This paper presents a new edge-protection algorithm and its very large scale integration (VLSI) architecture for block artifact reduction. Unlike previous approaches using block classification, our algorithm utilizes pixel classification to categorize each pixel into one of two classes, namely smooth region and edge region, which are described by the edge-protection maps. Based on these maps, a two-step adaptive filter which includes offset filtering and edge-preserving filtering is used to remove block artifacts. A pipelined VLSI architecture of the proposed deblocking algorithm for HD video processing is also presented in this paper. A memory-reduced architecture for a block buffer is used to optimize memory usage. The architecture of the proposed deblocking filter is verified on FPGA Cyclone II and implemented using the ANAM 0.25 µm CMOS cell library. Our experimental results show that our proposed algorithm effectively reduces block artifacts while preserving the details. The PSNR performance of our algorithm using pixel classification is better than that of previous algorithms using block classification.

Keywords: Deblocking filter, offset filter, edge-preserving filter, post-processing, very large scale integration (VLSI) architecture.
loop routines, such as POCS-based algorithms [9], [10], are only suitable for software because these loop routines require high delay which is not appropriate for real-time processing. Thus, most of deblending filters use spatial-domain filtering methods which have low computational complexity.

Recently, classification-based deblending method in the spatial domain has emerged as a promising approach due to low computational complexity, fast execution, and real-time ability. Jongho Kim and others [4] explicitly classify each deblending block (DB) which has an image made of 8×8 squares, into two metrics: horizontal activity and vertical activity. A hard threshold is applied to the two metrics to give four categories: uniform DB, horizontal DB, vertical DB, and complex DB. For each type of DB, different filtering methods are used to reduce all kinds of block artifacts. Ling Shao and others [5] proposed a deblending scheme using local entropy analysis. Each block is classified into detailed regions, flat regions, and intermediate regions based on entropy values. In the Amir Z. Averbuch’s method [6], blocks are categorized into 2 classes: uniform block and nonuniform block. Deblending of the uniform blocks is accomplished by applying three iterations of the weight adaptation by grading (WABG) scheme. Then, a fourth iteration is used in order to deblend the nonuniform blocks or detailed blocks. Since block artifacts are more visible in smooth regions, the authors’ purpose is to apply strong smoothing to uniform regions and little or no smoothing on detailed regions. Therefore, the detail in the image can be preserved while block artifacts are reduced.

In all these proposed methods, the authors attempt to categorize blocks into classes. Then, different filtering techniques for each region are applied based on the classification characteristics of the blocks. The strong smoothing is applied on smooth regions, and week smoothing is applied to textured regions. Therefore, the sharpness and detail of the image is preserved while the block artifact is reduced. However, the classification with fixed size blocks does not always give robust distinction between different regions. The drawback of these approaches is that fixed-size classified blocks may contain some smooth regions and edge regions with various shapes. For example, a block is classified into a smooth region when the intensity activity of the block is less than a predefined threshold. However, this block actually contains some edge pixels. When the deblending filter applies strong smoothing to the whole of this block, these edge pixels can be destroyed. As a result, the deblending filter may smooth inappropriate regions because of the fixed-size block classification. Figure 1 demonstrates the case when a block is classified into a smooth region, while containing some edge pixels.

In this paper, we propose a new pixel classification-based approach for block artifact reduction and also present an efficient architecture for its hardware implementation. In our proposed algorithm, we first categorize each pixel by an edge detection process instead of classifying each fixed-size block as in other techniques, and then generate three binary edge-protection maps for horizontal orientations, vertical orientations, and edge strength. Based on the horizontal and vertical maps, we apply an offset filter to reduce grid noise in a smooth region. In the next step, we employ an edge-preserving filter with an edge profile map to remove staircase noise and corner outlier noise. We also propose an efficient pipelined architecture for the algorithm. The hardware implementing architecture is designed practically for HD video real-time post-processing systems by using pipelined architecture and area-optimized block buffers to achieve the highest throughput as well as reduce memory usage. Our architecture is verified on Cyclone II FPGA and synthesized by using the ANAM 0.25 μm CMOS cell library.

The rest of this paper is organized as follows. In section II, we describe the proposed edge-protection deblending algorithm. Section III presents VLSI architecture with detailed block diagrams of the deblending filter. Section IV shows the experiment results including algorithm simulation and architecture verification, and the performance comparison with other approaches. Finally, we conclude in section V.

II. Proposed Edge-Protection Deblending Algorithm

The aim of the proposed approach is to preserve the detail of images by using an edge-protection map while filtering blocky noise. The edge-protection map is generated by pixel classification. Two adaptive filters are then applied to process pixels with different weighted factors according to the edge-protection map.

1. Pixel Classification

Let’s consider an 8×8 block as shown in Fig. 2. This block is
We classify every pixel in this block by an edge detection process. Various convolution kernels such as Sobel or Prewitt can be used for this operation. In this paper, for the purpose of real-time hardware implementation, we choose the Prewitt operator which needs low computation resources to estimate the magnitude and orientation of edges.

After applying the Prewitt operator for a 3×3 sliding window, we obtain the orientation magnitude \( G_x \) and \( G_y \) of the horizontal and vertical direction for each pixel. The magnitude, or edge strength, is then calculated using the formula

\[
G = |G_x| + |G_y|.
\]

Three binary edge-protection maps are then created from the \( x \)-direction gradient, \( y \)-direction gradient, and edge strength. We apply the threshold \( T \) and \( T_d \) to generate three binary edge maps, \( E_x \), \( E_y \), and \( E_z \), for the block. In the experiment, the threshold \( T \) and \( T_d \) are set to 20 and 10, respectively.

\[
\begin{align*}
E_{x_{i,j}} &= \begin{cases} 
0, & |G_x| < T_d, \\
1, & |G_x| \geq T_d,
\end{cases} \\
E_{y_{i,j}} &= \begin{cases} 
0, & |G_y| < T_d, \\
1, & |G_y| \geq T_d,
\end{cases} \\
E_{z_{i,j}} &= \begin{cases} 
0, & G < T, \\
1, & G \geq T.
\end{cases}
\end{align*}
\]

These maps indicate that a pixel belongs to a smooth region or an edge region if its binary value in these maps is zero or one, respectively. Therefore, the adaptive filter can use these maps to protect the detail of images.

2. Offset Filter for Grid Noise

After obtaining binary edge-protection maps for the block, we apply 1-D offset filtering for all pixels in the block to reduce blocky noise. The offset filter processes horizontally first, and then, vertically. The procedure of 1-D offset filtering is

\[
\begin{align*}
offset &= p_{i,j} - p_{i,5}, \\
p_{i,1} &= p_{i,1} - offset/16 \times E_{x_{i,1}} , \\
p_{i,2} &= p_{i,2} - offset/8 \times E_{x_{i,2}} , \\
p_{i,3} &= p_{i,3} - offset/4 \times E_{x_{i,3}} , \\
p_{i,4} &= p_{i,4} - offset/2 \times E_{x_{i,4}} - offset/4 \times E_{x_{i,5}} , \\
p_{i,5} &= p_{i,5} + offset/4 \times E_{x_{i,5}} + offset/4 \times E_{x_{i,6}} , \\
p_{i,6} &= p_{i,6} + offset/4 \times E_{x_{i,6}} , \\
p_{i,7} &= p_{i,7} + offset/8 \times E_{x_{i,7}} , \\
p_{i,8} &= p_{i,8} + offset/16 \times E_{x_{i,8}},
\end{align*}
\]

where \( p_{ij} \) with \( i=1:8 \) is the intensity value of the pixel at the position \( (i,j) \) in the 8×8 block as shown in Fig. 1, and \( E_{x_{i,j}} \) is the inversion of \( E_{x_{i,j}} \), that is, if \( E_{x_{i,j}} \) is zero, \( E_{x_{i,j}} \) is one.

In (3), we define “offset” as the amount of discontinuity at the block boundary. This value is calculated from the difference of the two nearest pixels at the block boundary. As with some previous methods [4], [5], we also use shifted offset values (that is offset/2, offset/4, and offset/8) to eliminate discontinuity at the block boundary. The pixels on the left \( (p_{11}, p_{12}, p_{13}, p_{14}) \) and pixels on the right \( (p_{51}, p_{52}, p_{53}, p_{54}) \) are updated by the way to balance the gap at the block boundary. Since block artifacts are more visible on a smooth region, we employ the oriented edge map to make the filter process only pixels in smooth regions except for two pixels at the block boundary \( p_{14} \) and \( p_{54} \). If these pixels are in the edge region, they are updated with the coefficient offset/4. The edge map \( E_z \) is applied for horizontal filtering, and \( E_z \) is applied for vertical filtering.

After this step, the blocky noise in the smooth region can be reduced, but staircase noise and corner outlier noise still remain in edge regions. A post-processing filter should be used to remove these noises.

3. Edge-Preserving Filter for Staircase Noise and Corner Outlier Noise

We propose an edge-preserving filter to remove staircase noise and corner outlier noise. Our approach is based on the fundamental idea that the bilateral filter is able to smooth along the edge and thus enhance the detail of images [15]. Consider a 3×3 spatial sliding window

\[
X = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix},
\]

where \( x_i, i=1,\cdots,9 \) is an intensity value, and \( x_j \) is a to-be-filtered
pixel. The intensity distances from the center pixel $x_i$ are
defined by

$$d_i = |x_i - x_c|, \quad i = 1, \ldots, 9. \quad (5)$$

The coefficients for the convolution mask of the filter are
extracted as follows:

$$c_i = (255 - d_i)^q. \quad (6)$$

The coefficient $c_i$ receives larger values for smaller values of
intensity distance $d_i$. That is, pixels which are very different in
intensity from the central pixel are weighted less than pixels
that are close intensity from the central pixel. This concludes to
the following convolution mask:

$$C = \frac{1}{Q+1} \begin{bmatrix}
  c_1 & c_2 & c_3 \\
  c_4 & c_5 & c_6 \\
  c_7 & c_8 & c_9
\end{bmatrix}. \quad (7)$$

Similar to a bilateral filter whose coefficients are weighted
based on intensity similarity with Gaussian distribution, the
proposed edge-preserving filter uses coefficients depending on
intensity distances with an exponent function. Therefore, the
convolution mask allows creating a smooth effect along an
edge and thus can remove staircase noise and corner outlier
noise. The factor $q$ is to control the amount of edge smoothing
and was chosen for the experiment. In our experiment, $q$ is set
to 8. The denominator $\Sigma c_i$ is a normalization factor to maintain
zero-gain for the output image. The output of the edge-
preserving filter is computed by applying spatial filtering with
the coefficient mask $C$ and the sliding window $X$.

The edge-preserving filtering is only applied for pixels in
edge regions in order to remove staircase noise and corner outlier
noise. Therefore, the binary edge map $E_x$ is used in the
filtering procedure as shown in (8).

$$p_{ij} = p_{ij} \times E_{x_{ij}} + \text{edge~filter}(X) \times E_{z_{ij}}. \quad (8)$$

In summary, our proposed algorithm for block artifact
reduction is described in the following procedure:
For each 8×8 block :

i) establish three binary maps, $E_x$, $E_y$, and $E_z$, for edge
protection,
ii) apply the horizontal and vertical offset filters to pixels of
the block according to procedure (3),
iii) apply the edge-preserving filter to pixels of the block
according to (8).

III. Efficient Architecture for Block Artifact Reduction

The proposed architecture is designed to perform real-time
processing, so a pipeline technique is employed. Parallel
processing is used for the offset filter and edge-preserving filter.
An 8-bit fixed-point numerical representation for pixel
intensity is chosen. HDTV 1080p with 30 Hz frame rate
processing is a target constraint, so the maximum resolution of
the input image can be 1,920×1,080 pixels. Therefore, the
required throughput is 1,920×1,080×30 ≈ 75.106 pixels/s. The
design core processes 1 pixel per 2 clock cycles. Hence, the
required operation clock is 150 MHz.

1. Top Block Design

The deblurring core is comprised of five basic functional
units: a block buffer, a pixel classifier, an offset filter, an edge-
preserving filter, and a flow controller. The top design
architecture is shown in Fig. 3.

The proposed deblurring filter requires input data pixel by
pixel. The block buffer converts raster scan input pixels into
blocks of 8×8 pixels for the offset filter, and blocks of 3×3
pixels for the pixel classifier. After being applied to the offset
filter, the 8×8 pixel blocks are converted to 3×3 pixel blocks for
the edge-preserving filter. The control unit implemented by a
finite state machine is responsible to control data flow. The
delay unit realized by FIFO memory is to keep synchronization
for pipeline stages.

2. Memory-Reduced Block Buffer

In the proposed architecture, the pixel classifier accepts 3×3
pixel blocks as the input, the offset filter receives 8×8 pixel
blocks, and the edge-preserving filter requires 3×3 pixel input
blocks. Therefore, we need two types of block buffer for 3×3
pixel blocks and 8×8 pixel blocks. For raster scan input format
as shown in Fig. 4(a), conventional buffers normally used for
image processing hardware [16] utilize 8 line-buffers to create
8×8 pixel blocks as shown in Fig. 5(a). Therefore, large FIFO
memory buffers are required if the input image has HD
resolution.

In order to reduce memory usage for block buffers, we
propose a new block buffer using modified raster format as
shown in Fig. 4(b). The new block buffer receives pixels in
vertical order and generates 8×8 pixel arrays for the offset filter
as in Fig. 5(b).

The new buffer architecture only requires 8-byte FIFO
memory to store temporal pixels. The total memory for the
new block buffer is 120 (7×16+8) bytes. Therefore, we can
save 99% memory usage of the block buffer if the input data
are 1080p image frames, with the cost of extra hardware for
address generator to access image memory in vertical raster
scan order, while the conventional block buffer requires 13,448
(7×1920+8) bytes.
Fig. 3. Block diagram of the deblocking filter.

![Diagram of deblocking filter](image)

Fig. 4. (a) Conventional raster scan order and (b) modified raster scan format.

![Diagram of raster scan orders](image)

Fig. 5. (a) Conventional 8×8 block buffer using horizontal raster scan format (W is the width of images) and (b) memory-reduced 8×8 block buffer using vertical raster scan format.

![Diagram of block buffers](image)

Fig. 6. Architecture of the 8×8 block-to-raster unit.

![Diagram of 8×8 block-to-raster unit](image)

The raster-to-3×3 block unit is similar to the 8×8 block buffer. It includes a 3×3 register array and FIFO memory with the length 13 (16–3). The 8×8 block-to-3×3 block unit is designed by combining the 8×8 block-to-raster and the raster-to-3×3 block units. The 8×8 block-to-raster unit is shown in Fig. 6.

3. Pixel Classification Unit

The pixel classification unit comprises a Prewitt operator and thresholding unit. We apply 3-stage pipeline architecture for pixel classification unit (Fig. 7). The pixel classification unit is able to generate three binary values, $E_x$, $E_y$, and $E_z$, for each pixel in every clock cycle.

4. Offset Filter

The design of the offset filter unit is based on a shifting parameter to reduce the hardware complexity. The offset filter receives two types of input data including a pixel vector and an edge map vector. Each clock cycle, the offset filter processes 8 pixels in vector format. Hence, it takes 8 clock cycles for the offset filter to complete an 8×8 pixel block.

Two identical offset filtering units are used to allow simultaneous on-the-fly filtering for vertical and horizontal
blocky noise. However, there is a difference between two these offset filters in the input data format. The horizontal offset filter receives the horizontal pixel vector and horizontal edge-protection map as input data, while the vertical offset filter accepts the vertical pixel vector and vertical edge-protection map as input data. The block diagram of the proposed architecture is shown in Fig. 8. The output results of the filter are registered by flip-flops for the pipeline stages.

5. Edge-Preserving Filter

The edge-preserving filter receives a 3×3 pixel block and an $E_z$ map as input data. The convolution mask of the input 3×3 block is calculated by three steps: intensity distance computation, coefficient extraction, and coefficient normalization. Then spatial filtering is performed by multiplying the convolution mask by the input neighbor pixel values.

The block architecture of the edge-preserving filter is shown in Fig. 9. The critical path includes 4 multipliers, 2 subtractors, 1 absoluter, 1 adder, 1 multiplexer, and a divider. To avoid long delays for the critical path, a 12-stage pipelined architecture is proposed for this unit (Fig. 9). The multiplier is designed with 8-bit architecture, and the output product is a 16-bit value. Therefore, we take 8 high bits of the product value to be the input for the next stage of computation. The precision is slightly decreased, but it is acceptable for visual quality.

6. Address Generator and Controller

The address generator is to generate an addressing counter in
vertical raster scan format. The address generator receives two input data: width size and height size of the input frame. We designed the address generator with a finite state machine (FSM) to count address values according to the raster scan as shown in Fig. 4(b).

The control unit is also realized by an FSM to control the enable signals, EN_x, EN_y, and EN_z for offset filter and edge-preserving filter. This FSM activates the enable signals when the 8×8 sliding blocks for the offset filter and the 3×3 sliding blocks for the edge-preserving filter are available at inputs of each filter unit. Four rows and four columns of pixels at the boundary of frames are not processed by the offset filter because of the lack of information on neighbor pixels.

The timing diagram of the deblocking filter core is shown in Fig. 10. The total number of pipeline stages of the design is 192, so the delay between input and output is 192 clock cycles. The first result comes out after several clock cycles, which is indicated by RDY signal.

IV. Experiment Results

In order to verify and evaluate the algorithm and its architecture for the deblocking filter, experiments for both algorithm simulation and VLSI architecture verification have been performed. The results from the reference model,
behavior simulation, and FPGA verification were compared and evaluated together. In our experiment, we first simulated the algorithm by Matlab 7.4.0, modeled the architectures in VHDL, simulated the waveform on ModelSim 5.6, and verified the design on FPGA platform. Finally, we synthesized the design by using 0.25 μm ANAM CMOS cell library.

1. Algorithm Simulation

For the algorithm simulation on Matlab, we first experimented with a set of test images including Lena, Barbara, and Peppers. All the test images are 512×512 with 8-bit grayscale resolution. PSNR metric was used to evaluate and compare the results with some previous deblocking schemes. To evaluate the performance of our proposed algorithm, we compared our method with some other deblocking techniques using spatial-domain classification-based approaches [4]-[7]. The algorithm in [4] proposed by J. Kim and others classifies pixel blocks by computing pixel activities and then applying offset-and-shift technique. The algorithm in [5] proposed by L. Shao and others uses local entropy analysis for block classification and low-pass filters for block artifact reduction in different regions. The algorithm in [6] proposed by A.Z. Averbuch and others classifies pixel blocks into uniform and nonuniform, then applies iterations of the WABG scheme for uniform blocks and a single iteration of the deblocking frames of variable size (DFOVS) method for nonuniform blocks. The algorithm in [7] proposed by T. Chen and others uses the sum of AC coefficient energy to classify blocks into a low- or high-activity class, then applies the adaptive filter for different classes of blocks. All these methods use fixed-size block classification before applying deblocking filtering. Table 1 shows the PSNR comparative results for different images at various bit rates using some previous algorithms and our proposed method. Figure 11 shows the deblocked images of grayscale Lena512 by applying some deblocking methods. As in the experiment result, our proposed method achieves the highest PSNR values for most images and bit rates.

2. Hardware Implementation

The proposed architecture of the deblocking filter was implemented in VHDL at RTL level, simulated in ModelSim 5.8, and verified the design on FPGA platform. Finally, we synthesized the design by using 0.25 μm ANAM CMOS cell library.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena 512</td>
<td>0.25</td>
<td>30.04</td>
<td>31.48</td>
<td>31.37</td>
<td>30.91</td>
<td>31.01</td>
<td>31.22</td>
</tr>
<tr>
<td></td>
<td>0.36</td>
<td>32.95</td>
<td>33.62</td>
<td>33.48</td>
<td>32.88</td>
<td>32.85</td>
<td>33.37</td>
</tr>
<tr>
<td></td>
<td>0.46</td>
<td>34.26</td>
<td>34.60</td>
<td>34.62</td>
<td>34.52</td>
<td>33.70</td>
<td>34.51</td>
</tr>
<tr>
<td>Peppers 512</td>
<td>0.25</td>
<td>30.18</td>
<td>31.29</td>
<td>31.19</td>
<td>30.62</td>
<td>30.74</td>
<td>30.75</td>
</tr>
<tr>
<td></td>
<td>0.37</td>
<td>32.47</td>
<td>33.24</td>
<td>33.06</td>
<td>32.43</td>
<td>32.45</td>
<td>32.72</td>
</tr>
<tr>
<td></td>
<td>0.47</td>
<td>33.61</td>
<td>34.14</td>
<td>34.02</td>
<td>33.12</td>
<td>33.22</td>
<td>33.72</td>
</tr>
<tr>
<td>Barbara 512</td>
<td>0.44</td>
<td>26.99</td>
<td>27.40</td>
<td>27.38</td>
<td>26.67</td>
<td>27.11</td>
<td>27.23</td>
</tr>
<tr>
<td></td>
<td>0.54</td>
<td>28.25</td>
<td>28.55</td>
<td>28.55</td>
<td>27.71</td>
<td>28.08</td>
<td>28.23</td>
</tr>
<tr>
<td></td>
<td>0.69</td>
<td>30.16</td>
<td>30.25</td>
<td>30.28</td>
<td>30.10</td>
<td>29.33</td>
<td>29.73</td>
</tr>
</tbody>
</table>

Fig. 11. Deblocking test image grayscale Lena512 using some previous approaches and the proposed method.

Fig. 12. The 15th frame of the 100 kbps sequence “Foreman” before and after applying the proposed deblocking algorithm.
Finally, the proposed architecture was synthesized with 0.25 μm ANAM CMOS cell library by Synopsys Design Compiler. The constraint delay of the critical path is set to 6.7 ns for HD video processing. Therefore, this design can work at a clock frequency up to 150 MHz and perform deblurring filtering for 1080p video sequences. Our design was successfully synthesized with the constraints.

Table 3 shows the comparison between our design and other designs using an in-loop deblurring filter for H.264/AVC. Our algorithm is more complex than an in-loop deblurring filter. Therefore, the hardware implementation requires more resources than a conventional deblurring filter. However, the PSNR improvement of our design is higher, thus the visual quality of the output images is better.

V. Conclusion

In this paper, we proposed an edge-protection deblurring algorithm and also presented its efficient VLSI architecture for a deblurring filter. Being different from previous block classification based approaches, the proposed method classifies each pixel into smooth or edge regions by an edge detection process and generates edge-protection maps. Based on these maps, two types of filters, including an offset filter and edge-preserving filter, are applied to remove grid noise, staircase noise, and corner outlier noise. Our experiments on some standard test images show that the proposed method can reduce block artifacts more effectively than some previous approaches while preserving the detail of the image.

The hardware architecture for the proposed deblurring algorithm is implemented. The pipeline technique is applied, and parallel structure for the offset filter and edge-preserving filter is used to achieve the highest throughput for real-time video processing. Memory-reduced architecture for the block buffer is employed to minimize memory usage. The synthesis results show that the architecture can process HD resolution and verified on FPGA Cyclone II. For architecture simulation, we used the library std.textio to read and write pixel data from and to a file. The result file was then compared with the simulation result from Matlab. The FPGA verification was realized on Kit DE2-70 using FPGA EP2C70F896 at 50 MHz for VGA image frames.

The verification model is shown in Fig. 13.

Table 2. PSNR improvement for test sequences.

<table>
<thead>
<tr>
<th>Test sequences</th>
<th>Bit rate (kbps)</th>
<th>Average PSNR (dB)</th>
<th>Average PSNR improvement (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman (CIF, 300fps)</td>
<td>100</td>
<td>27.64</td>
<td>0.612</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>27.71</td>
<td>0.592</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>27.84</td>
<td>0.561</td>
</tr>
<tr>
<td>News (CIF, 300fps)</td>
<td>100</td>
<td>28.10</td>
<td>0.360</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>28.17</td>
<td>0.333</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>28.23</td>
<td>0.284</td>
</tr>
<tr>
<td>Silent (CIF, 150fps)</td>
<td>100</td>
<td>28.13</td>
<td>0.542</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>28.19</td>
<td>0.525</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>28.48</td>
<td>0.476</td>
</tr>
</tbody>
</table>

Table 3. Synthesis result of proposed deblurring filter.

<table>
<thead>
<tr>
<th>Features</th>
<th>Ref. [17]</th>
<th>Ref. [18]</th>
<th>Our design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>In loop filter</td>
<td>In loop filter</td>
<td>Edge-preserving filter</td>
</tr>
<tr>
<td>PSNR improvement</td>
<td>0 to 1 dB</td>
<td>0 to 1 dB</td>
<td>0.2 to 1.5 dB</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Non</td>
<td>Non</td>
<td>192 stage</td>
</tr>
<tr>
<td>Process (μ)</td>
<td>0.25</td>
<td>0.18</td>
<td>0.25</td>
</tr>
<tr>
<td>Cycle/MB</td>
<td>615</td>
<td>250</td>
<td>512</td>
</tr>
<tr>
<td>Frequency required for HD at 30fps (MHz)</td>
<td>300</td>
<td>120</td>
<td>150</td>
</tr>
<tr>
<td>Memory (bytes)</td>
<td>640</td>
<td>30,720</td>
<td>310</td>
</tr>
<tr>
<td>Gate count (k)</td>
<td>20.66k</td>
<td>19.64k</td>
<td>50.1k</td>
</tr>
</tbody>
</table>
video frames with a 150 MHz operation clock. Furthermore, our hardware implementation can be integrated in HD video post processing systems.

References


Truong Quang Vinh received the BE in electronics engineering from Ho Chi Minh University of Technology, Vietnam, in 1999, and the ME in computer science from the Asian Institute of Technologies, Bangkok, Thailand, in 2003. He is now a PhD candidate at Chonnam National University, Gwangju, Korea. His research interests include digital circuits, SoC, embedded system design, and video post-processing.

Young-Chul Kim received his PhD from Michigan State University, USA, the MS from the University of Detroit, USA, and the BS in electronics engineering from Hanyang University, Korea. In 1993, he joined the Department of Electronics Engineering at Chonnam National University (CNU) where he is currently a professor. From 2000 to 2004, he was a director of IDEC at CNU. From 2004 to 2005, he was a Vice Dean of the College of Engineering in this university. Since 2004, he has become the chief of the LG Innotek R&D center at CNU. His research interests are SoC design and development using IPs and low power design.