A HIGH PERFORMANCE HARDWARE ARCHITECTURE FOR HALF-PIXEL ACCURATE H.264 MOTION ESTIMATION

Sinan Yalcin and Ilker Hamzaoglu
Faculty of Engineering and Natural Sciences, Sabanci University,
34956, Tuzla, Istanbul, Turkey
syalcin@su.sabanciuniv.edu, hamzaoglu@sabanciuniv.edu

Abstract—In this paper, we present a high performance and low cost hardware architecture for real-time implementation of half-pel accurate variable block size motion estimation for H.264 / MPEG4 Part 10 video coding. The proposed architecture includes a novel half-pel interpolation hardware that is shared by novel half-pel search hardwares designed for each block size. This half-pel accurate motion estimation 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 85 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 30 HDTV frames (1280x720) per second.

I. INTRODUCTION

Video compression systems are used in many commercial products. 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, a new international standard is developed. This new standard, called H.264 / MPEG4 Part 10, offers significantly better video compression efficiency than previous standards.

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 half-pel accurate 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 video encoders. In order to increase the performance of integer-pel motion estimation, half-pel accurate variable block size motion estimation is performed [4, 5, 6]. However, the amount of computation required by half-pel accurate variable block size motion estimation is even more than the amount required by integer-pel 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 half-pel accurate variable block size 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 half-pel accurate variable block size motion estimation for H.264 / MPEG4 Part 10 video coding. The proposed architecture includes a novel half-pel interpolation hardware that is shared by novel half-pel search hardwares designed for each block size. To the best of our knowledge, this is the first half-pel accurate variable block size motion estimation hardware in the literature for H.264 standard. This half-pel accurate motion estimation hardware is designed to be used as part of a complete H.264 video coding system for portable applications together with a previously published integer-pel motion estimation hardware [7]. The proposed architecture is

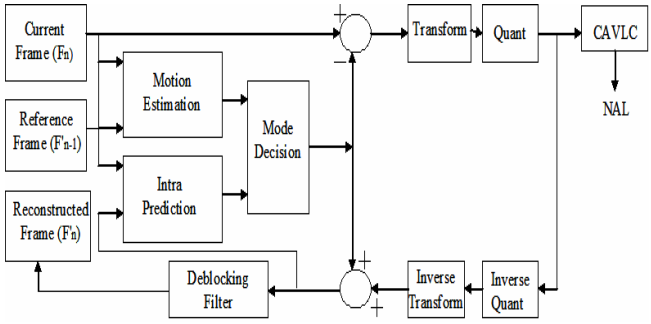

Figure 1. H.264 Encoder Block Diagram

implemented in Verilog HDL. The Verilog RTL code is verified to work at 85 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 30 HDTV frames (1280x720) per second.

Our hardware design is a cost-effective solution for portable applications. Interpolation datapath in the proposed hardware is shared by various block size modes and search datapaths use 28 processing elements for search operations.

The rest of the paper is organized as follows. Section II explains the half-pel accurate motion estimation algorithm. The proposed half-pel interpolation hardware is described in Section III and the proposed half-pel search hardware is described in Section IV. The implementation results are given in Section V. Finally, section VI presents the conclusions.

II. OVERVIEW OF HALF-PEL ACCURATE MOTION ESTIMATION ALGORITHM

The search locations for half-pel accurate motion estimation are shown in Figure 2. First, integer-pel motion estimation is performed at the integer-pel search locations and best integer-pel motion vector (MV) is determined based on a performance metric, e.g. minimum Sum of Absolute Difference (SAD). Then, half-pel motion estimation is performed at the half-pel search locations around the best integer-pel MV with a search range of [-1, 1], and the integer-pel MV is refined by the best half-pel accurate MV.

Before searching for the best half-pel accurate MV, half pixels in the half-pel search window are interpolated from neighboring pixels using a 6-tap FIR filter with weights (1/32, -5/32, 5/8, 5/8, -5/32, 1/32). First, the half pixels that are adjacent to two integer pixels are interpolated from 6 integer pixels. Then, the remaining half pixels are interpolated from 6 horizontal or 6 vertical half pixels. A half-pel interpolation example is shown in Figure 3. First, the half pixels a, b, c, d, e, f are interpolated from 6 corresponding horizontal integer pixels. For example, half pixel c is interpolated from the 6 horizontal integer pixels A, B, C, D, E, F (c = round ((A-5B+20C+20D-5E+F) / 32)). Then, the half pixels

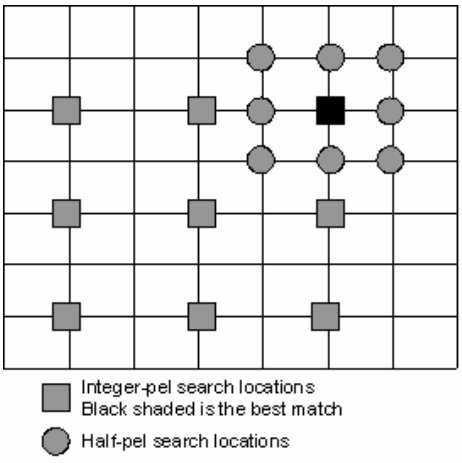

Figure 2. Half-Pel Search Locations

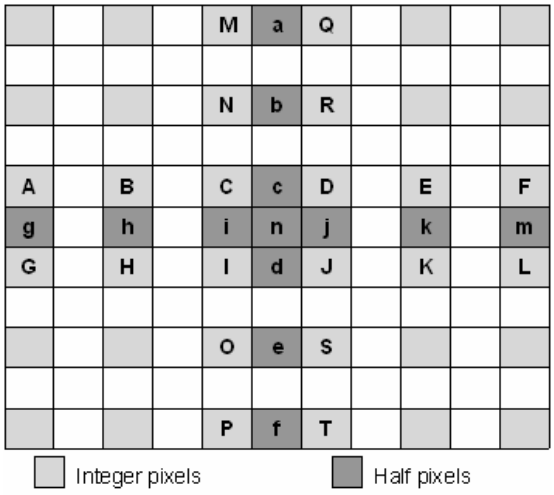

Figure 3. Half-Pel Interpolation Example

g, h, i, j, k, m are interpolated from 6 corresponding vertical integer pixels. For example, half pixel i is interpolated from the 6 vertical integer pixels M, N, C, I, O, P. Finally, half-pel n can be interpolated from either horizontal half pixels g, h, i, j, k, m or vertical half-pixels a, b, c, d, e, f.

III. PROPOSED HALF-PEL INTERPOLATION HARDWARE

The half-pel accurate variable block size motion estimation hardware performs half-pel interpolation and half-pel search for each block size. First, half-pel interpolation hardware calculates the half pixels in the half-pel search window of a block. Then, half-pel search hardware searches the half-pel search locations and determines the best half-pel accurate MV for that block. This process is repeated for all block sizes.

The proposed half-pel interpolation hardware for 4x4, 4x8 and 8x4 block sizes is shown in Figure 4. If a dedicated half-pel interpolation datapath is used for each block size, the half-pel interpolation datapaths would be idle during half-pel search. The proposed hardware, therefore, has a novel half-pel interpolation datapath that is shared by 4x4, 4x8 and 8x4 block sizes. During the half-pel search of 4x4 blocks, the half-pel interpolation datapath is used for half-pel interpolation of 4x8 and 8x4 blocks. This reduces the area of the half-pel interpolation hardware significantly without increasing the cycle count.

The half-pel interpolation hardware for 8x8, 8x16, 16x8 and 16x16 block sizes has its own half-pel interpolation datapath and it is similar to half-pel interpolation hardware for 4x4, 4x8 and 8x8 block sizes.

A. Half-Pel Interpolation Flow

The proposed half-pel interpolation flow for a 4x4 block is shown in Figure 5. The half-pel interpolation flows for the other block sizes are similar to this flow. The light gray rectangles denote integer pixels (e.g. 0-0) and dark gray rectangles denote half pixels (e.g. A00, B00, C00). The integer-pel MV for this 4x4 block points to the integer pixel 3-3. The half-pel search locations around this integer pixel (C00, A03, C01, B01, C11, A13, C10, B00) have to be searched to determine the best half-pel accurate MV. Therefore, the top-left corner of the half-pel search window is C00 and the bottom-right corner is C44. There are 9x9 (integer and half pixels) – 4x4 (integer pixels) = 65 half pixels in the half-pel search window and 100 integer pixels (0-0 to 9-9) are required to calculate these half pixels. The integer pixels are stored in the input register file and the half pixels in the half-pel search window are stored in the search register file.

The half pixels are grouped according to their calculation order; first set A half pixels, then set B half pixels and finally set C half pixels are calculated. Set A half pixels are interpolated from 6 corresponding vertical integer pixels. For example, A00 is interpolated from integer pixels 0-0, 1-0, 2-0, 3-0, 4-0, 5-0 and A10 is interpolated from integer pixels 1-0, 2-0, 3-0, 4-0, 5-0, 6-0. Therefore, the first column of set A half pixels (A00 to A40) are calculated using the first column of integer pixels (0-0 to 9-0). The remaining set A half pixels are calculated similarly.

Set B half pixels are interpolated from 6 corresponding horizontal integer pixels. For example, B00 is interpolated from integer pixels 3-0, 3-1, 3-2, 3-3, 3-4, 3-5 and B01 is interpolated from integer pixels 3-1, 3-2, 3-3, 3-4, 3-5, 3-6. Therefore, the first row of set B half pixels (B00 to B04) are calculated using the fourth row of integer pixels (3-0 to 3-9). The remaining set B half pixels are calculated similarly.

Finally, set C half pixels are interpolated from 6 corresponding horizontal set A half pixels. Therefore, in addition to the set A half pixels that are in the search window, the set A half pixels that are required to calculate set C half pixels are also calculated and stored in temporary register file. For example, C00 is interpolated from set A half pixels A00, A01, A02, A03, A04, A05 and C01 is interpolated from set A half pixels A01, A02, A03, A04, A05, A06. Therefore, the first row of set C half pixels (C00 to C04) are calculated using the first row of set A half pixels (A00 to A09). The remaining set C half pixels are calculated similarly.

B. Half-Pel Interpolation Datapath

The proposed half-pel interpolation datapath is shown in Figure 6. The datapath implements the 6-tap FIR filter round ((A-5B+20C+20D-5E+F) / 32). It takes 6 input pixels and calculates the corresponding half pixel. The datapath is pipelined into 2 stages using pipeline registers (P Reg) to increase the clock frequency and interpolation throughput. The multiplications with coefficients 5 and 20 are implemented with shifters and adders instead of multipliers to reduce area. For example, 5X is calculated by 2-bit left shift (4X) and addition (4X+X). The output of the datapath is clipped to range [0-255].

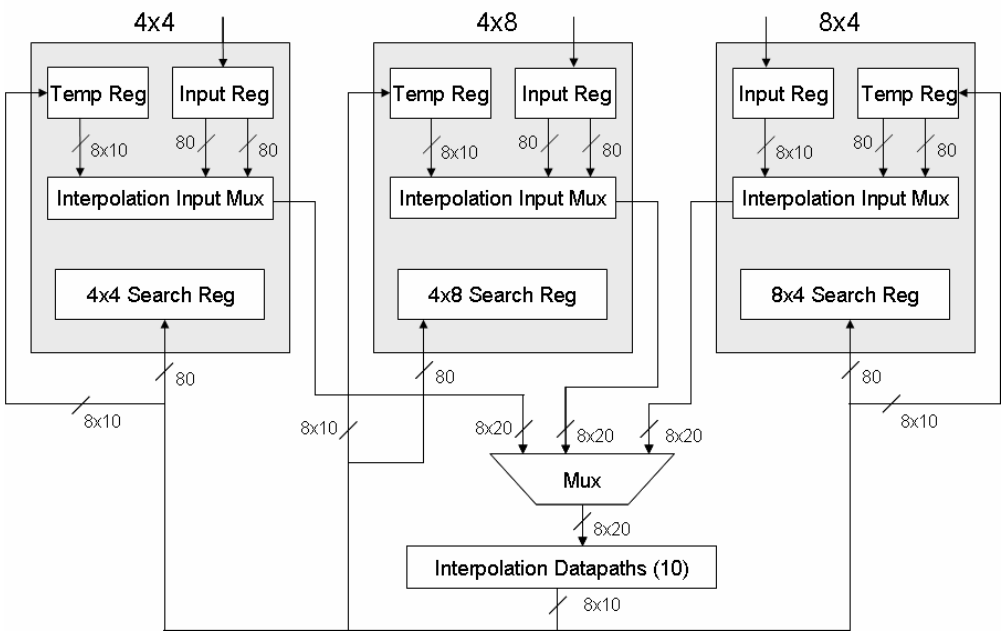

Figure 4. Half-Pel Interpolation Hardware

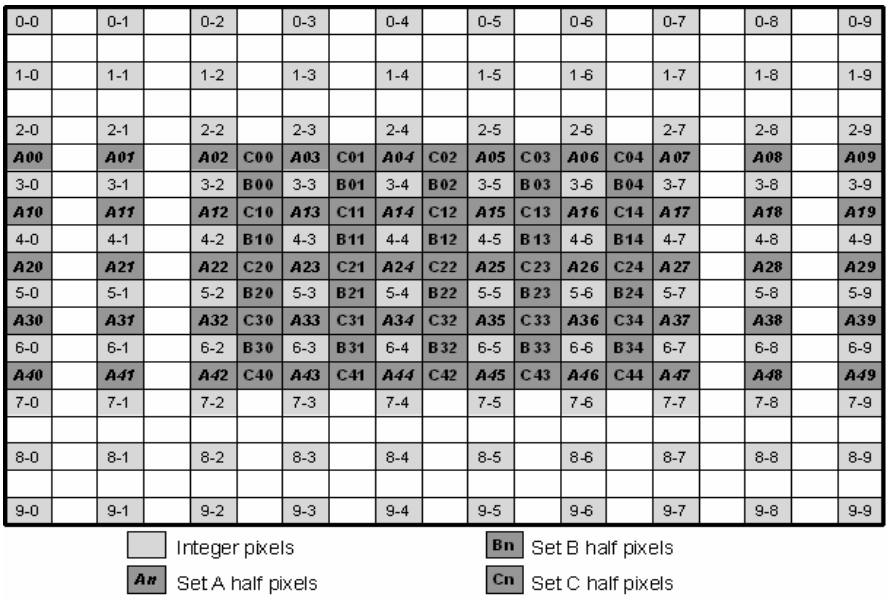

Figure 5. 4x4 Half-Pel Interpolation Flow

Since one set A half pixel is interpolated from 6 integer pixels, if we use 1 half-pel datapath, 5 set A half pixels will be interpolated in 5 clock cycles by accessing 30 integer pixels. Since one set A half pixel is interpolated from 6 integer pixels, if we use 1 half-pel datapath, 5 set A half pixels will be interpolated in 5 clock cycles by accessing 30 integer pixels. However, since one column of set A half pixels (5 pixels) can be calculated using one column of integer pixels (10 pixels), if we use 5 half-pel interpolation datapaths, 5 set A half pixels can be interpolated in 1 clock cycle by accessing 10 integer pixels. This reduces the number of input register file accesses by 3 and the number of clock cycles by 5. We used 10 half-pel interpolation datapaths to further reduce the clock cycle count. Therefore, two columns of set A half pixels (10 pixels) are calculated in 1 clock cycle by accessing 20 integer pixels.

Similarly, two rows of set B half pixels (10 pixels) are calculated in 1 clock cycle by accessing 20 integer pixels. However, since set C half pixels are interpolated from set A half pixels and accessing two rows of set A half pixels in 1 clock cycle increases the complexity of the register files, one row of set C half pixels (5 pixels) are calculated in 1 clock cycle by accessing one row of set A half pixels (10 pixels).

The half-pel interpolation hardware, therefore, calculates set A half pixels in 5 clock cycles, set B half pixels in 2 clock cycles and set C half pixels in 5 clock cycles. The half-pel interpolation for a 4x4 block, therefore, takes 12 clock cycles.

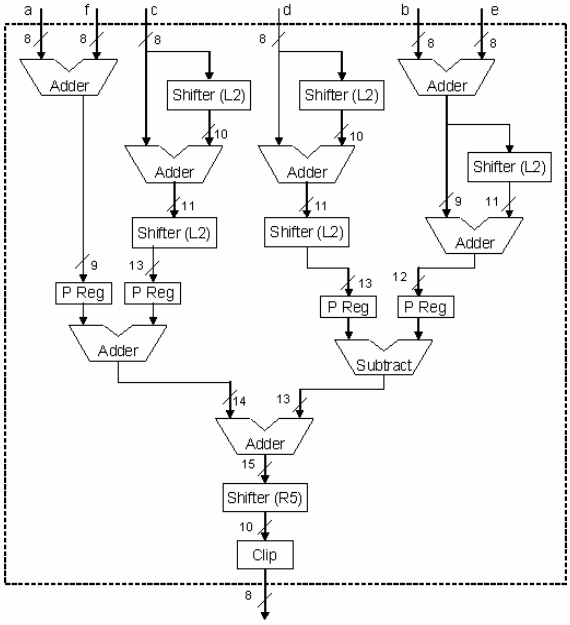

Figure 6. Half-Pel Interpolation Datapath

Since the half-pel interpolation datapaths access 20 integer pixels and produce 10 half pixels in 1 clock cycle, the input register files should have 20 read ports and search register files should have 10 write ports. However, in Xilinx Virtex II FPGAs, register files with more than two read and one write ports use very large area and distributed RAMs have two read and one write ports. Therefore, in order to use distributed RAMs, we used register files with a word length of 10 pixels (80 bits).

This requires organizing the half-pixels in the search register file such that 10 half pixels produced by half-pel interpolation hardware in one cycle can be written as one word to the search register file and the half pixels can be easily accessed by the half-pel search hardware. The proposed layout of the half-pixels in the 4x4 search register file is shown Figure 7. The rectangles labeled as 4-0, 0-0, 4-1, 2-0, X, 2-1, 4-5, 0-1, 4-6 correspond to C00, A03, C01, B00, 3-3, B01, C10, A13, C11 in Figure 5. Therefore, the half-pel search window in the 4x4 search register file corresponds to half-pel search window starting with C00 and ending with C44 in Figure 5.

As shown in Figure 7, first two columns of set A half pixels are written as an 80-bit word to address 0 and the other two columns of set A half pixels are written as an 80-bit word to address 1. Similarly, first two rows of set B half pixels are written as an 80-bit word to address 2 and the other two rows of set B half pixels are written as an 80-bit word to address 3. Similarly, set C half pixels are written to addresses 4, 5 and 6.

IV. PROPOSED HALF-PEL SEARCH HARDWARE

The proposed half-pel accurate variable block size motion estimation hardware has dedicated half-pel search hardware for each block size in order to perform the half-pel search faster. Each half-pel search hardware has 4 PEs. Since there are 7 block sizes, 28 PEs are used in the half-pel motion estimation hardware.

Half-pel search hardware for 4x4 block size is shown in Figure 8. The half-pel search hardwares for other block sizes are similar to this hardware. The SAD value for a search location is calculated by a processing element (PE) in 16 clock cycles. Since

4-0	0-0	4-1	0-5	4-2	1-0	4-3	1-5	4-4
2-0	X	2-1		2-2		2-3		2-4
4-5	0-1	4-6	0-6	4-7	1-1	4-8	1-6	4-9
2-5		2-6		2-7		2-8		2-9
5-0	0-2	5-1	0-7	5-2	1-2	5-3	1-7	5-4
3-0		3-1		3-2		3-3		3-4
5-5	0-3	5-6	0-8	5-7	1-3	5-8	1-8	5-9
3-5		3-6		3-7		3-8		3-9
6-0	0-4	6-1	0-9	6-2	1-4	6-3	1-9	6-4

Set A half pixels (Address 0 and 1)
Set B half pixels (Address 2 and 3)
Set C half pixels (Address 4, 5 and 6)

Figure 7. 4x4 Search Register File

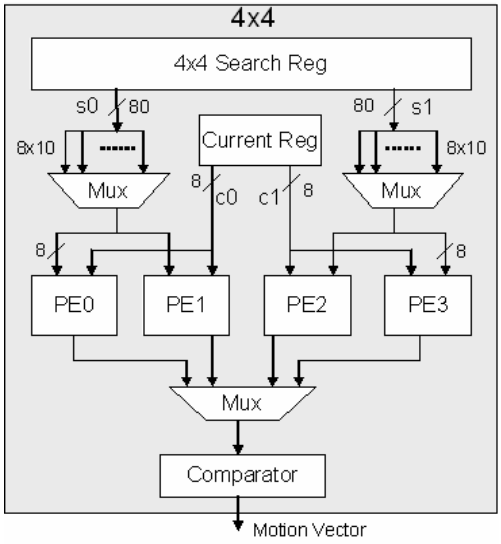

Figure 8. 4x4 Half-Pel Search Hardware

there are 8 half-pel search locations, half-pel search would take 8*16=128 clock cycles using one PE. We used 4 PEs in order to perform the half-pel search operation faster. Each PE calculates the SAD for two half-pel search locations in 2*16=32 clock cycles. The SADs calculated by PEs are sent to a comparator, and the comparator determines the minimum SAD and the corresponding best half-pel accurate MV.

The proposed half-pel search flow for a 4x4 block is shown in Figure 9. The half-pel search flows for the other block sizes are similar to this flow. The calculations done by each PE in this flow is organized to reduce the number of search register file and current register file read ports. The search register file has two 80-bit read ports, s0 and s1. During half-pel search, the proper pixel (8 bits) is selected by a multiplexer in each cycle among 10 pixels (80 bits) and sent to proper PEs. The current register file has two 8-bit read ports, c0 and c1. PE0 and PE1 use s0 and c0 ports, PE2 and PE3 use s1 and c1 ports.

The half-pel search locations are allocated to PEs as follows. First, 4 search locations that use set C half pixels are searched. PE0 calculates the SAD for the 4x4 block starting with pixel stored in address 4-0 and ending with pixel stored in address 5-8, PE1 calculates the SAD for the 4x4 block starting with 4-1 and ending with 5-9, PE2 calculates the SAD for the 4x4 block starting

Cycle	c0	c1	s0	s1	PE0	PE1	PE2	PE3
0	0	0	4	4	c0-s40	c0-s41	c0-s45	c0-s46
1	1	1	4	4	c1-s41	c1-s42	c1-s46	c1-s47
2	2	2	4	4	c2-s42	c2-s43	c2-s47	c2-s48
3	3	3	4	4	c3-s43	c3-s44	c3-s48	c3-s49
4	4	4	4	5	c4-s45	c4-s46	c4-s50	c4-s51
5	5	5	4	5	c5-s46	c5-s47	c5-s51	c5-s52
6	6	6	4	5	c6-s47	c6-s48	c6-s52	c6-s53
7	7	7	4	5	c7-s48	c7-s49	c7-s53	c7-s54
8	8	8	5	5	c8-s50	c8-s51	c8-s55	c8-s56
9	9	9	5	5	c9-s51	c9-s52	c9-s56	c9-s57
10	10	10	5	5	c10-s52	c10-s53	c10-s57	c10-s58
11	11	11	5	5	c11-s53	c11-s54	c11-s58	c11-s59
12	12	12	5	6	c12-s55	c12-s56	c12-s60	c12-s61
13	13	13	5	6	c13-s56	c13-s57	c13-s61	c13-s62
14	14	14	5	6	c14-s57	c14-s58	c14-s62	c14-s63
15	15	15	5	6	c15-s58	c15-s59	c15-s63	c15-s64
16	0	1	2	0	c0-s20	c0-s21	c1-s05	c1-s06
17	1	5	2	0	c1-s21	c1-s22	c5-s06	c5-s07
18	2	9	2	0	c2-s22	c2-s23	c9-s07	c9-s08
19	3	13	2	0	c3-s23	c3-s24	c13-s08	c13-s09
20	4	0	2	0	c4-s25	c4-s26	c0-s00	c0-s01
21	5	4	2	0	c5-s26	c5-s27	c4-s01	c4-s02
22	6	8	2	0	c6-s27	c6-s28	c8-s02	c8-s03
23	7	12	2	0	c7-s28	c7-s29	c12-s03	c12-s04
24	8	3	3	1	c8-s30	c8-s31	c3-s15	c3-s16
25	9	7	3	1	c9-s31	c9-s32	c7-s16	c7-s17
26	10	11	3	1	c10-s32	c10-s33	c11-s17	c11-s18
27	11	15	3	1	c11-s33	c11-s34	c15-s18	c15-s19
28	12	2	3	1	c12-s35	c12-s36	c2-s10	c2-s11
29	13	6	3	1	c13-s36	c13-s37	c6-s11	c6-s12
30	14	10	3	1	c14-s37	c14-s38	c10-s12	c10-s13
31	15	14	3	1	c15-s38	c15-s39	c14-s13	c14-s14

Figure 9. 4x4 Half-Pel Search Flow

with 4-5 and ending with 6-3 and PE3 calculates the SAD for the 4x4 block starting with 4-6 and ending with 6-4. During these calculations, port s0 provides the pixels stored in addresses 4 and 5 and these 20 pixels are used by PE0 and PE1. Port s1 provides the pixels stored in addresses 4, 5 and 6 and proper 20 pixels are used by PE2 and PE3.

Then, 2 search locations that use set A half pixels and 2 search locations that use set B half pixels are searched. PE0 and PE1 searches the two set B search locations and PE2 and PE3 searches the two set A search locations. PE0 calculates the SAD for the 4x4 block starting with pixel stored in address 2-0 and ending with pixel stored in address 3-8, PE1 calculates the SAD for the 4x4 block starting with 2-1 and ending with 3-9, PE2 calculates the SAD for the 4x4 block starting with 0-0 and ending with 1-8, PE3 calculates the SAD for the 4x4 block starting with 0-1 and ending with 1-9. During these calculations, port s0 provides the pixels stored in addresses 2 and 3 (set B) and these 20 pixels are used by PE0 and PE1. Port s1 provides the pixels stored in addresses 0 and 1 (set A) and these 20 pixels are used by PE2 and PE3.

V. IMPLEMENTATION RESULTS

The proposed half-pel motion estimation hardware for 4x4, 4x8 and 8x4 block sizes is implemented in Verilog HDL. The half-pel motion estimation for a 4x4 block takes 48 clock cycles; half-pel interpolation takes 14 clock cycles (2 cycles register file access and pipeline delays) and half-pel search takes 34 clock cycles (2 cycles register file access and pipeline delays). Since there are 16 4x4 blocks in a MB, half-pel motion estimation for a MB for 4x4 block size takes 16*48=768 clock cycles. The half-pel motion estimation for an 8x4 block takes 25+65=90 clock cycles. Since there are 8 8x4 blocks in a MB, half-pel motion estimation for a MB for 8x4 block size takes 8*90=720 clock cycles. Similarly, half-pel motion estimation for a MB for 4x8 block size takes 720 clock cycles. Therefore, half-pel motion estimation for a MB for 4x4, 4x8 and 8x4 block sizes take 768 clock cycles and the 4x4 block size is the bottleneck.

The implementation is verified with RTL simulations using Mentor Graphics ModelSim. 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 7.1.

The FPGA implementation is verified to work at 85 MHz under worst-case PVT conditions with post place and route simulations. The FPGA implementation can process an HDTV frame in 32.6 msec. (3600 MB * 768 cycles per MB * 11.8 ns clock cycle = 32.6 msec). Therefore, it can process 1000/32.6 = 30 HDTV frames (1280x720) per second.

The FPGA implementation including the distributed RAMs uses the following FPGA resources; 5627 Function Generators, 2814 CLB Slices, 3546 Dffs/Latches, i.e. %6 of Function Generators, %6 of CLB Slices, %3.7 of Dffs/Latches.

VI. CONCLUSIONS

In this paper, we presented a high performance and low cost hardware architecture for real-time implementation of half-pel accurate variable block size motion estimation for H.264 / MPEG4 Part 10 video coding. The proposed architecture includes a novel half-pel interpolation hardware that is shared by novel half-pel search hardwares designed for each block size. This half-pel accurate motion estimation 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 85 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 30 HDTV frames (1280x720) per second.

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] H. Mahdavi-Nasab, S. Kasaei, "Half-Pixel Accuracy Block Matching Motion Estimation Algorithms for Low Bitrate Video Communications", *Proc. IEEE Int Conf. on Internet*, 2005

[5] M. Rehan, P. Agathoklis, "Half-pixel accurate motion estimation using a flexible triangle search", *Proc. IEEE Conf. on Comm., Comp. and Signal Proc.*, pp. 257–260, 2005

[6] T. Dias, N. Roma, L. Sousa, "Efficient motion vector refinement architecture for sub-pixel motion estimation systems", *Proc. IEEE Workshop on Signal Proc. System Design and Implementation*, pp. 313-318, November 2005

[7] S. Yalcin, H. Ates, I. Hamzaoglu, "A High Performance Hardware Architecture for an SAD Reuse based Hierarchical Motion Estimation Algorithm for H.264 Video Coding", *Int. Conf. on Field Programmable Logic and Applications*, pp. 509–514, August 2005