A Novel Parallel Hardware Architecture for Inter Motion Estimation in HEVC

Canh Dinh, Toan Nguyen, Cuong Pham, Phong Nguyen,
Duc Duong, Ha Phung, Tien Pham, Thang Nguyen
School of Electronics and Telecommunications,
Hanoi University of Science and Technology, Hanoi, Vietnam.
thang.nguyenvu@hust.edu.vn

***Abstract*—High Efficiency Video Coding (HEVC) standard, generated by ITU, can provide compression ratio twice more than current H.264/ MPEG-4. To date, only a few hardware have been implementated for Integer Motion Estimation (IME) to date. In this paper, a parallel hardware architecture for IME in HEVC encoder is proposed. This design uses Rot-W-Diamond (RWD) algorithm to reduce computational load and parallelism to improve processing speed. Therefore, this design can reach 4K (4096×2160) video in real time at 60 frames per second (fps) and achieve the frequency of 125MHz.**

***Index Terms*—H.265/HEVC; H.264/MPEG-4; FPGA; Motion Estimation (ME); Inter Motion Estimation (IME).**

I. INTRODUCTION

The multimedia applications with diverse services have developed rapidly in recent years. This leads to higher requirements of video quality and resolution. However, in H.264 standard, the video coding standardization project of the ITU VCEG [1] is no longer relevant to high-resolution video processing. HEVC [1], the next generation of H.264 is currently expected to double the compression ratio to decrease the required bandwidth, while still ensuring video quality. Therefore, the transmission of high quality video like 4k (4096×2160) is completely possible.

In video coding, one of the most important widely research topic is to find methods to effectively reduce video space and temporal residual. Motion Estimation (ME), which often takes 66% to 77% of encoder complexity [2], is much known for reducing temporal redundancies by detecting the location of current Prediction Unit (PU) in reference frame and Motion Vector (MV) for each PU. Figure 1 describes the movement of a PU of current frame placed in reference frame.

ME is time-consuming although it has a major role in an encoder. The better motion estimation process, the better compression efficiency. ME process includes two stages: integer-pixel motion estimation (IME) and fractional pixel motion estimation (FME). The initial prediction is carried out by IME, which occupies 21.33% - 25.6% of total computational load of ME [3], to find the best-match integer point in reference frame, while FME is responsible for refining the identified results from the initial stage.

Recently, there have been many proposed fast search algorithms, such as Wide Diamond and Square pattern, Rot-W-Diamond [4], and parallel clustering tree search [5]. However, only a few hardware architecture designs for IME have been proposed. Xu Yuan [6] uses full search algorithm, which requires a great deal of search points and high complexity computation.

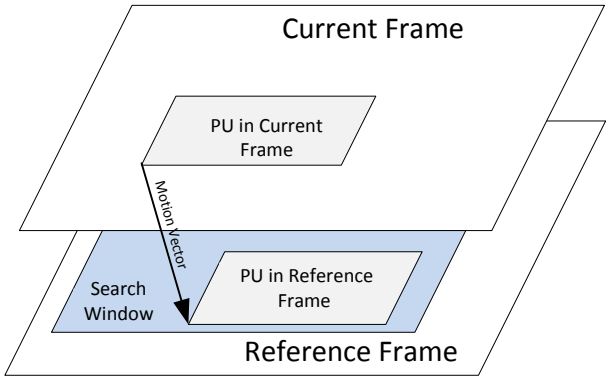

Figure 1: Motion estimation

Consequently, the design can only process medium resolution videos (Full-HD), which is incapable of making full use of H.265's video high compression. In [7], J. Byun uses full search algorithm with SRAM to increase reading memory speed. Thus, his hardware consumes large memory and has small search range (64×64). Although the architecture IME in [8] has been improved in using fast search algorithm to significantly reduce the number of search points, its hardware design fails in optimizing working speed. Therefore, the video resolution is only Full-HD. In [5], a hardware-oriented IME based on a new algorithm named "parallel clustering tree search" (PCTS) can support video applications of QFHD (3840×2160), but its frame rate achieves moderately 30 fps, much lower than 60fps in 4k standard. For that reason, the finding of appropriate algorithms to design high-speed hardware like 4k for IME is very important.

In the published algorithms, Rot-W-Diamond deems to be outstanding for its reduction of nearly 72% of computation load of ME while keeping bitrate and PSNR virtually unchanged. Moreover, Rot-W-Diamond algorithm is compatible with the implementation of hardware architecture, which is able to conduct fast search and process high-resolution videos like 4K. In this paper, we propose a novel IME architecture for HEVC video coding based on RDW algorithm. With the capability of parallel processing of all points in search range by using full PU sizes, the proposed architecture can process high-resolution videos, up to 4K at 60 fps.

The paper is organized as follows: Section II gives an overview of motion estimation, Inter motion estimation in HEVC encoders and describes RWD algorithm. The

hardware implementation is presented in Section III. Section IV shows the Synthesis results and the comparison with other designs. Finally, Conclusion will be given in Section V.

II. Overview of RWD Algorithm

The Rot-W-Diamond (RWD) has three main stages: finding starting position, fast search and second search, as shown in Figure 2.

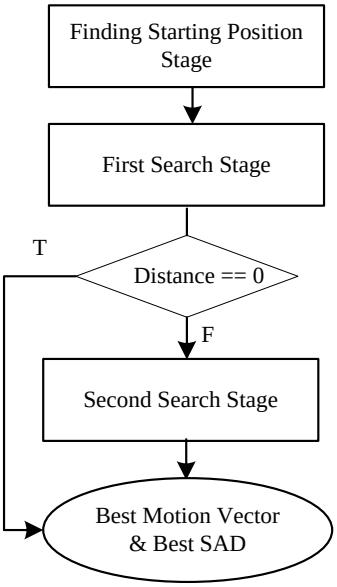

Figure 2: Brief overview of RWD searching algorithm

Step 1 – Starting Position Selection: Starting position is chosen between the ones of motion vector prediction (PMV) and zero motion vector: The position that has the smallest SAD will be chosen to be the initial starting point of First Search Stage.

Step 2 – First Search: In this stage, ME process uses diamond search pattern with a start position from the Step1 to find 40 points in Rot-W-Diamond Pattern with search range -64÷64, as shown in Figure 3. The best point with the smallest SAD will be chosen. If the distance between the best point and the starting point is 0, the search will be terminated. If the distance equals to 1 or 2, the neighboring points of this point will be checked to find out the best point of the First Search Stage by comparing SAD among these points. If the distance is larger than five, a raster search will divide search range into girds of 20x20 pixels to find out the best point.

Step 3 – Second Search: This is a refinement stage. This stage has the same procedure as the first search except that there is no raster search. It terminates when the best point and starting point are coincident.

Owing to the search point reduction and early termination conditions, the RWD algorithm with small calculation load is compatible with a hardware architecture that quickly processes and satisfies high-resolution videos.

From the approach, the block diagram of our IME unit that consists of *Integer Search Motion Estimation (ISME), Current PU RAM (CUR_RAM), Candidate Motion Vector RAM (CMV_RAM), Reference RAM (REF_RAM)* and *Sum of Absolute Differences & Comparator (SAD & Comparator)* is depicted in Figure 4.

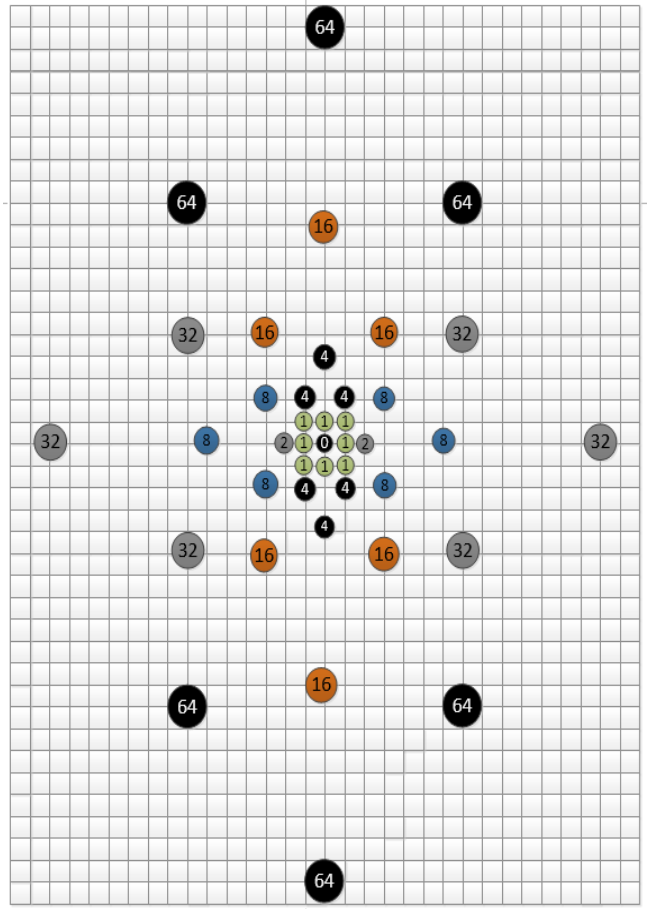

Figure 3: Rot-W-Diamond Pattern

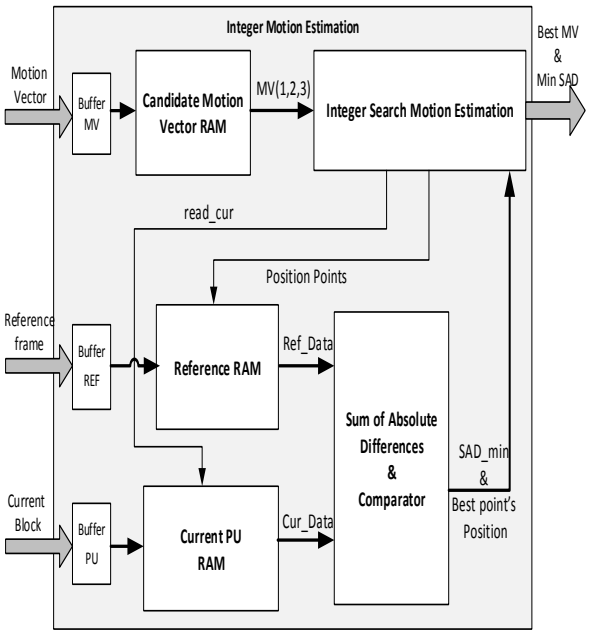

Figure 4: IME architecture

III. Hardware Architecture

As depicted in Figure 4, *IME* receives input data including the Candidate Motion Vectors, Reference Frame, and Current PU, which then stores them respectively into *CMV_RAM*, *REF_RAM* and *CUR_RAM*. At first, *CMV_RAM* sends the positions of the reference vectors to *ISME* to find the starting point for searching procedure.

Once this point is detected, *ISME* block instantly the transmit search point's position to *REF_RAM* to obtain corresponding Ref_Data (reference data). Then, the Ref_Data along with the Cur_Data (current PU) are sent to *SAD & Comparator* for the calculation of the smallest SAD. *ISME* will constantly update the best positions until the motion vector with the smallest SAD is identified. Details of each component will be described in this section.

A. Candidate Motion Vector RAM (CMV_RAM)

The CMV_RAM is created by Blocks memory that store the value of Candidate Median Vectors as predicted in Figure 5. All of MV vectors are dispatched concurrently to ISME module to calculate the starting point for the entire searching process.

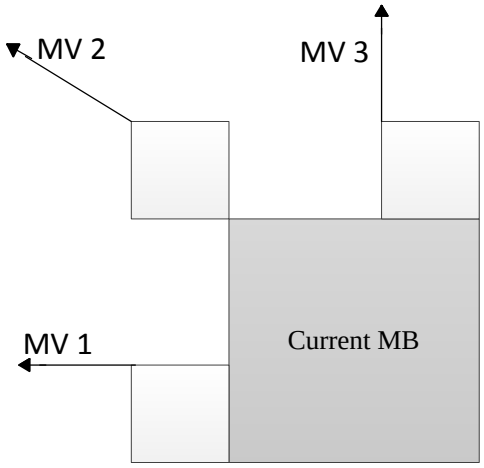

Figure 5: Positions of candidate median vectors

B. Integer Search Motion Estimation (ISME)

The ISME designed upon the RWD algorithm is the main control of the architecture is shown Figure 6. It comprises the Initial Point, Primary Search, Raster Scan and Neighbor Search.

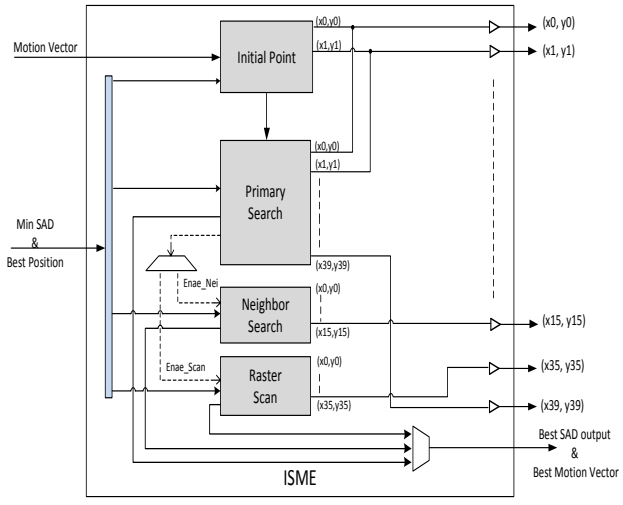

Figure 6: *ISME* architecture

In Figure 5, the *Initial point* module receives 3 vectors to calculate the median position (x1, y1), according to the Equation (1).

$$Median(x_1, y_1) = middle(MV_1, MV_2, MV_3). \quad (1)$$

This position is sent together with the position (x0, y0) of the Current PU to *REF_RAM* module to push the data to SAD module. The starting point of the *Primary Search* will select the one of smaller SADs between (x0, y0) and (x1, y1).

After determining the starting point, the *Primary Search* sends 40 positions (the tuples of (x0, y0) to (x39, y39)) to *REF_RAM*, as shown in Figure 6. Then, the data from *REF_RAM* and *CMB_RAM* are transferred to *SAD & Comparator* to select the point with the lowest SAD value.

i. If dis is 0 then the search is terminated.
ii. If the dis is larger than 5, the signal *Enae_Scan* will be set to enable the *Raster Scan*.
iii. If the dis is smaller than 5, the signal *Enae_Nei* will be set to enable the *Neighbor Search*.

The *Neighbor Search* determines 16 positions around the starting point following the algorithm and sends these positions (the tuples of (x0, y0) till (x15, y15)) to *REF_RAM*. The next process is the same as the *Primary Search.*

The *Raster Scan* defines the search window ± 64 pixels around the starting point, fragments it to 36 MBs [9] and sends these MBs' positions to RF RAM. The next process is like the *Primary Search*.

After the searching processes are completed, the ISME will find the Motion Vector and the SAD value of MB's best position. In each searching stage, all positions are sent simultaneously to *REF_RAM*, which help effectively and quickly speed up the searching process. However, the processing in *REF_RAM* is very complicated as it requires a full and accurate output.

C. Reference RAM (REF_RAM)

The main bottleneck introduced above is able to read the memory with many output ports at the same time. Therefore, instead of using Block memories or Dual Port RAM, which allow to read only one or two addresses at the same time, the new architecture for *REF_RAM* created by distributed RAM allows to read multiple addresses at the same time. It consists of 40 modules *Generate Address*, 40 modules *Reorder*, *and* a *Register Arrays module* that are shown as follows.

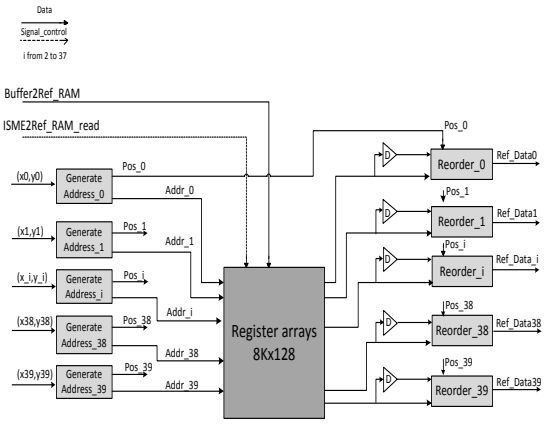

Figure 7: *Reference RAM* architecture

At the beginning of the process, the data of reference buffers are stored in 8Kx128 *Register arrays*. The *Generate Address* modules receive the positions coming from the *ISME* to calculate the addresses respectively. After that, the output data from the *Register arrays* are sent to *Reorder* modules to rearrange and obtain the correct data. Figure 8 describes the detailed design of the *Generate Address* and the *Reorder* module.

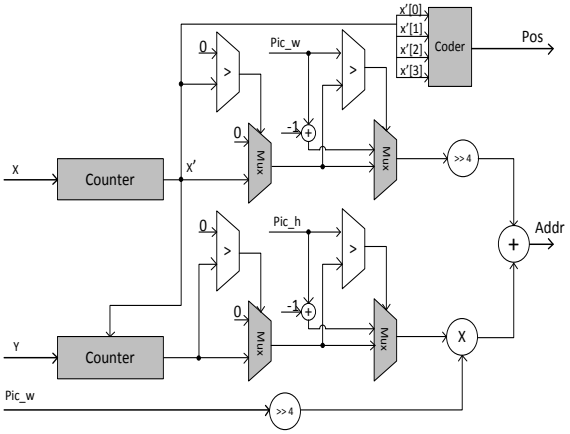

Figure 8: Generate address diagram

Outputs of the *Generate Address* module are calculated by the following assignments:

$$Addr = x\_clip >> 4 + y\_clip \times (Pic\_w >> 4). \quad (2)$$

$$Pos = x\%16. \quad (3)$$

where:

x_clip = Clip (0, Pic_w, x_offset),
y_clip = Clip (0, Pic_h, y_offset)
x_offset = x + offset_x, y_offset = y + offset_y.
offset_x, offset_y is increased after each cycle.

$$Clip(x, y, z) = \begin{cases} x; & z < x \\ y; & z > y \\ z; & \text{otherwise} \end{cases} \quad (4)$$

The data storage and address management of the *Register Arrays* are shown in Figure 9.

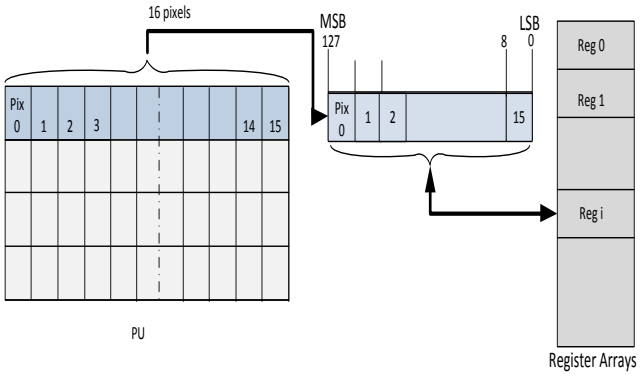

Figure 9: Storage and management of memory array

The *Register Arrays* includes many registers, each of which has 128 bits. The reading of 16 pixels per clock is more effective than one pixel, so each register will store 16 pixels of reference frame. However, the position of reference PU is at random in reference to the frame, so output data must be arranged by *Reorder* module to collect the correct data. This process is depicted in Figure 10.

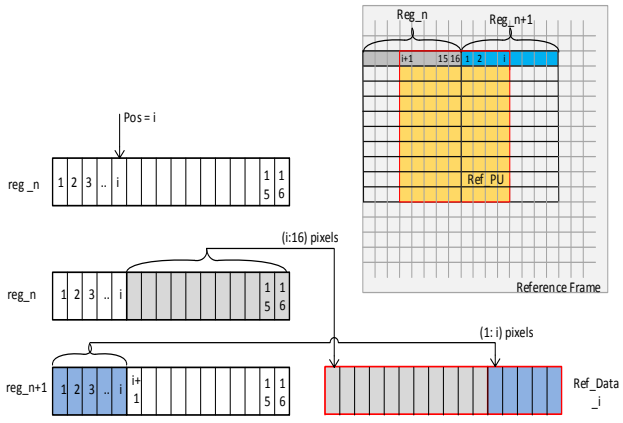

Figure 10: Reorder unit processing

There are 16 pixels in the register, so there are 16 position's pixels in the register (Pos). For instance, there is a 16×16 reference PU whose position is not at the top of register (Pos is not equal by zero), it needs two continuous registers to combine and arrange to create exact data.

By creating distributed RAM, which is capable of reading lots of input, we only take 256 cycles to read the 40 PUs of 64 × 64 size, which is 20 times faster than the conventional Dual port RAM; therefore, its ability of reading data is optimum and efficient.

D. Current RAM (CMB_RAM)

The *Current RAM* stores current PU, which adapts from 4x4 to 64x64. The way it stores data and manages address is resembling the *Reference RAM*; however, it is created by the Block memory. The configuration of this memory is shown in Figure 11.

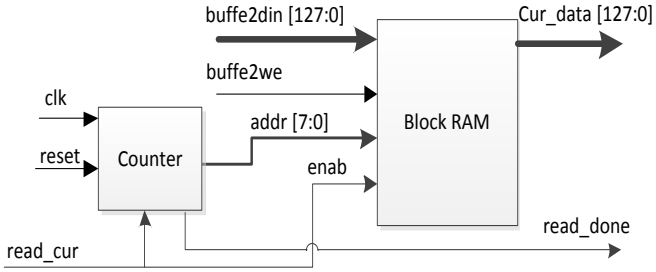

Figure 11: Current RAM diagram

To calculate the minimum residual value, all reference PUs are compared with only a current PU, so *Current RAM* does not claim to read multi addresses at the same time. Therefore, this architecture uses Block Memory that has one input, one output and rapid access.

E. Sum of Absolute Difference and Comparison (SAD and Comparator)

The *SAD & Comparator* calculates the sum of absolute differences (*SAD*) of 40 PUs to find out the smallest *SAD*. In order to parallel the process, the *SAD & Comparator* are divided into 40 *SAD_128* modules and the *Comparator* as as shown in Figure 12.

Each *SAD_128* includes 16 AD circuits processing 16

pixels (128 bit), as described in Figure 13 and Figure 14.

The absolute difference (AD) of two pixels are calculated by Equation (5).

$$|X - Y| = \begin{cases} X+Y'+1, if\ MSB=1 \\ X'+Y+1, if\ MSB=0 \end{cases} \quad (5)$$

In the Equation (5), MSB is the most significant bit, X' and Y' are the complements of X and Y, respectively [10].

The design uses parallel SAD computation blocks, in which, each adder is optimally designed by using 8-bit Full Adder Width Carry. Therefore, the calculation speed of the design is significantly improved. Thanks to the optimal design for each basic SAD to reduce hardware resource and concurrently run all SAD blocks, the calculation speed of the entire architecture has been increased significantly.

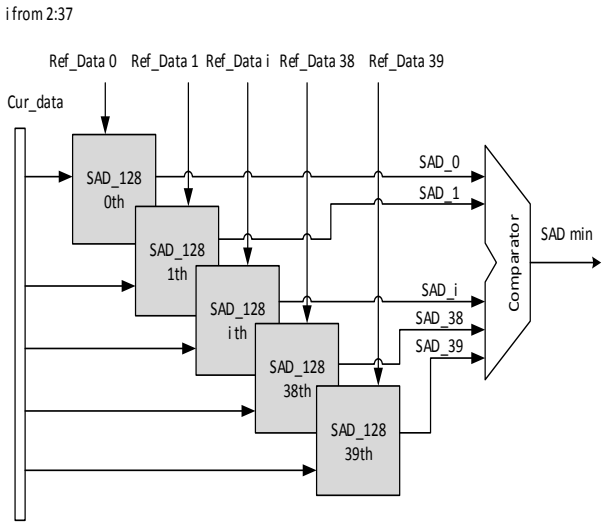

Figure 12: *SADC* architecture

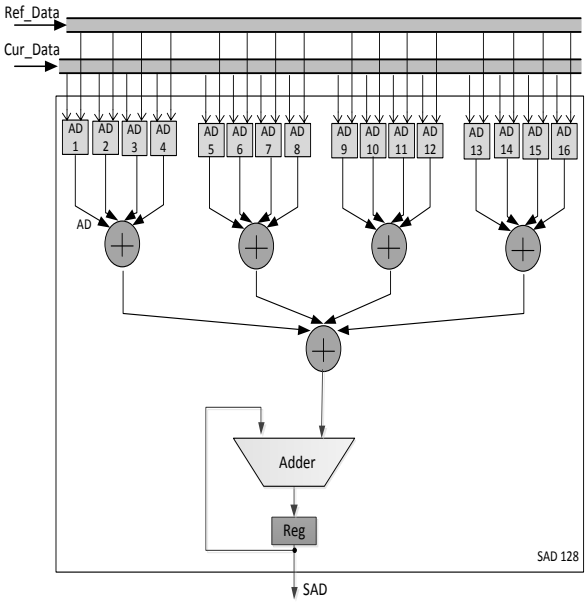

Figure 13: SAD_128 diagram

IV. Results and Comparison

The proposed architecture is implemented in Verilog HDL and synthesized into Xilinx Virtex-7 FPGA using Xilinx ISE tool. The synthesized results are given in Table 1.

The synthesized result shows that the implementation uses 294k LUTs, 14k Registers and 2 Block RAMs for one current PU. Because the SAD calculations of points lying in searching area are carried out parallel as much as possible, LUTs occupies a large number of hardware. However, the number of Registers and Block Rams is very low compared to 44k Registers and 83 Block RAMs of architecture [5].

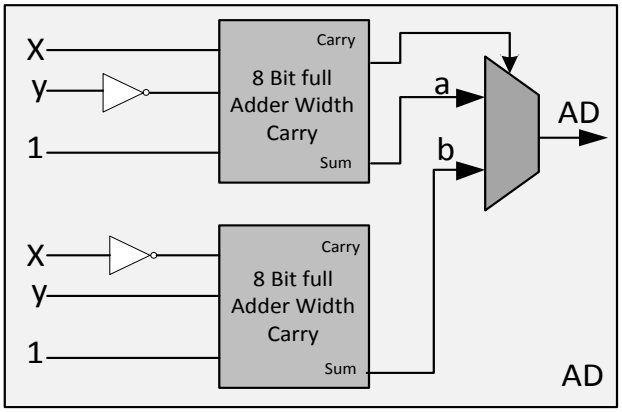

Figure 14: AD diagram

Table 1
Synthesized Results

Logic Utilization	Used	Available
Number of Slide Registers	15001	2443200
Number of LUTs	288918	1221600
Number of Block RAM	2	1292

Besides, following the RWD algorithm with three main stages, our architecture uses only 266 (5+261) cycles for step one, 261×2 cycles for step 2 and 261×2 cycles for step three to process one 64×64 PU. As a result, in total, it takes a maximum of 1310 clocks for one PU processing at system clock 100MHz. Moreover, the algorithm frequently stops at the first search with early termination conditions, and it only takes 788 (5+261+261×2) clocks for one 64×64 PU. Hence, calculating on average, it rapidly responses to the requirements of 4k@60 fps.

Based on the comparison shown in Table 2, by dint of using parallel architecture and optimization algorithms, the proposed design achieves the highest video resolution with the highest frame rate and the smallest memory compared to the others.

Table 2
Comparison between Proposed Design and Previous Works for HEVC

	[5]	[6]	[8]	Proposed
Algorithm	PCTS	Full search	DW	RWD
Resolution	3840× 2160	1920× 1080	1920× 1080	4096× 2160
Frame Rate	30	30	30	60
Supported PU size	All size	All size	64×64	All size
Search Range	128× 128	128× 128	128× 128	128× 128
Frequency (MHz)	200	110	200	125
Technology	Virtex6 40nm	XC6VLX -550T	Virtex6 40nm	Virtex7 40nm

V. Conclusion

In this paper, a relevant and efficient IME architecture given for HEVC is implemented on Virtex-7 at 100MHz

system clock. By combining with Rot-W-Diamond algorithms, the proposed parallel architecture is competent to support 4k videos at 60 fps in real time.

ACKNOWLEDGMENT

This work has been supported by project No B2013-01-59CT.

REFERENCES

[1] Sullivan, Gary J., Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. "Overview of the high efficiency video coding (HEVC) standard." *IEEE Transactions on circuits and systems for video technology* 22, no. 12, pp. 1649-1668, 2012.

[2] Zhao, Z. and Liang, P., "A statistical analysis of h. 264/avc fme mode reduction." *IEEE transactions on circuits and systems for video technology* 21, no. 1, pp. 53-61, 2011.

[3] Vanne, J., Viitanen, M., Hamalainen, T.D. and Hallapuro, A., "Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs" IEEE Transactions on Circuits and Systems for Video Technology 22, no. 12, pp. 1885-1898, 2012.

[4] Nguyen, P., Tran, H., Nguyen, H., Nguyen, X.N., Vo, C., Nguyen, B., Ngo, V.D. and Nguyen, V.T. "Asymmetric diamond search pattern for motion estimation in HEVC," *In Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on*, pp. 434-439, 2014.

[5] Ye, X., Ding, D. and Yu, L., "A hardware-oriented IME algorithm and its implementation for HEVC," *In Visual Communications and Image Processing Conference, 2014 IEEE*, pp. 205-208, 2014.

[6] Yuan, X., Jinsong, L., Liwei, G., Zhi, Z., and Teng, R. K. "A high performance VLSI architecture for integer motion estimation in HEVC". *In ASIC (ASICON), 2013 IEEE 10th International Conference on*, pp. 1-4, 2013.

[7] Byun, J., Jung, Y., and Kim, J. "Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD". *Electronics Letters*, vol. 49, no. 18, pp. 1142-1143, 2013.

[8] Vidyalekshmi, V. G., Yagain, D., and Rao, G.."Motion estimation block for HEVC encoder on FPGA". *In Recent Advances and Innovations in Engineering (ICRAIE)*, pp. 1-5, 2014.