A HIGH PERFORMANCE HARDWARE ARCHITECTURE FOR AN SAD REUSE BASED HIERARCHICAL MOTION ESTIMATION ALGORITHM FOR H.264 VIDEO CODING

Sinan Yalcin, Hasan F. Ates, Ilker Hamzaoglu

Faculty of Engineering and Natural Sciences, Sabanci University,
34956, Tuzla, Istanbul, Turkey
syalcin@su.sabanciuniv.edu,{hasanates, hamzaoglu}@sabanciuniv.edu

ABSTRACT

In this paper, we present a high performance and low cost hardware architecture for real-time implementation of an SAD reuse based hierarchical motion estimation algorithm for H.264 / MPEG4 Part 10 video coding. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 68 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames (640x480) or 82 CIF frames (352x288) per second.

1. INTRODUCTION

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing systems. These applications make the video compression hardware devices an inevitable part of many commercial products. To improve the performance of the existing applications and to enable the applicability of video compression to new real-time applications, recently, a new international standard for video compression is developed. This new standard, offering significantly better video compression efficiency than previous video compression standards, is developed with the collobaration of ITU and ISO standardization organizations. Hence it is called with two different names, H.264 and MPEG4 Part 10.

The video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools. As it is shown in the top-level block diagram of an H.264 Encoder in Figure 1, one of these tools is the variable block size motion estimation used in the baseline profile of H.264 standard [1, 2, 3]. Motion estimation is the most computationally demanding part of the encoders implementing the previous video compression standards. Variable block size motion estimation achieves better coding results than the fixed block size motion estimation used in the previous video compression standards. However, the amount of computation required by variable block size motion estimation is even more than the amount required by fixed

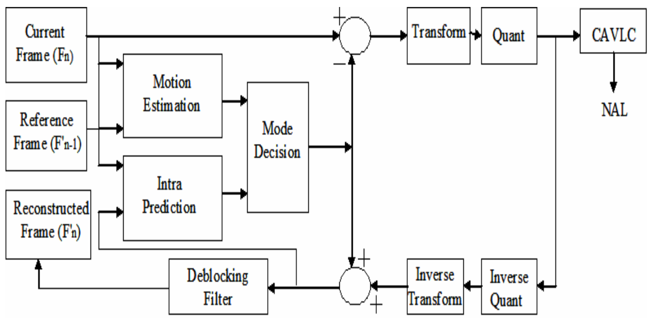

Fig. 1. H.264 Encoder Block Diagram

block size motion estimation. Therefore, this coding gain comes with an increase in encoding complexity which makes it an exciting challenge to have a real-time implementation of motion estimation for H.264 video coding.

In this paper, we present a high performance and low cost hardware architecture for real-time implementation of an SAD reuse based hierarchical motion estimation algorithm for H.264 / MPEG4 Part 10 video coding. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 68 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames (640x480) or 82 CIF frames (352x288) per second.

A hardware architecture for real-time implementation of a variable block size motion estimation algorithm for H.264 video coding is presented in [4]. This hardware achieves higher performance than our hardware design at the expense of a much higher hardware cost. Our hardware design is a more cost-effective solution for portable applications. They use 256 processing elements in their datapath as opposed to 36 processing elements in our datapath.

The rest of the paper is organized as follows. Section II explains the hierarchical motion estimation algorithm. Section III describes the proposed architecture in detail. The implementation results are given in Section IV. Finally, Section V presents the conclusions.

2. SAD REUSE BASED HIERARCHICAL MOTION ESTIMATION ALGORITHM

The amount of computation required by full-search method (FSM) is not practical for real-time implementation even for fixed block size motion estimation (ME). Variable block size ME allows dividing a 16x16 Macroblock (MB) into different size partitions and using a different motion vector (MV) for each partition. A 16x16 MB can be divided into two 8x16 or two 16x8 or four 8x8 partitions. Each 8x8 partition can further be partitioned into two 4x8, two 8x4 or four 4x4 partitions. A variable block size ME algorithm, therefore, has to find the best MVs for all partitions of the MB; ([1 MV for 16x16 MB] + [2 MVs for 16x8 partitions] + [2 MVs for 8x16 partitions] + [4 MVs for 8x8 partitions] + [8 MVs for 8x4 partitions] + [8 MVs for 4x8 partitions] + [16 MVs for 4x4 partitions] = 41 MVs). The best partition for the MB is determined by the Mode Decision algorithm based on these 41 MVs. Therefore, efficient algorithms are needed to reduce the computational cost for variable block size ME [5, 6].

In this paper, we propose to use an SAD reuse based hierarchical ME algorithm similar to the algorithm presented in [6]. The simulation results show that even though this algorithm has a much lower computational cost than FSM, it provides almost as good coding efficiency as FSM.

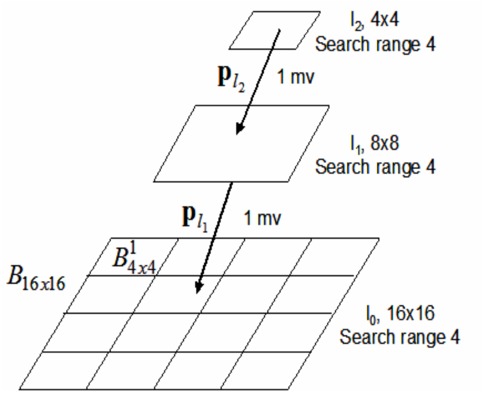

Fig. 2. Hierarchical Motion Estimation Algorithm

The algorithm is illustrated in Figure 2. It consists of the following four steps:

1 A 3-level pyramid is constructed using averages of the MB pixels. A 4x4 block at level l_2 corresponds to the 16x16 MB in level l_0.

2 A MV, $\mathbf{p}_{l_2}$, is predicted for the 16x16MB in level l_0 by performing full search for the 4x4 block in level l_2 within a search range of [-R/4, R/4] ([-R, R] is the search range of the FSM).

3 The MV prediction is refined by performing full search for the 8x8 block at the location pointed by the motion vector $2\mathbf{p}_{l_2}$ in level l_1 within a search range of [-R/4, R/4]. The refined MV prediction is $\mathbf{p}_{l_1}$.

4 The MVs for the 16x16 MB and for all of its partitions are determined by performing full search based on minimizing the Lagrangian cost ($\mathcal{J}(\mathbf{d})$) for all the partitions at both the location pointed by the motion vector $2\mathbf{p}_{l_1}$ and location (0,0) in level l_0 within a limited search range of ([-R/4, R/4]). The Lagrangian cost is computed using the following equations:

$$\mathcal{J}(\mathbf{d}) = \mathrm{SAD}_{\mathrm{B}_{\mathrm{mxn}}}(\mathbf{d}) + \lambda_M R(\mathbf{d} - \mathbf{p}_{med})$$

$$\mathrm{SAD}_{\mathrm{B}_{\mathrm{mxn}}}(\mathbf{d}) = \sum_{x=1,y=1}^{m,n} | c(x,y) - r(x+d_x, y+d_y) |$$

where $\mathrm{B}_{\mathrm{mxn}}$ is a partition of size mxn, (m,n) ∈ {(4,4), (4,8), (8,4), (8,8), (16,8), (8,16), (16,16)}, $\mathbf{d} = (d_x, d_y)$ is the MV, c and r are current and reference frames respectively, λ_{M} is the Lagrange multiplier for ME, $\mathbf{p}_{med}$ is the MV prediction used by H.264 video coding standard during the coding process, and $R(\mathbf{d} - \mathbf{p}_{med})$ specifies the bitrate spent for coding MV difference information.

The refined MV prediction in level l_1 constitutes a good initial prediction for the 16x16 MB and for all of its partitions in level l_0 when scaled by 2. Therefore, hierarchical motion vector prediction, $2\mathbf{p}_{l_1}$ is used as a MV prediction for the 16x16 MB and for all of its partitions. However, in some cases, $2\mathbf{p}_{l_1}$ is inaccurate for small partitions such as 4x4, using (0,0) vector as an additional MV prediction helps to alleviate this problem.

The full search for a 16x16 MB and for all of its partitions performed in level l_0 require computing the Sum of Absolute Differences (SADs) for all MVs within the search range for all partitions. However, since the full search for a 16x16 MB and for all of its partitions are performed starting at the same location in level l_0 (location pointed by the motion vector $2\mathbf{p}_{l_1}$ or location (0,0)) within the same size search range ([-R/4, R/4]), SADs computed for 4x4 partitions can be reused to compute the SADs for larger partitions, e.g. 8x8, 16x16. In other words, for a given MV $\mathbf{d} = (d_x, d_y)$, SAD of $\mathrm{B}_{\mathrm{mxn}}$ can be decomposed into the SADs of its 4x4 partitions:

$$\mathrm{SAD}_{\mathrm{B}_{\mathrm{mxn}}}(\mathbf{d}) = \sum_{\substack{k=1,\\ l=1}}^{m/4,n/4} \sum_{\substack{x=4k-3,\\ y=4l-3}}^{4k,4l} | c(x,y) - r(x+d_x, y+d_y) |$$

Since all summations on the right are evaluated at the same MV $\mathbf{d}$, computing $SAD_{B_{mxn}}(\mathbf{d})$ requires computing the SADs of all its 4x4 partitions for MV $\mathbf{d}$ and adding them up. This SAD reuse technique decreases the total number of computations significantly.

The SAD reuse based hierarchical ME algorithm is integrated into the Joint Model (JM) Reference Software Version 7.4 [7]. The updated software is then used to simulate the hierarchical ME algorithm for R=16 using video sequences carphone (QCIF), foreman (CIF), mobile (SIF) and flowergarden (SIF) at 30fps. All frames except the first one are coded as P-frames. One reference frame is allowed. The CAVLC entropy coder is used, with quantization parameter values QP = 24, 28, 32, 36. For comparison to FSM, average PSNR loss in dB and percentage change in bitrate are reported in Table 1. In addition, at equal bitrates, PSNR loss is observed to be less than 0.2 dB for all the tested sequences. These results confirm that even though our algorithm has a much lower computational cost than FSM, it provides almost as good coding efficiency as FSM.

Table 1. Performance Comparison with FSM

	δPSNR (dB)	δbitrate (%)
carphone (QCIF)	-0.04	+0.76
foreman (CIF)	-0.04	+3.11
mobile (SIF)	-0.02	+0.39
flowergarden(SIF)	-0.02	+0.73

3. PROPOSED HARDWARE ARCHITECTURE

In this section, we will explain the proposed hardware architecture for real-time implementation of the SAD reuse based hierarchical motion estimation algorithm described in section 2. The proposed hardware implements the algorithm for the case where R=16 and therefore the search ranges used in all 3 levels l_0, l_1 and l_2 are [-4, 4]. The search window for a [-4, 4] search range contains 9x9 = 81 search locations; 2*4+1 = 9 rows and 2*4+1 = 9 search locations in each row.

The current MB (16x16 pixels) and search window (64x64 pixels) are stored in block RAMs in the FPGA. The proposed hardware first constructs a 3-level pyramid by using the averaging datapath shown in Figure 3. The datapath is used to generate the current block and search window values in levels l_1 and l_2 by calculating the average of the corresponding pixels in the current MB and search range in level l_0. Each averaging unit calculates the average of 4 pixels in level l_0. The resulting values are stored in registers and they are used to perform full search for the 8x8 block in level l_1 within a search range of [-4, 4]. The averaging unit A5 calculates the average of the results produced by A1-A4 which corresponds to the average of 16

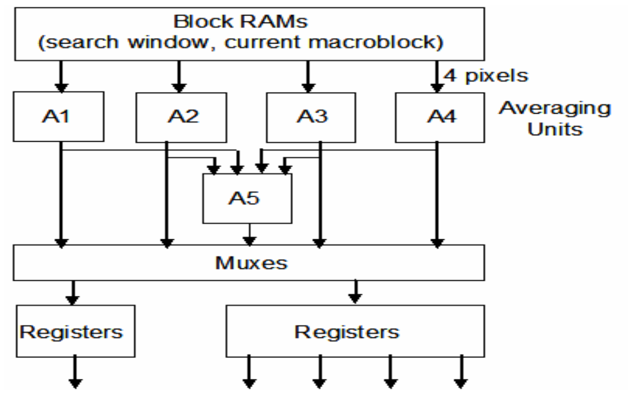

Fig. 3. Averaging Datapath

pixels in level l_0. The resulting values are stored in registers and they are used to perform full search for the 4x4 block in level l_2 within a search range of [-4, 4]. The averaging process takes 640 clock cycles.

The proposed hardware then performs both the hierarchical MV prediction in levels l_2 and l_1, and motion estimation with SAD reuse in level l_0 using the datapath shown in Figure 4. The datapath uses 36 PEs divided into four separate groups. Each group has an array of 9 PEs. The architecture of a PE and the organization of PEs in a group are shown in Figure 5. As we will explain in this section, the reason for using 36 PEs divided into four separate groups is to have an efficient real-time implementation of the motion estimation with SAD reuse in level l_0. The hierarchical MV prediction in levels l_2 and l_1 are implemented by utilizing the hardware resources used for the motion estimation with SAD reuse in level l_0.

The datapath is first used for the hierarchical MV prediction in level l_2 by performing full search for the 4x4 block in level l_2 within a search range of [-4, 4]. All 36 PEs in the datapath are used to perform the full search as follows. Each PE is used to calculate the SAD value for one search location in the search window. Since there are 9 search locations in one row of the search window, a PE group is used to calculate the SAD values for the search locations in one row of the search window. After each PE group finishes calculating the SAD values for the search locations in one row of the search window, it starts calculating the SAD values in another row of the search window. Therefore, each PE group together with a multiplexer and comparator is used to find the minimum SAD in two rows of the search window. All 4 PE groups are, therefore, utilized to find the motion vector $\mathbf{p}_{l_2}$ with the minimum SAD in the search window. This process takes 42 clock cycles.

The datapath is then used for the hierarchical MV refinement in level l_1 by performing full search for the 8x8 block at the location pointed by the motion vector $2\mathbf{p}_{l_2}$ in

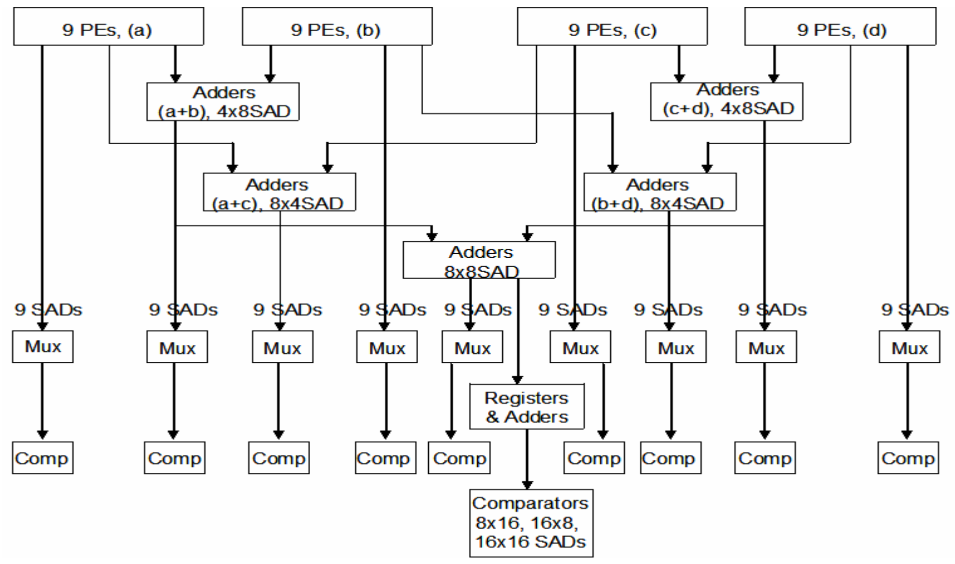

Fig. 4. Hierarchical Motion Estimation Datapath

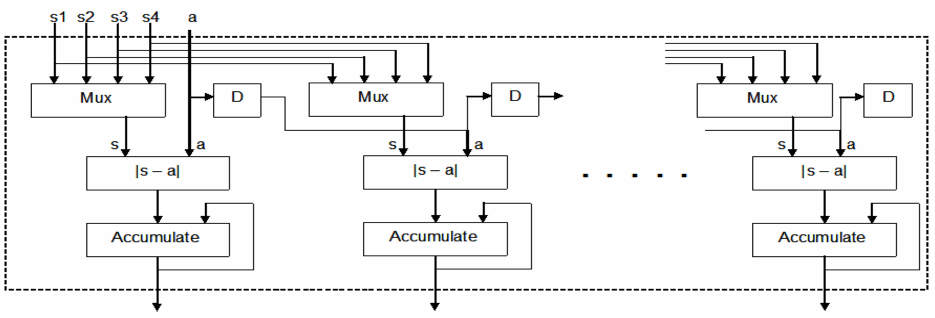

Fig. 5. Processing Element Group

level l_1 within a search range of [-4, 4]. Since there are four 4x4 partitions (a, b, c, and d) of the 8x8 block and there are 9 search locations in one row of the search window, each PE group is used to calculate the SAD values for a 4x4 partition for the search locations in one row of the search window. Each PE in a group calculates the SAD value for its 4x4 partition for one search location in one row of the search window. PE groups 0, 1, 2, and 3 are used for partitions a, b, c, and d respectively. After each PE group finishes calculating the SAD values for its 4x4 partition for the search locations in the current row of the search window, it starts calculating the SAD values for its 4x4 partition in the next row of the search window. After the corresponding processing elements in each PE group, e.g. processing element 0 in each PE group, calculate the SAD value for a search location for its 4x4 partition, the 4x8SAD and 8x8SAD adders in the datapath are used to calculate the SAD value for that search location for the 8x8 partition. The multiplexer and comparator at the outputs of the 8x8SAD adders are used to find the minimum SAD for the 8x8 partition and the corresponding motion vector $\mathbf{p}_{l_1}$ in the search window. This process takes 156 clock cycles.

The datapath is finally used for the motion estimation with SAD reuse in level l_0. It is used to perform full search based on minimizing the Lagrangian cost for the 16x16 current MB and for all of its partitions at both the location pointed by the motion vector $2\mathbf{p}_{l_1}$ and location (0,0) within a search range of [-4, 4] to determine the 41 best motion vectors for all partitions of the MB. The datapath is designed to use the SAD reuse technique for performing full search for a 16x16 MB and for all of its partitions within a search range of [-4, 4]. Each PE group in the datapath together with a multiplexer and comparator is used to perform full search for a 4x4 partition of the 16x16 MB within a search range of [-4, 4]. Since there are 9 search locations in one row of the search window, 9 PEs are grouped together to calculate the SAD values for a 4x4 partition for the search locations in one row of the search window. Each processing element in a group calculates the SAD value for a 4x4 partition for one search location in one row of the search window.

Cycle	Input Data						Processing Elements Inputs									
	a	b	s1	s2	s3	s4	PE0	PE1	...	PE7	PE8	PE9	PE10	...	PE16	PE17
0	a00		s00				a00,s00									
1	a01		s01				a01,s01	a00,s01								
2	a02		s02				a02,s02	a01,s02								
3	a03		s03				a03,s03	a02,s03								
4	a10	b00	s10	s04			a10,s10	a03,s04				b00,s04				
5	a11	b01	s11	s05			a11,s11	a10,s11				b01,s05	b00,s05			
:	:	:	:	:	:		:	:		:	:	:	:		:	
12	a30	b20	s30	s24	s18	s0C	a30,s30	a23,s24	...	a11,s18	a10,s18	b20,s24	b13,s18	...	b01,s0C	b00,s0C
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
15	a33	b23	s33	s27	s1B	s0F	a33,s33	a32,s33	...	a20,s27	a13,s1B	b23,s27	b22,s27	...	b10,s1B	b03,s0F
16	a00	b30	s10	s34	s28	s1C	a00,s10	a33,s34	...	a21,s28	a20,s28	b30,s34	b23,s28	...	b11,s1C	b10,s1C
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
23	a13	b03	s23	s17	s3B	s2F	a13,s23	a12,s23	...	a00,s28	a33,s3B	b30,s34	b23,s28	...	b11,s1C	b10,s1C
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
31	a33	b23	s43	s37	s2B	s1F	a33,s43	a32,s43	...	a20,s37	a13,s2B	b23,s37	b22,s37	...	b10,s2B	b03,s1F
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
:	:	:	:	:	:	:	:	:		:	:	:	:		:	:
143	a33	b23	sB3	sA7	s9B	s8F	a33,sB3	a32,sB3	...	a20,sA7	a13,s9B	b23,sA7	b22,sA7	...	b10,s9B	b03,s8F
144		b30		sB4	sA8	s9C		a33,sB4	...	a21,sA8	a20,sA8	b30,sB4	b23,sA8	...	b11,s9C	b10,s9C
:		:		:	:	:				:	:	:	:		:	:
:					:	:				:	:				:	:
155						sBF										b33,sBF

Fig. 6. Data Flow for Processing Elements PE0-PE17

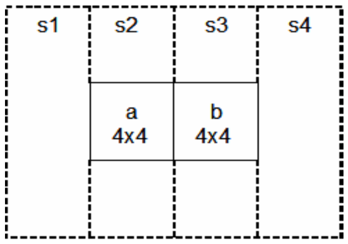

Fig. 7. Search Window Overlap of Two Neighboring 4x4 Partitions

As it is shown in Figure 6, in order to reduce the number of current block and search window register ports and number of accesses to these registers, each PE in a group starts calculating its SAD value one cycle later than the previous PE in that group so that PEs can reuse the current block value accessed by the first PE in the group and several PEs can use the same search window value in the same cycle. Since PE0 starts working in cycle 0, it finishes calculating its first SAD in cycle 15. The last PE in that group, PE8, finishes calculating its SAD in cycle 8 + 15 = 23. After each PE finishes calculating an SAD value for a 4x4 partition in the current row of the search window, it starts calculating an SAD value for the same 4x4 partition in the next row of the search window. Since there are 9 rows in the search window, the minimum SAD for a 4x4 partition and the corresponding motion vector is found in 8 + 9x16 = 152 cycles.

Since the full search for a 16x16 MB and for all of its partitions are performed starting at the same location in level l_0 (location pointed by the motion vector $2\mathbf{p}_{l_1}$ or location (0,0)) within the same size search range ([-4, 4]), the search windows of two neighboring 4x4 partitions (a, b) of the MB overlap as shown in Figure 7. The search window regions s1, s2 and s3 are used for partition a, and the search window regions s2, s3 and s4 are used for partition b. Therefore, the search window regions s2 and s3 are shared by both a and b partitions. In order to exploit this to reduce the number of search window register ports (from 3+3=6 to 4) and the number of accesses to search window registers, the full search for partitions a and b are performed simultaneously by using PE group 0 for partition a and PE group 1 for partition b. As it is shown in Figure 6, the processing elements in PE group 1 starts calculating their SADs 4 cycles later than the corresponding processing elements in PE group 0 so that several PEs in group 0 and group 1 can use the same search window value (in regions s2 or s3) in the same cycle. Therefore, the minimum SAD for partition b and the corresponding motion vector is found in 4+152=156 cycles.

As the PE groups 0 and 1 perform the full search for partitions a and b, PE groups 2 and 3 perform the full search for partitions c and d simultaneously based on the same data flow shown in Figure 6. Therefore, the minimum SADs for 4x4 partitions a, b, c and d and the corresponding motion vectors are found in 156 cycles.

After the corresponding processing elements in each PE group, e.g. processing element 0 in each PE group, calculate the SAD value for a search location for its 4x4 partition, the 4x8SAD, 8x4SAD and 8x8SAD adders in the datapath are used to calculate the SAD values for that search location for the 4x8 (a+b and c+d), 8x4 (a+c and b+d), and 8x8 (a+b+c+d) partitions by reusing the SAD values of the 4x4 partitions. In other words, as the full search for 4x4 partitions a, b, c, and d are performed, the full search for two 4x8 (a+b and c+d), two 8x4 (a+c and b+d), and one 8x8 (a+b+c+d) partition are also performed in parallel by using the 4x8SAD, 8x4SAD and 8x8SAD adders and the multiplexers and comparators at their outputs in the datapath. Therefore, by using the SAD reuse technique, the minimum SADs for two 4x8, two 8x4 and one 8x8 partition and the corresponding motion vectors are found as well in the same 156 cycles.

After the full search for the first four 4x4 partitions are performed, the four PE groups are used to perform the full search for the next four 4x4 partitions of the MB. Again, by using the SAD reuse technique, the full search for the corresponding two 4x8, two 8x4, and one 8x8 partition are performed in parallel. Since there are four 8x8 partitions in a MB, this process is repeated 4 times. Therefore, full search for all 4x4, 4x8, 8x4 and 8x8 partitions are performed in 4*156 = 624 clock cycles.

As the full search for 8x8 partitions are performed, the full search for 8x16, 16x8 and 16x16 partitions are also performed in parallel by using the 8x16SAD, 16x8SAD and 16x16SAD registers, adders, multiplexers and comparators in the datapath. Therefore, by using the SAD reuse technique, the minimum SADs for 8x16, 16x8 and 16x16 partitions and the corresponding motion vectors are found as well in the same 624 clock cycles.

After the full search for the 16x16 current MB and for all of its partitions at the location pointed by the motion vector $2\mathbf{p}_{l_1}$ within a search range of [-4, 4] are performed, the full search for the same MB and for all of its partitions are performed at location (0, 0) within a search range of [-4, 4] by using the same datapath with the same data flow. This process takes 624 clock cycles as well. Therefore, the 41 best motion vectors for all partitions of a MB are determined in 640 (averaging) + 42 (level l_2) + 156 (level l_1) + 2*624 (level l_0) = 2086 clock cycles.

4. IMPLEMENTATION RESULTS

The proposed architecture is implemented in Verilog HDL. The implementation is verified with RTL simulations using Mentor Graphics ModelSim SE. The Verilog RTL is then synthesized to a 2V8000ff1152 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE Series 5.2i.

The FPGA implementation is verified to work at 68 MHz under worst-case PVT conditions with post place and route simulations. The FPGA implementation can process a VGA frame in 36.8 msec. (1200 MB * 2086 clock cycles per MB * 14.7 ns clock cycle = 36.8 msec) Therefore, it can process 1000/36.8 = 27 VGA frames (640x480) per second. The FPGA implementation can process a CIF frame in 12.2 msec. (396 MB * 2086 clock cycles per MB * 14.7 ns clock cycle = 12.2 msec) Therefore, it can process 1000/12.2 = 82 CIF frames (352x288) per second.

The FPGA implementation including input, output and internal RAMs and register files uses the following FPGA resources; 14505 Function Generators, 7253 CLB Slices, 5227 Dffs/Latches, 13 Block RAMs, and 7 Block Multipliers (used for calculating λ_M * R), i.e. %15.5 of Function Generators, %15.5 of CLB Slices, %5.4 of Dffs/Latches, %7.7 of Block RAMs, and %4.1 of Block Multipliers.

5. CONCLUSION

In this paper, we presented a high performance and low cost hardware architecture for real-time implementation of an SAD reuse based hierarchical motion estimation algorithm for H.264 / MPEG4 Part 10 video coding. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 68 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 27 VGA frames (640x480) or 82 CIF frames (352x288) per second.

6. REFERENCES

1 T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003

2 I. G. Richardson, H.264 and MPEG-4 Video Compression, Wiley, 2003

3 Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003

4 Yu-Wen Huang, Tu-Chih Wang, Bing-Yu Hsieh, and Liang-Gee Chen, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264", *Proc. IEEE ISCAS*, May 2003

5 H. C. Tourapis and A.M. Tourapis, "Fast motion estimation within the H.264 codec," *Proc. IEEE Int. Conf. Multimedia and Expo*, vol. 3, pp. 517–20, July 2003

6 H. F. Ates and Y. Altunbasak, "SAD Reuse in Hierarchical Motion Estimation for the H.264 Encoder", *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, March 2005

7 Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Joint Model (JM) Reference Software Version 7.4, http://iphome.hhi.de/suehring/tml