International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 4, No.4 (Oct-2015)

http://dx.doi.org/10.12785/ijcds/040401

A Complete Verification of a Full Search Motion Estimation Engine

Yasser Ismail[1,2]

[1]*College of Information Technology, Department of Computer Engineering, University of Bahrain, Sakhair, Bahrain.*
[2]*Electronics and Communications Engineering Department – Faculty of Engineering – Mansoura University – Mansoura – Egypt.*

Received 10 May 2015, Revised 12 July 2015, Accepted 28 August. 2015, Published 1 October 2015

Abstract: A Full Search Motion Estimation architecture design is proposed and fully elaborated and tested in this paper. The proposed Motion Estimation architecture smartly reuses the data fetched from the main memory to be used in the search area. This allows using less memory I/O bandwidth. The proposed architecture guarantees a full utilization of all resources and not to have any stall at all during the Motion Estimation process. The proposed architecture guarantees high speed by performing the Motion Estimation process in adequate number of clock cycles. Additionally, high video quality is obtained using the proposed architecture. Both of the high speed and the high video quality are achieved by using an efficient algorithm to load the search area into a local memory. The local memory efficiently loads the processing array with the required search area and achieving two data reuse levels. We concentrate on elaborating and functionally testing the whole Motion Estimation architecture using VHDL verification language and provide a proof for the high accuracy of the designed architecture. The design of the local memory is implemented using only registers and a simple counter. This simplifies the design by avoiding the use of complicated addressing to write or read into/from the local memory. The proposed architecture has a regular data flow which leads to a simple VLSI implementation. The proposed architecture is flexible and can be used for low and high definition video sequences. Due to the high speed of the proposed architecture, it can be used for many real time video applications such as video phones, video conference, and HDTV broad casting.

Keywords: H.264/AVC, H.265/HEVC, Motion Estimation, video coding, VHDL

1. INTRODUCTION

HD-DVD, video conferencing, HDTV broadcasting, video-on-demand, multimedia messaging, and ultra frequency video transmission are real time video applications that have been spread nowadays. H.264/AVC (Advanced Video Coding) and H.265/HEVC (High Efficiency Video Coding) are recent standards used for such applications [1-4]. Such standards keep very low bit-rate as well as high video quality. This is achieved by adding some complexities to the encoder design of such standards. Multiple reference frames, half-pel and quarter-pel accurate Motion Estimation, parallel processing, and variable block sizes techniques are examples for such added complexities.

Full Search Motion Estimation (FSME) is the well known algorithm used in both H.264/AVC and H.265/HEVC standards for removing the temporal redundancy of the transmitted video signal.

Consequently, the encoder of such standards can achieve a high compression in the transmitted bit-rate. FSME guarantees high video quality and high compression in the transmitted bit-rate, however, it consumes most of the video encoding time [5]. Consequently, many fast Motion Estimation algorithms were developed to tackle the problem of high complexity of the FSME process. Three Step Search (TSS) [6, 7], New Three Step Search (NTSS) [8], Four Step Search (FSS) [9], Diamond Search (DS) [10], Cross Diamond Search (CDS) [11], Successive Elimination Algorithm (SEA) [12, 13], and Adaptive Search Window Size (ASWS) [14, 15] are examples for such fast Motion Estimation algorithms.

Most of the previous fast Motion Estimation algorithms are not implemented in VLSI due to the un-regularity of data flow. Although some of them are well implemented in VLSI, the transmitted video accuracy is low [16, 17]. As a result, Full Search Motion Estimation

E-mail: yasserali1977@gmail.com

is still used for video transmission. Due to its regular data flow, FSME algorithm is well implemented in VLSI. In this paper, a Full Search Motion Estimation architecture design is presented and fully elaborated and tested. Regularity of data flow, reducing the I/O bandwidth required for video transmission, reusing data that is fetched from the main memory, and fully utilizing the resources of the proposed design are the issue in this paper. We use the VHDL verification language to verify the functionality and accuracy of all components of the proposed Motion Estimation architecture.

The paper is organized as follows. Section 2 presents the problem formulation. The proposed Motion Estimation architecture is discussed in details in section 3. The whole data flow of the Motion Estimation architecture is discussed in section 4. Section 5 discusses the simulation results. Finally conclusion and future work are drawn in section 6.

2. PROBLEM FORMULATION

H.264/AVC (Advanced Video Coding) and H.265/HEVC (High Efficiency Video Coding) are the most recent video coding standards jointly by ITU-T VCEG and ISO/IEC MPEG [1-4]. Figure 1 shows the time complexity of the encoding process of both H.264/AVC and H.265/HEVC standards, respectively. It is very clear that Motion Estimation and Compensation process (MC) is the most exhaustive part which consumes up to 53% and 84% in case of using H.264/AVC and H.265/HEVC encoders. This is due to the very high number of operations required to perform such process. Following is a brief description of the Motion Estimation process.

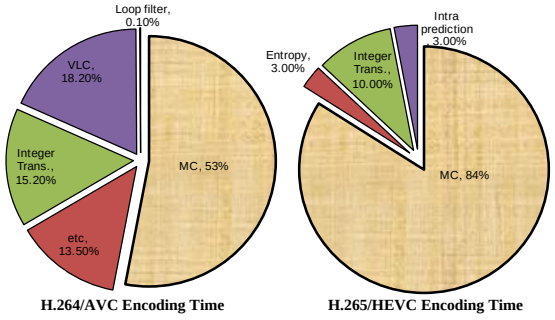

Figure 1: Time encoding complexity for H.264/AVC and H.265/HEVC encoders using one reference frame.

A. *Motion Estimation Process*

Motion Estimation (ME) is the process of finding the Motion Vector (MV) that defines the transformation of the current block image from the reference block one. Full Search Block Matching Motion Estimation (FSBM-ME) is the most popular ME algorithm [18]. In FSBM-ME algorithm, the current frame is divided into blocks, each of size N×N pixels; where N=16. Each block searches for its best match candidate block in the search area located at the reference frame. As seen in Figure 2, the best match candidate block using the FSBM-ME algorithm is calculated by searching each point in the search area represented by $2P_{max}\times 2P_{max}$.; where $2P_{max}$ is range of the selected search area. The point located at the smallest cost is selected as the best match candidate block. The cost can be measured using the Sum of Absolute Difference (SAD) metric. The displacement between the center of the search area and the best match reference block is represented by the Actual Motion Vector (AMV).

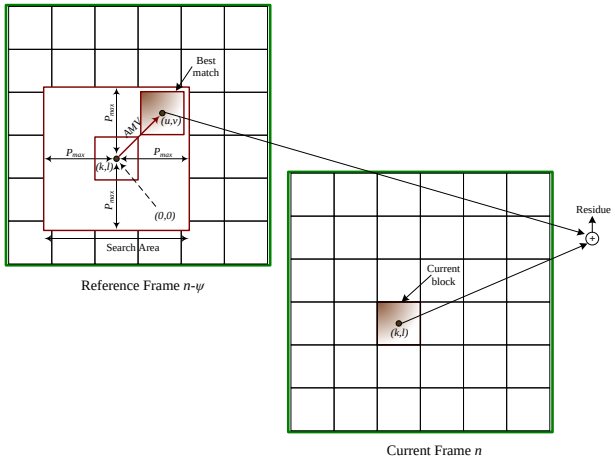

Figure 2: Full Search Block Matching Motion Estimation (FSBM-ME) algorithm.

Comparing video applications to other multimedia sources such as speech and text, it consumes much data. Table 1 illustrates some different video data formats. For SIF video sequences, an 32×32 search area is needed. While for SDTV and HDTV video sequences, an 64×64 search area is required [5, 19]. We concluded from the data in Table 1 two main important notations:

1- Higher number of search areas are needed from the memory as the frame size is increased or due to the increasing consumer demand for higher resolution [5, 20]. For example, UHDTV broadcasting requires much data to be fetched from memory than the Video Conferencing which uses SIF video format. Since the memory I/O bandwidth is limited, the proposed work is proposing and elaborating the use of architecture for better use of the available memory I/O

bandwidth. In this paper an architecture is proposed for performing FSBM-ME process. The elaborated architecture allows the data reuse of an existing data inside the ME co-processor. Consequently, no need for fetching large amount of data from the main memory.

2- The more the resolution of a video sequence is, the more the required computations to perform the FSBM-ME algorithm. These computations consume much encoding time. The proposed architecture allows parallel processing; consequently, higher speed of video transmission is obtained. Additionally, 100% utilization of the resources of the proposed architecture is achieved. Following sub-section is a brief description of the used data reuse principle.

TABLE 1: DIFFERENT FORMATS FOR VIDEO TRANSMISSION [19].

	Pixels/line	Lines/frame	Frames/sec
UHD 8k	7680	4320	30
UHD 4k	3840	2160	30
HDTV broadcast	1920	1080	30
SDTV broadcast (D1)	720	486	30
Video conferencing (SIF)	352	240	30

B. Data Reuse Principle

Compared to H.264/AVC standard [3], H.265/HEVC has accomplished up to 50% savings in the transmitted bit-rate. Consequently, 4K and Ultra High Definition TV (UHD-TV) resolutions can be achieved [19]. There are two main problems in both standards [1, 3, 5, 21, 22]. The huge number of pixels data required from the external memory is the first problem [5]. For a current block of size N×N pixels, a search area of size $2P_{max} \times (2P_{max} + N-1)$ pixels is required from the external memory. The second problem is the huge number of computations required for performing the full search Motion Estimation process. $2P_{max} \times (2P_{max} + N-1)$ absolute difference operations for a full Motion Estimation process per one current block is required. The huge number of data can be solved by using data reuse techniques [23-25]. In this work we use two different data reuse levels; i.e., Level A and Level B as follows:

Data reuse level A: In a single strip of the search area of size $2P_{max} \times 2P_{max}$, consecutive candidate blocks are overlapping in (N×N-1 pixels) within the same strip as seen in Figure 3. As a result, the overlapped area can be reused for the future candidate block #2 and only one column is needed from the external memory for such future candidate block#2.

Data reuse level B: There are overlapped pixels between two consecutive strips (i.e., strip#1 and strip#2) as seen in Figure 3. Consequently, while processing strip#2, most of the pixels used in strip#1 can be reused. It means, only one row of pixels is needed from the external memory to complete strip#2.

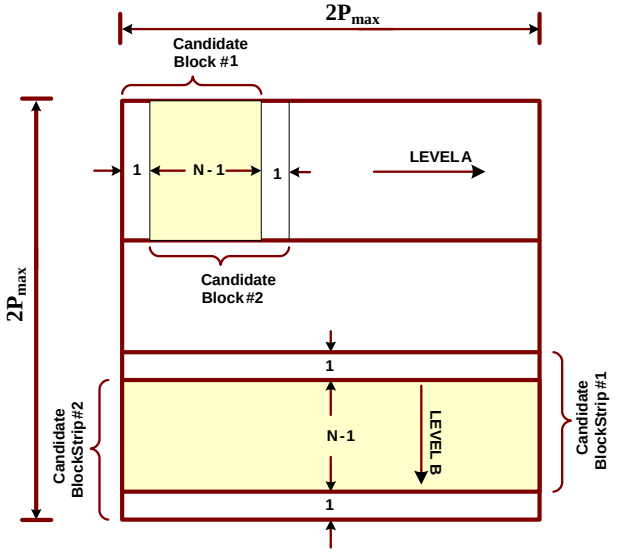

Figure 3: Data reuse levels A and B.

3. PROPOSED MOTION ESTIMATION ARCHITECTURE

The whole proposed ME architecture is shown in Figure 4. This architecture is mainly used for the H.264/AVC standard. The search area fetched from memory is $2P_{max} \times (2P_{max} + N-1)$ and the current block size is N×N. N and P_{max} are chosen to be 16. The ME operation starts when the De-multiplexer (Demux) receive the pixels of both the Current Block (CB) and the search area from the external memory. The Demux distributes the data to either the Local Memory or the PE Array. The Local Memory consists of three sub-memories. Local Memory send candidate blocks to the Processing Array which contains the data of both the current and the candidate blocks. After the absolute differences are calculated inside the PE array, they will be sent to the Adder Tree to get the Sum of Absolute Address (SAD). The SAD value is then sent to the Compare Unit to find the minimum SAD between the CB and all candidates in the search area. After the comparison, the position of the final minimum SAD is stored in the motion vector memory. The motion vector memory sends all the stored actual motion vectors to the main processor. The Control Unit controls all those activities of the components.

It is worth mentioning that this architecture is scalable one, so it can be easily used for the H.265/HEVC standard. Local memory will have same size but the PE

array will be extended to be 32×32 in order to be suitable for the ME of the H.265/HEVC standard.

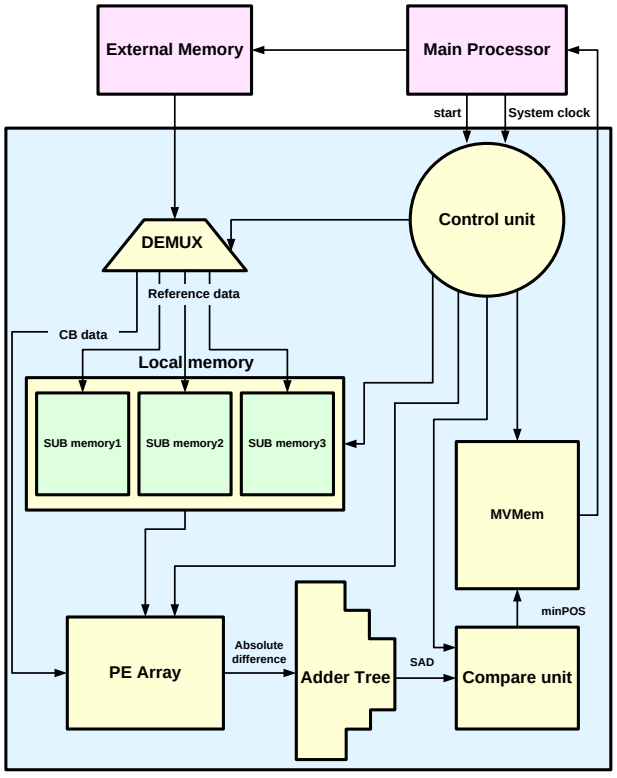

Figure 4: The proposed ME architecture.

A. *PE Array*

The Processing Element (PE) array is the factory of getting the Absolute Difference (AD) values between the current block and the candidate block in the search area. It consists of 16 PE Rows as seen in Figure 5 to form the PE array in Figure 6. The current block data pixels and the candidate block data pixels enter the 16 rows in parallel via the terminals CBR_{in} and RBR_{in}, respectively. Every clock cycle, one data pixel enters the least significant PE of each row of Figure 5. Since the pixel value ranged from 0 to 255 gray levels, the number of bits per pixel is chosen to me 8. As a result, each PE row has 128 bits for the whole ADs in one row. It is worth mentioning that the data enters the first PE and each PE sends its stored data to the next PE. There is an exception for the last PE which does not need to send any data to any next PE. All of the PEs calculates the absolute difference in parallel.

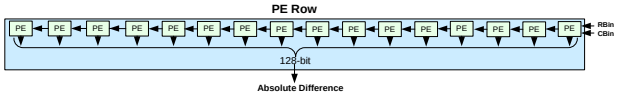

Figure 5: PE row.

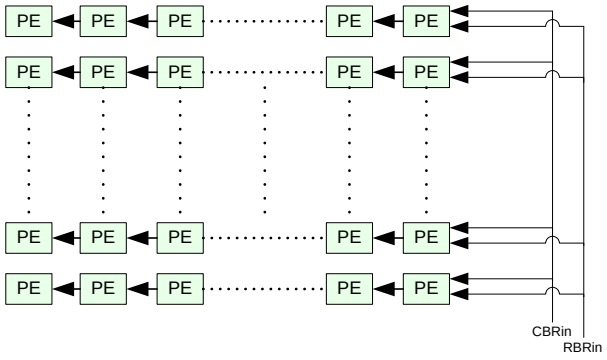

Figure 6: PE array.

B. *Adder Tree*

The output of the PE Array is 256 AD values that need to be summed in a very fast fashion. Using normal adders result in a huge delay that may prevent the proposed architecture to be used in the real time video applications. Adder tree architecture is a good choice that uses parallel processing to add many values in one clock cycle [5, 26]. The main unit in adder tree is the 4-2 compressor shown in Figure 7. It is used to add 4 bits at a time.

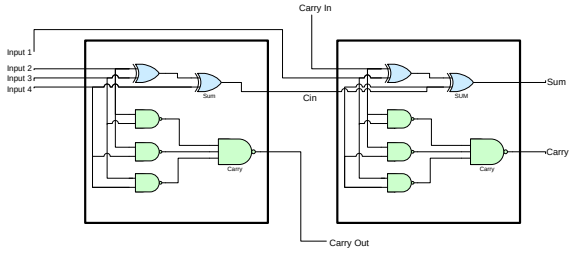

Figure 7: 4-2 compressor.

Assume that we have four Bytes:

Byte1: $a_7\, a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0$

Byte2: $b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0$

Byte3: $c_7\, c_6\, c_5\, c_4\, c_3\, c_2\, c_1\, c_0$

Byte4: $d_7\, d_6\, d_5\, d_4\, d_3\, d_2\, d_1\, d_0$

These four Bytes will enter to 4-2 compressors as seen in Figure 8. The value of carry out for the current stage i will be C_{in} for the next stage ii. The final result will be obtained by using 9-bits adder which adds the output of the adder tree in Figure 8 as follows:

Carry out	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
				+					
C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	0

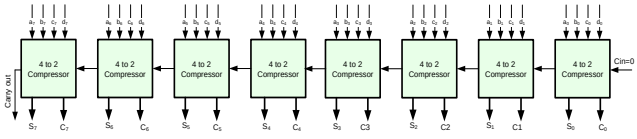

Figure 8: 4-Bytes adder tree.

C. Sum of Absolute Difference (SAD) Unit

The 256 AD values, result from the PE array, are divided into 16 4×4 groups. Each group uses the adder tree principle to add all of its AD values. All groups are working in parallel and the final result is a 16-bits SAD value as seen in Figure 9.

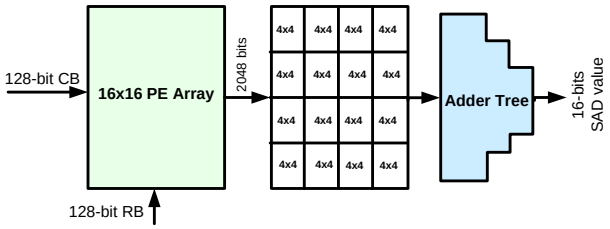

Figure 9: The SAD unit.

D. Local Memory

The main idea of data reuse principle is performed by using the Local Memory unit. It is used to save the data of the search area as well as data that may be reused in the future. Consequently, no need for fetching such reused data again from the main memory. The Local Memory unit consists of two main units: The Demultiplexer (Demux) and the sub-memory units as seen in

Figure 11.

Figure 10 shows the required search area for a 16×16 current block. The last pixels in part 2 and part 4 required additional 16×15 pixels for completing the search process. This is the reason for using the last sub-memory 3 in Figure 11. The additional pixels (dashed area in **Error! Reference source not found.**), which are required for searching the pixels in part 3 and part 4, can be fetched using the three sub-memories 1, 2, and 3, consequently. Each sub memory contains a 16×16 register array as seen in Figure 12. Each register is eight bits in length and saves a value of one pixel in the search area. The data enter as 16 pixels row by row from down to top direction. Each clock cycle one row enters from bottom and shift one row to the upper register row. Data outputs from sub-memory column by column starting from the left column and move forward to the right direction. Selecting a specific column is done by using a counter.

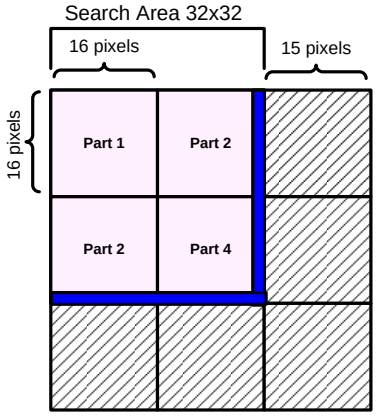

Figure 10: The search area needed for a 16×16 current block.

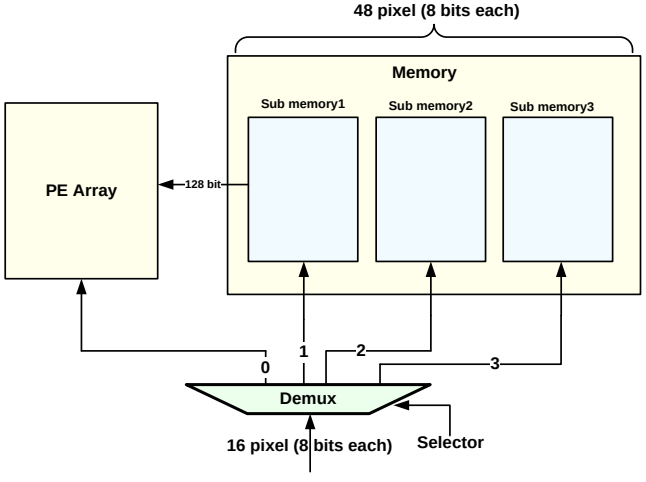

Figure 11: Local Memory.

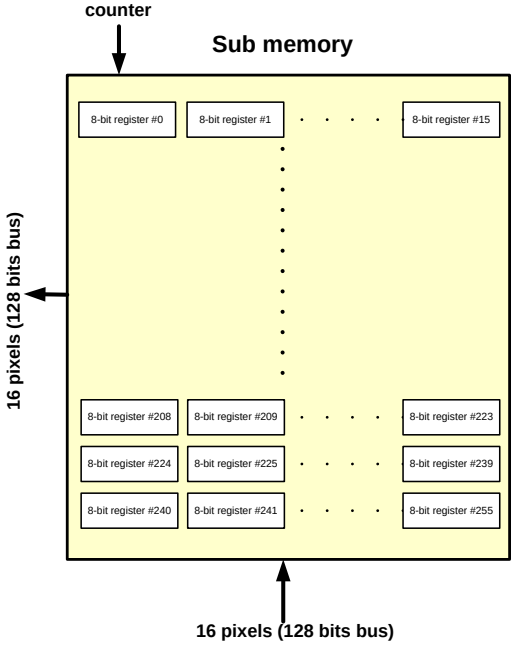

Figure 12: The sub-memory architecture.

The Demux is acting as the interface between the external memory and both the PE array and the local memory. Data is transferred from the external memory using 128 bits data bus (16 pixels wide). The PE array starts filling its registers with the Current Block data fetched from the external memory once per ME search operation when the select of the DEMUX is set to 0. The search area is filled starting by sub memories 1, 2, and 3, respectively, when the select terminal is in positions1, 2, and 3.

Back to the whole architecture of the local memory in Figure 11 and the search area in Figure 10, the whole operation will be as follows. During the first 16 clock cycles, the select terminal of the DEMUX will be 0. The PE array starts getting the values of the current block row by row as 16 pixels (128 bits) in the upward direction. In the next 16 clock cycles, the select terminal will be 1 and sub-memory 1 start to be filled in the upward direction with part 1 of the search area. In clock cycle number 33, the counter will refer to the most significant column of sub-memory 1 and select terminal will be 2. Additionally, sub-memory 2 starts to be filled with part 2 of the search area. The counter keeps increasing until clock cycle # 48. At clock cycle # 48, all part 1 of search area is moved to the PE array and group 2 is filled in sub-memory 2. PE array will give 256 Absolute Difference (AD) values at clock cycle # 49. The AD values will be added by the adder tree to get the final SAD value at clock cycle # 50. Level A data reuse is achieved by moving the counter to the first left column of part 2 of the search. Once the counter is selecting this column, it will be entered to the left column of the PE array to give another 256 AD values. The process will continue until the first strip of level A data reuse is done. It is worth mentioning that on clock cycle # 49, the select terminal will be 3 to start filling the sub-memory 3 with the dashed area of first strip level A of the search area in Figure 10. Level B data reuse [21] will be achieve by filling only one row from part 3 ad part 4 into sub-memories 1, 2, and 3, respectively. The counter will be updated to cover all points in the search area in Figure 10. It is worth mentioning that the SAD value is 16 bits length.

E. Motion Vector Memory

The output of the adder tree is a SAD value between the current block and the candidate block (SAD_current). The compare unit stores the value of the minimum SAD so far and its corresponding position. The compare unit compares the SAD_current with the minimum SAD. Id SAD_current is less than the minimum SAD, the compare unit will update its minimum SAD value with SAD_current and its new position. After all candidate blocks in the search area are processed, the final position will be sent to a motion vector memory in Figure 13.

The proposed ME architecture is flexible one. It means it can be used for doing ME process for many formats of video sequences. For example, QSIF, SIF, and SDTV video sequences [5]. For Motion Estimation, the current block should be divided into 16×16 and each current block should have an actual motion vector (position of the minimum SAD). These actual motion vectors (AMV) are stored in a motion vector memory shown in Figure 13.

The size of the SDTV video sequence is 720×486 pixels per frame. If divided into 16×16 current blocks, 1395 AMVs are needed. Motion vector memory is simply a FIFO system that contains 1395 registers. We simply used 32×32 search area in our simulation. Consequently, the input to the motion vector memory is 11-bits in length. The first position is stored in the bottom register and shifts in the upper direction every new AMV. The reset terminal (Rst) is enabled once per current frame. The enable terminal (En) is enabled at the end of each Motion Estimation process to store an AMV for a current block.

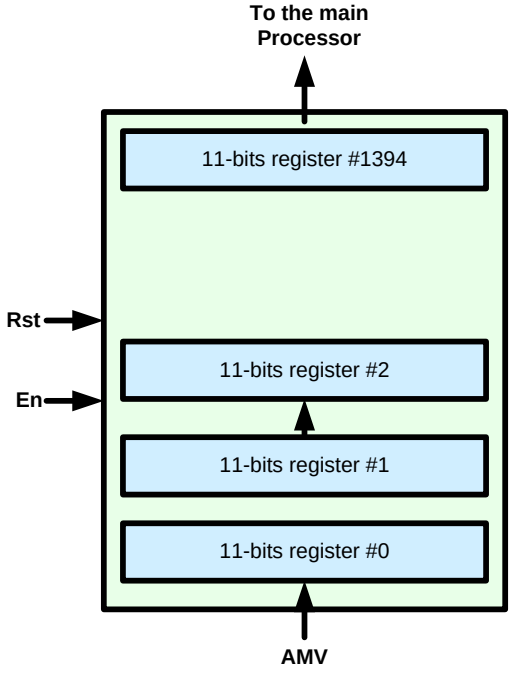

Figure 13: Motion Vector Memory.

F. Control Unit

The Control Unit is the most important and complex part of the design. It produces all the required control signals for the whole components of the ME architecture. The control unit consists of two important parts: the up counter and the control signals controller. The Control Unit has three inputs; i.e., enable, reset, and the system clock. The outputs of such unit are all the needed control signals.

The up counter is used to count the clock cycles needed for each ME process start from the top left pixel to the bottom right one in the search area. For example, for a search area of 32×32, the up counter starts from 000H to 400H. To start counting, an enable, reset and system clock

are needed as inputs, and the number of clock is the output of the up counter. The up counter value is reset with every new ME process. The output of such counter represents the position of the candidate block inside the search area. This value should be matched to the whole frame axis before storing the value of the best match candidate position in the motion vector memory.

The control signals controller takes the output of the up counter as its input. The output of such controller is the control signals that initiate all component of the whole ME architecture.

4. THE DATA FLOW OF THE ME ARCHITECTURE

The ME process starts by getting start control signal and the system clock from the main processor. PE array is filled with the current block pixel values in the first 16 clock cycles. This filling operation occurs by set the select terminal of the DEMUX to 0. The second 16 clock cycles, the sub-memory 1 will be filled by 16×16 pixels of search area (group 1) as seen in Figure 14. This will be done by set the select terminal of the DEMUX to 1. In clock cycle number 33, PE array starts read data of group 1 and sub-memory 2 also starts reading 16×16 search area group 2 by setting the select terminal to 2. At clock cycle number 48 the PE array gives 256 absolute differences to the adder tree and sub-memory 3 starts getting its 16×16 search area pixels by setting the select terminal to 3. In clock cycle number 49 the adder tree will give the SAD value to the compare unit and the PE array gets the first column of group 2 in Figure 14 which achieve data reuse level A. In clock cycle number 50, the compare unit is done by its update operation. It is worth mentioning that sub-memory 3 finishes filling its pixel values at clock cycle number 64. It means each sub-memory requires 16 clock cycle to be filled. After filling sub-memory 3, in clock cycle number 65, only 16 pixels (group 4) will fill the bottom row of sub-memory 1. All values in sub-memory 1 will be shifted upward to achieve level B data reuse. After filling the contents of group 3 into the PE array, new candidate value of group 1 starts to enter the PE array. Groups 5 and 6 will be filled in clock cycles 66 and 67, respectively. Operations will be repeated by entering the remaining search area values in the sub-memories and read them accordingly into the PE array. It is worth mentioning that sub-memories 1 and 2 require 16 clock cycles to read data column by column from each one. Sub-memory 3 only fills 15 columns into the PE array.

It is clear from previous discussion that data enters the sub-memories row by row to achieve level B data reuse. Level A data reuse is achieved by switching the read operations between sub-memories. Thus, 16 clock cycles are needed to read from sub-memory 1 while writing sub-memory 2 row by row. And 16 clock cycles to read from sub-memory 2 while writing sub-memory 3 row by row. Finally, 15 clock cycles are needed read from sub-memory 3 while writing the bottom row of sub-memory 1 and sub-memory 2 (level B data reuse). That is a total of 47 clock cycles. Those 47 clock cycles are repeated, with resetting the counter of the local memory for each row, for 32 rows of the search area. Adding 16 clock cycles for loading the current block into the PE array, another 16 clock cycles for loading the first candidate block inside the PE array, two clock cycles for getting the first SAD, one clock cycle for the compare unit, and one clock cycle for resetting all registers at the beginning of the ME process, the total setup clock cycles are 36. The number of clock cycles required for the whole ME operation are (32 row of search area) × (47 clock cycles for reading one slice using sub memories 1, 2, and 3) + 36 (setup clock cycles) which are 1540 clock cycles.

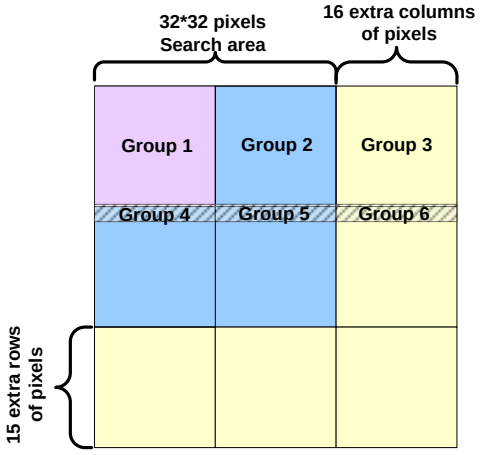

Figure 14: Data flow in the ME architecture.

5. SIMULATION RESULTS

The frames are divided into blocks of size 16×16. The used search area has a size of 32×32 pixels. The proposed Motion Estimation architecture is tested using VHDL verification language and full functional verification was performed using Modelsim tool.

In our simulations, we simulate each part of the whole ME architecture in Figure 4. Figure 15 and Figure 16show the test benches for the current block (CB) and the candidate or reference block (RB) used to test the PE array. If we fill both of the CB and the RB at same time to the PE array, it takes 16 clock cycles to be loaded inside the registers of the PE array. Since for each PE, there two registers, two more clock cycles are needed. The expected results of the absolute difference shown in Figure 17can be obtained after clock cycle number 18. The simulation result in Figure 20 confirms the absolute difference values in Figure 17 of the PE array at clock cycle number 18.

$$CB = \begin{bmatrix}
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
07&03&04&03&03&02&03&08&05&03&0A&03&07&0C&0F\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00
\end{bmatrix}$$

Figure 15: The current block (CB).

$$RB = \begin{bmatrix}
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
01&01&05&07&05&06&05&05&05&04&05&01&0D&03&05&01\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00
\end{bmatrix}$$

Figure 16: The reference block (RB).

$$|CB - RB| = \begin{bmatrix}
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
06&02&01&04&02&04&02&03&05&01&02&09&0A&04&06&0E\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00\\
00&00&00&00&00&00&00&00&00&00&00&00&00&00&00&00
\end{bmatrix}$$

Figure 17: The expected absolute difference values.

The second part that we checked is to join the PE array with the adder tree to calculate the SAD value. The used CB and RB in this case are more complex and are shown in Figure 18 and Figure 19, respectively. The expected SAD value is 200H after 18 clock cycles. Figure 21 confirms such results.

$$CB = \begin{bmatrix}
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03\\
03&03&03&03&03&03&03&03&03&03&03&03&03&03&03&03
\end{bmatrix}$$

Figure 18: The CB used in SAD calculation.

$$RB = \begin{bmatrix}
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01\\
01&01&01&01&01&01&01&01&01&01&01&01&01&01&01&01
\end{bmatrix}$$

Figure 19: The RB used in SAD calculation.

The whole architecture is checked using a search area start at (R0,C0) and has a search area size of 32×32. The whole Motion Estimation process consumes 1540 clock cycles (16 clks for loading CB into PE array, 16 clks for loading sub-memory 1, 47 clocks for loading each strip into the PE array, 2 clks for getting the SAD, 1 clk for reset the whole registers, and 1 clk for the compare unit to store into the MV memory. It is worth mentioning that we have 32 strips of search area that should be loaded into the PE array. As seen in Figure 22, the best match is the block in red that is located in the position (R30, C27). Using the calculation above, we expect that the Actual Motion Vector (AMV) of the best match candidate block is to be located at clock cycle # 1487. Figure 23 shows the simulation results of performing the ME process and finding the best match at clock cycle # 1478. This is considered as the AMV which is stored in the MV memory after the whole ME process is done at clock cycle # 1540.

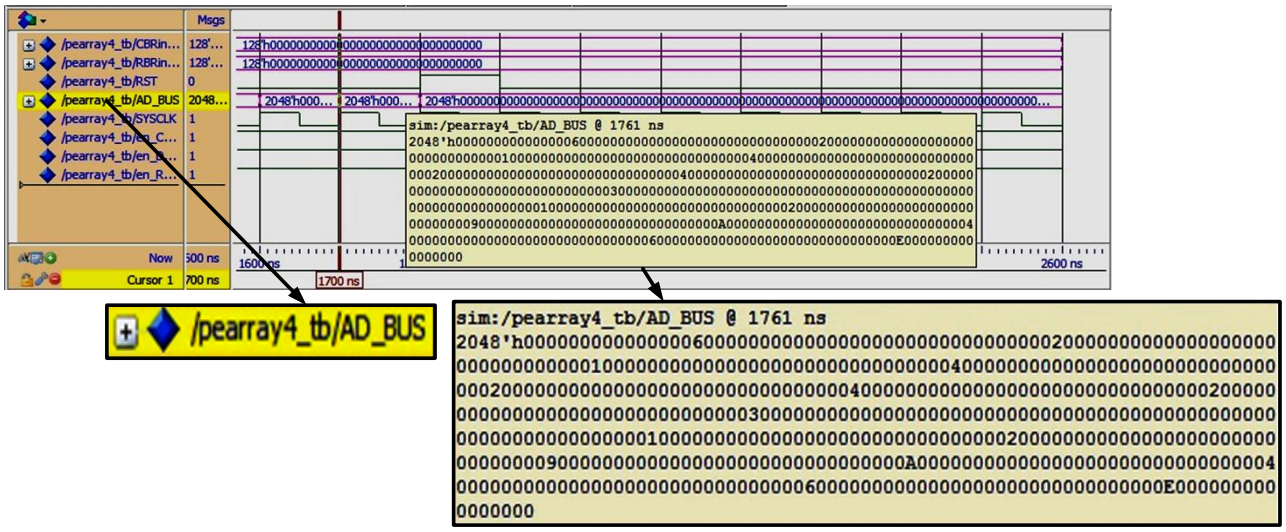

Figure 20: Simulation result of the PE array.

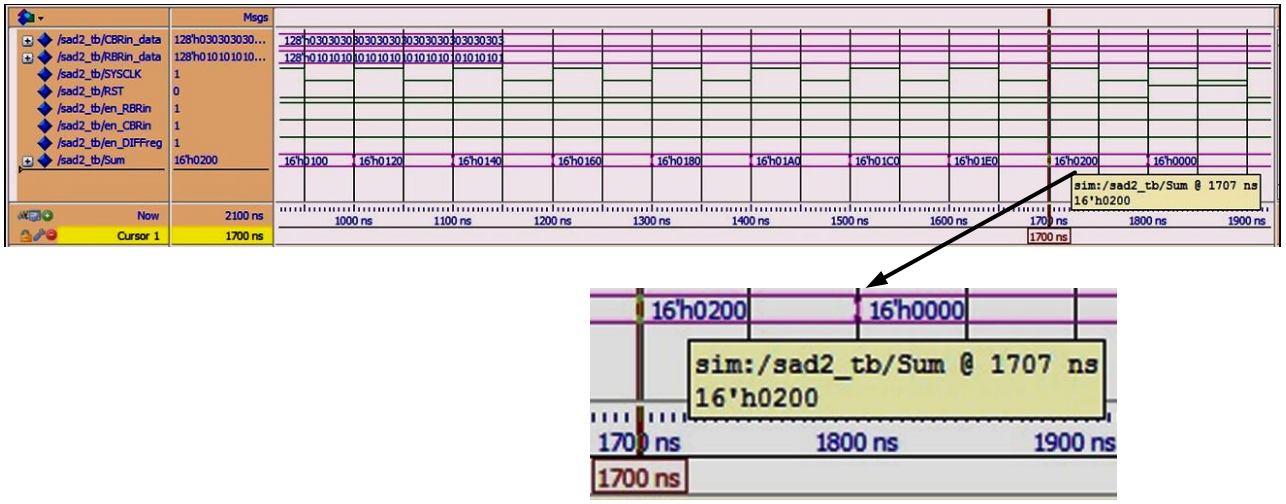

Figure 21: Simulation result of SAD unit.

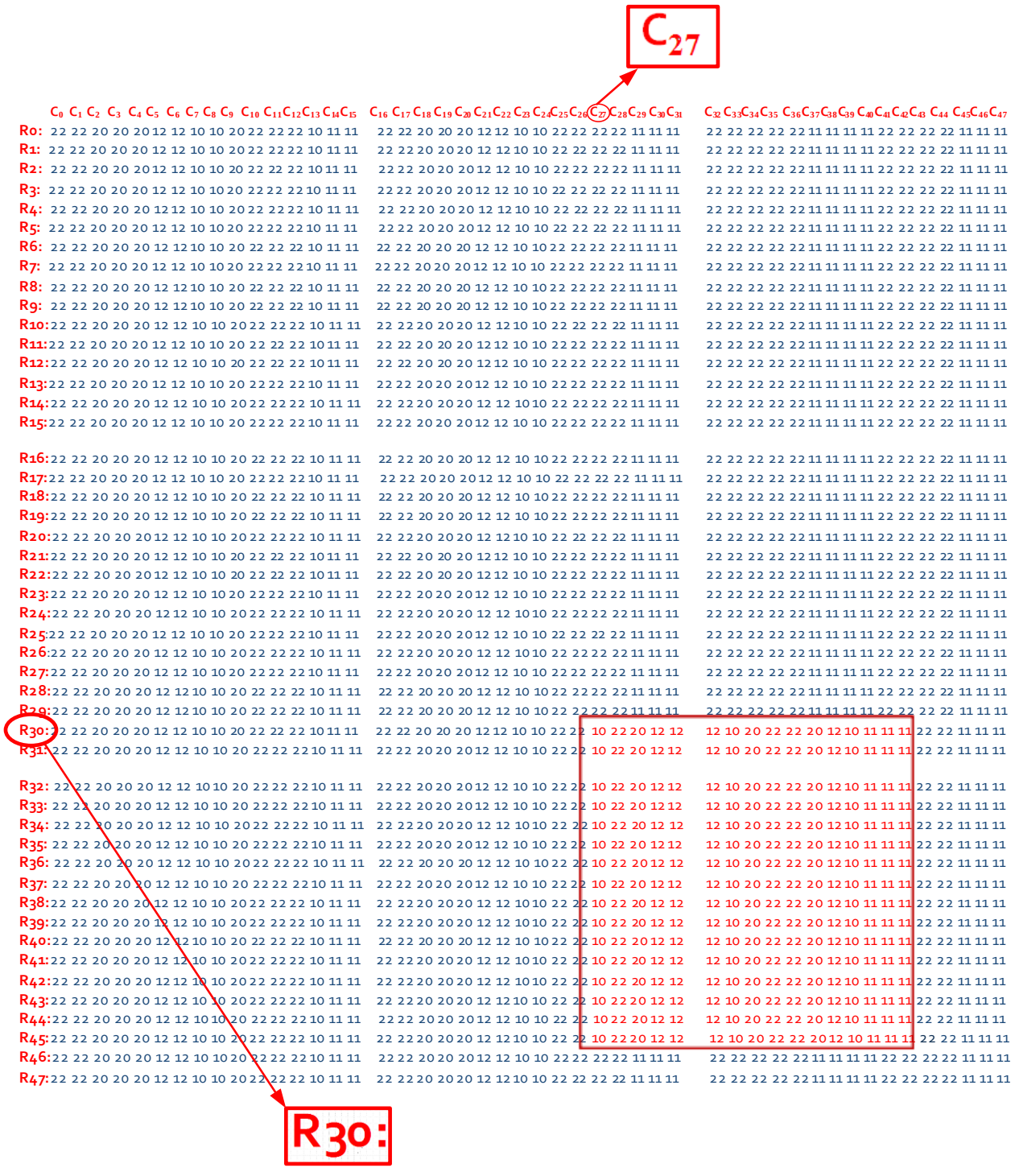

Figure 22: The best match is selected at (R_{30}, C_{27}) for a search area of size 32×32.

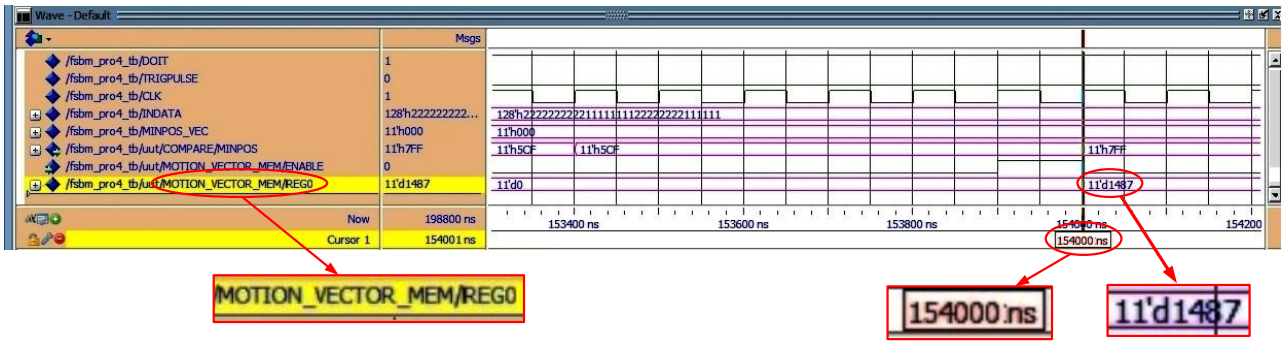

Figure 23: The simulation result of the best match is located at clock # 1487.

CONCLUSION AND FUTURE WORK

Full Search Motion Estimation architecture is proposed and fully functionally tested using VHDL verification language. Simulation results show that the whole Motion Estimation process can be performed using 1540 clock cycles (it means the transmission throughput is $\frac{1}{1540}$) with 100% utilization of all resources. Data reuse is achieved using smart data flow as well as small internal local memory. Simulation results show that the proposed architecture can find the exact AMV with 100% success rate. The future work will include the calculation of hardware cost and comparing the proposed design with the state of the art ME architectures. Since the proposed architecture uses less number of components and has a regular data flow, this is expected to positively affect on speed, area, and power consumption. A comprehensive comparison between the architecture in this paper and the future work is considered.

ACKNOWLEDGMENT

The author acknowledges the support of the Deanship of Scientific Research – University of Bahrain – Bahrain for supporting this work under the project number 43308016. The author thanks his student, Najeeba Mohammed Jaffar, for her efforts in simulations.

REFERENCES

[1] J. Ohm, G. J. Sullivan, H. Schwarz, T. Thiow Keng, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 22, pp. 1669-1684, 2012.

[2] Z. Hao and M. Zhan, "Fast Intra Mode Decision for High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 24, pp. 660-668, 2014.

[3] Y. Ismail, J. B. McNeely, M. Shaaban, H. Mahmoud, and M. A. Bayoumi, "Fast Motion Estimation System Using Dynamic Models for H.264/AVC Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* , vol. 22, pp. 28-42, 2012.

[4] "High Efficiency Video Coding (HEVC) Text Specification Draft 6," *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 WP3*, Feb. 2012.

[5] Y. Ismail, W. El-Medany, H. Al-Junaid, and A. Abdelgawad, "High Performance Architecture for Real-time HDTV Broadcasting," *Journal of Real-Time Image Processing, Springer, ISSN: 1861-8200 (print version), and ISSN: 1861-8219 (electronic version)*, May 27, 2014.

[6] H. Amirpour, A. Mousavinia, and N. Shamsi, "Predictive Three Step Search (PTSS) algorithm for motion estimation," in *2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*, 2013, pp. 48-52.

[7] H. A. Choudhury and M. Saikia, "Reduced three steps logarithmic search for motion estimation," in *2014 International Conference on Information Communication and Embedded Systems (ICICES)*, 2014, pp. 1-5.

[8] L. Renxiang, Z. Bing, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, pp. 438-442, 1994.

[9] P. Lai-Man and M. Wing-Chung, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 313-317, 1996.

[10] Z. Shan and M. Kai-Kuang, "A new diamond search algorithm for fast block matching motion estimation," in *Proceedings of 1997 International Conference on Information, Communications and Signal Processing, 1997. ICICS.*, 1997, pp. 292-296 vol.1.

[11] C. Chun-Ho and P. Lai-Man, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 1168-1177, 2002.

[12] C. Changryoul and J. Jechang, "Successive Elimination Algorithm for Constrained One-bit Transform Based Motion Estimation Using the Bonferroni Inequality," *IEEE Signal Processing Letters*, vol. 21, pp. 1260-1264, 2014.

[13] L. Hwal-Suk, J. Jik-Han, and P. Dong-Jo, "An effective successive elimination algorithm for fast optimal block-matching motion estimation," in *15th IEEE International Conference on Image Processing, 2008. ICIP 2008*, 2008, pp. 1984-1987.

[14] Y. Ismail, M. Shaaban, J. B. McNeely, and M. A. Bayoumi, "An Efficient Adaptive High Speed Manipulation Architecture for Fast Variable Padding Frequency Domain Motion Estimation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 1239-1248, 2011.

[15] S. Goel, Y. Ismail, and M. A. Bayoumi, "Adaptive search window size algorithm for fast motion estimation in H.264/AVC standard," in *48th Midwest Symposium on Circuits and Systems, 2005*, 2005, pp. 1557-1560 Vol. 2.

[16] J. Sung-Tae and L. Sang-Seol, "A 4-way pipelined processing architecture for three-step search block-matching motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 50, pp. 674-681, 2004.

[17] D. Xu, J. M. Noras, and W. Booth, "A simple and efficient VLSI architecture for a very fast high performance three step search algorithm," in *IEE Colloquium on High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197)*, 1998, pp. 6/1-6/6.

[18] L. Yeong-Kang and C. Lien-Fei, "A high data-reuse architecture with double-slice processing for full-search block-matching algorithm," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.* , 2003, pp. II-716-II-719 vol.2.

[19] P. Davis and S. Marikkannan, "Implementation of Motion Estimation Algorithm for H.265/HEVC," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 3, Special Issue 3, April 2014.

[20] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *Trans. Circuits Syst. Video Technol.*. vol. 12, pp. 61-72, Jan. 2002.

[21] D. Meng, C. Canhui, and M. Kai-Kuang, "A novel multiple description video coding based on data reuse," in *2013 20th IEEE International Conference on Image Processing (ICIP)*, 2013, pp. 1928-1932.

[22] T. Muralidhar Reddy, P. Muralidhar, and C. B. Rama Rao, "New fast search block matching Motion Estimation algorithm for H.264 /AVC," in *2014 International Conference on Recent Trends in Information Technology (ICRTIT)*, 2014, pp. 1-5.

[23] G. He, D. Zhou, Y. Li, Z. Chen, T. Zhang, and S. Goto, "High-Throughput Power-Efficient VLSI Architecture of Fractional Motion Estimation for Ultra-HD HEVC Video Encoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, pp. 1-1, 2015.

[24] N. Aarthi, K. S. Athishkarthic, N. Madhan Kumar, and P. Jayakrishnan, "A high performance 2-dimensional VLSI architecture for H.264/AVC Variable Block Size integer motion estimation," in *2013 International Conference on Emerging Trends in Communication, Control, Signal Processing & Computing Applications (C2SPCA)*, 2013, pp. 1-4.

[25] W. Yansheng, L. Leibo, Y. Shouyi, Z. Min, C. Peng, Y. Jun, and W. Shaojun, "On-Chip Memory Hierarchy in One Coarse-Grained Reconfigurable Architecture to Compress Memory Space and to Reduce Reconfiguration Time and Data-Reference Time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 983-994, 2014.

[26] W. Shugang, "Residue checker using optimal signed-digit adder tree for error detection of arithmetic circuits," in *2014 IEEE Region 10 Conference - TENCON 2014* 2014, pp. 1-6.

Dr. Yasser Ismail received the B.Sc.degree in Electronics & Communications Engineering from Mansoura University, Mansoura, Egypt, in 1999, the M.Sc. degree in Electrical Communications from Mansoura University, Mansoura, Egypt, in 2002, the M.Sc. degree in Computer Engineering from University of Louisiana at Lafayette, Louisiana, USA, in 2007. Dr. Yasser Ismail got his Ph.D. from the University of Louisiana at Lafayette in May 2010. Dr. Yasser Ismail worked as an assistant professor in Umm Alqura University – KSA from 2010 to 2012. He is currently working as an assistant professor in University Of Bahrain (UOB) - Bahrain. Dr. Yasser permanently working at the Electronics and Communications Engineering Department – Faculty of Engineering – Mansoura University – Mansoura – Egypt. Dr. Yasser is served as a reviewer for several conferences and journals, including ISCAS 2010, ICIP 2010, ICIP 2011, ICECS2013, Transaction on Circuit and System for Video Technology (TCSVT), and IEEE Transactions on Image Processing, and Signal Processing. He has also gained many valuable projects from KSA, NSF, and Bahrain. Dr. Yasser served in the organizing committee of 2013 IEEE International Conference on Electronics, Circuits, and Systems (ICECS2013). His research of interest includes video processing, digital signal processing, Robotics, RFID, Localization, VLSI, FPGA, wireless communication systems, and low power embedded systems.