An Efficient Hardware Architecture for Quarter-Pixel Accurate H.264 Motion Estimation

Serkan Oktem and Ilker Hamzaoglu

Faculty of Engineering and Natural Sciences, Sabanci University
34956, Tuzla, Istanbul, Turkey
oktem@su.sabanciuniv.edu, hamzaoglu@sabanciuniv.edu

Abstract

In this paper, we present an efficient hardware architecture for real-time implementation of quarter-pixel accurate variable block size motion estimation for H.264 / MPEG4 Part 10 video coding. The proposed hardware performs quarter-pixel interpolation dynamically, i.e. only the quarter pixels necessary for performing quarter-pixel accurate search at the location pointed by the half-pixel motion vector are calculated. This reduces the amount of computation performed for quarter-pixel interpolation, and therefore reduces the power consumption of the quarter-pixel accurate motion estimation hardware. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed hardware architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 60 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 34 VGA frames (640x480) 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 systems an inevitable part of many commercial products. To improve the performance of video compression systems, recently, H.264 / MPEG4 Part 10 video compression standard, offering significantly better video compression efficiency than previous video compression standards, is developed with the collobaration of ITU and ISO standardization organizations.

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 motion estimation algorithm used in the baseline profile of H.264 standard [1, 2]. Motion Estimation (ME) is the most computationally demanding part of the encoders implementing the previous video compression standards. Variable block size ME achieves better coding results than the fixed block size ME used in the previous video compression standards. However, the amount of computation required by variable block size ME is even more than the amount required by fixed block size ME.

In order to increase the performance of integer-pel ME, sub-pel (half-pel and quarter-pel) accurate variable block size

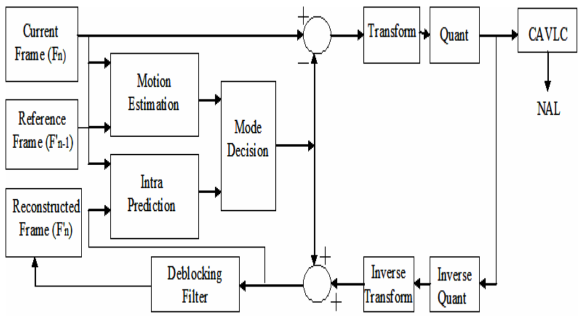


Figure 1. H.264 Encoder Block Diagram

ME is performed [1, 2]. However, the amount of computation required by sub-pel ME is even more than the amount required by integer-pel ME. It is shown that sub-pel ME accounts for about 68% of CPU usage of a H.264 video encoder in fast motion estimation mode which is already 65% faster than full search mode [3]. Therefore, the coding gain obtained by sub-pel ME comes with an increase in encoding complexity which makes it an exciting challenge to have a real-time implementation of sub-pel accurate variable block size ME for H.264 video coding.

In this paper, we present an efficient hardware architecture for real-time implementation of quarter-pel accurate variable block size ME for H.264 video coding. The proposed hardware performs quarter-pixel interpolation dynamically, i.e. only the quarter pixels necessary for performing quarter-pixel accurate search at the location pointed by the best half-pixel motion vector are calculated, for reducing the amount of computation performed for quarter-pixel interpolation and therefore reducing the power consumption.

The proposed quarter-pel accurate ME hardware is designed to be used as part of a complete H.264 video coding system for portable applications together with the half-pel accurate ME hardware presented in [4]. The proposed hardware architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 60 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 34 VGA frames (640x480) per second.

Several hardware architectures for real-time implementation of sub-pel accurate variable block size ME for H.264 video coding are presented in the literature [5, 6]. The hardware architecture presented in [5] uses less hardware than our hardware design and has lower performance than our hardware design. The hardware architecture presented in [6] achieves higher performance than our hardware design at the expense of a much higher hardware cost. It uses much more FIR filters (64 vs. 28) and processing elements (144 vs. 56) than our hardware design in order to process 30 HDTV frames (1280x720) per second. Our hardware design is a more cost-effective solution for portable applications.

The rest of the paper is organized as follows. Section 2 explains the quarter-pel accurate motion estimation algorithm. Section 3 describes the proposed architecture in detail. The implementation results are given in section 4. Finally, section 5 presents the conclusions.

2. Overview of Quarter-Pel Accurate Motion Estimation Algorithm

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

Before searching for the best HP accurate MV, half pixels in the HP 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 HP 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 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, HP 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.

Before searching for the best QP accurate MV, quarter pixels in the QP search window are interpolated from neighboring pixels using a bilinear filter. A QP interpolation example is shown in Figure 3. For example, quarter pixel cc is interpolated from the integer pixel C and half pixel c (cc = (C+c+1)>>1), quarter pixel cn is interpolated from the half pixels c and n (cn = (c+n+1)>>1), and quarter pixel cj is interpolated from the half pixels c and j (cj = (c+j +1)>>1).

3. Proposed Quarter-Pel Accurate Motion Estimation Hardware

The proposed QP accurate ME hardware for 4x4 block size is shown in Figure 4. The QP ME hardwares for other block sizes are similar to this hardware. For each 4x4 block in a MB, first, HP ME hardware finds the best HP MV by performing

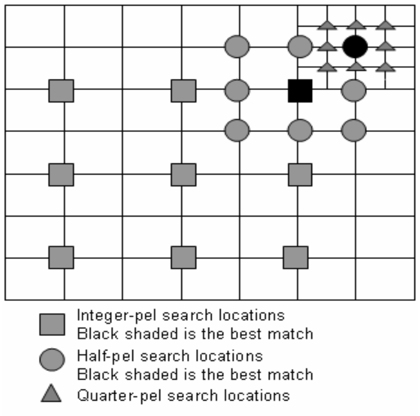

Integer-pel search locations
Black shaded is the best match
Half-pel search locations
Black shaded is the best match
Quarter-pel search locations

Figure 2. Half-Pel and Quarter-Pel Search Locations

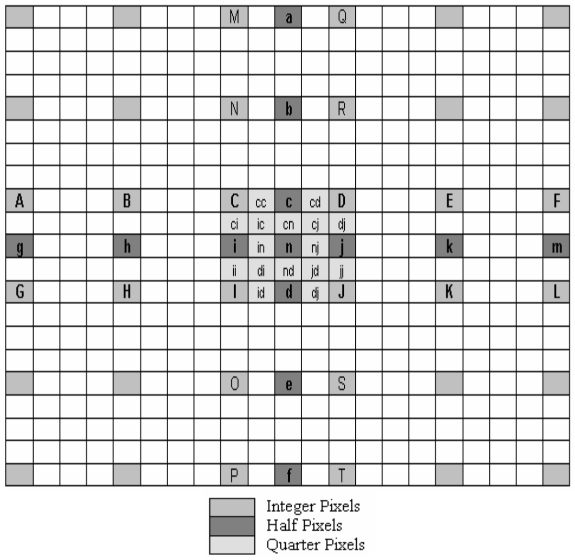


Figure 3. Half-Pel and Quarter-Pel Interpolation Example

half-pel interpolation (HPI) and half-pel search (HPS) and sends this HP MV to QP ME hardware. Then, QP ME hardware finds the best QP MV for that 4x4 block by performing quarter-pel search (QPS) around the location pointed by this HP MV with a search range of [-1, 1].

As the HP ME hardware is performing HPI and HPS, the integer and half pixels necessary for QP accurate ME are send to the search window register file (SWRF) by the HP ME hardware. The proposed layout of the integer and half pixels in the 4x4 SWRF, when the location pointed by the best integer-pel MV is location 17, is shown Figure 5.

Since the HP ME will be performed at the HPS locations 8, 9, 10, 16, 18, 24, 25 and 26, the best HP MV will point to one of these locations and the QP ME will be performed at the eight QPS locations around that location. For example, if the best HP MV points to location 8, QP ME will be performed at the QPS locations 8_1, 8_2, 8_3, 8_4, 8_5, 8_6, 8_7 and 8_8.

The control unit sends the read addresses to SWRF based on the best HP MV for accessing the necessary integer and half pixels. Since there are eight HPS locations and there are eight

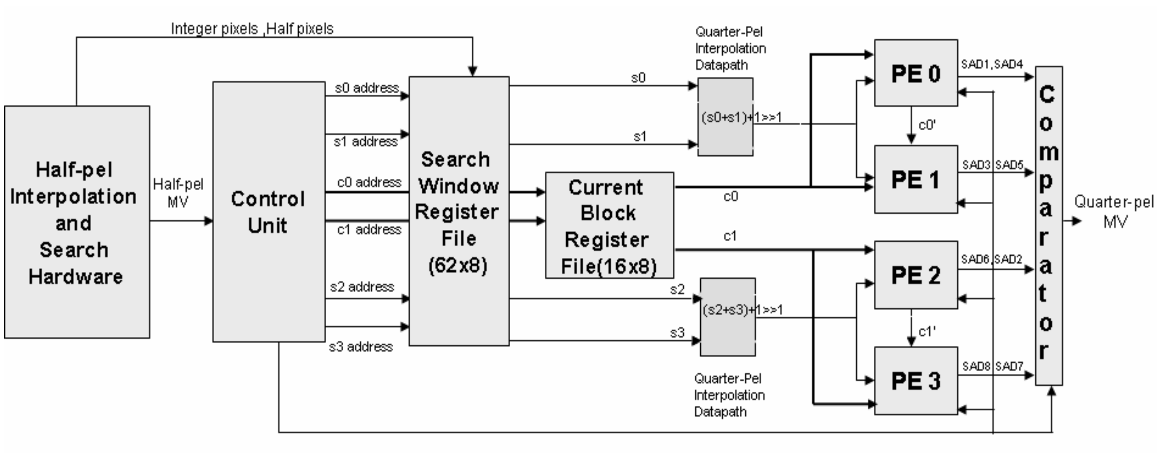

Integer pixels ,Half pixels
Half-pel Interpolation and Search Hardware
Half-pel MV
Control Unit
s0 address
s1 address
c0 address
c1 address
s2 address
s3 address
Search Window Register File (62x8)
Current Block Register File(16x8)
s0
s1
c0
c1
s2
s3
Quarter-Pel Interpolation Datapath
(s0+s1)+1>>1
(s2+s3)+1>>1
Quarter-Pel Interpolation Datapath
PE 0
PE 1
PE 2
PE 3
c0'
c1'
SAD1,SAD4
SAD3 SAD5
SAD6 SAD2
SAD8 SAD7
Comparator
Quarter-pel MV

Figure 4. Proposed Quarter-Pel Accurate Motion Estimation Hardware

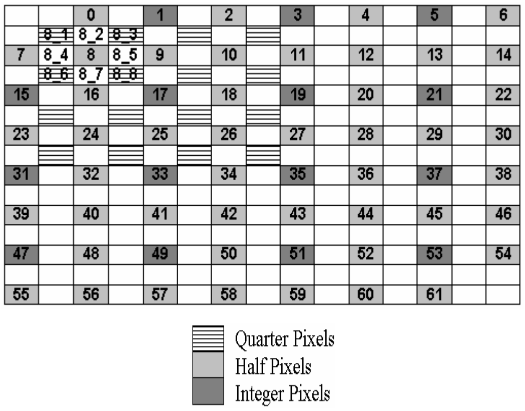


Figure 5. Search Window Register File

QPS locations for each HPS location, the control unit must be able to generate read addresses for 64 QPS locations (8_1, 8_2, 8_3, … , 26_6, 26_7, 26_8). The QPI datapaths generate the quarter pixels and send them to processing elements (PE). The SAD values for QPS locations are calculated by the processing elements PE0, PE1, PE2 and PE3. The quarter pixels necessary for calculating the SAD value for the QPS location 8_1 are shown in Figure 5.

The proposed layout of the integer and half pixels in the 4x4 SWRF provide a good correlation between the read addresses of 64 QPS locations. The read address correlations of 64 QPS locations are shown in Figure 6. For example, the read addresses of the integer and half pixels used for generating the quarter pixels necessary for QPS location 8_8 are 9 more than the read addresses of the integer and half pixels used for generating the quarter pixels necessary for QPS location 8_1. In Figure 6, this read address correlation between QPS locations 8_1 and 8_8 is shown by writing the read address for location 8_8 as 8_1 + 9. The read address correlations of 64 QPS locations are similarly shown in Figure 6. Therefore, the control unit generates the read addresses of 64 QPS locations by using the read addresses of the QPS locations 8_1, 8_2, 8_3, 8_4 and the read address correlations of 64 QPS locations.

The SAD value for a QPS location is calculated by a PE in 16 clock cycles. Since there are 8 QPS locations, QPS would take 8*16=128 clock cycles using one PE. We used 4 PEs in order to perform the QPS operation faster. Each PE calculates the SAD for two QPS locations. The SADs calculated by PEs are sent to a comparator, and the comparator determines the minimum SAD and the corresponding best QP accurate MV.

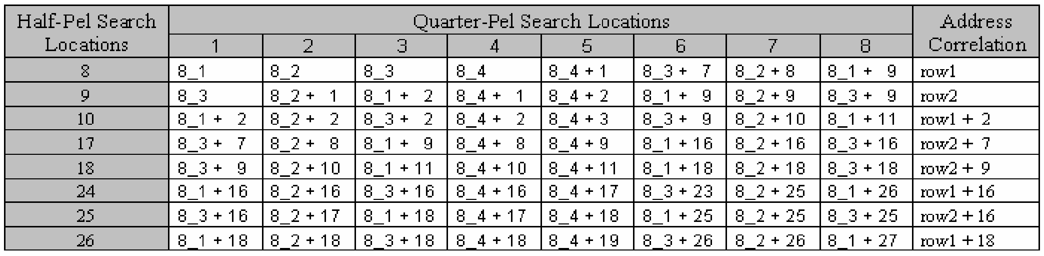
The proposed QP interpolation and search flow for a 4x4 block is shown in Figure 7. The QP interpolation and search flows for the other block sizes are similar to this flow. The calculations done by each PE in this flow are organized to reduce the number of read ports of the search window and current block register files and to reduce the number of read accesses to these register files.

Because of the proposed allocation of QPS locations to PEs and the proposed flow, the SWRF has four 8-bit read ports (s0, s1, s2 and s3), and the current block register file has two 8-bit read ports (c0 and c1). PE0 and PE1 use s0, s1 and c0 ports, PE2 and PE3 use s2, s3 and c1 ports. PE1 can reuse the current block pixel accessed by PE0 in a previous clock cycle (c0'). Similarly, PE3 can reuse the current block pixel accessed by PE2 in a previous clock cycle (c1'). In addition, PE0 and PE1 can use the same search window pixels in the same clock cycle. Similarly, PE2 and PE3 can use the same search window pixels in the same clock cycle. In order to achieve these, PEs do not perform any calculation in some clock cycles.

4. Implementation Results

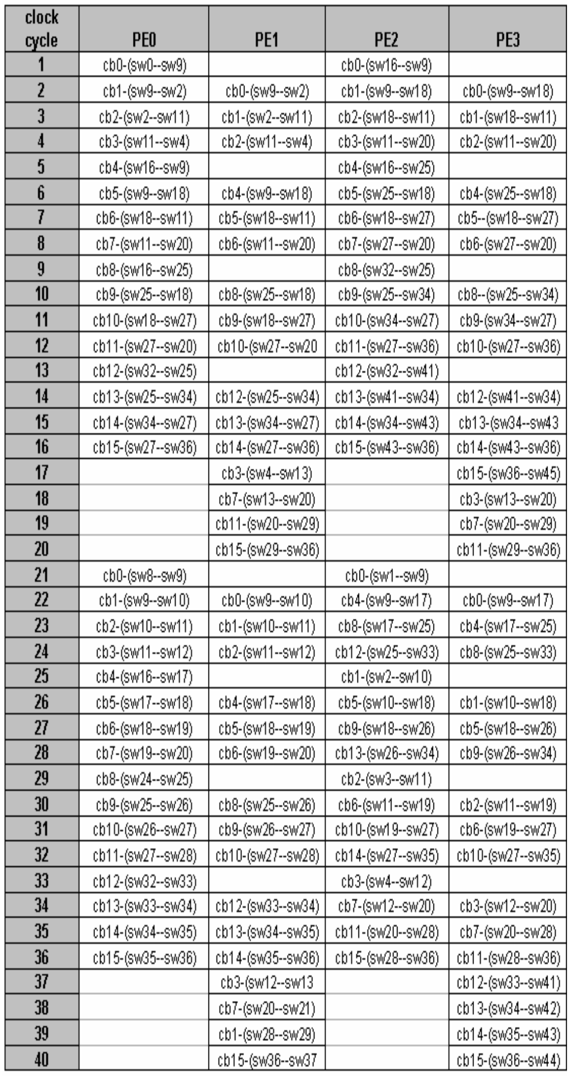
The proposed QP ME hardware is implemented in Verilog HDL. QP interpolation and search take 44 clock cycles for a 4x4 block. Since there are 16 4x4 blocks in a MB, QP ME for a MB for 4x4 block size takes 16*44 = 704 clock cycles. QP interpolation and search for an 8x4 block size take 80 clock cycles. Since there are 8 8x4 blocks in a MB, QP ME for a MB for 8x4 block size takes 8*80 = 640 clock cycles. Similarly, QP ME for a MB for 4x8, 8x8, 16x8, 8x16 and 16x16 block sizes take 608, 576, 576, 560 and 544 clock cycles respectively. Therefore, 4x4 block size is the bottleneck.

The HP interpolation and search take 48 clock cycles for a 4x4 block and 4x4 block size is the bottleneck for HP accurate ME hardware as well [4]. Therefore, sub-pel ME for a 4x4 block takes 48+44 = 92 clock cycles and sub-pel ME for a MB takes 16*92=1472 clock cycles.



Half-Pel Search Locations	Quarter-Pel Search Locations								Address Correlation
	1	2	3	4	5	6	7	8	
8	8_1	8_2	8_3	8_4	8_4 + 1	8_3 + 7	8_2 + 8	8_1 + 9	row1
9	8_3	8_2 + 1	8_1 + 2	8_4 + 1	8_4 + 2	8_1 + 9	8_2 + 9	8_3 + 9	row2
10	8_1 + 2	8_2 + 2	8_3 + 2	8_4 + 2	8_4 + 3	8_3 + 9	8_2 + 10	8_1 + 11	row1 + 2
17	8_3 + 7	8_2 + 8	8_1 + 9	8_4 + 8	8_4 + 9	8_1 + 16	8_2 + 16	8_3 + 16	row2 + 7
18	8_3 + 9	8_2 + 10	8_1 + 11	8_4 + 10	8_4 + 11	8_1 + 18	8_2 + 18	8_3 + 18	row2 + 9
24	8_1 + 16	8_2 + 16	8_3 + 16	8_4 + 16	8_4 + 17	8_3 + 23	8_2 + 25	8_1 + 26	row1 + 16
25	8_3 + 16	8_2 + 17	8_1 + 18	8_4 + 17	8_4 + 18	8_1 + 25	8_2 + 25	8_3 + 25	row2 + 16
26	8_1 + 18	8_2 + 18	8_3 + 18	8_4 + 18	8_4 + 19	8_3 + 26	8_2 + 26	8_1 + 27	row1 + 18

Figure 6. Address Correlation of Quarter-Pel Search Locations



clock cycle	PE0	PE1	PE2	PE3
1	cb0-(sw0--sw9)		cb0-(sw16--sw9)	
2	cb1-(sw9--sw2)	cb0-(sw9--sw2)	cb1-(sw9--sw18)	cb0-(sw9--sw18)
3	cb2-(sw2--sw11)	cb1-(sw2--sw11)	cb2-(sw18--sw11)	cb1-(sw18--sw11)
4	cb3-(sw11--sw4)	cb2-(sw11--sw4)	cb3-(sw11--sw20)	cb2-(sw11--sw20)
5	cb4-(sw16--sw9)		cb4-(sw16--sw25)	
6	cb5-(sw9--sw18)	cb4-(sw9--sw18)	cb5-(sw25--sw18)	cb4-(sw25--sw18)
7	cb6-(sw18--sw11)	cb5-(sw18--sw11)	cb6-(sw18--sw27)	cb5--(sw18--sw27)
8	cb7-(sw11--sw20)	cb6-(sw11--sw20)	cb7-(sw27--sw20)	cb6-(sw27--sw20)
9	cb8-(sw16--sw25)		cb8-(sw32--sw25)	
10	cb9-(sw25--sw18)	cb8-(sw25--sw18)	cb9-(sw25--sw34)	cb8--(sw25--sw34)
11	cb10-(sw18--sw27)	cb9-(sw18--sw27)	cb10-(sw34--sw27)	cb9-(sw34--sw27)
12	cb11-(sw27--sw20)	cb10-(sw27--sw20	cb11-(sw27--sw36)	cb10-(sw27--sw36)
13	cb12-(sw32--sw25)		cb12-(sw32--sw41)	
14	cb13-(sw25--sw34)	cb12-(sw25--sw34)	cb13-(sw41--sw34)	cb12-(sw41--sw34)
15	cb14-(sw34--sw27)	cb13-(sw34--sw27)	cb14-(sw34--sw43)	cb13-(sw34--sw43
16	cb15-(sw27--sw36)	cb14-(sw27--sw36)	cb15-(sw43--sw36)	cb14-(sw43--sw36)
17		cb3-(sw4--sw13)		cb15-(sw36--sw45)
18		cb7-(sw13--sw20)		cb3-(sw13--sw20)
19		cb11-(sw20--sw29)		cb7-(sw20--sw29)
20		cb15-(sw29--sw36)		cb11-(sw29--sw36)
21	cb0-(sw8--sw9)		cb0-(sw1--sw9)	
22	cb1-(sw9--sw10)	cb0-(sw9--sw10)	cb4-(sw9--sw17)	cb0-(sw9--sw17)
23	cb2-(sw10--sw11)	cb1-(sw10--sw11)	cb8-(sw17--sw25)	cb4-(sw17--sw25)
24	cb3-(sw11--sw12)	cb2-(sw11--sw12)	cb12-(sw25--sw33)	cb8-(sw25--sw33)
25	cb4-(sw16--sw17)		cb1-(sw2--sw10)	
26	cb5-(sw17--sw18)	cb4-(sw17--sw18)	cb5-(sw10--sw18)	cb1-(sw10--sw18)
27	cb6-(sw18--sw19)	cb5-(sw18--sw19)	cb9-(sw18--sw26)	cb5-(sw18--sw26)
28	cb7-(sw19--sw20)	cb6-(sw19--sw20)	cb13-(sw26--sw34)	cb9-(sw26--sw34)
29	cb8-(sw24--sw25)		cb2-(sw3--sw11)	
30	cb9-(sw25--sw26)	cb8-(sw25--sw26)	cb6-(sw11--sw19)	cb2-(sw11--sw19)
31	cb10-(sw26--sw27)	cb9-(sw26--sw27)	cb10-(sw19--sw27)	cb6-(sw19--sw27)
32	cb11-(sw27--sw28)	cb10-(sw27--sw28)	cb14-(sw27--sw35)	cb10-(sw27--sw35)
33	cb12-(sw32--sw33)		cb3-(sw4--sw12)	
34	cb13-(sw33--sw34)	cb12-(sw33--sw34)	cb7-(sw12--sw20)	cb3-(sw12--sw20)
35	cb14-(sw34--sw35)	cb13-(sw34--sw35)	cb11-(sw20--sw28)	cb7-(sw20--sw28)
36	cb15-(sw35--sw36)	cb14-(sw35--sw36)	cb15-(sw28--sw36)	cb11-(sw28--sw36)
37		cb3-(sw12--sw13		cb12-(sw33--sw41)
38		cb7-(sw20--sw21)		cb13-(sw34--sw42)
39		cb1-(sw28--sw29)		cb14-(sw35--sw43)
40		cb15-(sw36--sw37		cb15-(sw36--sw44)

Figure 7. Quarter-Pel Interpolation and Search Flow

The Verilog HDL implementation of the QP ME hardware 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 6 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 60 MHz under worst-case PVT conditions with post place and route simulations. The FPGA implementation can process a VGA frame in 29.32 msec (1200 MB * 1472 cycles per MB * 16.6 ns clock cycle = 29.32 msec). Therefore, it can process 1000/29.32 = 34 VGA frames (640x480) per second.

The FPGA implementation uses the following FPGA resources; 18566 CLB Slices, 37131 Function Generators and 21339 DFFs, i.e. %39 of CLB Slices, %39 of Function Generators and %22 of DFFs.

5. Conclusion

In this paper, we presented an efficient hardware architecture for real-time implementation of quarter-pixel accurate variable block size ME for H.264 video coding. This quarter-pixel accurate ME hardware is designed to be used as part of a complete H.264 video coding system for portable applications. The proposed hardware architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 60 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can process 34 VGA frames (640x480) 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] K. Minoo, T. Q. Nquyen, "Reverse, Sub-Pixel Block Matching: Applications within H.264 and Analysis of Limitations", *IEEE Int. Conf. on Image Processing,* pp. 3161-3164, 2006.

[4] S. Yalcin, I. Hamzaoglu, "A High Performance Hardware Architecture for Half-Pixel Accurate H.264 Motion Estimation", *14th Int. Conf. on VLSI-SoC,* October 2006.

[5] AT.C.Chen, Y.W.Huang, L.G.Chen, "Fully untilized and reusable architecture for fractional motion estimation of H.264/AVC", *IEEE ICASSP*, pp. 9-12, 2004.

[6] C. Yang, S. Goto, T. Ikenaga, "High performance architecture for fractional motion estimation in H.264 for HDTV", *IEEE ISCAS*, 2006.