An Efficient Hardware Architecture for H.264 Adaptive Deblocking Filter Algorithm

Mustafa Parlak
Sabanci University
Tuzla, Istanbul
34956, Turkey
mparlak@su.sabanciuniv.edu

Ilker Hamzaoglu
Sabanci University
Tuzla, Istanbul
34956, Turkey
hamzaoglu@sabanciuniv.edu

Abstract

This paper presents an efficient hardware architecture for real-time implementation of adaptive deblocking filter algorithm used in H.264 video coding standard. This hardware is designed to be used as part of a complete H.264 video coding system for portable applications. We use a novel edge filter ordering in a Macroblock to prevent the deblocking filter hardware from unnecessarily waiting for the pixels that will be filtered become available. The proposed architecture is implemented in Verilog HDL. The Verilog RTL code is verified to work at 72 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can code 30 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 International standards, is developed with the collaborations 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 adaptive deblocking filter (DBF) algorithm [1, 2, 3].

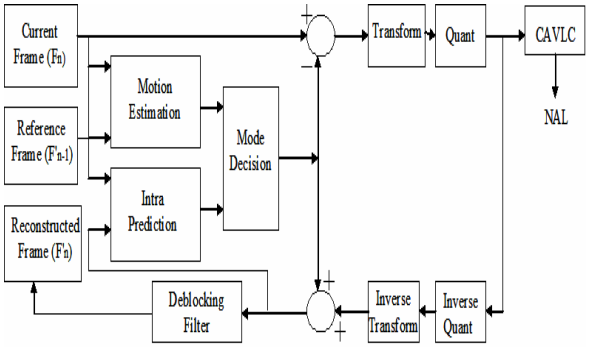


Figure 1 H.264 Encoder Block Diagram

As shown in Figure 1, deblocking filter is applied to each decoded Macroblock (MB) after inverse quantization and inverse transform. Deblocking filter improves the visual quality of decoded frames by reducing the visually disturbing blocking artifacts and discontinuities in a frame due to coarse quantization of MBs and motion compensated prediction [4]. Since the filtered frame is used as a reference frame for motion-compensated prediction of future frames, deblocking filter also increases coding efficiency resulting in bit rate savings [4].

The deblocking filter algorithm used in H.264 standard is more complex than the deblocking filter algorithms used in previous video compression standards. First of all, the H.264 deblocking filter algorithm is highly adaptive. Second, it is applied to each edge of all the 4x4 luma and chroma blocks in a MB. Third, it can update 3 pixels in each direction that the filtering takes place. Fourth, in order to decide whether the deblocking filter will be applied to an edge, the related pixels in the current and neighboring 4x4 blocks must be read from memory and processed. Because of these complexities, the deblocking filter

algorithm can easily account for one-third of the computational complexity of an H.264 decoder [4, 5].

In this paper, we present an efficient hardware architecture for real-time implementation of H.264 adaptive deblocking filter algorithm. 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 72 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can code 30 CIF frames (352x288) per second.

Several hardware architectures for real-time implementation of H.264 adaptive deblocking filter algorithm are presented in the literature [6, 7]. These architectures achieve 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. We achieved real-time performance for portable applications by only using one 12-bit adder, one 12-bit comparator, a few shifters, two's complementers and multiplexers in our datapath.

The rest of the paper is organized as follows. Section II presents a brief overview of adaptive deblocking filter algorithm used in H.264. Section III describes the proposed hardware architecture in detail. The implementation results are given in Section IV. Finally, Section V presents the conclusions.

2. Overview of H.264 Adaptive Deblocking Filter Algorithm

H.264 adaptive deblocking filter removes visually disturbing block boundaries created by coarse quantization of MBs and motion compensated prediction. MBs in a frame are filtered in raster scan order. Filtering is applied to each edge of all the 4x4 luma and chroma blocks in a MB. The 4x4 luma and chroma blocks in a MB are shown in Figure 2. The 4x4 block edges in a MB are filtered in the order specified in the H.264 standard [3]. First, the vertical edges in the MB are filtered in the order a, b, c, d, i and j. Then, the horizontal edges in the MB are filtered in the order h, g, f, e, l and k.

The deblocking filter algorithm for an edge is shown in Figure 3 [3, 4]. This figure clearly shows the adaptive nature of the deblocking filter algorithm. There are several conditions that determine whether a 4x4 block edge will be filtered or not. There are

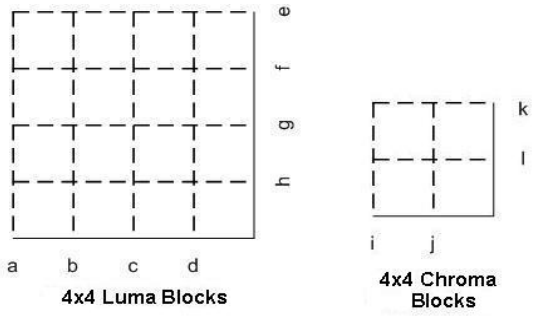


Figure 2 4x4 Blocks in a MB

additional conditions that determine the strength of the filtering for the 4x4 block edges that will be filtered. As shown in the figure, boundary strength (BS) parameter, α and β threshold values and the values of the pixels in the edge determine the outcomes of these conditions, and the values of up to 3 pixels on both sides of an edge can be changed depending on the outcomes of these conditions.

The deblocking filter algorithm is adaptive in three levels; slice level, edge level and sample level [3, 4]. Slice level adaptivity is used to adjust the filtering strength in a slice to the characteristics of the slice data. The filtering strength in a slice is adjusted by encoder using the offset-a and offset-b parameters. The α and β threshold values that determine whether a 4x4 block edge will be filtered or not and how strong the filtering will be for an edge are a function of quantization parameter (QP) and these two offset parameters.

Edge level adaptivity is used to adjust the filtering strength for an edge to the characteristics of that edge. The filtering strength for an edge is adjusted using the BS parameter. Every edge is assigned a BS value depending on the coding modes and conditions of the 4x4 blocks. The conditions used for determining the BS value for an edge between two neighboring 4x4 blocks are summarized in Table 1 [3, 4]. The strength of the filtering done for an edge is proportional to its BS value. No filtering is done for the edges with a BS value of 0, whereas strongest filtering is done for the edges with a BS value of 4.

Sample level adaptivity is used to adjust the filtering strength for an edge to the characteristics of the pixels in that edge in order to distinguish the true edges from those created by quantization. The filtering strength for an edge is therefore determined by comparing pixel gradients in that edge with α and β threshold values for that edge.

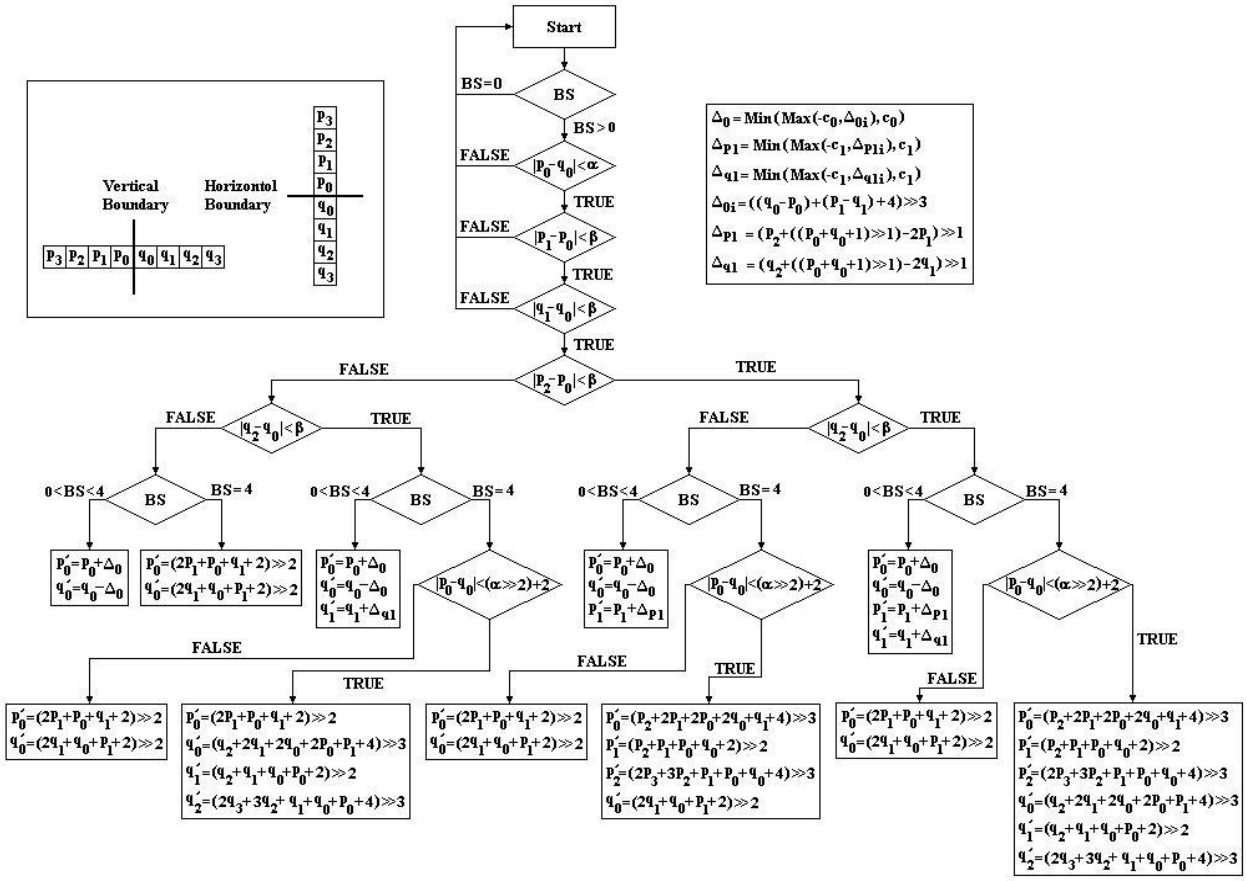


Figure 3 H.264 Deblocking Filter Algorithm

Table 1 Conditions for Determining BS Value

Coding modes and conditions	BS value
One of the blocks is intra and the edge is a macroblock edge	4
One of the blocks is intra	3
One of the blocks has coded residuals	2
Difference of block motion ≥ 1 luma sample distance	1
Motion compensation from different reference frames	1
Else	0

3. Proposed Hardware Architecture

The proposed DBF architecture is shown in Figure 4. It includes a datapath, a control unit, an address generator, one 384x8 register file and two dual port internal SRAMs to store partially filtered pixels. There is also an input buffer to store the pixels that will be filtered and an output buffer to store the filtered pixels.

In a complete H.264 video encoder, the input buffer is loaded with the reconstructed MB generated by the inverse transform and quantization hardware. The inverse transform and quantization hardware generates the reconstructed MB one 4x4 block at a time [8]. In order for the DBF hardware start filtering the available edges as soon as a new 4x4 block is ready, we proposed a novel filtering order which is shown in Figure 5. This filtering order allows overlapping the execution of the inverse transform and quantization hardware and the DBF hardware, and this improves the encoder throughput. This filtering order produces the same results as the filtering order specified in the H.264 standard [3].

In a complete H.264 video encoder, the filtered MBs generated by the DBF hardware form a reference frame which is used for motion compensated prediction of future frames. Since the pixels in a reference frame have to be stored in raster scan order, DBF hardware transfers the pixels in a filtered MB from the output buffer to reference frame memory in raster scan order.

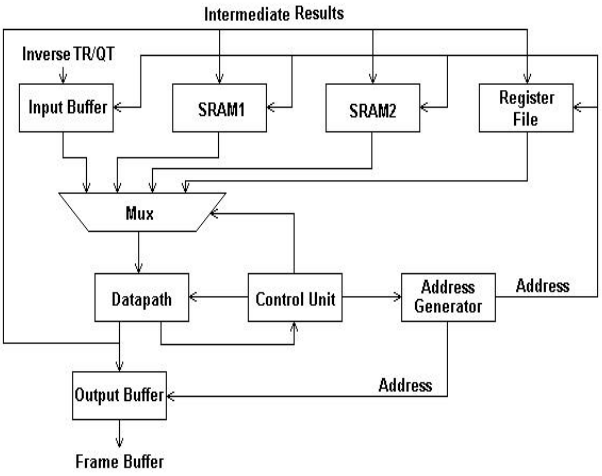


Figure 4 DBF Hardware Block Diagram

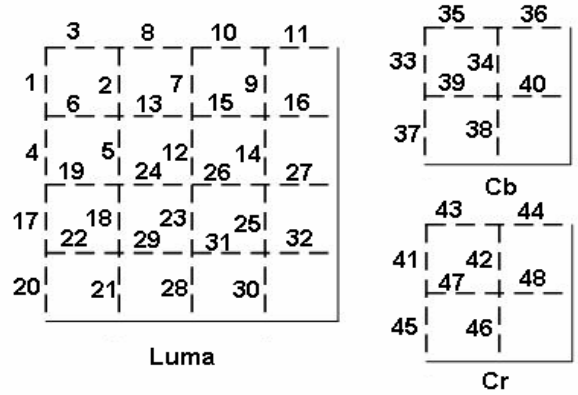


Figure 5 Proposed Filtering Order for Luma and Chroma Blocks

The DBF datapath is shown in Figure 6. It is implemented as a two stage pipeline to improve the clock frequency and throughput. The first pipeline stage includes one 12-bit adder and two shifters to perform numerical calculations like multiplication and addition. The second pipeline stage includes one 12-bit comparator, several two's complementers and multiplexers to determine conditional branch results.

As the DBF algorithm is highly adaptive, the control unit and address generator designs are quite complex. The address generator is implemented as a two stage pipeline to improve the clock frequency. Since the DBF algorithm includes several conditional branches, control unit sometimes has to wait for a branch outcome to continue its execution. In order to avoid datapath pipeline stalls, pre-computation calculations that are independent of these branch outcomes are executed in these cycles.

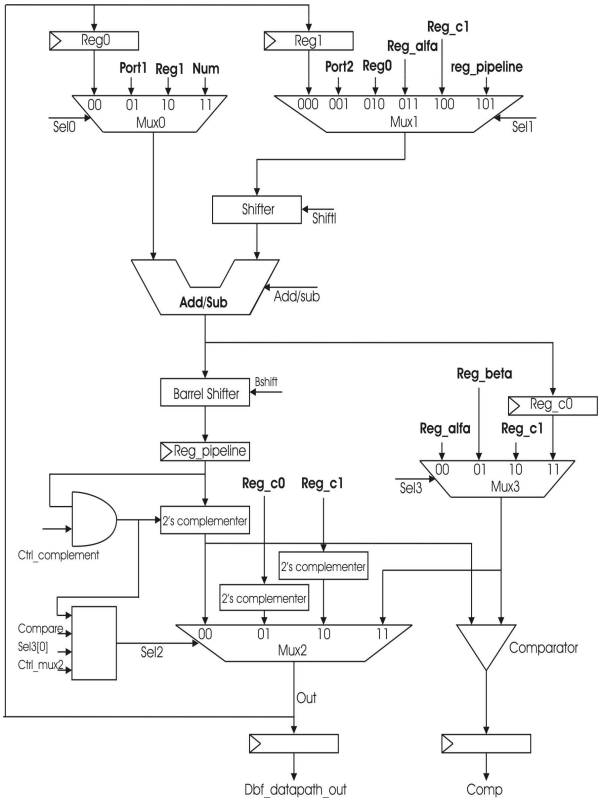
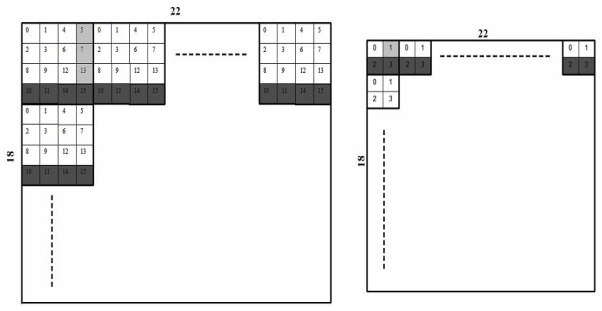


Figure 6 DBF Datapath

Since each 4x4 block in a MB has 4 edges, a pixel in a 4x4 block may be read or updated four times before the 4x4 block is filtered completely. Since the partially filtered pixels of a MB (256 luma and 128 chroma pixels) are accessed quite frequently during the filtering process of that MB, in order to reduce the power consumption, we used a 384x8 register file for storing them instead of using a large internal SRAM.

The organization of the luminance and chrominance components of the MBs in a CIF frame is shown in Figure 7. During the filtering process of a MB, the bottom edges of the 4x4 blocks in the last row of the MB (blocks 10, 11, 14 and 15 of the luminance component, and blocks 2 and 3 of the chrominance Cb and Cr components) cannot be filtered, because the lower neighboring reconstructed MB is not yet generated by the inverse transform and quantization hardware. Therefore, until the lower neighboring reconstructed MBs become available for filtering, the partially filtered pixels in the luminance components of all the MBs in one row of the frame are stored in one internal dual port SRAM and the partially filtered pixels in the chrominance components of all the MBs in the same row of the frame are stored in another internal dual port SRAM.



(a) Luma **(b) Chroma Cb and Cr**

Figure 7 Macroblocks in a CIF Frame

Similarly, during the filtering process of a MB, the right edges of the 4x4 blocks in the last column of the MB (blocks 5, 7, 13 and 15 of the luminance component, and blocks 1 and 3 of the chrominance Cb and Cr components) cannot be filtered, because the right neighboring reconstructed MB is not yet generated by the inverse transform and quantization hardware. Therefore, until the right neighboring reconstructed MB becomes available for filtering, the partially filtered pixels in the luminance and chrominance components of the current MB are stored in the same 384x8 register file which is used for storing the partially filtered pixels in a MB during the filtering process of that MB.

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 2V8000ff1157 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 7.1i. The FPGA implementation is verified to work in a Xilinx Virtex II FPGA on an Arm Versatile Platform development board.

The FPGA implementation including input and output register files as well is placed and routed at 72 MHz under worst-case PVT conditions. It takes 6144 clock cycles in the worst-case to process a MB. The FPGA implementation can process a CIF frame in 33.8 msec. (396 MB * 6144 clock cycles per MB * 13.9 ns clock cycle = 33.8 msec) Therefore, it can process 1000/33.8 = 30 CIF frames (352x288) per second.

The FPGA implementation including input and output register files as well used the following FPGA resources; 3909 Function Generators, 1955 CLB Slices, 313 DFFs, and 2 Block RAMs, i.e. 4.19% of Function Generators, 4.20% of CLB Slices, 0.32% of DFFs, and 1.19% of Block RAMs.

5. Conclusion

In this paper, we presented an efficient hardware architecture for real-time implementation of H.264 adaptive deblocking filter algorithm. 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 72 MHz in a Xilinx Virtex II FPGA. The FPGA implementation can code 30 CIF frames (352x288) 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. 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] Peter List, Anthony Joch, Jani Lainema, Gisle Bj0ntegaard, and Marta Karczewicz, "*Adaptive Deblocking Filter*", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, pp.614-619, 2003

[5] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "*H.264/AVC baseline profile decoder complexity analysis*", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, 715-727, 2003

[6] Yu -Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang, and Liang-Gee Chen, "*Architecture Design for Deblocking Filter in H.264/JVT/AVC*", Proc. IEEE Conf. on Multimedia and Expo, pp. 693-696, 2003

[7] Bin Sheng, Wen Gao and Di Wu, "*An Implemented Architecture of Deblocking Filter for H.264/AVC*", IEEE International Conference on Image Processing (ICIP'04), Vol.1, 24-27, pp. 665-668, October 2004

[8] Ozgur Tasdizen and Ilker Hamzaoglu, "*A High Performance and Low Cost Hardware Architecture for H.264 Transform and Quantization Algorithms*", Proc. European Signal Processing Conference, September 2005