A High Performance H.264 Deblocking Filter

Vagner Rosa, Altamiro Susin, and Sergio Bampi

Federal University of Rio Grande do Sul – Informatics Institute
Av. Bento Gonçalves, 9500 - Campus do Vale - Bloco IV
Bairro Agronomia - Porto Alegre - RS -Brasil
CEP 91501-970 Caixa Postal: 15064
vsrosa@inf.ufrgs.br, Altamiro.Susin@ufrgs.br, bampi@inf.ufrgs.br

Abstract. Although the H.264 Deblocking Filter process is a relatively small piece of code in a software implementation, profile results shows it cost about a third of the total CPU time in the decoder. This work presents a high performance architecture for implementing a H.264 Deblocking Filter IP that can be used either in the decoder or in the encoder as a hardware accelerator for a processor or embedded in a full-hardware codec. A developed IP using the proposed architecture support multiple high definition processing flows in real-time.

1 Introduction

The standard developed by the ISO/IEC MPEG-4 Advanced Video Coding (AVC) and ITU-T H.264 experts set new levels of video quality for a given bit-rate. In fact, H.264 (from this point the standard will be referred only by its ITU-T denomination) outperforms previous standards in bit-rate reduction. In H.264 an Adaptative Deblocking Filter is included in the standard to reduce blocking artifacts, very common in very high compressed video streams. In fact, most video codecs use some filtering as a pre/post-processing task. The main goal of the inclusion of this kind of filter as a part of the standard was to put it inside the feedback loop in the encoding process. As a consequence, a standardized, well tuned, and inside the encoding loop filter could be designed, achieving better objective and subjective image quality for the same bit-rate. The H.264 Deblocking Filter is located in the DPCM loop as shown in Figure 1 for the decoder (it is also called Loop Filter by this reason). Exactly the same filter is used in the encoder and the decoder. The Deblocking Filter in the H.264 standard is not only a single low pass filter, but a complex decision algorithm and filtering process with 5 different filter strengths. Its objective is to maintain the sharpness of the real images while eliminating the artifacts introduced by intra-frame and inter-frame predictions at low bit-rates, while mitigating the characteristic “blurring” effect of the filter at high bit-rates.

The filter itself is very optimized, with very small kernels of simple coefficients, but the complexity of the decision logic and filtering process makes the H.264 Deblocking Filter responsible for about one third [4] of the processing power needed in the decoder process.

T. Wada, F. Huang, and S. Lin (Eds.): PSIVT 2009, LNCS 5414, pp. 955–964, 2009.

In this paper an architecture for the H.264 deblocking filter is proposed. The architecture was developed focusing FPGA, aiming making a balanced use of the resources (logic, registers, and memory) available in FPGA architectures, although it can be synthesized using an ASIC flow. The performance goal was to exceed 1080p requirements when synthesized to a XILINX Virtex II FPGA. A Xilinx Virtex II pro FPGA was used to validate the developed IP.

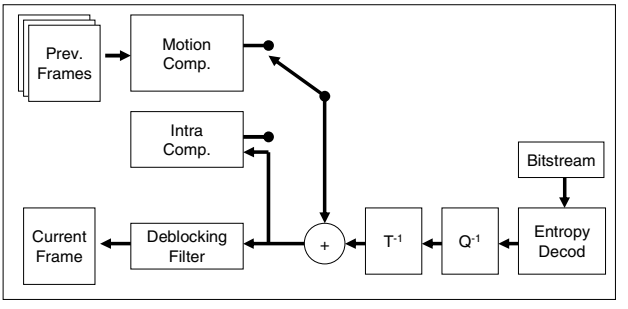

Fig. 1. H.264 Decoder

The rest of this paper is organized as follows. The section 2 describes the standardized algorithm for the H.264 deblocking filter. Section 3 presents the proposed architecture and section 4 the results obtained. Finally the section 5 presents the conclusions and future work.

2 Deblocking Filter Algorithm

In the H.264 standard the image is divided in small units called blocks. Each block is 4x4 pixels. The color format is YCbCr 4:2:0 (main profile), meaning the crominance (croma) components being sub-sampled to half the sample rate of the luminance (luma) in both directions. The blocks are then grouped in macroblocks which is a 4x4 block matrix for luma and 2x2 matrix for each croma component. Each block edge has to be filtered. The Deblocking Filter is applied to each decoded block of a given macroblock for luma and croma samples in raster scan order. For each block, four different edges are filtered separately, in the sequence presented in Figure 2.

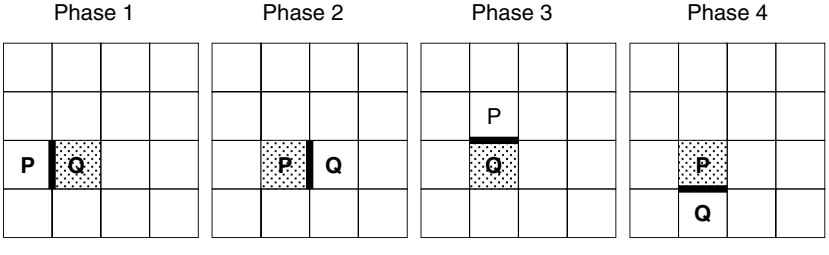

Fig. 2. Edge positions for a given 4x4 block inside a 16x16 macroblock

For each block edge, the filter is applied to the pixel component values perpendicular to that edge. The naming conventions for the pixels around the edge are showed in figure 3. Pixel components in both the current (Q) and the previews (P) block can have values changed. Pixels already modified during a filter stage can be modified again in a subsequent filter operation (this causes some data dependencies). The filtering algorithm is adaptive, so that the pixel values, the position of a block inside the macroblock, the type of prediction employed (inter or intra), the motion vectors (inter prediction) and the quantization parameter are taken into account for the boundary strength calculation.

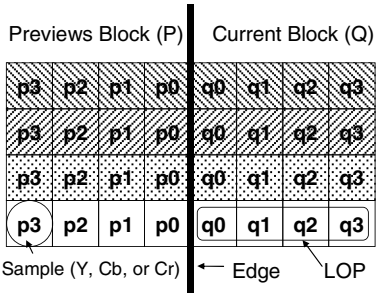

Fig. 3. Filter conventions

The boundary strength (BS) can assume five different values from 0 (no filtering) to 4 (strongest filtering) and is defined as follows.

- If the edge is not a slice edge (the filter is not applied across slice edges) then the filter can be applied, else BS=0. Based on pixel values, it can be found that a edge can contain a natural discontinuity in the values, making the BS to be set to 0.
- If the block is intra coded, then BS=3 if it is an internal edge or BS=4 if it is a external edge. Internal edges are those involving two blocks of the same macroblock, while external edges are those involving two blocks from different macoblocks.
- If neither blocks are intra-coded and at least one contain coded coefficients (non zero transform residues) then BS=2
- BS=1 is used when none of the above conditions are satisfied and the reference frames of two blocks are different or when the reference frames are the same but any component of the two motion vectors has difference more than 4 (1 pixel sample).

The Boundary Strength for croma is the same as for the corresponding luma block, but the filters employed for luma and croma are different. The quantization parameter (QP) and the pixel values are taken into account. The parameters α and β are QP dependent and set thresholds for filtering to be applied. Saturation functions (clip and clip3) need do be used in some steps of the calculations. This makes the BS decision logic a complex set of sequential calculations. The BS calculation needs to be done for every LOP (Line Of Pixels – figure 3) pair. More details on Deblocking Filter process can be obtained in [1] and a complete flowchart in [3].

3 Proposed Architecture

The proposed architecture was developed to exceed HDTV 1080p resolution requirements (1920x1080x30) in the H.264 Main Profile. The initial architectural concept was initially based on the work developed by [4], but was evolved to be a datapath block, different from the coprocessor block proposed by [4]. The primary differences from this work related to others found in the literature ([4], [5] and their references) are the use of local memory for the whole process, instead of accessing the main memory. The use of FPGAs as a primary implementation device also leaded to some architectural decisions that leaded to a more efficient resources usage and higher speed. In this scenario, the development of high-depth pipeline architecture was straightforward: each Logic Element in current FPGA can behave as a look-up-table (LUT), a single bit flip-flop (register), and a carry logic at the same time. The unneeded parts of the logic elements (ex. Flip-flops in combinational logic block) are bypassed and can not be used in other blocks due to routing limitations in typical FPGA architectures.

3.1 Numbering and Color Conventions

Figure 4 illustrates the block numbering and macroblock color convention adopted from this point.

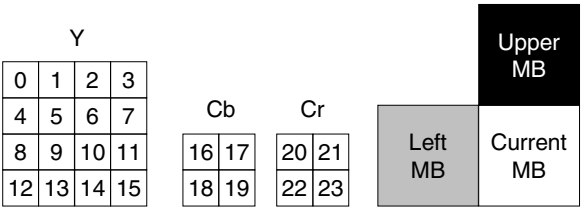

Fig. 4. Luma and Croma block enumeration; Macroblock color diagram

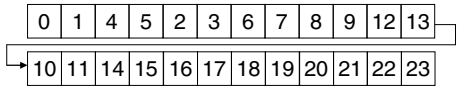

Fig. 5. Input block sequence in the input buffer of the Deblocking Filter

The input data for a given luma/croma macroblock arrives in the sequence presented in Figure 5: Y first, then Cb and Cr; double Z scan for luma.

3.2 Proposed Filter Architecture

In the Deblocking Filter process, the edge filter is the heart of the process. It is responsible for all filter functionality, including the thresholds and BS calculation and filtering itself. The remaining of the process is only sequence control and memory for block ordering.

The Edge Filter architecture can accept one LOP per cycle for Q and P blocks and produce the filtered Q' and P' LOP. This process is illustrated in Figure 6 (parameter lines are omitted for simplicity). Using this scheme, an entire block border will enter in the Edge Filter each four cycles (one block border is four LOPs tall, as illustrated in Figure 3).

Fig. 6. Edge Filter

Based on the diagram presented in Figure 6, a pipelined architecture for the Edge Filter was designed. As stated before, the procedure for calculation of all parameters needed to decide whether to filter or not and what BS to use requires a lot of sequential calculations (data dependencies). The pipelined architecture was designed so that only one arithmetic or logic operation is to be done every stage of the pipeline. This lead to an 11-stage pipeline only to calculate the BS, used to select the correct filter. All the filters could be done in parallel to BS calculation, so a 12 stage pipeline would be enough to achieve the maximum possible operation frequency. However, if an entire column of luma blocks (for vertical edges) that consist of four blocks stacked are processed before the next one, the first LOP of the first Q block (the uppermost) will be the first LOP of the first P block after 16 LOP cycles. If an Edge Filter architecture with a 16 stage pipeline can be designed, the output P' could be connected directly to the input Q. The croma Cb and Cr, wich are half the height (only 2 blocks tall), can be stacked so they can be processed as a luma block. This approach makes the implementation of the control logic much simpler. A 16-stage pipelined Edge Filter, presented in Figure 7, was then designed to meet the above criteria.

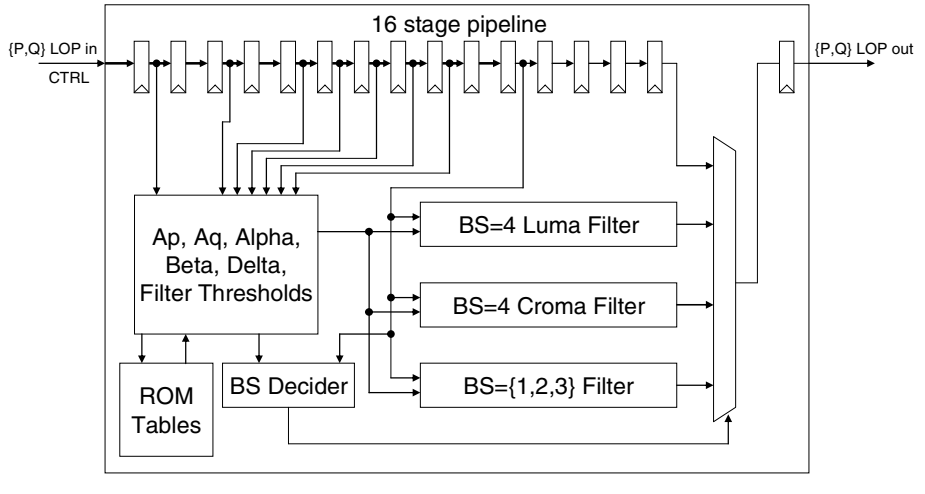

Fig. 7. Edge Filter

The architecture presented in figure 7 consumes two LOPs (one for Q and other for P blocks) and their respective parameters (QP, offsets, prediction type and neighborhood information) every clock cycle, producing two filtered LOPs (Q' and P') 16 clock cycles later.

This pipelined edge filter can be then encapsulated in such way only the Q input and P' outputs are visible, as illustrated in Figure 8.

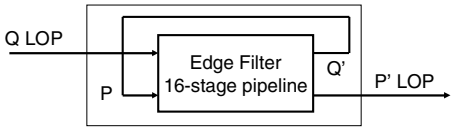

Fig. 8. Filter encapsulation

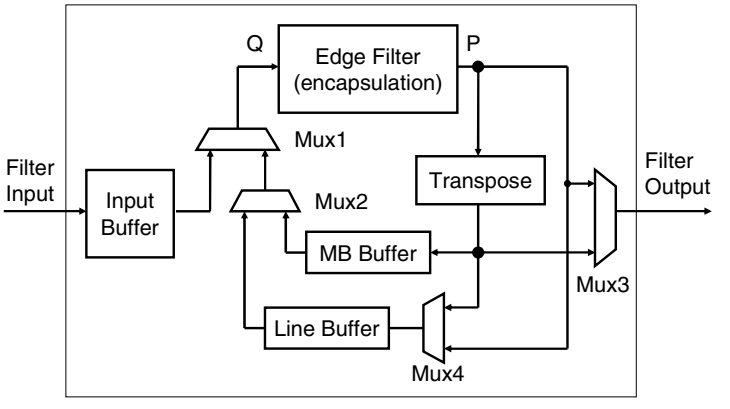

Fig. 9. Proposed Deblocking Filter architecture

Using this encapsulation turns the control logic simpler, but at a cost of a small overhead: P data can only be fed by the Q input, so the first P data need to be fed 16 cycles before the filtering process can start. During this 16 cycles, the BS should be forced to 0, so that no filtering is applied to P data while they passes through the Q datapath. After finishing the processing of a macroblock, the Edge filter must be emptied, so another 16 cycle have to be spent. Fortunately, the empting and filling phases stages can be overlapped, so the overhead is much lower.

Finally, the architecture of the proposed deblocking filter is presented in Figure 9. As stated in section 2, the filter has to be applied to both vertical and horizontal edges. In the proposed architecture, a single Edge Filter filters both horizontal and vertical edges. A transpose block is employed to convert vertically aligned samples in the block into horizontally aligned ones, so that horizontal edges can be filtered by the same Edge Filter.

The input buffer is needed only for data reordering, as input blocks arrive in a different order they are consumed in the deblocking process.

The MB and Line buffers are used to store blocks and block information data (QP, type, filter offset, state) which are not completely filtered (horizontal or vertical

filtering is missing). The size of MB buffer is 32 blocks (512 samples plus 32 block information data) and the line buffer depends on the maximum frame size the filter is supposed to process (7680 samples, plus 480 block information data for a 1080p HDTV frame).

3.3 Filter Operation

The filter data flow description follows. First, pixel and control data are fed to the filter toward the Input Buffer. This buffer is needed because data is read from input buffer in a different order it comes in. Also this buffer provides some burst capability, as it accept one LOP per clock cycle. Once a complete luma/croma macroblock is available in the input buffer, the filtering process can be started.

The Encapsulated Edge Filter entity contains a 16 stage pipeline edge filter where the input P is connected to the output Q. The mux1 and mux2 are set so that MB buffer data is fed to the input Q of the Encapsulated Edge Filter. The data read from the MB buffer is the blocks 3, 7, 11, and 15 from the left macroblock. As the data is read one LOP at a time, it takes four clock cycles to read an entire block into the Encapsulated Edge Filter.

Exactly 16 cycles are needed to read four 4x4 blocks. During this phase the filter is set to bypass (BS=0), so pixel data fed are not filtered. As stated before, the Q output of the Edge Filter is connected to the P input in the encapsulated edge filter. The next clock cycle after the load of the aforementioned blocks will put then into the edge filter again, but in the P input. At this point, the mux1 is switched so the Encapsulated Edge Filter can receive data from the Input Buffer. The blocks 0, 4, 8, and 12 is read from input buffer, a LOP at a time.

The Edge Filter Process now is being fed with the data needed for the filtering process for the external vertical edge to start. Immediately after the block 12 the blocks 1, 5, 9, and 13 can be fed and then the blocks 2, 6, 10, 14 and 3, 7, 11, 15. 32 cycles after the block 3 from the left MB started being read from the MB buffer, the filtered block 3 will appear at the P output of the Encapsulated Edge Filter.

The data can take 3 different destinations: the MB Buffer, The Line Buffer or the output of the filter. In the case of the blocks 3, 7 and 11, there is no other processing to be done for these blocks, and the mux3 is selected to output directly from the output P of the Encapsulated Edge Filter. The block 15 from the left macroblock is fed to the Line Buffer toward the Transpose process to be used latter in the horizontal filtering process. The blocks 0 to 15 are sent to the MB buffer toward the Transpose to be filtered horizontally. The Transpose takes for cycles for reading the entire block and makes it available in the transposed form.

After filtering the vertical edge of all luma blocks the job for the horizontal edge can be done. In order to obtain the maximum throughput, the croma vertical filtering is done immediately after the luma. Then, blocks 17, 19, 21, and 23 from the left Cb and Cr macroblock are read from the MB Buffer. Notice that the Cb and Cr blocks are stacked in order to achieve the 16 LOPs needed to fill the Edge Filter pipeline. As with the luma blocks, this initial operation only loads the Edge Filter with the P blocks and then the filter is deactivated. Then the croma blocks 16, 18, 20, and 22 are read from Input Buffer and then the blocks 17, 19, 21, and 23. The blocks 17 and 21 from the left Cb and Cr macroblocks, respectively are completely filtered and then

can be sent to the output. The blocks 19 and 23 from the left Cb and Cr macroblocks respectively are send to the Line Buffer toward the Transposer in order to be used in the horizontal external edge filtering. The blocks from the current Cb and Cr macroblocks are stored transposed in the MB Buffer for the horizontal filtering process.

At this moment, the horizontal edge filtering for luma and croma can take place. The luma data dependency was solved by the luma/croma interleaving and the blocks 12, 13, 14, and 15 from the upper macroblock stored in the Line Buffer are fed to the Encapsulated Edge Filter, followed by the blocks 0, 1, 2 and 3 from the current macroblock stored in the MB Buffer. The process follows by reading the blocks 4, 5, 6, and 7 and then 8, 9, 10, 11 and finally, 12, 13, 14 and 15. The output of the Encapsulated Edge Filter has different destinations. The blocks 12, 13, 14, and 15 from the upper macroblock are completely filtered and can be output from the filter, as well as the blocks 0, 1, 2, 4, 5, 6, 8, 9, 10. The remaining blocks takes two destinations: The blocks 3, 7, 11, 15 goes to the MB Buffer toward the Transpose for the next external vertical edge filtering; The blocks 12, 13, 15 goes to the Line Buffer, without being transposed (they are actually transposed and will be used as upper blocks in the filtering process for the macroblock below them).

The last phase of the filtering is the application of horizontal edge filtering to the croma blocks. As in the vertical edge filtering, the croma macroblocks need to be aligned and filtered together in order to fit in the 16 stage pipeline edge filter. The horizontal edges filtering for croma samples starts by reading the blocks 18, 19, 22, and 23 from the upper macroblocks stored in Line Buffer followed by reading the blocks 16, 17, 20, and 21 from the current macroblock stored in the MB Buffer, and finally the blocks 18, 19, 22, and 23. The blocks 16 and 20 are completely filtered and can be output from the filter. The blocks 17, 19, 21, and 23 are stored in the MB Buffer toward the Transpose and the blocks 18 and 22 are stored in the Line Buffer.

A total of 256 clock cycles are needed to process an entire 4:2:0 macroblock (24 blocks). If data is not available at the beginning of a 256 cycle operation, a bubble is inserted in the pipeline. All pipelines are emptied and the filter operation stops until there is data available in the input buffer. The stop cycle is also a 256 cycle operation.

Figure 10 illustrates the output sequence of blocks (as enumerated in figure 4) for the implemented filter architecture. Observe that for the 24 blocks output corresponding to an entire luma/croma macroblock processing cycle, the output have blocks belonging to three different macroblocks interleaved. The destination of that data is the reference frames (or the output video, when a frame is ready to display). A simple look-up-table can be implemented to ease the reference frame memory address.

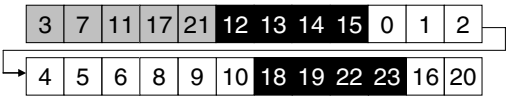

Fig. 10. Filter Output block sequence

4 Implementation and Results

The architecture presented in Section 3 was described in VHDL. About 3,500 lines of code were written. The design behavior was validated by simulation using some

testbench files and data extracted from the JVT reference software using some public domain video sequences. The validated behavioral design was then synthesized, the post place and route was validated and performance results were obtained for a Xilinx Virtex2-pro FPGA.

Using the reference software CODEC, extracted data before and after the Deblocking Filter process was used to ensure the correctness of the implemented architecture. Table 1 presents the number of Xilinx LUTs and BRAMs used to synthesize the developed Deblocking Filter. Observe the balance between the amount of logic (LUTs) and memory employed, related to the total amount available in the target device.

Table 1. Synthesis results

Device	XC2VP30	XC5VLX30
LUTs	4008/27392 (14%)	4275/19200 (21%)
BRAMs	20/136 (14%)	7/32 (21%)
Fmax (MHz)	148	197
FPS@1080p	71	95

Running at 148MHz in the Virtex II Pro device, this implementation is 2.36 times faster than the requirement for HDTV (1080p). For the Virtex-5, running at 197MHz, it is 3.14 times the requirement for HDTV. This IP can be used to build an encoder or decoder for a systems with the following characteristics:

- Ultra-high definition (4K x 2K pixel @ 24fps);
- High definition stereo video (two HDTV streams);
- Multi stream surveillance (up to 2K CIF streams);
- Scalable high definition;
- Low-power HDTV, where the device can operate at lower frequency, lower voltages and still achieve HDTV requirements.

Table 2 presents some performance comparison. The IP implemented with the developed architecture only loses to [5], but [5] do not include the external memory access penalty needed to obtain the upper MB data.

Table 2. Literature comparison

	Cycles/MB	Memory type	fps@1080p
our	256	Dual-port	95
[4]	variable	Two-port	45
[5]	96	Two-port	100
[6]	192	Dual-port	30

The maximum resolution achievable by the proposed architecture is only limited to the size of the line buffer (Figure 9) implemented. This buffer represents a significant amount of the total memory employed by this design and impacts the number of Block RAM (BRAM) used by the IP. The results presented in Table 1 is for a 2048

pixel wide frame, including the memory necessary to store block parameters, needed by the BS decision process. The maximum picture width is determined by a parameter in the synthesizable code and the height is unlimited.

5 Conclusion

This work presented a high performance architecture for H.264 Deblocking Filter IP targeted to exceed HDTV requirements in FPGA. The primary contribution of this work was the high performance deep pipeline architecture employed to improve the speed in the Boundary Strength decision and at the same time reducing the control logic. The proposed architecture stores all intermediate information in its own memory, differently from most works in literature that rely on external memory to store some blocks not completely filtered in a line of macroblocks. The developed IP based on the proposed architecture was synthesized to a Virtex II Pro and for a Virtex 5 device and prototyped in a XUP-V2Pro (Virtex II-Pro XC2VP30 device). Results showed its capability to exceed the processing rate for HDTV, reaching 71 frames per second in the Virtex II Pro device and 95 frames per second in the Virtex 5 device at 1080p (1920x1080) resolution.

Future work will address the support for high profiles, scalability and multi-view amendments of the H.264 standard, which require small modification in the BS decision logic and in the data width for pixel values.

References

1. Draft ITU-T Recommendation and Final Draft international Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC) (March 2003)
2. List, P., Joch, A., Lainema, J., Bjotergaard, G., Karczewicz, M.: Adaptative deblocking filter. IEE trans. Circuits Syst. Video Technol. 13, 614–619 (2003)
3. Puri, A., Chen, X., Luthra, A.: Video coding using the H.264/MPEG-4 AVC compression standard. Signal Processing: Image Communication 19, 793–849 (2004)
4. Sima, M., Zhou, Y., Zhang, W.: An Efficient Architecture for Adaptative Deblocking Filter of H.264/AVC Video Coding. IEEE Trans. On Consumer Electronics 50(1), 292–296 (2004)
5. Lin, H.-Y., Yang, J.-J., Liu, B.-D., Yang, J.-F.: Efficient deblocking filter architecture for H.264 video coders. In: 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006, May 21–24 (2006)
6. Khurana, G., Kassim, A.A., Chua, T.-P.: M.B.A Mi Pipelined hardware implementation of in-loop deblocking filter in H.264/AVC. IEEE Transactions on Consumer Electronics 52(I.2), 536–540 (2006)