic Processing Unit (GPU) Geforce GTX 780 with 3 GB DRAM, with CUDA language. The proposed algorithm gives up to 172.6 speedup comparing the original HM10.0 test model. In 2015, Dongkyu et al. introduce, in [18], two different algorithms into the ME, providing an execution time 130 × faster than that of the HM version 10.0. The CPU and the GPU used in this work are an Intel Core i7-2600 @ 3.4, 8 GB memory, and a Geforce GTX 780 with 3 GB DRAM, respectively. In [6], Jongho et al. reduce the ME time by around 50%, using the bi-prediction method, while maintaining the same image quality and bit-rate.

For the hardware reduction method, many efforts have been made, but a lot of them are dedicated to the previous video compression standard (H.264/AVC). In [19], a rapidly SAD algorithm is proposed on Xilinx SpartanII and Virtex-2 FPGA families for block sizes ranging from  $4 \times 4$  to  $16 \times 16$ . [20] presents a ME design executing a variety of block-matching search techniques. The performance measurements of the proposed architecture have been validated on Xilinx Virtex II Pro (XCV2P40) and Spartan 2 (XC2S150) FPGA. Kthiri et al. in [21–23], propose two new hardware architectures of fast-search block matching ME algorithm using LDPS for the H.264/AVC standard. The serial and parallel hardware designs reaches a maximum frequency of 390 and 420 MHz, respectively. The VHDL code is tested on ALTERA Stratix II and Xilinx Virtex-5 FPGA family.

Some hardware implementations reduce the motion estimation time of the HEVC encoder by reducing the SAD computational time. In [24], the authors propose a hardware SAD design for gray scale images. This proposed design allows the parallel calculation of all SAD of one defined block of  $4 \times 4$  dimension. The 16 SAD calculation units generate a total of 16 values that are added in a carry select adder to give a 12 bit result. A Dadda multiplier is used as a partial product reduction (PPR) for addition. The syntheses and the implementation by means of the Xilinx FPGA, Virtex-2 XC2V1000, show that the maximum frequency obtained is 133.2 MHz for  $4 \times 4$  blocks sizes. Nalluri et al., in [25, 26], propose two novel architectures to compute SADs with and without parallelism on FPGA Xilinx Virtex 5. The achieved results show that the parallel architecture is 3.9 times faster than the non-parallel. In [27], Yuan et al. propose a VLSI architecture for HEVC IME that supports coding tree block (CTB) structure with the asymmetric motion partition (AMP) mode. The architecture is achieved on Virtex-6 XC6VLX-550T with a 110 MHz system clock. Through the Xilinx Virtex-7 XC7VX550T FPGA and to compute the SAD values of the

prediction blocks, the authors in [28] propose a highly parallel SAD for HEVC ME based on 64 PUs working in parallel. In [29], Alcocer et al. propose a hardware HEVC ME implementation composed of memory areas for current CU and reference search area pixels, 64 processing units, one new SAD Adder Tree Block (SATB), and one block of comparison for saving the SAD with the minimum value and its corresponding MV. By properly grouping the  $64 \times 64$  pixel distortions obtained before, the SATB computes the SADs for all the possible prediction units. The proposed architecture is synthesized and implemented on Virtex-7 XC7VX550T-3FFG1158 FPGA. Their design provides 247 MHz frequency.

When analyzing these previous works, we remark that the parallelization of one diamond algorithm reduces the computational complexity of the DS [8], the CDS [9], the HDS [10], and the LDPS [11], which leads to the decrease of the ME in terms of execution time, hence the decrease of the total HEVC encoding time.

### 3 Diamond search in motion estimation process

In HEVC video coding, ME adopts two principal algorithms:

- Full search (FS) algorithm [30].
- TZSearch algorithm [8].

The FS process is considered as a native algorithm. It is based on the SAD computation of all pixels in the search area, in order to extract the minimum SAD, which corresponds to the best motion vector. ME occupies more than 96% of the intended encoding time with the FS as the main algorithm for the inter-prediction [31].

On the other hand compared to full search algorithm, TZ-Search reduces the computational complexity and increases the speed-up around 23 times [32], with smaller quality losses than FS algorithm. Moreover, despite this, as mentioned in the introduction, this algorithm occupies 70% of the inter prediction time [6]. It is an algorithm achieved in the reference software HM (16.2). This algorithm is divided into four distinct main steps: The PMV (prediction motion vector), the first research using the diamond or the square pattern, the raster search and the refinement. These steps are summarized as shown in Fig. 3. In the first step and in order to compute the median predictor of the corresponding block, three predictors are used: the left predictor, the up, and the upper right one. The second step consists of performing the first search, determining the “searchrange” and the search pattern. This step aims to find the search window using a diamond or square pattern. The raster search, the third step, consists in choosing the best distance corresponding to the best-matched point

obtained from the last search. Depending on this distance denoted “BestDistance”, three different cases are presented: If the best distance is equal to zero, then the process is stopped. If  $1 < \text{Bestdistance} < \text{iRaster}$ , a refinement stage is directly proceeded, where the “iRaster” is an adjustable parameter in the configuration file that should not be exceeded. If  $\text{Bestdistance} > \text{iRaster}$  which is set appropriately, a raster scan is achieved using the iRaster’s value as the stride length. The raster search is proceeded when the difference obtained between the starting position and the motion vector, obtained from the first phase, is too large. This raster search is carried out on the complete search window (named Full search). When the motion vector’s distance obtained in the previous steps is not equal to zero, the refinement step is performed to extract the position.

The raster search is rarely used in the algorithm, since the condition of transition to this stage ( $\text{bestdist} < \text{iraster}$ ) is practically never verified. However, the most intense stages in terms of execution time and computational complexity in TZSearch, are First search and Refinement.

Indeed, as shown in Fig. 4, the majority of ME time is taken by the First search and the Refinement part. Which represents 75 and 68%, respectively for Vidyo1 and Kimono, of the total TZSearch time. This test is done using the Random Access (RA) configuration using the two sequences (Kimono and Vidyo1) recommended by JCT-VC [33], with two resolutions ( $1920 \times 1080$ ) and ( $1280 \times 720$ ), respectively.

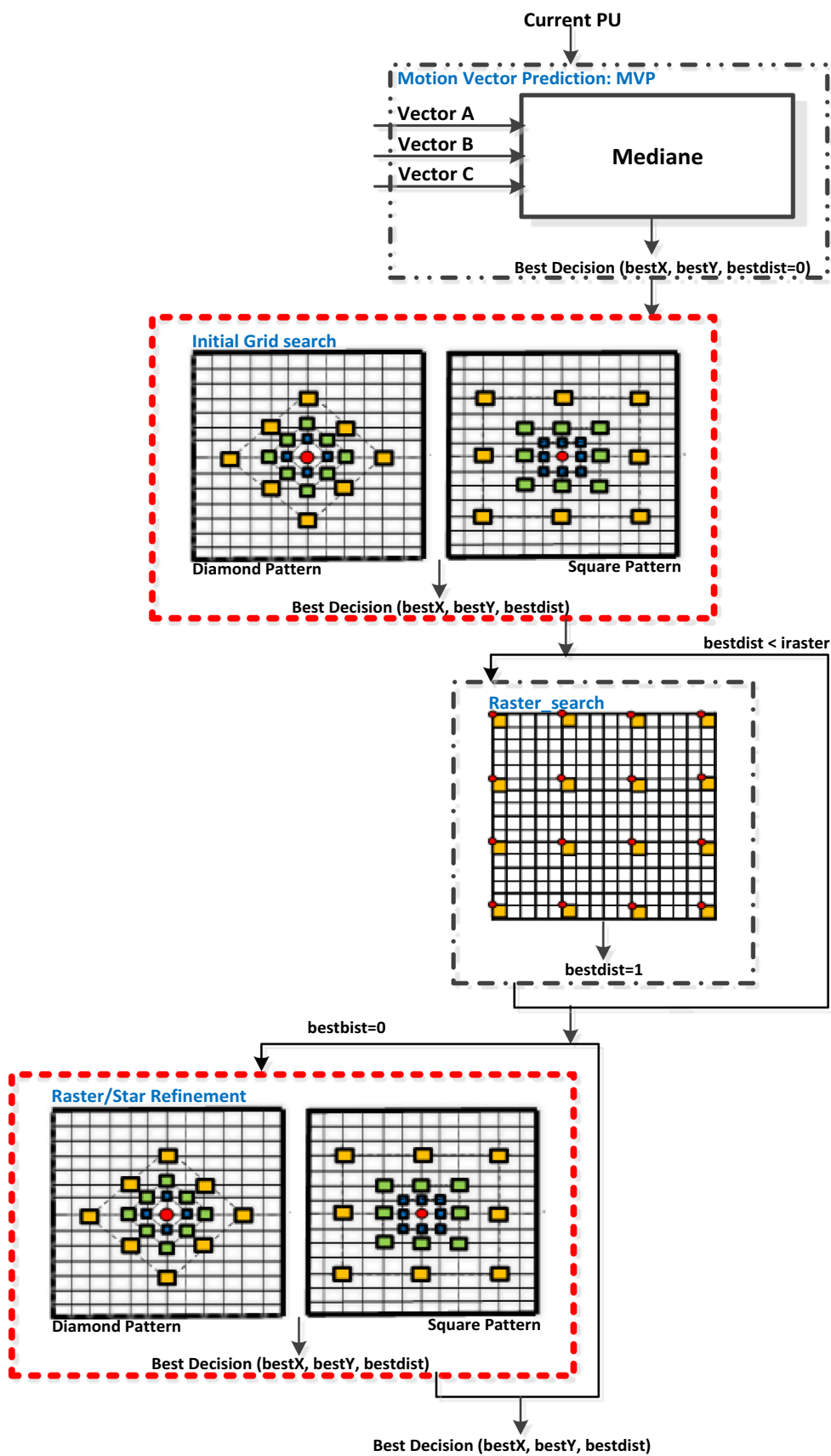
Both first search and refinement are based on the diamond pattern. In ME algorithm, the diamond pattern is repeated many times with different distances ranging from 1 to 64.

So, two types of diamond pattern are tested and calculated in DS algorithm: the small diamond pattern and the large one [34].

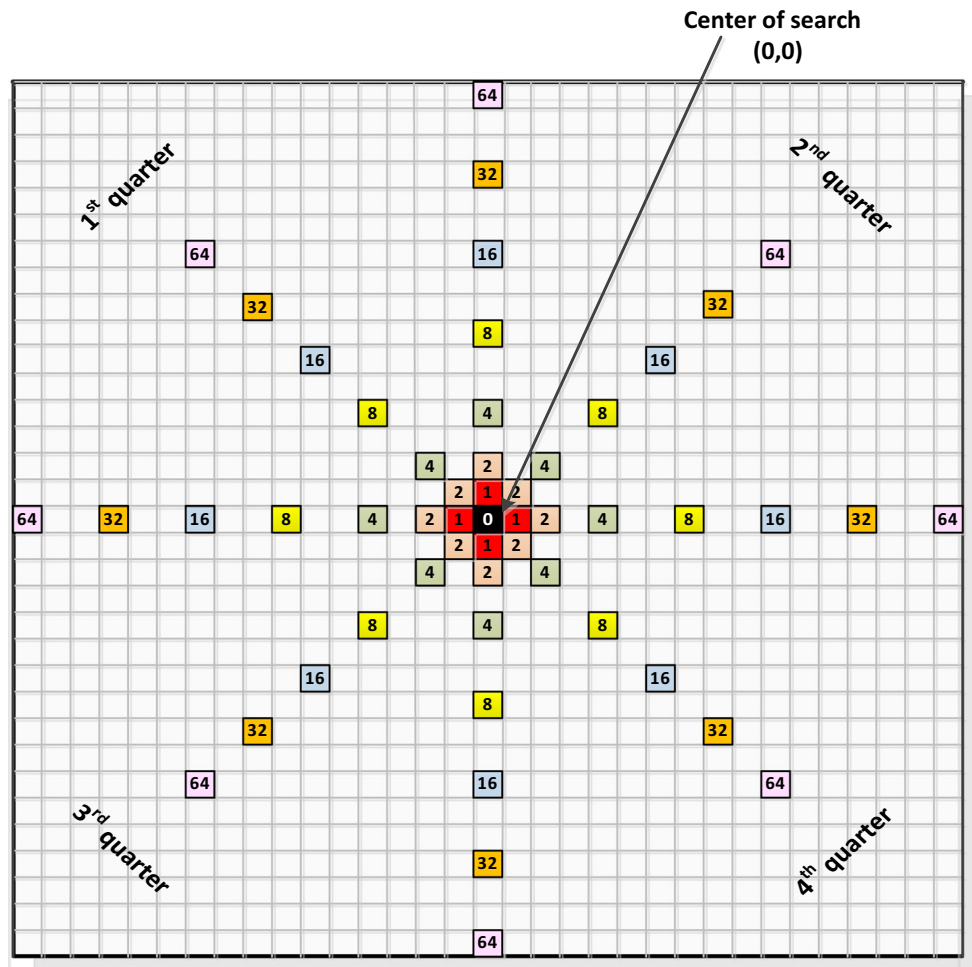
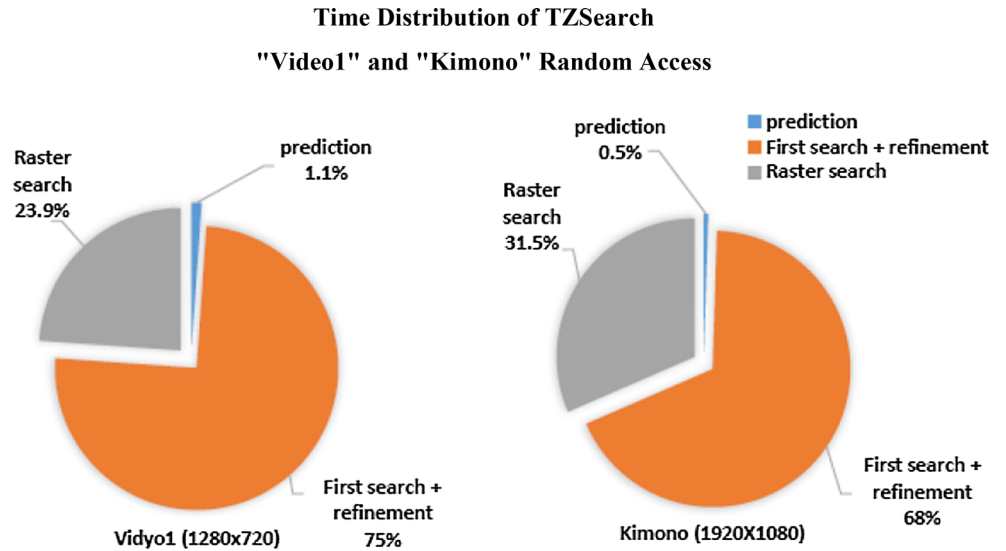
- The small diamond search pattern (SDSP): consisting of four search points forming a small diamond shape. This form is used only in the beginning, (the red one in Fig. 5).
- The large diamond search pattern (LDSP): consists of nine search points from which eight points encircle the center point to compose a diamond shape. This diamond form is used hundreds of times in a same search.

The search path method using the diamond search algorithm is described in Fig. 5. This pattern is axially symmetric and have the same priority in both horizontal and vertical direction. For each and every point, the SAD value is determined and a comparison is effectuated to extract the best one of every specified distance –with the optimal value-. The “distance” parameter represents the

**Fig. 3** Diagram of TZSearch algorithm



**Fig. 4** TZSearch profiling results



**Fig. 5** Diamond search points

distance from the central point to each point of the small or the large diamond (from distance 2 to 64). At the beginning, the SDSP is placed at the search window's center

(0,0), called original point (black point in Fig. 5). This first operation is performed for "distance" equal to 1 (red points in Fig. 5).

After that, the SAD is calculated for the eight blocks corresponding to the large diamond with “distance = distance  $\times$  2”. Then, we repeat the process recursively for the next LD, by multiplying the distance by two in each iteration. These operations are repeated until the SAD changes from one iteration to another. The process is stopped when the minimum SAD is unchanged after two iterations, or when the distance equals to search range (For HM16.2, the maximum window is set to 64). Figure 6 gives the DS flow chart, which describes the diamond search steps.

The DS algorithm is considered as the primary research objective to reduce computational complexity.

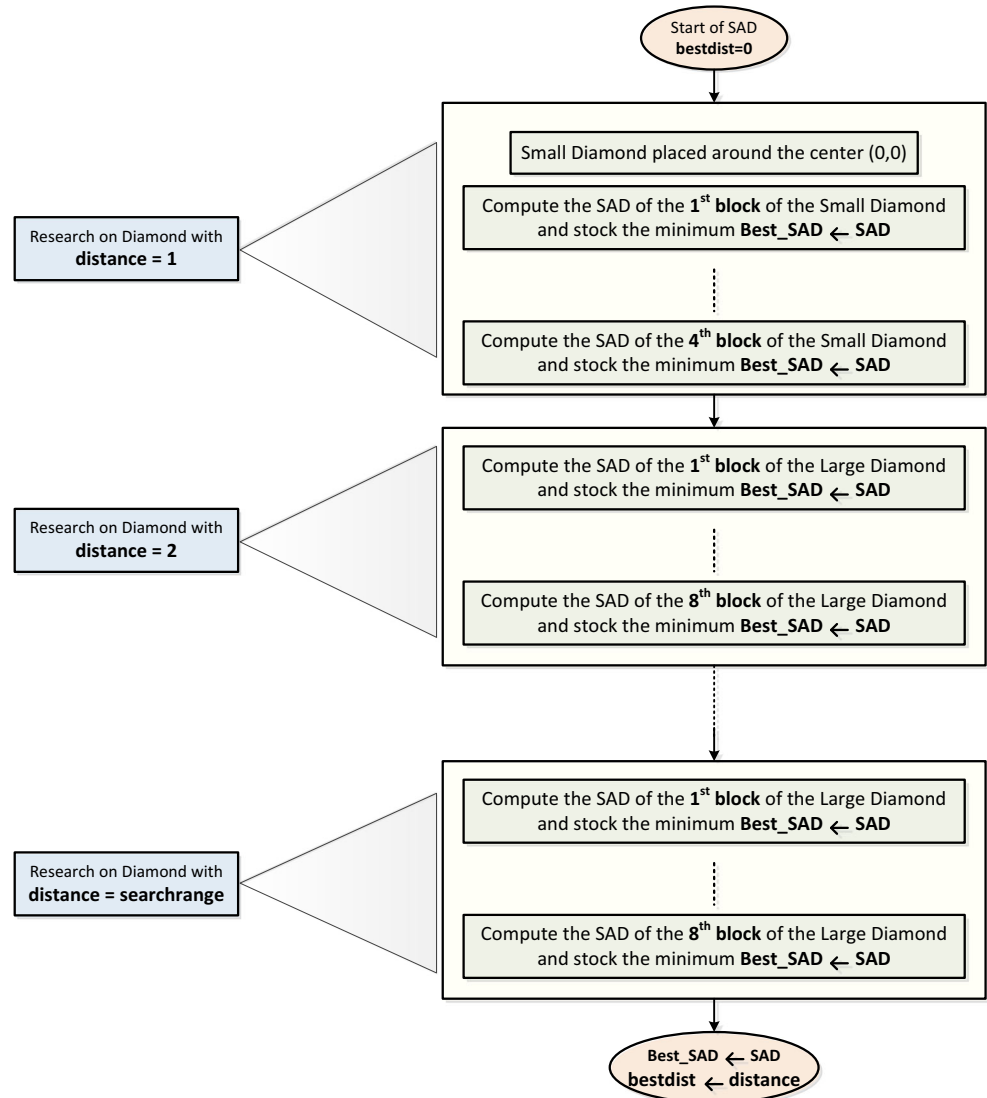
The diamond search algorithm is based on several diamond computation with different coordinates, i.e. diamond with different distances, distance = 1, 2, 4, 8, 16, 32 and 64. Thus, when optimizing the time of one diamond, just by modifying the distance parameter, the times of the other

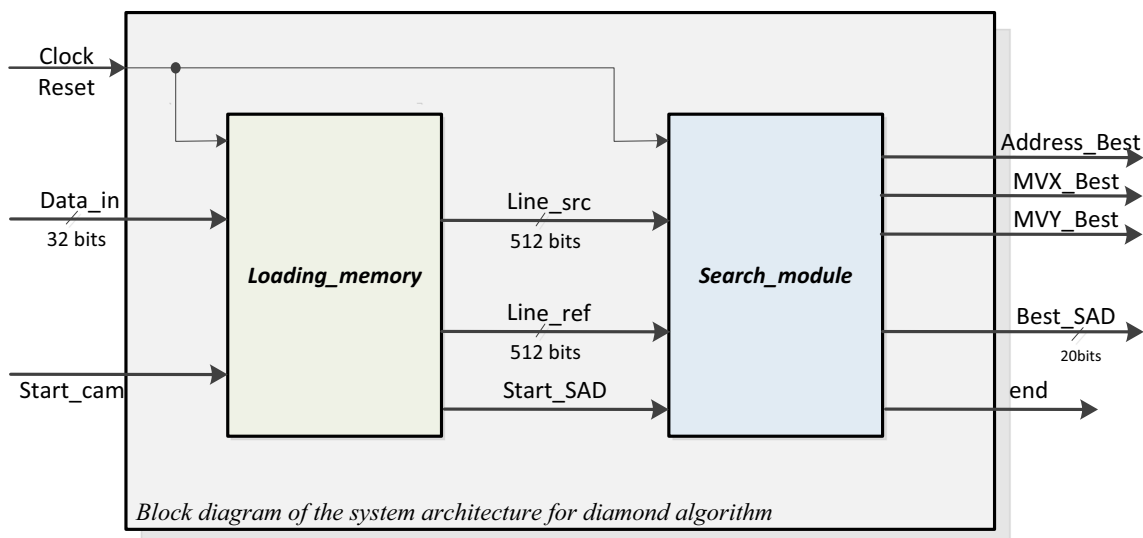
one will be reduced, which leads to the reduction of all the DS time and the ME time will be saved.

### 4 Proposed architecture

To reduce the HEVC computational complexity, the ME time will be reduced. One method to do this is to accelerate the diamond algorithm using FPGA device. Two hardware diamond algorithms (sequential and parallel architecture) are presented and compared, in this section, to obtain higher frequencies accompanied with the least possible area cost. Figure 7 illustrates the proposed architecture and its decomposition into sub-modules. Through the first module named “loading\_module”, the current CTU and the reference block are loaded, line by line. While through the search module, which represents the second module,

**Fig. 6** Diamond search algorithm





**Fig. 7** Block diagram of diamond algorithm

the motion vector and its coordinates according to the diamond algorithm are calculated.

These two sub-circuits are different for the proposed sequential architecture and the parallel one. Therefore, their details are described and detailed in the next subsections.

#### 4.1 Sequential calculation architecture

The sequential technique is based on a serial hardware architecture for the computation of the motion vector.

##### 4.1.1 Loading\_module

The reference search area proposed in our architecture, consists of  $144 \times 144$  pixels, is shown in Fig. 8. Where, the horizontal size of the search area is equal to horizontal size of the search window (64 pixels) in addition to the CTU line size (64 pixels) more 16 pixels. The same way is applied for the vertical size where 144 pixels are used instead of 64.

On the other hand, the source block is composed of  $64 \times 64$  pixels. Figure 9 shows the data disposition of the source block.

The loads of data from the camera are synchronized through the “Adaptator\_module” by delivering the control signal “start\_cam”. This is shown in Fig. 10. To store the reference search area and the source area in the memory, and since the data-bus is 32 bits, the transfer of the data is carried out at the rate of 32 bits per every one-clock cycle, until a line is complete. This mechanism is done line by line. Each memory point contains one source line followed by one reference line. Therefore, to load the first line of the source and the first line of the reference, 52 clock cycles

are needed ( $52 = (144 + 64) \times 8/32$ ). Where 36 clock cycles are needed for storing a reference line and 16 clock cycles for one source line. The lines loading of reference and source blocks are described as in Fig. 10.

The start of writing in the memory is reported with “start\_RAM” signal, delivered by the Adaptator\_module. This mechanism is repeated 64 times to load the source block. As to what concerns the reference search area, it requires 144 times to load. That is to say 64 times plus 80 times. It is worth clarifying that the loading for the source and the reference blocks is simultaneous. The writing end is known by the signal Read when “Read” = ‘1’. This is done in order to indicate that the data is ready for computation in the next module. Furthermore, if “Read” = ‘1’, from the memory, we can read the different stockpiled pixels by giving just the address of the preferred pixel, delivered by the control\_unit. Hence, limiting the access to the memory. The outputs of this Loading\_module, which are reference and source lines of 64 pixels, will be the input of the search module. Additionally, the loading module is controlled by the control\_unit. This control unit signals the start of finding the appropriate MV to the search module by a “start\_SAD” signal.

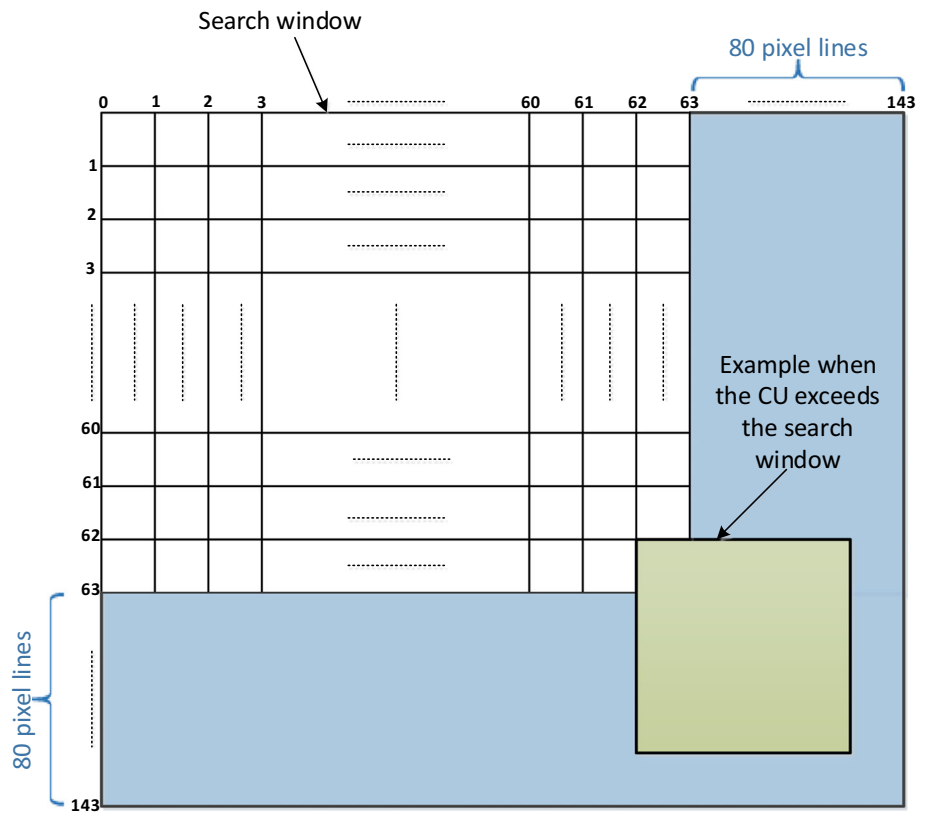
##### 4.1.2 Search\_module

Using the hardware component presented in Fig. 11, we can calculate the SAD value for the eight-reference block of the diamond.

This is done to choose the best estimated motion vector having the best SAD. The proposed architecture is made up of four sub-modules. First, we have the control\_unit, the most sensitive circuitry throughout our application, since, it is responsible for the addressing, the timing and the



**Fig. 8** Search area structure



Pixel(0)	.....	Pixel(63)
Pixel(64)	.....	Pixel(127)
Pixel(128)	.....	Pixel(191)
⋮	⋮	⋮
Pixel(4032)	.....	Pixel(4095)

**Fig. 9** Source block repartition

synchronization between all sub-modules: the Loading\_module, the SAD\_module and the comparator\_module.

Additionally, the control\_unit selects the pixels associated to each reference block. The “donefinal” signal indicates the calculation end of the corresponding SAD of the current block. Once the calculation of the first block is completed (“donefinal” = ‘1’), we move to the next block in order to compute its appropriate SAD value. The x and y signals on 7 bits are the coordinate addresses of the top-right-point of the block whose corresponding SAD we want to find. This mechanism will be repeated until the eight SADs of one diamond are computed. Finally, these eight

SAD values enter the comparator\_module to extract the Best\_SAD. The “end” signal indicates the end of calculation for all the blocks of the diamond.

**4.1.2.1 SAD\_module** Figure 12 shows the hardware component, SAD\_module, and its sub-modules. This module is used to compute the absolute difference and the sum between all reference and source pixels. The SAD\_module is composed of two sub-circuits: “Difference module” and the “Accumulator”, to calculate their addition.

We have to do the  $SAD_{64 \times 64}$ : i.e., we have 64 reference pixels minus 64 source pixels. Hence, we need an Abs\_Diff\_module composed of 64 Diff\_Abs units. Indeed, all pixels from the first current and reference pixels [Pixel\_ref(0), Pixel\_src(0)] to the latest current and reference pixels [Pixel\_ref(63), Pixel\_src(63)], are introduced to the SAD\_module. The entrance of the data in the Difference\_module is simultaneous and carried out two by two: the first reference [Pixel\_ref(0)] and first current [Pixel\_src(0)] pixels enter the first “Abs\_Diff (0)”, and the last reference pixels [Pixel\_ref(63)] and last current pixels [Pixel\_src(64)] enter the 64th “Abs\_Diff (63)”. All the results obtained from “Abs\_Diff (0)” to “Abs\_Diff (63)” blocks enter the following module; the “Accumulator”, where a sum of all the input values is carried out two by

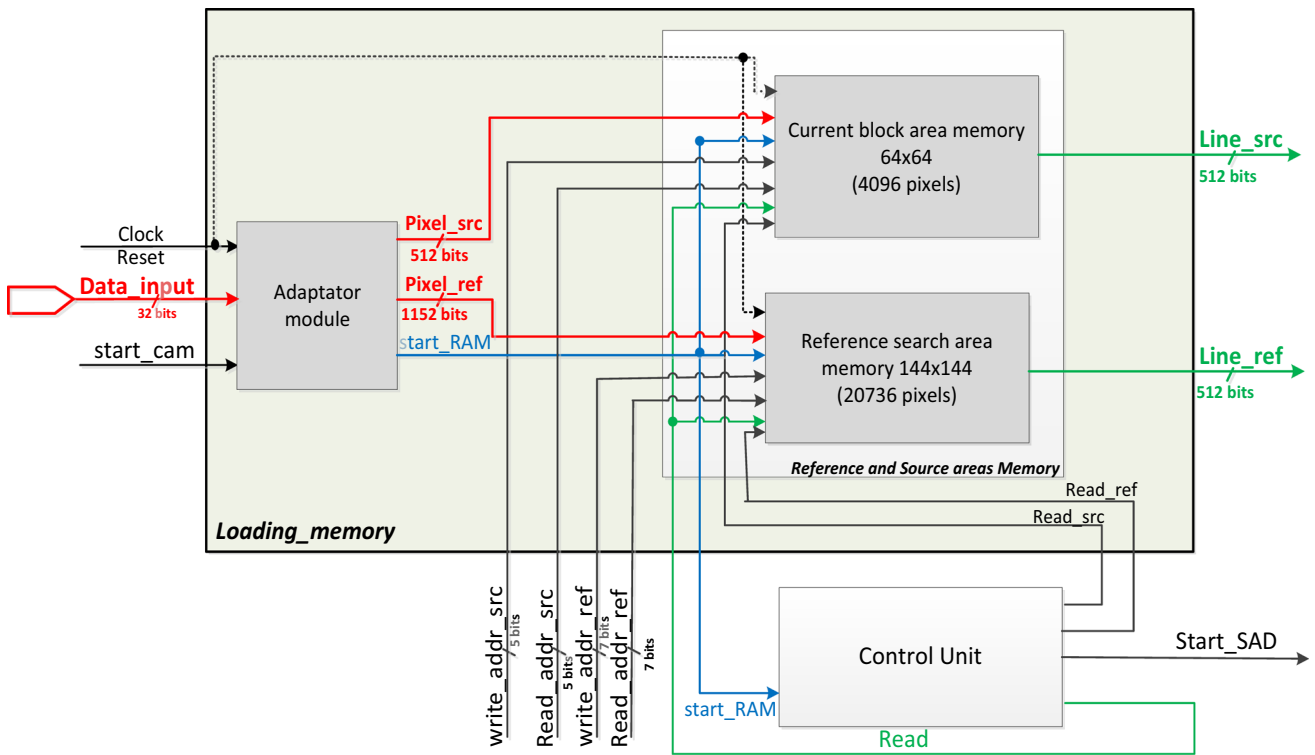


Fig. 10 Loading\_module

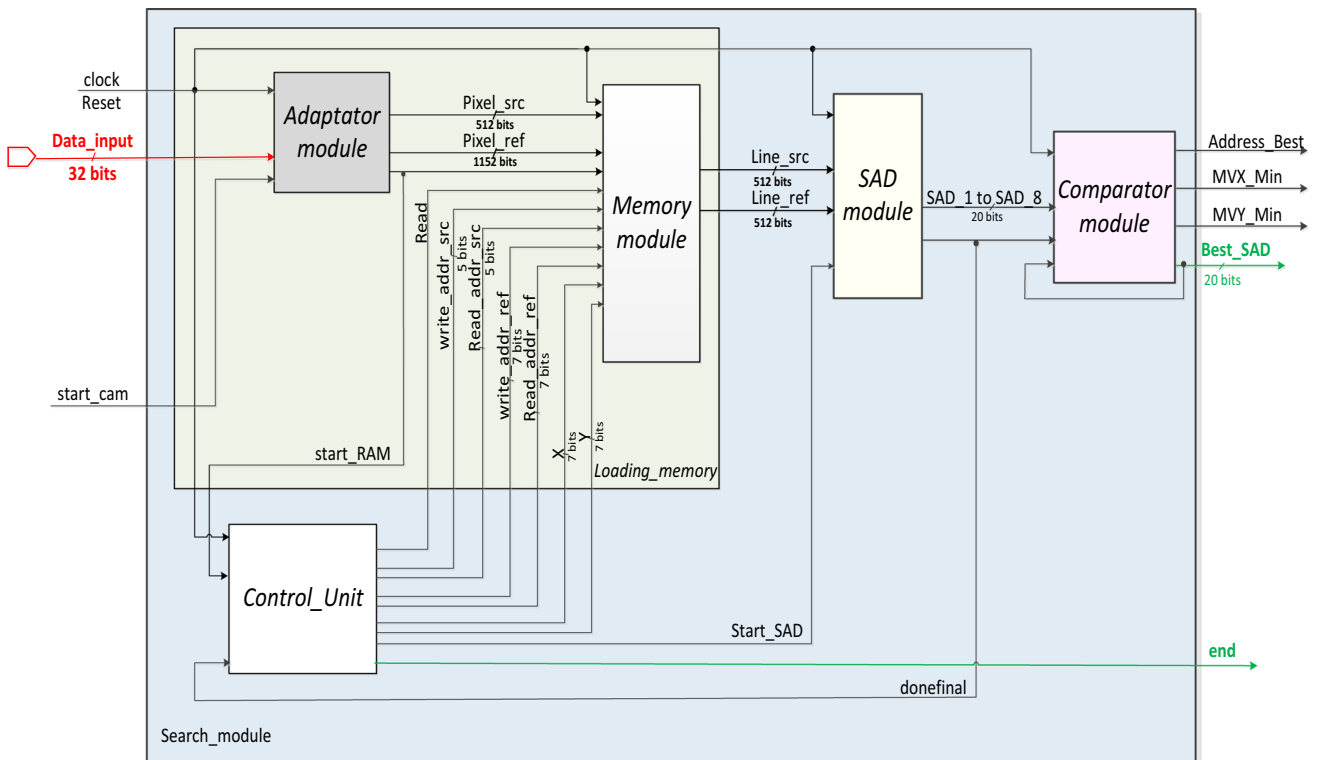


Fig. 11 Search\_module

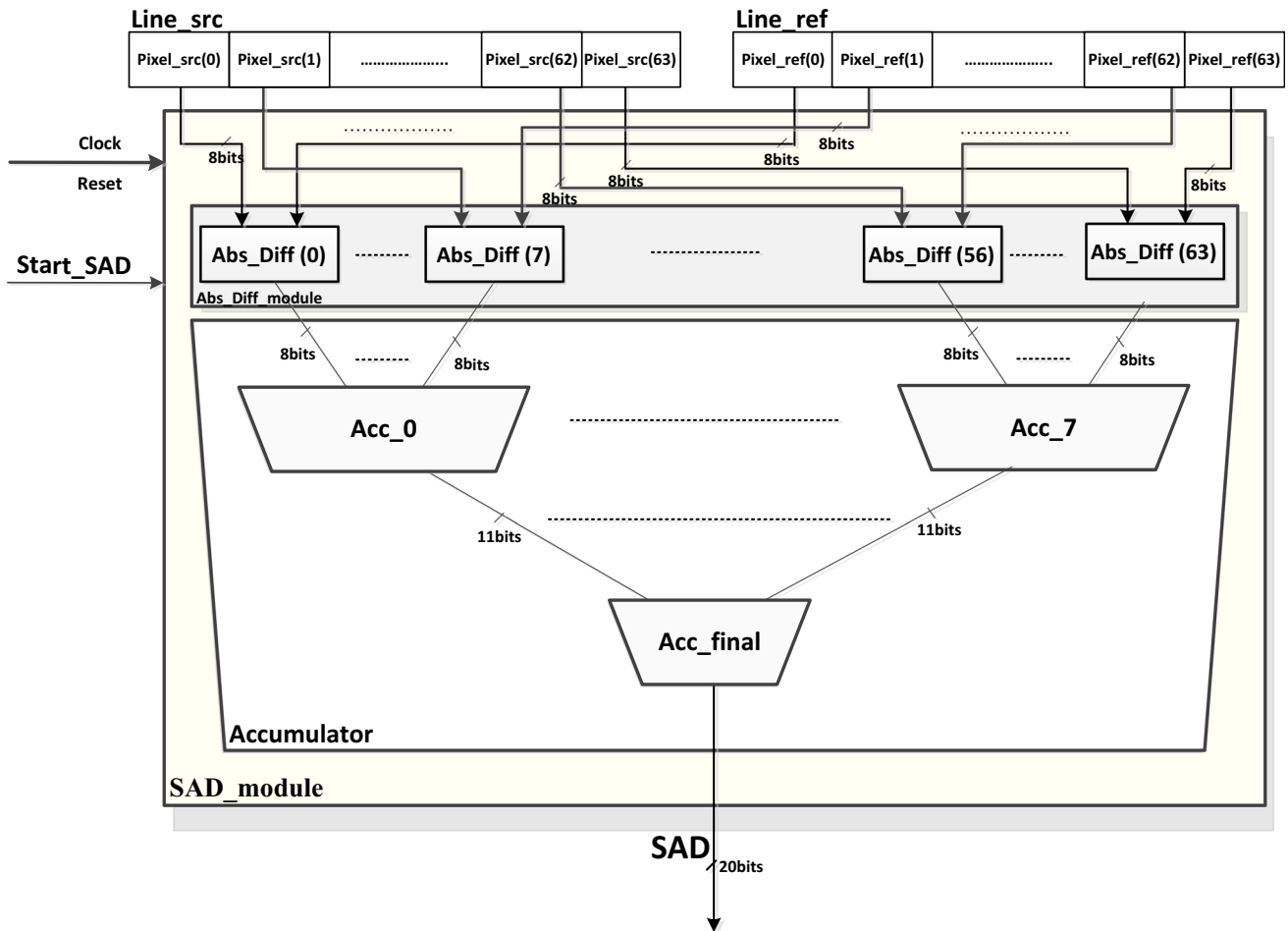


Fig. 12 SAD design

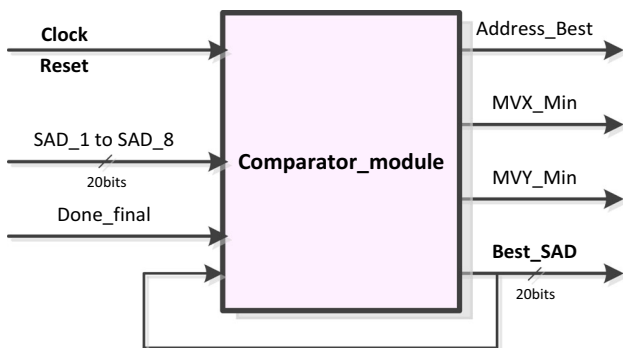


Fig. 13 Comparator\_module

two. Finally, we obtain the SAD value in three clock cycles.

**4.1.2.2 Comparator\_module** Using the comparator\_module, presented in Fig. 13, each obtained SAD value will be compared to the previous one in order to extract their minimum. The position details of the MV are the output of the control unit, which delivers the “end” signal

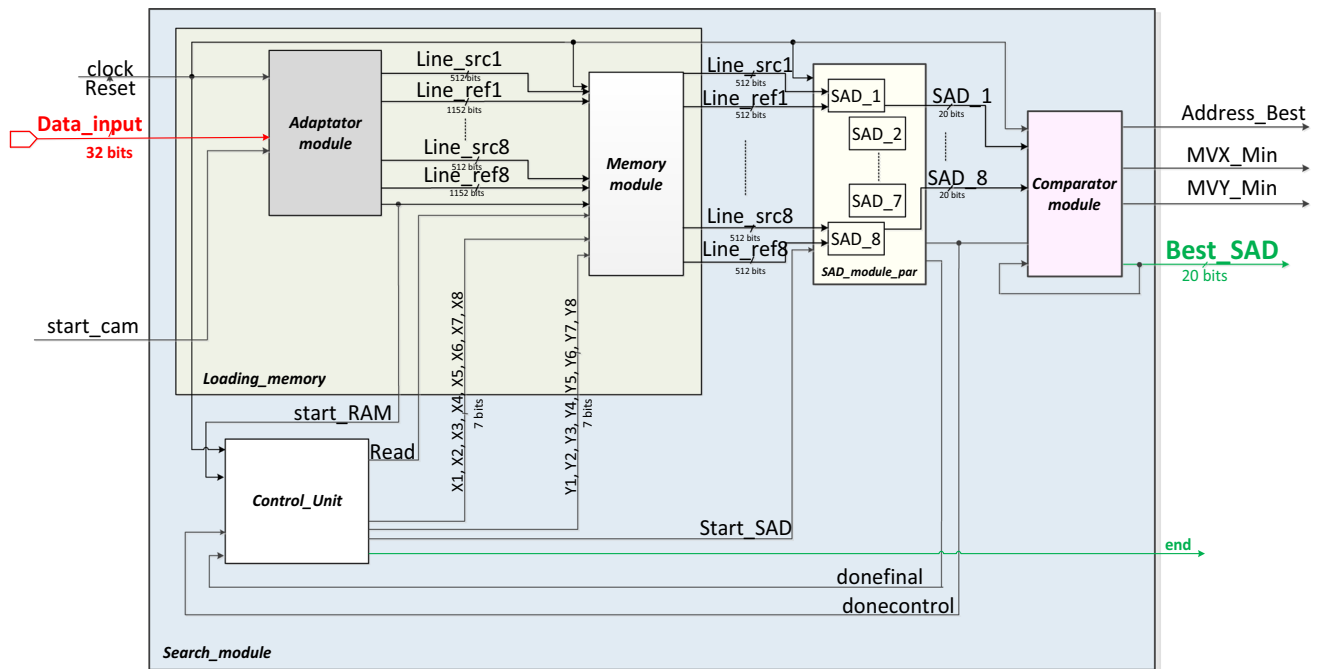
indicating the end of calculation of all the blocks of the diamond. This permits to choose the new line search for the following diamond with another distance (as described in the previous sub-section).

### 4.2 Parallel calculation architecture

The parallel method uses a parallel hardware architecture for the calculation of the MV. To obtain the best real-time performances, this parallel architecture uses some parallel Processing Elements (PE) executed in parallel.

Indeed, this architecture permits a parallel calculation of eight SAD values.

This is done by the extraction of the eight lines corresponding to each reference block, at the same time. All modules which differs from those defined previously, are described in this sub-section. The major differences between both architectures (sequential and parallel) used in this work, are the Loading\_module, the SAD\_module and the comparator\_module. Figure 14 shows the parallel architecture of our algorithm.



**Fig. 14** Parallel\_Search\_module

#### 4.2.1 Loading module

The only difference between the Loading\_module for the sequential and the parallel architectures is the number of output blocks. Instead of a reference and a source block, we have now eight-source and eight-reference search areas.

The eight reference blocks (from Block\_ref\_1 to Block\_ref\_8) are extracted in parallel. Thus, eight line pixels related to each reference block are selected at the same time. This extraction needs one clock cycle. Then, these eight lines pass to the next module (SAD\_module\_par) to compute their corresponding SAD values. In Fig. 15, an example of large diamond extraction, with a distance equal to 2, positioned in the center (0,0) in the reference area, is shown.

#### 4.2.2 SAD\_module\_par

The SAD\_module described in the previous section, is replaced by another component “SAD\_module\_Par” in parallel mode, which uses some parallel processing elements (PE) executed in parallel. This parallel module, SAD\_module\_Par, shown in Fig. 16, is made up of eight sub-blocks (PE1,...,PE8) for calculating the eight SAD values (from SAD\_1 to SAD\_8) corresponding to each block position in the diamond pattern. The computation is performed in parallel. After four clock cycles, the eight sub-blocks results are obtained and “donefinal” signal passes to ‘1’. Then, “donecontrol” signal passes in turn to

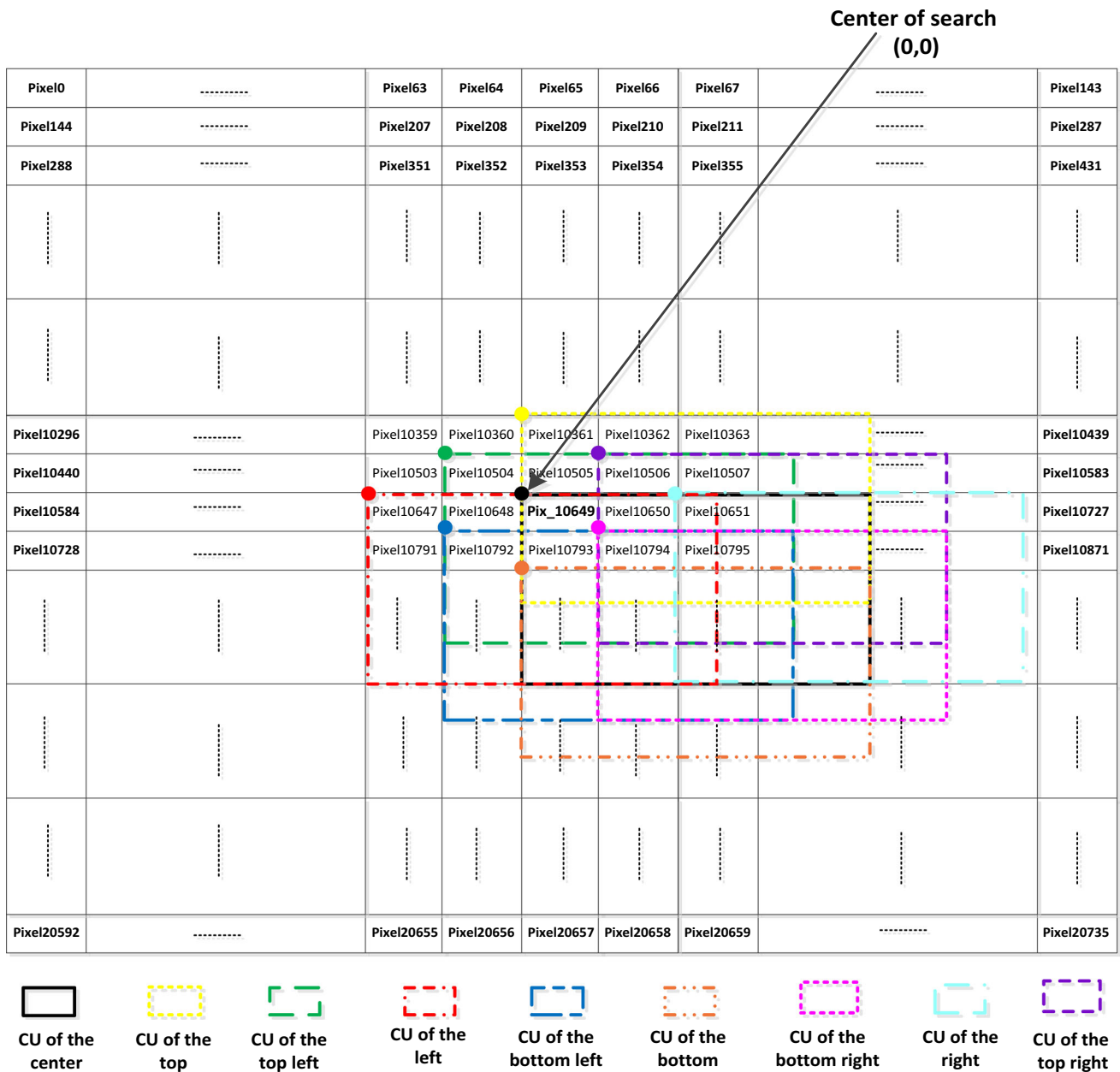
‘1’, indicating the activation of the comparator\_module to extract the best\_SAD of all the eight.

#### 4.2.3 Comparator\_module

This module permits the determination of the MV and its coordinates. This is known when extracting the best SAD with minimum value obtained from several positions of the diamond. The position information are supplied again to the control unit to select the next line search for the following diamond with another distance. Figure 17 shows the comparator module used in the parallel architecture.

## 5 Implementation and performance results

The proposed architecture is first, implemented and designed with VHDL language (VHSIC Hardware Description Language). Next, it is verified with RTL simulations using Mentor Graphics ModelSim. A VHDL test bench is then used to send pixel data to the diamond architecture and to store computation results. Afterwards, the implementation is synthesized, placed and finally routed to a Virtex-7 FPGA, using Xilinx ISE 13.1 [35]. The eight SADs and their obtained minimum are also compared to the results found by software for some reference and source blocks, to validate the results. Both architectures are suitable for  $64 \times 64$  CTU size and  $144 \times 144$  search area size.



**Fig. 15** Extraction of the suitable reference CU in the reference block for one diamond pattern with distance equal to 2

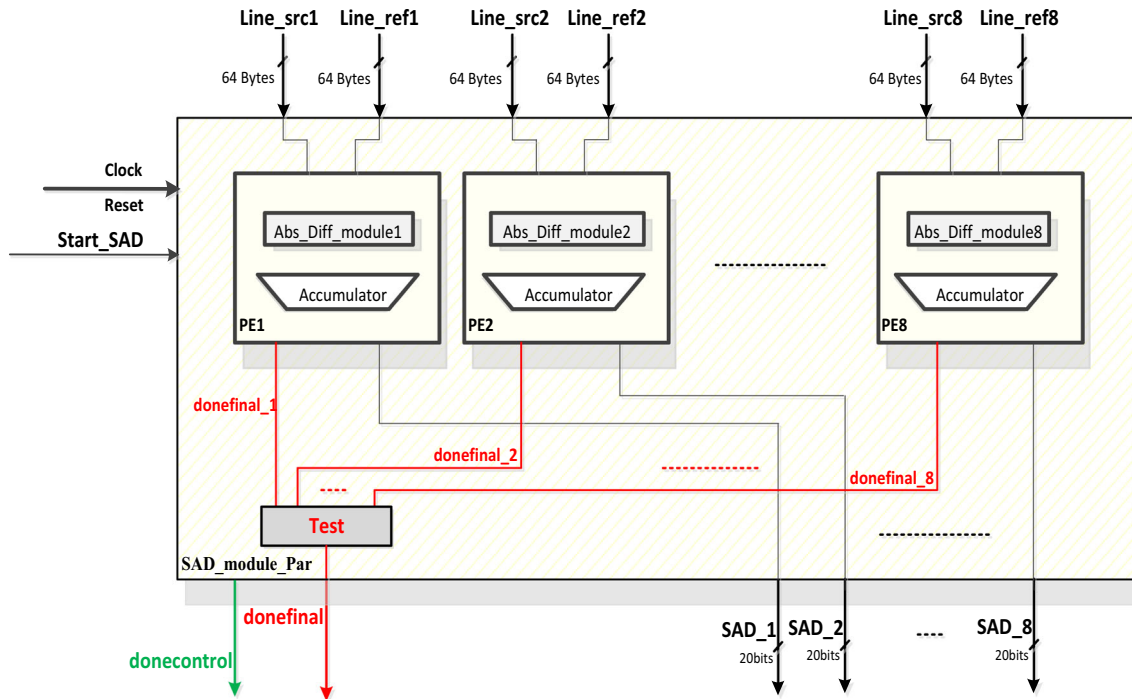
The two proposed hardware architectures do not have the same results in terms of area and execution time. The synthesis results comparison for the proposed architectures, with and without parallelism, are given in Table 1.

Comparing these two implementations on Virtex-7, the number of occupied pins for each architecture is 56, which represents 9% of the total number. In addition, these results show that the parallel architecture occupies 21 and 2% more than the sequential version, for ALTU and Register, respectively. But the maximum frequency of the parallel version is reduced by around 21 MHz compared to the sequential one.

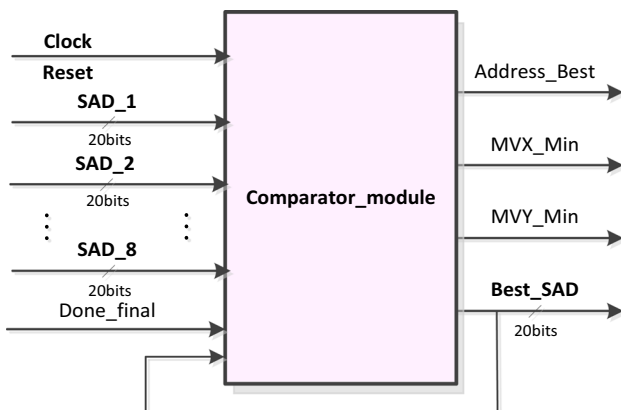
Moreover, the delay of the parallel version is 423.6 ns less than the sequential version. To process one block of  $64 \times 64$  pixels, the total delay can be calculated using Eq. (2).

$$delay_{fm} = N_{64 \times 64} \times \left( \frac{1}{Max\ freq} \right) \tag{2}$$

We note that  $N_{64 \times 64}$  represents the minimum clock cycles number required for the calculation of one  $64 \times 64$  pixels block, which is equal to 108 and 13.5 clock cycles respectively for the non-parallel and the parallel architectures, (these values are obtained in the synthesis).



**Fig. 16** Parallel SAD architecture



**Fig. 17** Comparator\_module

Table 2 compares the two proposed architectures; the parallel and the sequential one, against previous works.

The proposed sequential and parallel architectures were compared against previous works [24–29], as shown in Table 2. The area results are not ever comparable as they are implemented on different technologies. It is obvious, from Table 2, that our architectures outperform the architecture of [24–27]. In fact, for [24], when Rehman et al. proposed a hardware architecture for the SAD technique, which provides the correct position of the target within the frame/image, the maximum frequency found is lower than our frequency, by around 86.49 and 65.48 MHz, for the

sequential and the parallel architectures, respectively. Their benefit is in LTU resources, almost 6% of the total, since the design is for  $4 \times 4$  block sizes. Furthermore, comparing to the three SAD architectures suitable for motion estimation of HEVC video encoding, proposed by Nalluri et al. [25, 26] on Virtex-5 against our parallel architecture, the maximum frequency of our design is 45.062, 47.788 and 54.165 MHz less than the three SAD architectures. Additionally, it is 24.06, 26.786 and 33.163 MHz less than our sequential architecture. For the total clock cycles required for the calculation of one block of  $64 \times 64$  pixels, our two architectures are the best. Indeed, our sequential architecture requires 148 clock cycles, less than the sequential architecture proposed in [25]. As to our parallel architecture, it requires 50.5 and 3.5 clock cycles, less than that of the first and the second stage parallel architecture proposed in [25]. Yuan et al. [27] presented another architecture, based on a parallel VLSI architecture for IME. The synthesis results achieved by Virtex-6 XC6VLX-550T, using Xilinx ISE tool, give a decrease in the frequency of almost 88.733 MHz compared to our results, but their architecture is suitable only for  $32 \times 32$  block sizes. Regarding the obtained results for the  $64 \times 64$  CTU size, the authors in [28] present a parallel SAD architecture designed for FS algorithm with a search area of  $104 \times 104$  pixels. The synthesis results, obtained by Virtex-5 technology, give a frequency around 348 MHz.

**Table 1** Synthesis results of the proposed architecture on Virtex-7

Device	Xc7v2857-1ffg1157	Xc7v2857-1ffg1157	–
Design	Non-parallel DS architecture	Parallel DS architecture	Difference (for parallel architecture)
Total pins	56 (9%)	56 (9%)	0
Total LTUs (out of 178,800)	11,701 (6%)	49,258 (27%)	21% More
Total registers (out of 35,760)	4697 (1%)	13,351 (3%)	2% More
Max frequency (MHz)	219.735	198.733	21.002 MHz less
Clock cycles required for calculation of one block $64 \times 64$	108	13.5	85% Less
Total delay for one $64 \times 64$ block (ns)	491.5	67.93	423.57 Less

**Table 2** Comparative table of the proposed architectures with other designs

Design	Architecture	FPGA	LTUs	Frequency (MHz)	Clocks cycles	Delay (ns)	CTU size
Rehman et al. [24]	SAD Arch.	Virtex-2	657 (6%)	133.245	–	–	$4 \times 4$
Nalluri et al. [25]	1-Stage Paral. Arch.	Virtex-5	15,453 (22%)	171.947	64	372.2	$64 \times 64$
	2-Stage Paral. Arch.	Virtex-5	29,484 (43%)	165.57	16	96.63	$64 \times 64$
Nalluri et al. [26]	Sequential arch	Virtex-5	11,124 (16%)	174.673	256	1465.59	$64 \times 64$
	1-Stage Paral. Arch.	Virtex-5	15,453 (22%)	171.947	64	372.2	$64 \times 64$
	2-Stage Paral. Arch.	Virtex-5	29,484 (43%)	165.57	16	96.63	$64 \times 64$
Yuan et al. [27]	VLSI architecture	Virtex-6	55,346 (16%)	110	–	–	$32 \times 32$
Medhat et al. [28]	Parallel SAD Arch.	Virtex-5	25,173(36%)	348	–	–	$64 \times 64$
Alcocer et al. [29]	SAD Arch.	Virtex-5	–	159	112	704.4	$64 \times 64$
		Virtex-7	47,063(100%)	247	112	453.44	$64 \times 64$
Proposed	DS Sequential	Virtex-7	11,701 (6%)	219.735	108	491.5	$64 \times 64$
	DS Parallel	Virtex-7	49,258 (27%)	198.733	13.5	67.93	$64 \times 64$

Compared to our proposed architecture, their frequency is higher than ours, since our design uses a search area bigger than  $104 \times 104$  pixels, exactly  $144 \times 144$  pixels. In addition, our benefit is in LUTs, since for our architecture we use 6035 and 22915 slice logic, for the sequential and the parallel architectures, respectively, which is lower than the 25173-slice logic used in [28]. As Alcocer et al. [29], the Virtex-5 and 7 technology are used for synthesis and implementation of the integer motion estimator, in the case of a  $104 \times 104$  search area, where the processing is done in parallel on all sizes from  $64 \times 64$  up to  $4 \times 4$ . For surface level, we have a big advantage (27% against 100%). Also in the case of the speed level, we have a lower cycle number (13.5 cycles against 112 cycles). So, for [29], the SAD calculation of one block size of  $64 \times 64$  pixels required ( $14 \times 8=112$ ) clock cycles. In addition, our sequential delay is almost the same as that obtained by [29], but when comparing the delay of our parallel implementation, which is equal to 67.93 ns, it is lower than that in [29], which is 453.4 ns, since our frequency is lower (198.733 MHz against 247 MHz). This is may be due to

the search area used in the test (144 pixels against 104 pixels) and to our proposed calculation of the eight SADs in parallel.

On the other hand, our architecture can meet the real-time processing of the FHD @ 30 frames per second when running at 198.733 MHz.

Taking into account to the obtained results, our parallel design presents better efficient implementation of available resources on FPGA.

## 6 Conclusions

According to its high coding performance, the HEVC video standard, which replaces the H.264/AVC standard, is mostly suitable for application in high definition videos. However, this performance is accompanied by a high computational complexity due, mainly, to ME algorithm, which takes the largest part of the whole encoding time. Whence, many fast algorithms have been proposed in order to reduce computation, but, the majority, do not study how

they can be effectively implemented by hardware. Two new diamond hardware units suitable for HEVC based motion estimation engine, with sequential and parallel architectures, have been proposed in this paper to decrease the HEVC complexity in terms of execution time. These architectures are implemented on Virtex-7 FPGA Xilinx ISE. The synthesis results show that the parallel architecture requires the same time of one SAD obtained by the sequential architecture, which allows a gain of 8 times, compared to the sequential one. In addition, the Clock cycles required for the calculation of one block  $64 \times 64$  and the delay of the parallel version obtained, are better than all the results found in previous work. As a conclusion, our parallel design provides better efficient implementation of available resources on FPGA.

We think that the proposed diamond module can be adopted for variable block sizes with some modifications and can give excellent results, as the variable block sizes proposed for the H.264/AVC in [36]. In addition, we will make efforts to implement our proposed hardware design in the HM reference software.

## References

- Xiong, J., Li, H., Meng, F., Wu, Q., & Ngan, K. N. (2015). Fast HEVC inter CU decision based on latent SAD estimation. *IEEE Transactions on Multimedia*, 17(12), 2147–2159. <https://doi.org/10.1109/TMM.2015.2491018>.
- Cheung, N. M., Fan, X., Au, O., & Kung, M. C. (2010). Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2), 79–89. <https://doi.org/10.1109/MSP.2009.935416>.
- Fröjdh, P., Norkin, A., & Sjöberg, R. (2013). Next generation video compression. In *Ericsson Review, The Communications Technology Journal*, 1–8. <https://telcogroup.ru/files/materials-pdf/ericsson-hevc-h265.pdf>.
- Sullivan, G. J., Ohm, J. R., Han, W. J., & Wiegand, T. (2012). Overview of the high efficiency video coding HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1649–1668.
- Wiegand, T., Sullivan, G. J., Bjontegaard, G., & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 560–576.
- Kim, J., Jun, D. S., Jeong, S., Cho, S., Choi, J. S., Kim, J., et al. (2012). An SAD-based selective bi-prediction method for fast motion estimation in high efficiency video coding. *ETRI Journal*, 34(5), 753–758.
- Yoo, H. M., & Suh, S. J. W. (2013). Fast coding unit decision algorithm based on inter and intra prediction unit termination for HEVC. In *IEEE international conference on consumer electronics (ICCE)* (pp. 300–301).
- Tham, Y. J., Ranganath, S., Ranganath, M., & Kassim, A. A. (1998). A novel unrestricted center-biased diamond search algorithm for block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(4), 369–377.
- Samet, A., Souissi, N., Zouch, W., Ben Ayed, M. A., & Masmoudi, N. (2006). New horizontal diamond search motion estimation algorithm for H.264/AVC. In *Second symposium on communication, control and signal processing, ISCCSP, Marrakech, Morocco* (pp. 13–15).
- Cheung, C. H., & Po, L. M. (2002). A novel cross-diamond search algorithm for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12), 1168–1177.
- Werda, I., Chaouch, H., Samet, A., Ben Ayed, M. A., & Masmoudi, N. (2007). Optimal DSP-based motion estimation tools implementation for H.264/AVC baseline encoder. *International Journal of Computer Science and Network Security*, 7(5), 141–150.
- Belghith, F., Kibeya, H., Loukil, H., Ben Ayed, M. A., & Masmoudi, N. (2014). A new fast motion estimation algorithm using fast mode decision for high-efficiency video coding standard. *Journal of Real-Time Image Processing*, 11(4), 675–691.
- Kibeya, H., Belghith, F., Ben Ayed, M. A., & Masmoudi, N. (2016). Fast coding unit selection and motion estimation algorithm based on early detection of zero block quantified transform coefficients for high-efficiency video coding standard. *IET Image Processing*, 10(5), 371–380.
- Nalluri, P., Alves, L. N., & Navarro, A. (2015). Complexity reduction methods for fast motion estimation in HEVC. *Signal Processing: Image Communication*, 39, 280–292. <https://doi.org/10.1016/j.image.2015.09.015>.
- Khemiri, R., Bahri, N., Belghith, F., Sayedi, F. E., Atri, M., & Masmoudi, N. (2016). Fast motion estimation for HEVC video coding. In *IEEEIPAS'16: International image processing applications and systems conference* (pp. 1–4).
- Khemiri, R., Kibeya, H., Sayadi, F. E., Bahri, N., Atri, M., & Masmoudi, N. (2017). Optimisation of HEVC motion estimation exploiting SAD and SSD GPU-based implementation. *IET Image Processing*. <https://doi.org/10.1049/iet-ivr.2017.0474>.
- Lu, D., Sim, D. G., & Oh, S. J. (2014). Integer-pel motion estimation for HEVC on compute unified device architecture (CUDA). *IEIE Transactions on Smart Processing and Computing*, 3(6), 397–403. <https://doi.org/10.5573/IEIESPC.2014.3.6.397>.
- Lu, D., Sim, D. G., Cho, K., & Oh, S. J. (2016). Fast motion estimation for HEVC on graphics processing unit (GPU). *Journal of Real-Time Image Processing*, 12(2), 549–562. <https://doi.org/10.1007/s11554-015-0522-6>.
- Olivares, J., Hormigo, J., Villalba, J., Benavides, I., & Zapata, E. L. (2006). SAD computation based on online arithmetic for motion estimation. *Microprocessors and Microsystems*, 30, 250–258.
- Babionitakis, K., Doumenis, G. A., Georgakarakos, G., Lentaris, G., Nakos, K., Reisis, D., et al. (2008). A real-time motion estimation FPGA architecture. *Journal of Real-Time Image Processing*, 3(1), 3–20. <https://doi.org/10.1007/s11554-007-0070-9>.
- Kthiri, M., Loukil, H., Werda, I., Ben Atitallah, A., Samet, A., & Masmoudi, N. (2009). Hardware implementation of fast block matching algorithm in FPGA for H.264/AVC. In *IEEE 6th international multi-conference on systems, signals and devices* (pp. 1–4). <https://doi.org/10.1109/SSD.2009.4956714>
- Kthiri, M., Loukil, H., Ben Atitallah, A., Kadionik, P., Dallet, D., & Masmoudi, N. (2012). FPGA architecture of the LDPS motion estimation for H.264/AVC video coding. *Journal of Signal Processing Systems*, 68(2), 273–285.
- Kthiri, M., Kadionik, P., Lévi, H., Loukil, H., Ben Atitallah, A., & Masmoudi, N. (2010). An FPGA implementation of motion estimation algorithm for H.264/AVC. In *IEEE 5th international symposium on communications and mobile network (ISVC)* (pp. 1–4). <https://doi.org/10.1109/ISVC.2010.5654826>.
- Rehman, S., Young, R., Chatwin, C., & Birch, P. (2009). An FPGA based generic framework for high speed sum of absolute



difference implementation. *European Journal of Scientific Research*, 33(1), 6–29.

25. Nalluri, P., Alves, L. N., & Navarro, A. (2013). A Novel SAD architecture for variable block size motion estimation in HEVC video coding. In *IEEE international symposium on system on chip (SoC)* (pp. 1–4), Tampere.
26. Nalluri, P., Alves, L. N., & Navarro, A. (2014). High speed SAD architectures for variable block size motion estimation in HEVC video coding. In *IEEE international conference on image processing (ICIP), Paris* (pp. 1233–1237).
27. Yuan, X., Jinsong, L., Liwei, G., Zhi, Z., & Teng, R. K. F. (2013). A high performance VLSI architecture for integer-motion estimation in HEVC. In *IEEE 10th international conference ASIC (ASICON), Shenzhen* (pp. 1–4).
28. Medhat, A., Shalaby, A., Sayed, M. S., Elsabrouty, M., & Mehdi-pour, F. (2014). A highly parallel SAD architecture for motion estimation in HEVC encoder. In *IEEE Asia Pacific conference circuits system (APCCAS), Ishigaki* (pp. 280–283). <https://doi.org/10.1109/APCCAS.2014.7032774>.
29. Alcocer, E., Gutierrez, R., Lopez-Granado, O., & Malumbres, M. P. (2016). Design and implementation of an efficient hardware integer-motion estimator for an HEVC video encoder. *Journal of Real-Time Image Processing*. <https://doi.org/10.1007/s11554-016-0572-4>.
30. Richardson, I. E. (2002). Full search motion estimation. In I. E. G. Richardson (Ed.), *Video codec design* (pp. 99–101). New York: Wiley.
31. Felipe, S., Sergio, B., Mateus, G., Luciano, A., & Julio, M. (2012). Motion vectors merging: Low complexity prediction unit decision heuristic for the inter-prediction of HEVC encoders. In *IEEE international conference on multimedia and expo* (pp. 657–662).
32. Nalluri, P., Alves, L. N., & Navarro, A. (2012). Fast motion estimation algorithm for HEVC. In *IEEE international conference on consumer electronics, Berlin, Germany* (pp. 34–37). <https://doi.org/10.1109/ICCE-Berlin.2012.6336494>.
33. Bossen, F. (2013). *Common test conditions and software reference configurations*. Technical report JCTVC-L1100.
34. Jakubowski, M., & Pastuszak, G. (2013). Block-based motion estimation algorithms—A survey. *Journal of Opto-Electronics Review*, 21(1), 86–102. <https://doi.org/10.2478/s11772-013-0071-0>.
35. Xilinx, 7 Series FPGAs Data Sheet: Overview, DS180 (v2.2). (2016). [https://www.xilinx.com/support/documentation/data\\_sheets/ds180\\_7Series\\_Overview.pdf](https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf).
36. Elhamzi, W., Dubois, J., Miteran, J., & Atri, M. (2014). An efficient low-cost FPGA implementation of a configurable motion estimation for H.264 video coding. *Journal of Real-Time Image Processing*, 9(1), 19–30. <https://doi.org/10.1007/s11554-012-0274-5>.



**Randa Khemiri** received her M.S. and Ph.D. degree in Micro and Nano-electronics in 2013 and 2017 from the Faculty of Science of Monastir at the Laboratory of Electronics and Micro-Electronics (LE $\mu$ E)-FSM. Her current research interests include image and video processing, with emphasis on HEVC standard, currently focus on motion estimation for real-time video coding through FPGA and GPU (Graphic Processing Unit).



**Hassan Kibeya** received his Ph.D. degree in 2017 from the National School of Engineering (ENIS), Sfax, Tunisia, at the Laboratory of Electronics and Information Technology. His main research activities are focused on image and video signal processing, HEVC standard.



**Hassen Loukil** received electrical engineering degree from the National School of Engineering-Sfax (ENIS) in 2004. He received his M.S. and Ph.D. degrees in electronics engineering from Sfax National School of Engineering in 2005 and 2011 respectively. He is currently an assistant professor at Higher Institute of Electronic and Communication of Sfax (Tunisia). He is teaching Embedded System conception and System on Chip. He is currently researcher in the Laboratory of Electronics and Information Technology and an assistant at the University of Sfax, Tunisia. His main research activities are focused on image and video signal processing, hardware implementation and embedded systems.



**Fatma Ezahra Sayadi** received her Ph.D. degree in Micro-electronics from Faculty of Science of Monastir, Tunisia in collaboration with the LESTER Laboratory, University of South Brittany Lorient FRANCE., in 2006. She is currently a member of the Laboratory of Electronics and Micro-electronics. Her research includes image and video processing in graphics processor, motion tracking and pattern recognition, circuit and system design.



**Mohamed Atri** received his Ph.D. degree in Micro-electronics from Faculty of Science of Monastir, Tunisia, in 2001 and his Habilitation in 2011. He is currently the director of the Electronics and Micro-electronics Laboratory. His research includes circuit and system design, pattern recognition, image and video processing.



**Nouri Masmoudi** received his electrical engineering degree from the Faculty of Sciences and Techniques—Sfax, Tunisia, in 1982, the DEA degree from the National Institute of Applied Sciences—Lyon and University Claude Bernard—Lyon, France in 1984. From 1986 to 1990. He received his Ph.D. degree from the National School Engineering of Tunis (ENIT), Tunisia in 1990. He is currently a professor at the electrical engineering department—ENIS. Since 2000,

he has been a group leader ‘Circuits and Systems’ in the Laboratory of Electronics and Information Technology. Since 2003, he is responsible for the Electronic Master Program at ENIS. His research activities have been devoted to several topics: Design, Telecommunication, Embedded Systems, Information Technology, Video Coding and Image Processing.