The CSI Journal on
Computer Science and Engineering
Vol. 15, No. 2, 2018
Pages 12-17
Regular Paper

A Parallel Architecture for Motion Estimation in HEVC Encoder

Zaid Hanoosh Hoda Roodaki

K. N. Toosi University of Technology, Tehran, Iran

Abstract

Nowadays, the use of hardware/software codesign has grown dramatically in the design of embedded systems since it can improve the processing power, system efficiency, and the total cost of production. In this method, some parts of the system are implemented in hardware and the other parts are implemented in software in order to satisfy the system constraints, including power consumption, area and processing time. This paper proposes a parallel architecture for motion estimation in HEVC encoder. In the proposed method, the motion estimation part of the encoder, which has a high computational complexity, is implemented in hardware, and the computational complexity of this part is improved using parallel processing. The hardware implementation of motion estimation part is much less complex than the adopted HM reference software, making it more suitable for embedded systems. Experimental results show a significant improvement over software implementation.

Keywords: Motion Estimation, Parallel Processing, Computational Complexity, HEVC.

1. Introduction

In recent years, video codecs have been adopted in various embedded systems, such as digital cameras, digital TVs, and so on. Different video coding standards have been developed such as H.264 and HEVC. High Efficiency Video Coding (HEVC) standard is the most recent video coding standard that can achieve the best compression ratio compared to the previous video coding standards at the expense of high computation complexity [1].

HEVC has been designed to address all existing applications of previous standards and particularly focuses on increased video resolution (e.g. 4k×2k or 8k×4k resolution). Many techniques are adopted in HEVC, including intra-prediction, inter-prediction, motion compensation, discrete cosine transform, quantization and entropy coding to improve the coding efficiency. Depending on the coding mode of each macroblock, the predicted macroblock can be generated using inter-coding or intra-coding. Each inter-coded macroblock can be partitioned in various block sizes ranging from 64 × 64 samples to 4 × 4 samples. Intra-coding supports 33 directional modes that are encoded based on the previous decoded neighboring blocks [1].

The residual blocks of the intra or inter prediction are the difference between the original block and the predicted block. This residual blocks are then transformed and the transform coefficients are quantized, entropy coded, and transmitted to the decoder together with the prediction information [1]. Each of the mentioned processes have various computational complexity. We have arranged an experiment to measure the computational complexity of various units of HEVC codec. Table 1 shows the percentage of required time for the main units in HEVC codec relative to the total processing time as a measure of computational complexity.

As we can see in the table, the high complexity of HEVC video encoder is mainly initiated from inter-frame prediction that requires computationally intensive motion estimation process. The

next process in terms of computational complexity at encoder side is Transformation.

Table 1. The percentage of required time for the main units of HEVC codec relative to the total processing time

The processing unit	Execution Time (%)
Motion Estimation (Encoder Side)	8.09%
Transform NxN (Encoder Side)	7.24%
Deblocking filter (Decoder Side)	2.83%
Invtransform NxN (Decoder Side)	33.04%
Motion Compensation (Decoder Side)	7.1%

In this paper, we have selected motion estimation process for hardware implementation since it can benefit more from parallel processing. According to the Amdahl law [2], in a multiprocessor system, sometimes we cannot obtain a speedup value equal to the number of processors because the selected algorithm has some parts that cannot be parallelized, and have to be executed sequentially by a single processor. As we will show later, motion estimation process has more capability to parallelism compared to transformation process. So, we have selected this part for parallel implementation in our paper.

In motion estimation process, in each frame, the encoder should test all block sizes to find the optimal motion vector using the previous encoded frames. Then, the best coding mode for each block and the corresponding motion vector are sent to the decoder. Because of the computational complexity of this process, this standard do not meet the real-time requirement [3].

For real-time video transmission applications, such as mobile phones, the computational complexity should be controlled. Since the pure software implementations usually do not meet the real-time requirement, hardware implementation should be used to speed up the encoding process.

In this paper, we propose an efficient implementation for motion estimation function in HEVC. Then, some parallel process approaches will be used in hardware implemented part to speed up the execution process.

The rest of this paper is organized as follows. In Section 2, we review the hardware implementation of various units in video coding standards in the literature. In Section 3, the proposed method that is used to speed up the motion estimation process is described. The experimental results are presented in Section 4. Finally, Section 5 contains the conclusion.

2. Related Work

In this section, we review the methods that are proposed in the literature for hardware implementation of motion estimation process in HEVC video coding. In [4], a sub-pixel motion estimation process is selected for hardware implementation using Verilog HDL. In [5], a parallel motion estimation architecture is proposed for HEVC in which a sum of absolute difference values are calculated in parallel. A fast search motion estimation algorithm and its hardware implementation is proposed in [6] for integer motion estimation. In [7] a diamond search motion estimation design is introduce for HEVC video codec with focus of real-time processing HD video. As an another method, [8] suggests a hardware implementation method for motion estimation in HEVC which tries to reduce the area and bandwidth costs. Finally, in [9] the motion estimation algorithm is refined and a motion estimation process is implemented that consists of integer and fractional motion estimation.

As we can see, the proposed approaches for hardware implementation of motion estimation process in HEVC are limited and some of them tries to refine the motion estimation process in order to reduce the computational complexity. The method proposed in this paper, tries to cover some of the mentioned methods in this section using parallel processing. In other words, our proposed approach supports parallel processing for sum of absolute difference calculation in motion estimation process for variable block sizes in HEVC video coding standard. In addition, the proposed approach tries some kinds of pipeline approach to reduce the critical path in hardware implementation using some additional register in order to disrate the system clock pulse without affecting the total deadline of the system.

3. Proposed Method

As we explained, HEVC supports various block sizes for motion estimation process ranging from 16 × 16 to 4 × 4. In general, the smaller the block size, the accuracy of the motion estimation algorithm and the computational complexity is greater. Since, each block should be compared with the higher number of corresponding blocks in the previous frame. The hardware implementation of the motion estimation algorithm in this paper can take into account all block sizes specified in the HEVC standard.

First, the motion estimation algorithm for full search in HEVC video coding standard are implemented in hardware. Then, in order to improve the coding efficiency and increase the speed of the motion estimation algorithm, we decided to apply parallelization methods to the portions of the code that can run in parallel. The parallelization methods will be applied in three steps.

1) First, as we have explained before, each block of the current frame should be compared with all of the blocks of reference frame to find the most similar block. So, each pixel of the current frame should be subtracted from the corresponding pixel in reference frame. The subtraction of different pixels in a specific block can be done in parallel. Hence, the larger the block size, we have more pixels in the block and the number of parallel operations performed at this stage can be increased. Figure 1 shows the related code. Then, Table 2 shows the corresponding resources that used to synthesize the code for the smallest block sizes (2×2) on Spartan6 FPGA, model XC6SlX9.

As we can see, the logic utilization is small even for the smallest block size. The only limitation is the IO consumption. In other

```
library ieee;
use ieee.std_logic_1164.all;

package new_data_type is
 constant data_width : integer := 8 ;
 type vector_array is array (natural range <>) of
std_logic_vector(data_width-1 downto 0);
end new_data_type;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use work.new_data_type.all;

entity SUB_32in24_VP is
 GENERIC(   N: positive := 4);
 port(
        clk    : in std_logic;
        A_in   : in   vector_array(N-1 downto 0);
                        B_in    : in  vector_array(N-1 downto 0);
        C_out   : out vector_array(N-1 downto 0)
            );
 end SUB_32in24_VP;
architecture Behavioral of SUB_32in24_VP is

 signal A_INT : vector_array(N-1 downto 0);
 signal B_INT : vector_array(N-1 downto 0);
 signal C_INT : vector_array(N-1 downto 0);

 begin
  C_out <= C_INT;
    G1:  for i in 0 to N-1 generate
       begin
         process(clk)
           begin
              if  rising_edge(clk) then

                 A_INT(i) <= A_in(i);
                 B_INT(i) <= B_in(i);
                 C_INT(i) <= A_INT(i) - B_INT(i);
               end if;
            end process;
       end generate;

 end Behavioral;
```

Figure 1. Hardware implementation of motion estimation algorithm in HEVC for various block sizes

words, when the size of the block increases, IO consumption rises rapidly. So, by using another FPGA with higher number of IOs, our proposed approach can simply support various block sizes of HEVC standard.

Since we have used full search, each block of the current frame should be compared with the corresponding block in the reference frame. If the search area has a size equal to x × y, then the number of blocks that each block of current frame should be compared to is x × y. Since all the necessary subtractions for calculating SAD of each block are done in parallel and at one clock cycle, the total time needed to compare a block of current frame with all the blocks in the search area in the reference frame can be extracted from equation (1). The clock cycle at this step is 20 ns.

Table 2. The resources used to synthesize the code for the smallest block size on Spartan6 FPGA, model XC6SlX9

Slice Logic Utilization	
Number of Slice Registers	96 out of 11440 (0%)
Number of Slice LUTs	32 out of 5720 (0%)
Slice Logic Distribution	
Number of LUT Flip Flop pairs used	96
Number with an unused Flip Flop	0 out of 96 (0%)
Number with an unused LUT	64 out of 96 (66%)
Number of fully used LUT-FF pairs	32 out of 96 (33%)
IO Utilization	
Number of IOs	97 out of 102 (95%)

$$Time_{search} = number\ of\ blocks\ in\ the\ search\ range\ area \times Clk\ cycle \quad (1)$$

Then the total time for motion estimation process will be as follows:

$$Time\_Total = number\ of\ blocks\ in\ current\ frame \times Time\_search \quad (2)$$

2) At the next step, we have tried to improve the total time of the motion estimation process. As we can see in equations (1) and (2), the total time is dependent on clock cycle of the system. Therefore, if we could reduce the clock cycle of the system, the final processing time can be improved. So, we propose a pipeline approach for this step, and the circuit will be modified in order to reduce the critical path of the circuit. In this case, a lower clock cycle can be applied to the circuit without affecting the final deadline of the system. For this purpose, we propose to add some registers in different points of the critical path of the circuit to shorten it. As we know, the pipeline design can be produced by inserting registers between every iteration step. The registers should be added in a way that the new data could be processed before the prior data has finished. In our design, registers are added to the inputs of subtraction units. Therefore, the new inputs data can be logged in before the calculation of the subtraction of the previous data is completed. A part of modified schematic for this step are shown in Figure 2 for four inputs. Using this technique, the clock cycle will be decreased to 2.2ns.

3) Finally, at the third step, we tried to perform the search process to find similar blocks concurrently for a number of blocks, i.e. blocks of each row of the frame. The required resources to implement this algorithm for three concurrent blocks on Vertex6 FPGA, model 6vcx75tff484-2, are shown in Table 3.

As we can see, the only limitation is the number of IOs. Hence, using an FPGA with additional number of IOs, more concurrent processes can be performed.

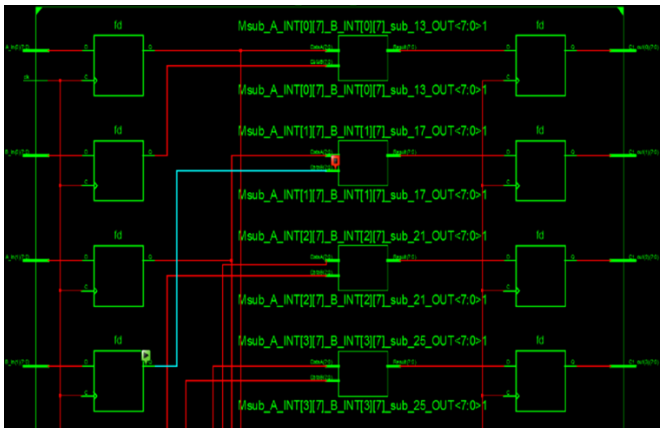

Figure 2. The pipeline schematic for motion estimation algorithm to improve execution time of the system

Table 3. The resources used to synthesize the code for the smallest block size on Vertex6 FPGA, model 6vcx75tff484-2

Slice Logic Utilization	
Number of Slice Registers	224 out of 93120 (0%)
Number of Slice LUTs	96 out of 46560 (0%)
Slice Logic Distribution	
Number of LUT Flip Flop pairs used	224
Number with an unused Flip Flop	0 out of 224 (0%)
Number with an unused LUT	128 out of 224 (57%)
Number of fully used LUT-FF pairs	96 out of 224 (42%)
IO Utilization	
Number of IOs	225 out of 240 (93%)

4. Simulation Results

In order to measure the efficiency of our proposed method, we used two video sequences Ballet and Breakdancer. Table 4 summarizes the properties of these two test sequences [10].

Table 4. Properties of the Test Sequences

Sequence	**Original Resolution**	**Frame rate (fps)**
Ballet	1024 × 768	15
Break-dancer	1024 × 768	15

Ten frames of each video are selected. The algorithm was tested in a way that the first frame was selected as the reference frame and the second frame as the current frame, coded by the reference frame, and the algorithm was executed with these two frames. Each of the current frame blocks were compared to all reference frame blocks, and the most similar block was obtained. The results of simulation for 10 blocks with the smallest sizes for Ballet and Breakdancer are shown in Table 5 and Table 6. The same algorithm is tested with the HEVC coding reference software [11].

As we explained before, in motion estimation process, each block in the current frame is compared to blocks in the reference frame and the best match for this block is found as the reference block. The Mean Square Error (MSE) between the current block and reference block, is commonly used as the distance metric since it can be implemented efficiently. To calculate MSE, the corresponding pixels of the current and reference blocks are subtracted. Then the obtained values are summed up and divided by the number of pixels. The smaller MSE between the current and reference blocks means the better the match has been found for the current block in the reference frame. Therefore, the smaller MSE means that the accuracy of the motion estimation algorithm is better. The average MSE for the Ballet and Break-dancer video sequences for 10 frames, is 0.9 and 0.5 in software and hardware implementations, respectively. So, we lose a small amount of accuracy in hardware implementation.

Table 5. The Extracted MSE for 10 blocks with the smallest sizes for Ballet

Reference Frame	**Current Frame**	**Block Number**	**MSE**
1	2	1	1
		2	1.5
		3	1.25
		4	0.5
		5	1
		6	0.75
		7	0.5
		8	0.75
		9	0.25
		10	0.5

Table 6. The Extracted MSE for 10 blocks with the smallest sizes for Breakdancer

Reference Frame	**Current Frame**	**Block Number**	**MSE**
1	2	1	2.25
		2	0.25
		3	0.5
		4	0.75
		5	0.25
		6	0.25
		7	0.5
		8	1
		9	0.5
		10	0.25

At the next step, we compare the required time to run algorithm in hardware and software implementation. The software implementation of algorithm was run on Intel (R) core (TM) i73517U CPU @ 1.90GHz 2.40GHz processor. The running time of the software algorithm for one frames is 21353.9 seconds on average. While according to the calculations above, equations (1) and (2), the execution time of the hardware is about 3405 seconds for the smallest size of block, 2 × 2, which is much smaller than

software. The total required time in hardware implementation for one frame and various block sizes are shown in Table 7.

Table 7. The total required time for motion estimation in hardware implementation for one frame and various block sizes

Block size	Total required time in hardware
4 × 4	85 s
8 × 4	42 s
16 × 16	5.3 s
64 × 32	0.6 s
64 × 64	0.3 s

Therefore, the simulation results show that in the hardware version, we have obtained a much better computational complexity by losing a small amount of accuracy. The total required time for larger block sizes is proper for real-time execution.

Similar methods have been proposed in literature for the previous standards, such as H.264, which include smaller sizes in terms of the size of the predicted blocks compared to the HEVC standard. In addition, as discussed, the HEVC standard is proposed for large-dimensional video, with much higher computational complexity of the motion estimation algorithm. We have also compared our results with the results of one of the previous works proposed in [12]. This paper proposes a highly parallel motion estimation architecture for HEVC [12]. Since various papers usually uses various sequences with different resolutions to extract their results, in order to be able to compare the results, we have used the "number of processed points per block size per second" metric as proposed in [12] to compare our results. The "number of processed point per block size per second" is estimated as the resolution of video sequence multiplied by the supported fps and the maximum search window, and can show the performance of the algorithm effectively. In [12], it is reported that for 4 × 4 and 8 × 8 block sizes and search range equal to 55 × 55, the number of processed points per block size per second is on average 188.2 G. We have tested our algorithm for our tested sequences and for search ranges 55 × 55 and 4 × 4 and 8 × 8 block sizes, and the number of processed points per block size per second are 2.36 and 176.2 G, respectively. The results show that our proposed algorithm can outperform the method proposed in [12] for high search range 55 × 55. Meanwhile, the pipeline technique that was presented in this paper can improve the computational complexity greatly.

5. Conclusion

In this paper, a method is proposed for hardware implementation of motion estimation function in HEVC video encoder. At first, the computational complexity of different coding units in HEVC video encoder was investigated and the units with higher computational complexity, i.e. motion estimation unit, were extracted. This unit was implemented in hardware to improve computational complexity using parallel processing techniques. The simulation results show that the hardware implementation with approximately the same accuracy has much lower computational complexity than software implementation.

References

[1] G. J. Sullivan, J.R. Ohm, W.J. Han, and T. Wiegand*,"*Overview of the High Efficiency Video Coding (HEVC) Standard,*" IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, Issue: 12, pp. 1649 – 1668, 2012.

[2] J. Mathew, and R. Vijayakumar, "The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend," *International Journal of Computer Science and Information Technologies*, vol. 2, no. 6, pp. 2796-2799, 2011.

[3] N. Purnachand, L. N. Alves, and A. Navarro, "Fast Motion Estimation Algorithm for HEVC," *IEEE International Conference on Consumer Electronics – Berlin*, pp. 34-37, 2012.

[4] A. Can Mert, E. Kalali, and I. Hamzaoglu, "Low Complexity HEVC Sub-Pixel Motion Estimation Technique and Its Hardware Implementation," *IEEE International Conference on Consumer Electronics – Berlin*, pp. 1-4, 2016.

[5] A. Medhat, A. Shalaby, M. S. Sayed, M. Elsabrouty, and F. Mehdipour, "A highly parallel SAD architecture for motion estimation in HEVC encoder," *IEEE Asia Pacific Conference on Circuits and Systems*, pp. 280-283, 2014.

[6] M. E. Sinangil, A. P. Chandrakasan, V. Sze, and M. Zhou, "Hardware-aware motion estimation search algorithm development for high-efficiency video coding (HEVC) standard," *IEEE International Conference on Image Processing*, pp. 1529-1532, 2012.

[7] R. Khemiri, H. Kibeya, H. Loukil, F. E. Sayadi, M. Atri, and N. Masmoudi, "Real-time motion estimation diamond search algorithm for the new high efficiency video coding on FPGA," *Analog Integrated Circuits and Signal Processing*, vol. 94, Issue 2, pp. 259-276, 2018.

[8] M. E. Sinangil, V. Sze, M. Zhou, and A. P. Chandrakasan, "Cost and Coding Efficient Motion Estimation Design Considerations for High Efficiency Video Coding (HEVC) Standard," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, Issue: 6, pp. 1017 – 1028, 2013.

[9] D.B. Lim, Y.K. Choi, H.J. Lee, and S.I. Chae, "A fast fractional motion estimation algorithm for high efficiency video coding," *International Conference on Electronics, Information, and Communications*, pp. 1-4, 2016.

[10] http://research.microsoft.com/en-us/um/people/sbkang/3dvideodownload, August 1, 2018.

[11] G. Sullivan, J. M. Boyce, Y. Chen, J.R. Ohm, C. Andrew Segall, and A. Vetro, "Standardized extensions of high efficiency video coding (HEVC)," *IEEE Journal of selected topics in Signal Processing*, vol. 7, no. 6, pp. 1001-1016, 2013.

[12] A. Medhat, A.Shalaby and M. S. Sayed, "High-Throughput Hardware Implementation for Motion Estimation in HEVC Encoder," *International Midwest Symposium on Circuits and Systems*, pp. 1-4, 2015.

Zaid Hanoosh received his M.Sc. from the Faculty of Computer Engineering of K. N. Toosi University of Technology, in 2018.

Email: zaidhanoosh@gmail.com

Hoda Roodaki is a faculty member at the Faculty of Computer Engineering of K. N. Toosi University of Technology.

Email: hroodaki@kntu.ac.ir

Paper Handling Data:

Submitted: 06.24.2018
Received in revised form: 07.15.2018
Accepted: 07.31.2018

Corresponding author: Hoda Roodaki
Affiliation of the corresponding author:
K. N. Toosi University of Technology