An Optimal Mobile Hardware Design for Inter Motion Estimation in HEVC

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

***Abstract*—This paper presents a hardware design for the Integer Motion Estimation (IME) compatible with the High Efficiency Video Coding (HEVC) standard. The hardware designed was targeted to meet 1080p@30fps real-time video coding. The architectures were described in Verilog HDL and synthesis on Xilinx Virtex VI. The proposed techniques significantly reduce the area and energy consumption of the proposed hardware on FPGA.**

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

I. INTRODUCTION

Today, the explosion of mobile device and 3G/4G broadband technology has grown significantly. In 2014, the number of mobile devices exceeded the number of people on earth and keep increasing [1]. Following that trend, the demand for information exchange, especially video services has been booming. Therefore, requirements for a new video compression standard that can support a variety of video resolution and quality. HEVC is the newest video compression standard that is expected to meet demands of the new services. However, this standard required huge computational complexity, which makes HEVC consume high power and large hardware. Moreover, since these services are used in mobile devices, low power and small hardware are also required.

Motion estimation (ME) is one of the most important component of HEVC. The purpose of ME block is to find the best matching block to the current block to increase the compression performance while maintain computational complexity. ME block is comprised of Integer Motion Estimation (IME) and Fractional Motion Estimation (FME). While ME makes up 60 – 90% of computation load [2], Integer Motion Estimation (IME) process account for about 21% - 26% total computational load of ME [3]. As a result of these observations, aim to reduce the ME computational effort targeting hardware solutions and suitable resolution for mobile device are in need of being researched.

Until now, several architectures targeting at IME in HEVC have been reported, J. Byun et al. proposed architecture only for 4K-Ultra High Definition (UHD) and used full search algorithm, which lead to high complexity in implementation [4]. It is not compatible with mobile devices because the high computational complexity and high resolution consumes large power consumption and waste hardware. Vidyalekshmi et al. show the sequential FPGA implementation of diamond search motion estimation algorithm [5]. It's a fast search algorithm, but architecture is completely sequential which can be further improved. Xu Yuan et al. has proposed a VLSI Architecture for Integer Motion Estimation in HEVC [6]. However, it requires a lot of hardware resources and not compatible with mobile devices.

This paper proposed a new architecture for IME block based on the *Rotating-W-Diamond* algorithm for H.265/HEVC video coding standard [8]. The *Rot-W-Diamond* reduces the computational time of the ME by nearly 69.5% and 72% comparing to the recommend search and pattern of square pattern respectively while the bitrate and PSNR are virtually unchanged. The proposed design targets 1080p resolution at 30fps which is the most efficient for most of mobile devices. The proposed hardware is implemented in Verilog HDL and synthesized with Virtex-6 FPGA. The achievements have met 1080p@30fps at 148 Hz with low hardware resource.

The rest of the paper is organized as follows. In Section II, we describe the steps of *Rotating-W-Diamond* algorithm. Section III showcase the hardware architecture approach. The implementation results and comparisons are listed in Section IV and finally, the conclusion is presented in Section V.

II. RELATED WORK

The *Rotating-W-Diamond* algorithm for H.265/HEVC bases on the following steps in Figure 1.

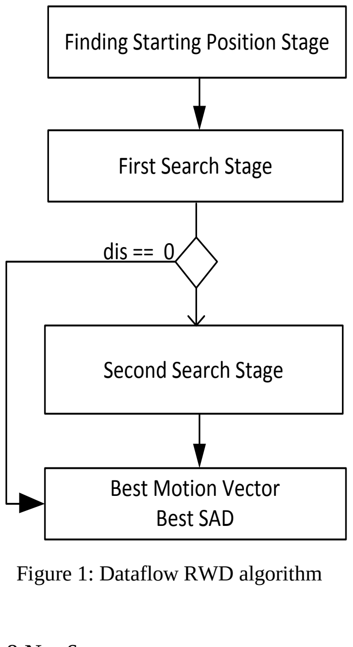

Figure 1: Dataflow RWD algorithm

In the Finding Starting Position, the best motion vector is chosen between motion vector of motion vector prediction (PMV) and zero motion vector by comparing SAD. The block has the smallest SAD will be chosen to be the initial starting block of First Search Stage.

In the First Search Stage, Rot-W-Diamond pattern is used in Figure 2.

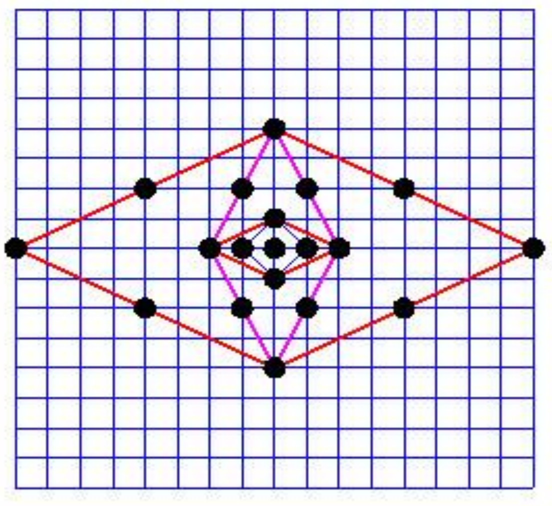

Figure 2: Rot-W-Diamond Pattern

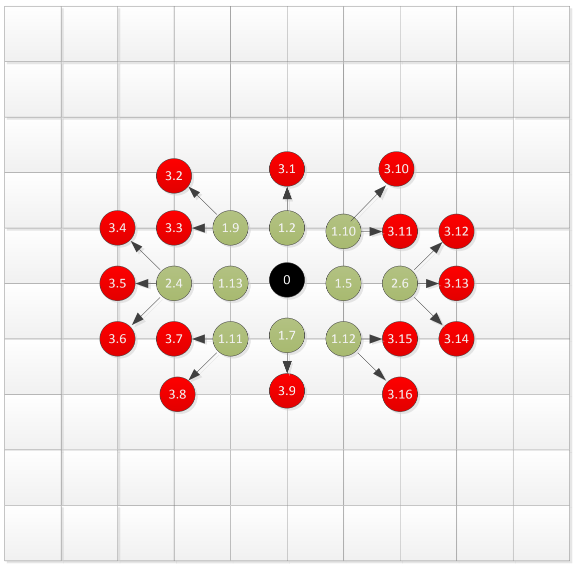

Figure 3: Neighbor Points

There are 40 points in Rot-W- Diamond Pattern. The process will compare the SAD of these points in each round of searching gradually to find the best point. If the best point has a distance (dis) of zero to the starting point, the search will be terminated. If the dis equal 1 or 2, Neighbor Points in Figure 3 of this point will be checked to find out the best point of First Search Stage by comparing SAD among these point. If the dis is larger five, a raster search will be taken with the gird of 20 to find out the best point of First Search Stage.

The last stage Second Search Stage is a refinement stage. The Second Search stage has the same behavior as the first search stage except that there is no raster search. This stage terminates when best point and the starting point are the same.

III. Hardware Architecture

In this paper, based on *Rotating-W-Diamond* algorithm, a hardware architecture, which can save hardware resource significantly, process faster and use search range up to 128x128 and support HD standard 1080p@30fps, is proposed. The block diagram of our IME unit is shown in Figure 4.

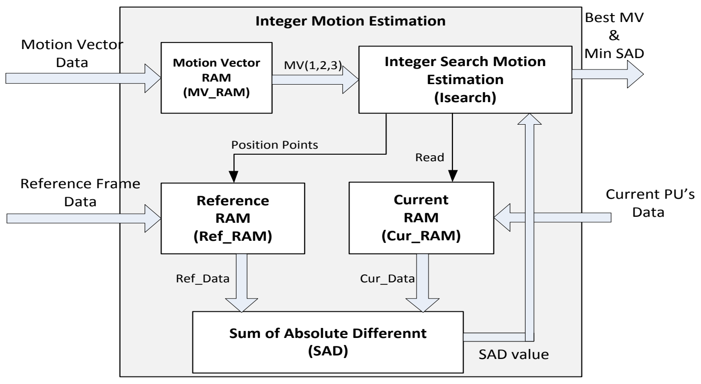

Figure 4: Top Block Diagram of IME

Reference frame, PU's of current frame and motion vector are stored in Ref_RAM Cur_RAM, MV_RAM blocks, respectively. The Ref_RAM and Cur_RAM blocks receive control signals and search points of the RWD algorithm from the Integer Search Motion Estimation (ISearch module) for generation of the reference PU address follow with the current PU address and generate data to send to SAD. *Module SAD* receives data from Ref_RAM and Cur_RAM and then caculates SAD between reference and current *PU* to create total SAD value. Then, the SAD value is sent to the Integer Search Motion Estimation – *Isearch* which implement point search belongs to *Rot-W-Diamond* algorithm. Then, *ISearch* compares SAD value to find the smallest SAD value and decides whether the smallest SAD value to be updated or not. At the end of the algorithm, the output of *IME* is smallest SAD value and its corresponding Motion Vector.

A. Integer search motion estimation (isearch)

The ROT_D_W algorithm is showed by FSM in Figure 5. State machine starts with *IDLE* state, when received searching signal, the next state is *START_POINT* will search the middle vector and send the previous point of PU to calculate SAD. After that, the state will be *WAIT_SAD*, this state waits the SAD result done and then moves to the *SAD_MV_MEDIAN*. In *SAD_MV_MEDIAN* state the median point will be sent to calculate SAD. The next state is *WAIT_SAD*, where the SAD after calculation, will be compared to choose the smallest SAD value. This point is chosen to be the starting point of the next search. The *First search* will send the pixels similarly in algorithm and update the best point. Finally, the distance among the best points will be considered to move to the *SEARCH SCAN20* or *SEARCH NEIGHBOR*. After that, it returns the 2nd search and complete searching.

Reading and processing data could be image by timing being executed by IME in Figure 6 and 7.

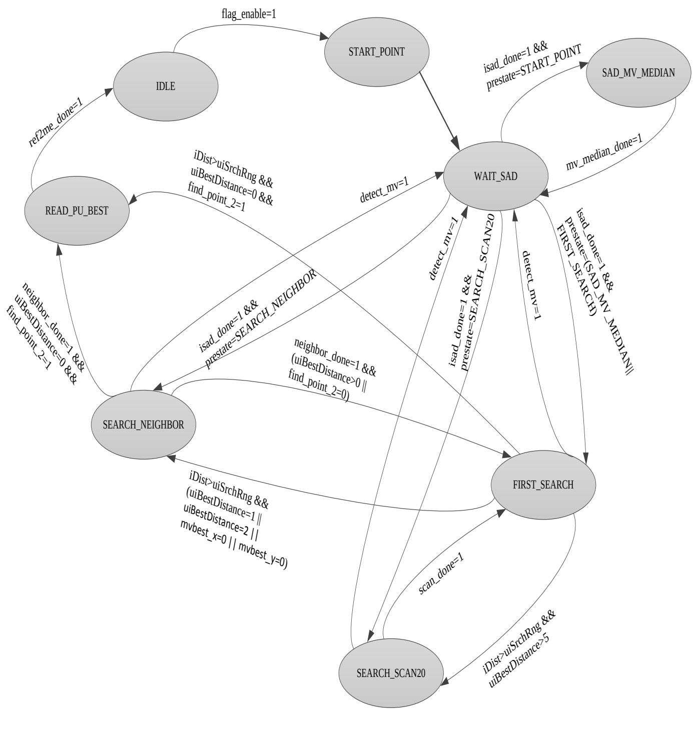
IDLE
START_POINT
SAD_MV_MEDIAN
WAIT_SAD
READ_PU_BEST
SEARCH_NEIGHBOR
FIRST_SEARCH
SEARCH_SCAN20
flag_enable=1
ref2me_done=1
isad_done=1 && prestate=START_POINT
mv_median_done=1
detect_mv=1
iDist>uiSrchRng && uiBestDistance=0 && find_point_2=1
isad_done=1 && prestate=SEARCH_NEIGHBOR
neighbor_done=1 && uiBestDistance=0 && find_point_2=1
neighbor_done=1 && (uiBestDistance>0 || find_point_2=0)
isad_done=1 && prestate=SEARCH_SCAN20
isad_done=1 && prestate=(SAD_MV_MEDIAN|| FIRST_SEARCH)
iDist>uiSrchRng && (uiBestDistance=1 || uiBestDistance=2 || mvbest_x=0 || mvbest_y=0)
scan_done=1
iDist<uiSrchRng && uiBestDistance>5

Figure 5: Finite state machine Integer Search

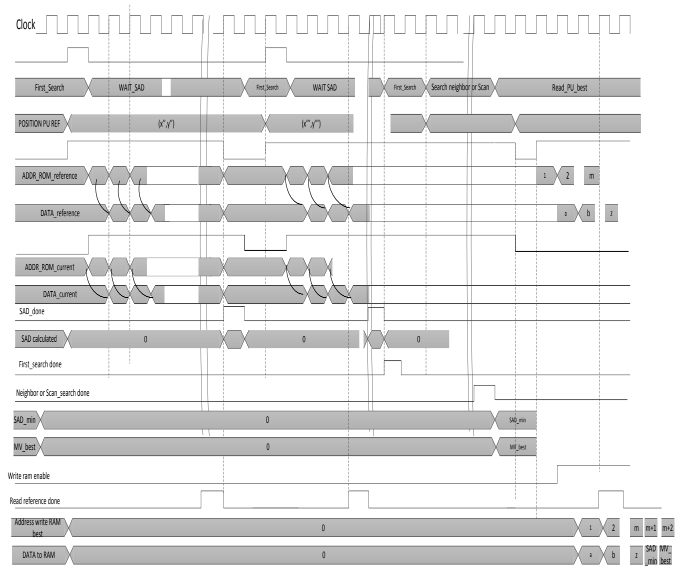

Figure 7: Reading RAM reference and RAM current in wait sad state, SAD calculating and the input data writing

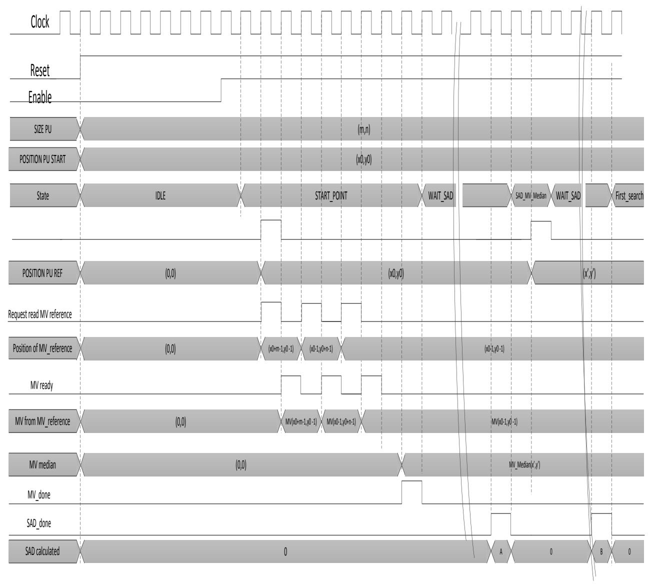

Figure 6: Reading MV Rom process in Start point state

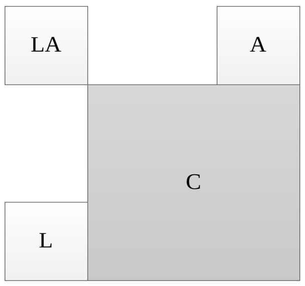

Figure 8: Median Vector

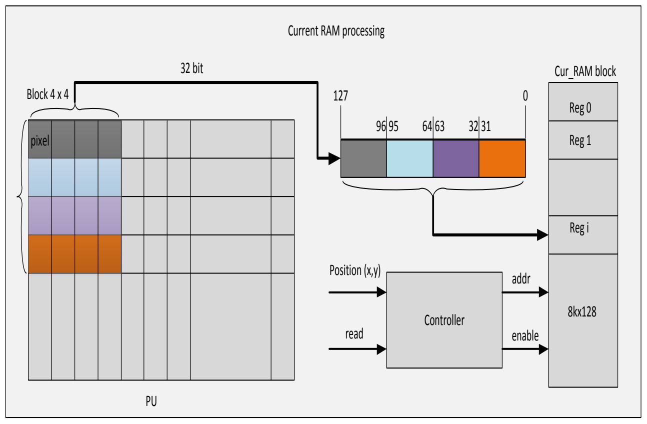

Figure 9: Current RAM

B. Motion Vector RAM

MV_RAM stores the value of candidates' motion vector (all motion vectors are previously searched) to search median vector in the *START_POINT* state. The first, median vector M = median (MV_L, MV_A, MV_{LA}) is calculated, and then compared SAD of M with SAD of C in Figure 8. Next, the smaller SAD is chosen to be the start point search.

C. Current RAM

The *Cur_RAM* stores current images with 16 pixels being saved on 128 bits register, so data will be read faster than saving one by one pixel.

As shown in Figure 9, Current PU is divided into 4x4 blocks, then each block stored on the 128 bit register. Therefore, One 4x4 pixels block need only 1 cycle to read data from *Cur_RAM*, so one 64x64 PU require 16 x 16 = 256 cycles. Data from *Cur_RAM* is sent to *SAD* module in order to calculate the SAD value. Controller receives *read signal* from ISearch and increase address after each cycle.

IV. REFERENCE RAM

The Reference RAM includes Address Generation and four Ref_Ram blocks. The Ref_Ram block stores the value of the reference picture with 4 pixels by horizontal reference frame into a 32 bit register.

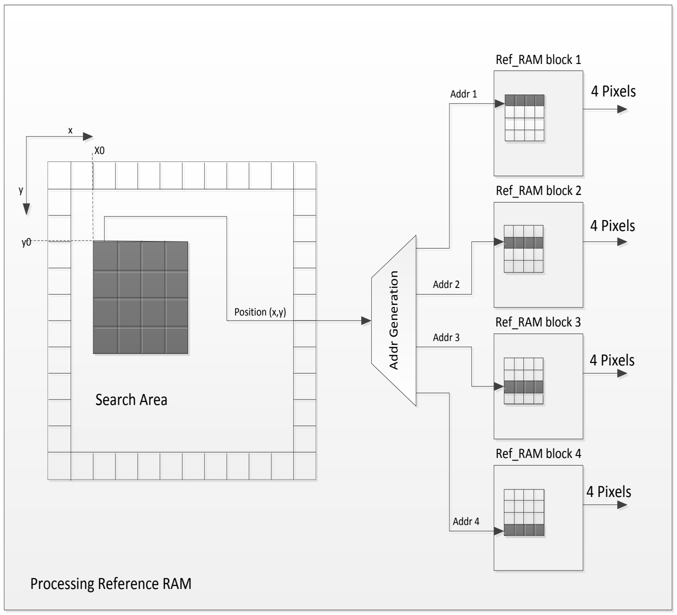

Figure 10: The Reference RAM

As shown in Figure 10, the ISearch Module sent position of search point to Reference RAM, thus the Addr Generation work out 4 addresses. And then, it is sent to 4 RAM blocks to get data and simultaneously increase address in the next cycle. So the first 2 cycles are necessary to read 4x4 pixels block from Reference RAM. After the first 4x4 pixel block is read, it needs only one cycle to read the next 4x4 pixel block. Totally, it takes 16x16 + 1 = 257 cycles for one 64x64 PU processing. In order to send data from the Cur_RAM and the Ref_RAM to SAD, The *signal read* has to be sent to the Cur_RAM before one cycle.

D. Sum of absolute difference (SAD)

SAD module receives data from Cur_RAM block and four Ref_RAM block in order to caculate sum of absolute difference between Ref_RAM and Cur_RAM. The processing unit specified in Figure 11.

The four *SAD_core*s simultaneously compute the SAD for each 16 pixels, generating in parallel 16 absolute differences (AD) and then summing all *SAD_core_out*s.

Each the SAD_core, whose architecture is shown in Figure 12, compute 4 absolute differences by processing 4 input pixels from the Ref_RAM block and the Cur_RAM block. The absolute differences calculation is shown in Equation (1).

$$|A-B| = \begin{cases} A+B'+1, if\ MSB = 1 \\ A+B'1, \quad if\ MSB\ = 1, \end{cases} \tag{1}$$

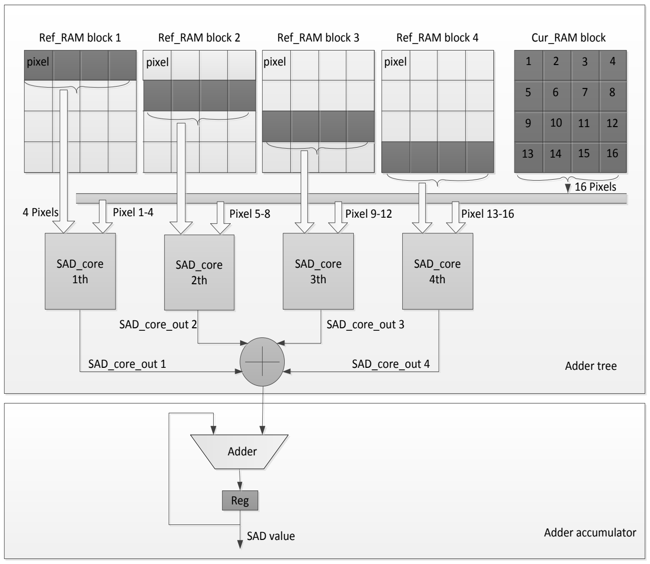

Figure 11: Processing unit

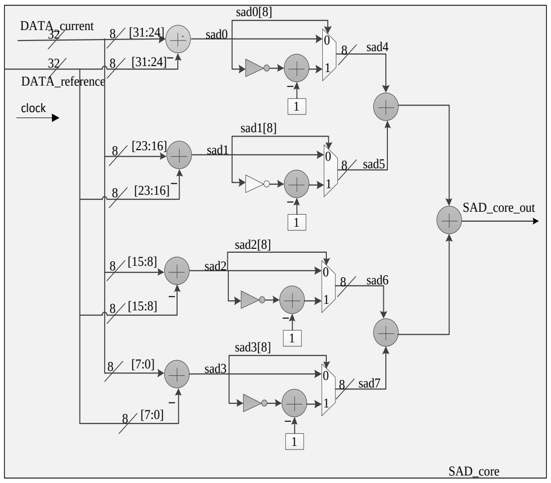

Figure 12: Architecture of the SAD_core

Where MSB is the most Significant bit, A’ and B’ are the complements of X and Y, respectively [9].

By the way of summing all absolute differences, the adder tree can calculate partial SADs. Hence, a 4x4 sub block is calculated each 4 cycles and final SAD value of a 64x64 PU is computed after 16x16= 256 cycles. After completing calculation of 1 PU, the SAD value will be sent to the ISearch to compare with other.

V. RESULT AND COMPARISON

The proposed architecture is designed, implemented and tested in Verilog HDL. It is synthesized and implemented for a Xilinx Virtex6. The comparison of the published IME hardware architectures is presented in Table 1.

Based on the above comparison Table, the proposed design achieves the smallest hardware resource at the same resolution and frame rate compared to the others by using a maximum optimization in designing architectures combined with fast algorithm.

Table 1
Comparison of hardware of IME papers

Features	[5]	[6]	[7]	Proposed
Algorithm	DS	-	DS	RWDS
Supported PU sized	-	-	16 x 16	All Sizes
Search Range	128 x 128	-	128 x 128	128 x 128
Technology	40nm	40nm	90nm	40nm
Frequency	200 MHz	110 MHz	41.3 MHz	148 MHz
Slice LUTs	-	55 K	-	12.8K
Slice Register	-	19.7 K	-	3.6 K
Resolution	1920 x 1080	1920 x 1080	1920 x 1080	1920 x 1080
Frame rate	30	30	30	30

DM: Diamond search
RWDS: Rotating wide diamond search

VI. CONCLUSION

The present paper proposed architectures implement fast search algorithm for real time HEVC integer motion estimation engines. The achieved results show that the architecture requires less hardware resource and meet 1080p@30fps real-time video coding.

Further work will focus on the optimization of this architecture to cable to process videos 4K by made several solution. We can save over picture reference and current in two array registers because the read data in register is faster than RAM. To decrease the number of clocks, we can implement parallel by calculated many point in advanced search algorithm on same time, mean fix the hardware for the points. In addition to, to design pipeline for many PU current input, the proposed architecture can separate states in old FSM to the modules which have functions same there states.

VII. REFERENCES

[1] Zachary Davies Boren (2014), There are officially more mobile devices than people in the world. http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html. Accessed in 7 Oct 2014.

[2] Z.Zhao and P. Liang (2011), “A Statistical Analysis of H.264/AVC FME Mode Reduction,” *IEEE TCSVT*, Vol. 21, No. 1, 53-61.

[3] JarnoVane et al. (2012), “Comparative Rate-Distortion Complexity Analysis of HEVC and AVC Video Codecs”, *IEEE Transactions on circuits and systems for video Technology*, vol.22, No.22.

[4] J. Byun et al. (2013), “Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD”, *Electronic Letter*, vol. 49, no. 18.

[5] Vidyalekshmi V.G et al. (2014), “Motion estimation block for HEVC encoder On FPGA,” *Recent Advances and Innovations in Engineering (ICRAIE)*

[6] Xu Yuan et al. (2013) “A High Performance VLSI Architecture for Integer Motion Estimation in HEVC” *ASIC (ASICON), IEEE 10th International Conference*.

[7] Phong Nguyen et al. (2014) "Asymmetric diamond search pattern for motion estimation in HEVC, "*Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on*, vol., no., pp.434,439.

[8] Nalluri, P et al. (2013) "A novel SAD architecture for variable block size motion estimation in HEVC video coding, " System on Chip (SoC), *2013 International Symposium on* , vol., no., pp.1,4.