A High-throughput, Area-efficient Hardware Accelerator for Adaptive Deblocking Filter in H.264/AVC

Muhammad Nadeem[1], Stephan Wong[1], Georgi Kuzmanov[1], Ahsan Shabbir[2]
[1] Delft University of Technology, Delft, The Netherlands
{M.Nadeem, J.S.S.M.Wong, G.K.Kuzmanov}@tudelft.nl
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
A.Shabbir@tue.nl

Abstract*—In this paper, we present a high-throughput, area-efficient, hardware accelerator for the deblocking filter in H.264/AVC video compression standard. In order to achieve this goal, we start with algorithmic optimization and propose a novel decomposition of the filter kernels for the deblocking filter. The proposed decomposition reduces the number of adders by 51% and thereby greatly reduces the area requirement for its implementation. Subsequently, at architecture level, while using two identical filtering units, the transpose units are realized by efficient reuse of hardware resources to further reduce the area requirement. The two filtering units process the horizontal and vertical edges of the macro-block simultaneously and therefore further enhance the throughput of the hardware accelerator. Several other optimization techniques, such as reuse of intermediate results, pipelining, and merging of processing blocks on critical path, result in a hardware accelerator for deblocking filter with high throughput at one hand and less area in terms of equivalent gates count on the other, when compared with existing state-of-the-art hardware accelerators in the literature. While working at clock frequency of 166 MHz, synthesized under 0.18 μm CMOS standard cell technology, it easily meets the throughput requirements of all the levels in H.264/AVC video coding standard and consumes only 12.06 K gates (excluding SRAM).

I. Introduction

The latest video coding standard H.264/AVC [4], jointly developed by ITU-T and ISO/IEC MPEG, significantly outperforms previous video coding standards (like H.263 and MPEG-2/4) in terms of bit-rate reduction. H.264/AVC offers perceptually the same video quality with at least 2 times better compression when compared with MPEG-2 [1], [2], and up to 30% better compression when compared with H.263+ and MPEG-4 Advanced Simple Profile (ASP) [3]. The H.264/AVC video coding standard has already been adopted for wide range of applications, from low bit-rate mobile video to high-definition TV.

The block diagram of the H.264/AVC encoder is depicted in Figure 1. The improved encoding efficiency achieved in H.264/AVC is not a result of any single feature in the new standard, rather it is a combination of a number of advanced coding tools at the expense of increased complexity. Multi-mode intra-prediction, integer discrete cosine transform (DCT), multi-frame variable-block-size inter-prediction with up to quarter pixel accuracy, context adaptive binary arithmetic

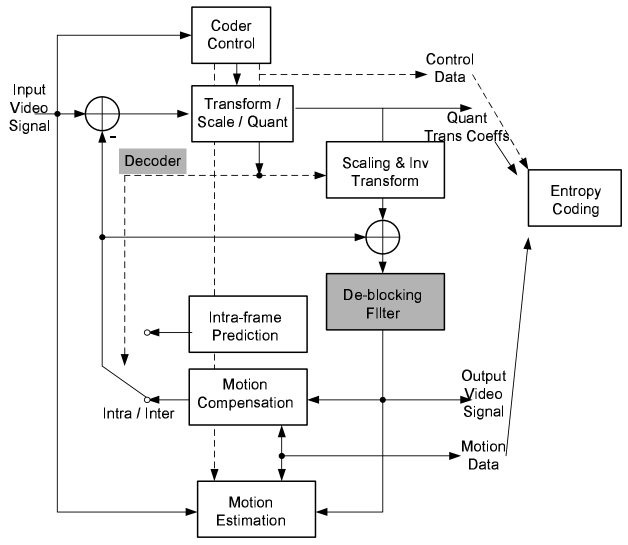


Fig. 1. Block diagram of H.264/AVC Encoder/Decoder

coding (CABAC) and deblocking filter are some of these tools in H.264/AVC.

Like other block-based video coding standards, the H.264 also suffers from the blocking artifacts in the reconstructed video. The block-based discrete cosine transform (DCT) and block-based motion compensation (MC) are two major sources of these blocking artifacts [5], [6]. The H.264/AVC uses an adaptive in-loop deblocking filter to remove such artifacts in the reconstructed video frame.

Since the deblocking filter is in the encoding loop of the H.264 video encoder, it does not only provide the perceptually improved video quality by removing the blocking artifacts, but also helps to reduce the bit-rate typically between 5-10% [7]. However, this improvement in video quality and reduction in the video bit-rate is achieved at the cost of increased complexity of the deblocking filter algorithm. According to the analysis of run-time profile of the H.264 decoder sub-functions, the deblocking filter consumes about one-third of the computational resources [8]. These demanding characteristics suggest a hardware implementation for such a deblocking filter for high definition video applications, where even larger frame sizes at higher frame rates are to be processed in real-time.

Several different hardware accelerators have been presented in the literature during the last few years for efficient hardware realization of the deblocking filter in H.264/AVC video standard. Most of these hardware accelerators use single filter unit to carry out the filtering operations in both directions (horizontal and vertical). This approach though requires less area but mostly fails to meet the throughput requirement for the real-time processing of high definition video (4096×2304, 16:9). Whereas the solutions based on multiple filtering units, provide better throughput at the cost of additional area. Yet, most of them do not meet the real-time processing requirement of all levels (level 1-5.1) offered by the video coding standard H.264/AVC. With this work, we address the above problems and provide the following specific contributions:

- The deblocking algorithm is further optimized through decomposition of filter kernels and the number of additions are reduced by 51%, when compared with that of original filter equations.
- Identification of common inter filter mode operations and a proposal for common data paths for both the filtering modes (strong filter mode and weak filter mode).
- Area-efficient design of transpose units achieved by aggressive reuse of on-chip hardware resources (memory modules).
- Efficient pipeline stage design to reduce the number of storage registers for intermediate results between the pipeline stages.
- Optimizations to reduce the critical path by merging certain processing blocks. This enables us to achieve significantly higher throughput and meet the real-time processing requirement of all levels (level 1-5.1) in video coding standard H.264/AVC.

It is worth mentioning that the increase in throughput is not at the cost of additional on-chip area. In most cases, it requires less area in terms of equivalent gate count when compared with the existing state-of-art hardware accelerators for the deblocking filter.

The organization of the paper is as follows. Related work is presented in Section II. A brief overview of the deblocking filter algorithm followed by the algorithm level optimizations is provided in Section III. Section IV describes the top level design, the data flow and the optimizations carried out at this level. Internal architecture details and related optimizations for throughput improvement and area reduction are provided in Section V. Results are discussed in Section VII and Section VIII concludes this paper.

II. Related Work

A large number of deblocking filter accelerators utilize a single filter unit. For instance, in [9], Shih et. al., propose a 5-stage pipelined hardware architecture for the deblocking filter. They employ a novel filtering order and data reuse strategy to reduce the number of cycles, memory traffic and required area for their implementation. In [13], the authors rearrange the data flow to significantly reduce the memory size requirement and propose an in-place architecture which re-uses the intermediate data as soon as it is available and therefore are able to reduce the intermediate data storage to four 4×4 blocks instead of a complete 16×16 macro-block (MB). Similarly, Chang [11] also reorders the computing flow to efficiently utilize the intermediate data between adjacent edges.

The hardware architecture in [15] implements a parallel-in parallel-out reconfigurable FIR filter to carry out the filtering operations and employs a dual-ported SRAM for intermediate data storage. Li, et. al., [10] adopt a 2-dimensional parallel memory scheme for parallel access in both the horizontal and vertical directions to speed up the filtering process and to eliminate the need of a transpose circuitry by using this memory scheme efficiently.

A hybrid filter scheduling (described in [12]) reduces the required number of clock cycles for filtering and therefore improves the system throughput while using the column-of-pixel data arrangement to facilitate the memory accesses and reusing the pixel value. A 5-stage pipelined architecture proposed in [16] for simultaneous processing of strong and weak filtering modes uses a novel transpose design to reduce the hardware cost and an alternate processing order of vertical and horizontal edges to reduce the on-chip memory requirement.

The area requirement for some of these hardware accelerators [10], [11], [20] is low due to their reduced functionality as these architectures do not implement boundary strength (BS) computation module.

Some of the hardware solutions based on multiple filter units have been introduced in the literature quite recently. These solutions provide higher throughput at the cost of additional on-chip area requirement, but still most of these solutions do not meet the throughput requirements of all the levels (level 1-5.1) offered by the Video Codec H.264/AVC. For instance, F. Tobajas [19] proposed a hardware architecture based on a double-filter strategy, and use a raster scan filtering order. Cheng [14], [17] proposes a configurable window-based architecture to simultaneously filter in both the directions. The main idea is to reduce the number of memory references through simultaneous processing architecture (SPA) using the vertical processing order instead of the raster scan order. A similar architecture is proposed in [18] by Venkatraman, et. al..

Some efforts have been made to optimize the filter kernels by removing redundant operations, however the strong filter mode is the focal point in most of these attempts. For instance, in [22], the author suggests 3 different decompositions of the filter kernels in the strong filter mode to reduce the number of addition operations. The author however, does not consider similar decompositions for weak filter mode.

In short, single filter unit based solutions require less area in terms of equivalent gate count fort their implementation but fail to meet the processing requirements of high definition video in real-time. In some cases, area reduction is achieved through reduced functionality offered by these hardware accelerators. The solutions based on multiple filtering units, on the other hand, though provide better throughput at the cost of additional area but still do not meet the real time processing requirement of all the levels offered by the video coding

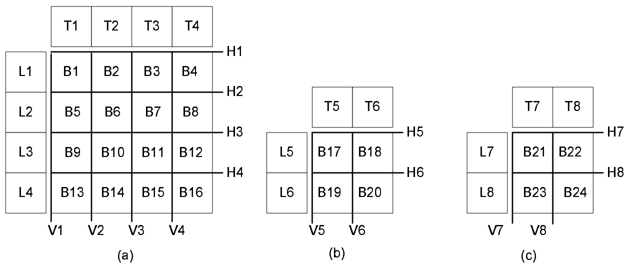

T1 T2 T3 T4
L1 L2 L3 L4
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 B16
H1 H2 H3 H4
V1 V2 V3 V4
(a)
T5 T6 L5 L6 B17 B18 B19 B20 H5 H6 V5 V6
(b)
T7 T8 L7 L8 B21 B22 B23 B24 H7 H8 V7 V8
(c)

Fig. 2. Vertical and Horizontal 4x4 block edges in a MB (a) Y component of MB (b) U component of MB (c) V component of MB

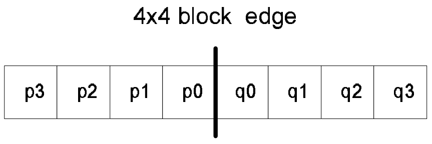


Fig. 3. Convention for describing the pixels across Vertical/Horizontal edge

standard H.264/AVC. We present a high-throughput hardware accelerator for deblocking filter to meet the processing requirement of all the levels in H.264/AVC while keeping the area requirement as low as possible. With our design we address the above discussed problems of related works and provide several optimal design solutions for them.

III. The H.264/AVC Deblocking Filter Algorithm

This section provides a brief overview of the adaptive deblocking filter algorithm in H.264/AVC. The detailed description of the algorithm can be found in [4].

The filtering operation is performed on macro-block (MB) basis after reconstruction of the picture, with all MBs in a picture processed in order of increasing MB address. The filtering is applied to all the 4×4 block edges except the edges at the boundary of the picture or for which the filtering is disabled explicitly. The filtering process is invoked for luma and chroma components of the MB separately. For each MB and for each component, vertical edges are filtered first starting with the left most edge and proceeding through the edges in their geometrical order. The horizontal edges are filtered afterwards in a similar fashion, starting with the top most edge and proceeding through the edges in their geometrical order [4] as depicted in Figure 2. The filtering process also requires pixels from left and top neighbor macro-blocks in order to filter the left most edges (V1, V5 and V7) and the top most edges (H1, H5 and H7), respectively. A macro-block at 4×4 block level is depicted in Figure 2. Blocks B1 through B16 belong to Y (luma) component of the current MB whereas B17-B20 and B21-B24 are from U, V (chroma) components of the same MB respectively. The blocks (T1-T8) are the top neighbor 4×4 blocks and (L1-L8) represent the left neighbor 4×4 blocks of a MB in Figure 2. The convention to describe the pixels across the horizontal and vertical edge is depicted in Figure 3. The bold line between pixels p0 and q0 is either a vertical or horizontal edge between two adjacent 4×4 blocks. Pixels q0-q3 represent the pixels in the current 4×4 block where as pixels p0-p3 are from corresponding left

```
(a)     Pixel Level Filter Control Flags
FSF = (Bs ≠ 0) And (|p0-q0|<α) And(|p1-p0|<β) And (|q1-q0|<β)  (1)
SFF_P = (Luma edge) And (|p2-p0|<β) And (|p0-q0|<((α/4)+2))    (2)
SFF_Q = (Luma edge) And (|q2-q0|<β) And (|p0-q0|<((α/4)+2))    (3)
WFF_P = (Luma edge) And (|p2-p0|<β)                            (4)
WFF_P = (Luma edge) And (|q2-q0|<β)                            (5)
```

```
(b)     Strong Filter (BS= 4)
p0' = (p2 + 2*p1 + 2*p0 + 2*q0 + q1 + 4) >> 3                  (6)
p1' = (p2 + p1 + p0 + q0 + 2) >> 2                             (7)
p2' = (2*p3 + 3*p2 + p1 + p0 + q0 + 4) >> 3                    (8)
p0'' = (2*p1 + p0 + q1 + 2) >> 2                               (9)

q0' = (p1 + 2*p0 + 2*q0 + 2*q1 + q2 + 4) >> 3                  (10)
q1' = (q2 + q1 + q0 + p0 + 2) >> 2                             (11)
q2' = (2*q3 + 3*q2 + q1 + q0 + p0 + 4) >> 3                    (12)
q0'' = (2q1 + q0 + p1 + 2) >> 2                                (13)
```

```
(c)     Weak Filter (BS= 3)
delta = clip(-c1, c1, ((((q0-p0)<<2) + (p1-q1) +4)>>3))        (14)
p0' = clip(0,255,p0 + delta)                                   (15)
q0' = clip(0,255,q0 - delta)                                   (16)

p1' = p1+clip(-c0,c0,(p2+((p0+q0+1)>>1)-(2*p1))>>1)            (17)
q1' = q1+clip(-c0,c0,(q2+((p0+q0+1)>>1)-(2*q1))>>1)            (18)
```

Fig. 4. (a) Pixel level filter controls flags Eqs. (b) Strong Filter mode Eqs.(c) Weak Filter mode Eqs.

or top neighbor 4×4 block across the vertical or horizontal edge respectively.

The H.264/AVC deblocking filter is a highly adaptive filter. It adapts at slice level, 4×4 block edge level and set of pixels level within a 4×4 block. At the slice level, a set of threshold parameters control the filtering operation, whereas at the block edge level, the filtering strength is computed on the basis of parameters, e.g., encoding mode, motion vector difference and coded residual of 4×4 block. The boundary strength (BS) parameter controls the filtering strength at 4×4 block level and varies from 4 (strong filtering) to 0 (no filtering). For BS values 1, 2 and 3, a weak filtering process is invoked on the pixels across the block edge. The decision process to compute the BS value is given in [4].

The selected filtering mode is turned ON or OFF depending upon the value of Filter Sample Flag (FSF). The FSF is derived using Eq. (1) in Figure 4(a). The value of FSF is based on the local edge information and a set of quantization parameters dependent on thresholds α(QP) and β(QP).

In strong filter mode (BS=4), at most 3 pixels are modified on either side of the edge. The new pixel values for the pixels on left or top side of the edge (p-pixels) are computed using Eqs. (6-8) Figure 4(b), provided the strong filter flag for p-pixels (SFF_P) is set. If the SFF_P flag is not set, only one pixel (p0) is filtered and the new value for this pixel is derived by Eq.(9). Similarly, the pixels in the current block (q-pixels) are computed using Eqs. (10-12) Figure 4(b), provided the corresponding strong filter flag (SFF_Q) is set. In case SFF_Q flag is not set, only one pixel (q0) is filtered using Eq. (13) and rest of the pixels remain unchanged. The values for SFF_P and SFF_Q flags are derived from Eq.(2) and Eq.(3), respectively.

In the weak filter mode (BS=1,2 or 3), at most 2 pixels on either side of the edge are filtered. The new filtered values for the pixels p0 and q0 across the edge are derived from Eq. (15) and Eq. (16) respectively, whereas Eq. (17) and Eq. (18) are used to provide the filtered values for pixels p1 and q1, if the corresponding weak filter flag (WFF_P, WFF_Q) is set. In

Strong Filter (BS= 4)

p0' = (u1 + u5) >> 3	(19)
p1' = (u1) >> 2	(20)
p2' = (u1 + 2*t1) >> 3	(21)
p0'' = (u3 + t3) >> 2	(22)
q0' = (u2 + u5) >> 3	(23)
q1' = (u2) >> 2	(24)
q2' = (u2 + 2*t2) >> 3	(25)
q0'' = (u3 + t4) >> 2	(26)

Weak Filter (BS= 3)

delta = Clip(-c1, c1, ((4*u0 + 3) + u3 + 1)) >> 3)	(27)
p0' = Clip(0,255,p0 + delta)	(28)
q0' = Clip(0,255,q0 - delta)	(29)
p1' = p1+clip(-c0,c0,(u1)>>2)	(30)
q1' = q1+clip(-c0,c0,(u2)>>2)	(31)

Where

u1 = (2*p2 + p0 + 1) + (q0 – 4*p1) When Bs ? 4 and	(32)
u1 = (p2 + p0 + 1) + (q0 + p1 + 1) When Bs = 4	(33)
u2 = (2*q2 + q0 + 1) + (p0 – 4*q1) When Bs ? 4 and	(34)
u2 = (q2 + q0 + 1) + (p0 + q1 + 1) When Bs = 4	(35)
u3 = p1 – q1 When Bs ? 4 and	(36)
u3 = (p1 + q1 + 1) When Bs = 4	(37)
u4 = p0 + q0 + 1	(38)
u5 = u3 + u4	(39)
t1 = (p3 + p2 + 1)	(40)
t2 = (q3 + q2 + 1)	(41)
t3 = p1 + p0 + 1	(42)
t4 = q1 + q0 + 1	(43)

Fig. 5. Decomposed filtering equations for Strong and Weak filter modes

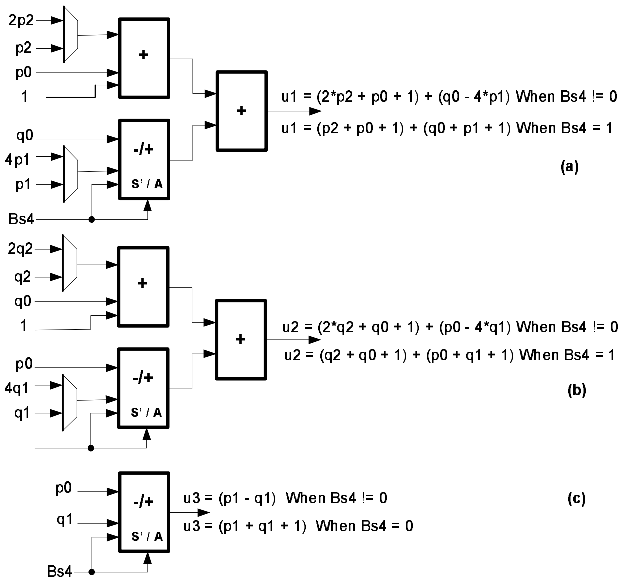
case the weak filter flag is not set, the filtering operation for these pixels is turned OFF. The values of WFF_P and WFF_Q are derived from Eq. (4) and Eq. (5), as shown in Figure 4(a).

No filtering is applied for BS = 0.

Algorithm Level Optimizations for Deblocking Filter: In strong filter mode, the filtered pixel values are derived from Eqs. (6)-(13) as shown in Figure 4(a). Similarly the filtered pixel values for weak filter mode are derived from Eqs. (14)-(18). It is clear from these equations that the strong filter mode requires 36 additions; whereas 13 additions and 5 clip operations are required for the weak filter mode. This results in *49 additions and 5 clip operations* in all while excluding the operations needed to compute the pixel level filter flags.

As mentioned in the related work, the author of [22] suggested 3 different decompositions of the filtering equations in the strong filter mode to reduce the number of operations. The author, however, did not consider the reduction in number of operations for the weak filter mode. The suggested decomposition with least number of operations in [22] requires 22 addition in case of strong filter mode. Another 13 additions along with 5 clip operations are, therefore, required for weak filter mode case. This results in *35 additions and 5 clips operations* to perform the filtering. Similarly the deblock filter design for the strong filter mode proposed in [24] requires 23 additions. Therefore, *36 additions and 5 clip operations* are required to implement this design for both fitering modes in H.264/AVC.

Decomposition of filter kernels: An important contribution of this paper is the removal of redundant operations in both the filtering modes through novel decomposition of filtering Eqs. (6)-(18) in Figure 4 into a set of modified filtering Eqs. (19)-(43) given in Figure 5. Note that the rounding constants in original filter Eqs. (6)-(14) are efficiently distributed among



2p2
p2
p0
1
q0
4p1
p1
Bs4
+
-/+
S' / A
u1 = (2*p2 + p0 + 1) + (q0 - 4*p1) When Bs4 != 0
u1 = (p2 + p0 + 1) + (q0 + p1 + 1) When Bs4 = 1
(a)
2q2
q2
q0
1
p0
4q1
q1
u2 = (2*q2 + q0 + 1) + (p0 - 4*q1) When Bs4 != 0
u2 = (q2 + q0 + 1) + (p0 + q1 + 1) When Bs4 = 1
(b)
p0
q1
Bs4
u3 = (p1 - q1) When Bs4 != 0
u3 = (p1 + q1 + 1) When Bs4 = 0
(c)

Fig. 6. Overlapped data paths for u1, u2 and u3 in Strong and Weak Filter modes

several intermediate results in the form of $x + y + 1$. This operation, however, requires only one adder for its hardware implementation. The term $4u0 + 3$ in Eq. (27) can be realized by appending bits '11' as least significant bits in $u0$ and therefore do not require any additional adder. Similarly, re-use of intermediate results significantly reduces the number of additions. The proposed decomposition requires only 31 additions.

Inter filter mode optimization: Since both the filtering modes are mutually exclusive, only one of them is used for filtering of pixels data across the 4x4 block edge at any time. Therefore, inter-filter-mode redundancy can be removed by designing the data paths for Eqs. (32)-(37) in such a way that they overlap as much as possible in terms of arguments for processing units(adders in this case). The proposed realization of Eq. (32)-(37) using overlapped data paths is illustrated in Figure 6(a)(b)(c). The inter filter mode optimization reduces another 7 adders at the cost of only 4 multiplexer modules. The suggested decomposition in this paper along with inter filter mode optimization, therefore, requires only 24 additions and 5 clip operations. Hence the number of additions are reduced by 51% when compared with that of original filtering equations in [4], by 31%, when compared with decomposition suggested in [22], and by 33%, when compared with [24]. A comparison of number of operations required to carry out the filtering process by [4], [22], [24] and the proposed in this paper, is given in Figure 8. In Section VI, we further elaborate on how these overlapped data paths help to reduce the number of intermediate storage registers required between the two pipeline stages.

IV. The Hardware Accelerator Organization

In this section, different building blocks that constitute the top level design of the hardware accelerator are first introduced with a brief rationale for each of them. Subsequently, the pixel

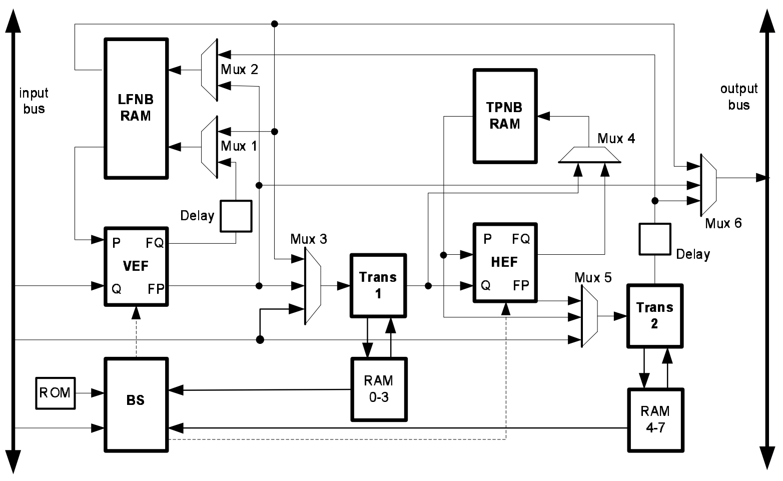


Fig. 7. High level organization of the proposed deblock filter accelerator.

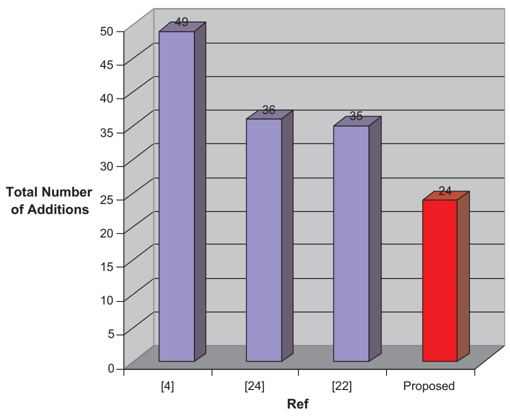


Fig. 8. Comparison: Addition operations in Strong and Weak filter modes

data flow, at 4×4 block level, is explained to provide an insight in the functioning of the hardware accelerator.

The block diagram of the deblocking filter hardware accelerator is depicted in Figure 7. All solid line data paths are 32-bit wide (4 pixels) and the dotted lines represent the control signals. This accelerator is based on two identical filter units Vertical Edge Filter (VEF), Horizontal Edge Filter (HEF), two transpose units (Trans1, Trans 2), Boundary Strength computation unit (BS) and Left/Top neighbor RAM units (LFNB RAM, TPNB RAM). The VEF unit processes the pixel-rows across the vertical block edge in horizontal direction, whereas the HEF unit processes the pixel-columns across the horizontal block edge in vertical direction to remove the blocking artifacts in the MB. The LFNB RAM stores the pixel-rows of the left neighbor 4×4 blocks (L1)-(L8) for the vertical edge filtering process in VEF. Similarly The TPNB RAM stores the pixel-columns of the top neighbor 4×4 blocks (T1)-(T8) for the horizontal edge filtering process in HEF.

During the filtering process of the internal block edges, partially filtered pixels of the current 4×4 block are used as Left/Top neighbor pixels for the next 4×4 block along horizontal/vertical direction respectively. Therefore the LFNB/TPNB RAM units also serve as a temporary storage to hold the partially filtered intermediate pixels results.

The BS unit computes the boundary strength for all the 4×4 block edges in a MB in both directions. The required configuration data, e.g., encoding type, motion vectors and quantization parameters of 4×4 blocks for the computation of BS value is stored in the RAM units as depicted in Figure 7. The RAM unit is composed of two 8 × (4×8-bit) dual-ported SRAMs and each one of these is used to store either top or left neighbor block configuration data during boundary strength computation process. The BS unit provides the computed boundary strength values along with the filter control thresholds (α, β, Tc0), stored in the ROM table, to the VEF and HEF filtering units.

The "Trans1" unit, at the input of HEF, transposes the incoming pixel-rows of the 4×4 block into pixel-columns. The other transpose unit "Trans2", at the output of the HEF, converts the filtered pixel-columns back to pixel-rows before sending these filtered pixels back to the external picture buffer through the output bus.

Data Flow: Before starting the filtering process for any MB, the configuration data and the top neighbor 4×4 blocks(T1)-(T8) are first transferred from the external memory to the hardware accelerator. During this phase, the BS unit computes the boundary strength for all the block edges in the current MB. The pixel data of luma component for the current MB follows the configuration data and is transferred in raster scan order on 4×4 block level (B1, B2, .., B16). The filtering phase starts as soon as the first pixel-row of block B1 arrived at the input of VEF filter unit. The input pixel data is first filtered for vertical block edges in the VEF unit and later by HEF unit for the block edges in the horizontal direction. The chroma 4×4 blocks (B17 B18, ..., B24) follow the luminance blocks and are filtered in the same fashion.

Once the filtering operation is completed in both directions by VEF and HEF units for the luma/chroma blocks, these blocks are transferred to the external picture buffer via the output bus. The data flow at block level is depicted in Figure 9. The VEF(Q-input) is always provided with the input blocks (B1, B2, B3, ... , B24) from the external unfiltered picture buffer. The corresponding left neighbors (L1, B1, B2, B3, L2, ...) are fed from the LFNB RAM unit into the VEF (P-input)

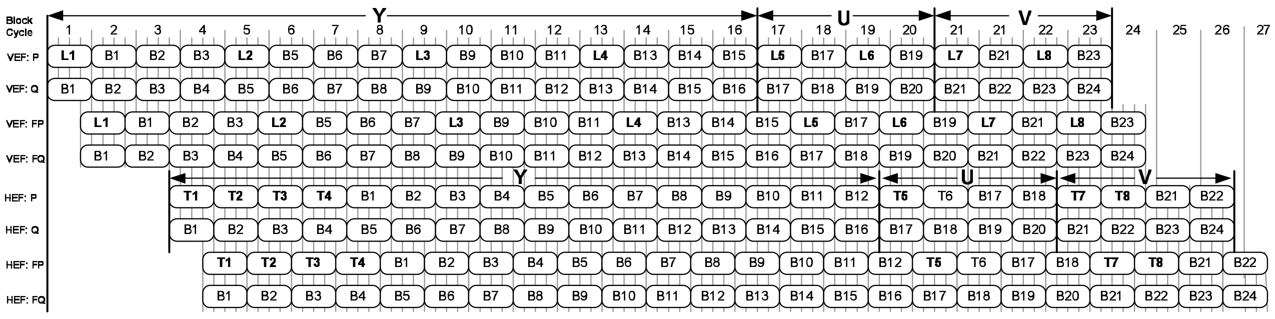


Fig. 9. 4×4 block level data flow through the VEF, HEF units in the proposed hardware architecture.

filter unit.

The partially filtered blocks (B1, B2, B3, B4,..., B24) from VEF (Q-output) are always stored in the LFNB RAM and are used as left neighbor block in the next filter block cycle. The filtered blocks (B1, B2, B3, B5..., B23), after completion of the vertical edge filtering operation from VEF unit, are sent to the next filtering unit HEF for horizontal edge filtering operation. The last 4×4 block in each block row of the current MB (B4), (B8), (B12) temporarily stored in the LFNB RAM, are also sent to the HEF unit via the same data path before start of next block row. Hence the HEF unit also receives the 4×4 blocks of current MB, after being filtered in the horizontal direction by the VEF unit, in the same sequence as that of VEF-unit (Q-input), as shown in Figure 9.

The last column of 4×4 blocks (B4), (B8), (B12), (B16), (B18), (B20), (B22) and (B24) in the current MB is stored in the LFNB RAM as (L1)-(L8) for next macro-block after completion of the filter operation in both directions. The blocks in the last luma/chroma block rows (B13), (B14), (B15), (B17), (B19), (B21), (B23), temporarily stored in the TPNB RAM module are sent to the external picture buffer. These blocks are transposed from pixel-columns to pixel-rows orientation during the transfer process by the "Trans2" unit. The left neighbor blocks of the previous MB are directly sent from LFNB RAM.

When the vertical filtering process is completed for the current MB, the transfer of the configuration data and the top neighbor blocks (T1)-(T8) in the next MB is initiated, in parallel with the processing of HEF unit for current MB, to maximize the throughput of the deblock filter.

V. Architecture Level Optimization

Our deblock filter hardware accelerator utilizes two identical filter units to enhance its throughput. However, because of identical filter units, two transpose units are required to be introduced. One transpose unit each before and after the HEF filter unit as depicted in Figure 7. These transpose units are an overhead of such an arrangement and cost significant amount of area for their implementation. Different techniques have been used in the literature to reduce the area requirement of the transpose units. All of these techniques, in one way or another, require a separate temporal register file for its realization.

We, however, follow a different approach in our design to minimize this overhead. We identified that the transpose

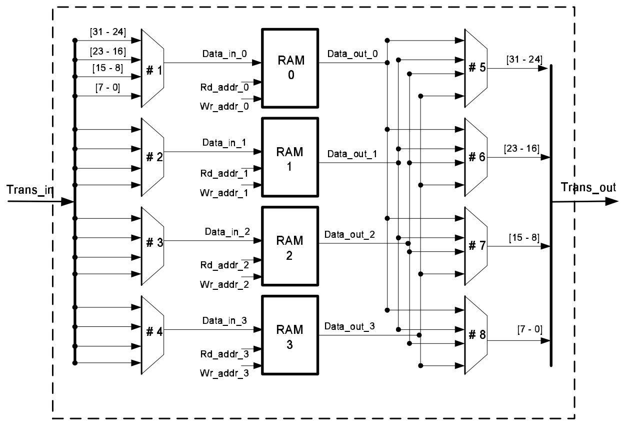


Fig. 10. Functional block diagram of Transpose unit

process and the boundary strength computation are two mutually exclusive set of operations. The storage used for BS parameters is a free resource during the transpose phase. Therefore, in our design we re-use the RAM locations of the BS parameters as a storage during the transpose phase. This saves us precious area when compared to the use of dedicated register files, during the transpose phase.

The transpose unit in our design, does not require any separate temporal buffer for its implementation. It rather uses the same RAM units used by the BS unit for temporary storage of pixel-rows of the 4×4 block during the transpose process. This architecture level optimization requires some pre- and post-storage pixel data re-arrangements for implementation of the transpose unit. The design and implementation details of transpose units using configuration RAM units is explained in the remainder of this section.

Transpose unit implementation: The transpose unit consists of 2 sets of 4 multiplexers connected to the RAM units as shown in Figure 10. RAM units are shared between the transpose and BS units to serve as temporary storage location. The cost of a register file consisting of 14 32-bit registers [19] is about 3.3k gates whereas the multiplexers depicted in Figure 10 cost only 500 gates (implemented on 0.18 μm CMOS standard cell technology). This optimization delivers area saving. The transpose mechanism is explained in the following.

The transpose unit takes 8 cycles to complete the transpose process of a 4×4 block. In the first 4 cycles, block "n" is stored at "set 1" address locations (0, 1, 2 and 3). Before storage, the pixel-rows of the block “n” are arranged in a manner that no two pixels of any pixel-column of the transposed 4×4 block shall be in the same storage RAM unit. This is achieved through multiplexers (Mux1)-(Mux4) at the input of these storage RAMs as shown in Figure 10. The control signals for these multiplexers and the data stored in the RAM units are depicted in Figure 11(a). In the next 4 cycles, block “n+1” is stored after the same pixel re-arrange process at “set 2” address locations (4, 5, 6 and 7), whereas the pixels of the previously stored block “n” at “set 1” address locations are read from the storage RAMs using appropriate addresses, as depicted in Figure 11(b).

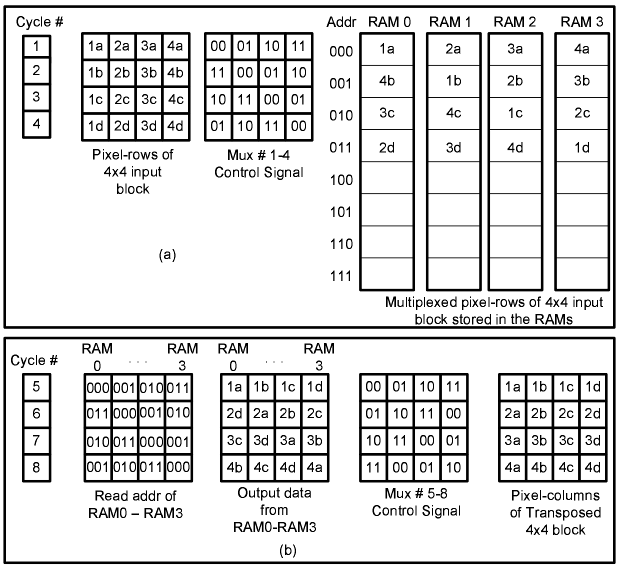


Fig. 11. Transpose mechanism for 4x4 block

These pixels are subsequently re-positioned to form the pixel-columns of the 4×4-transposed block using the multiplexers (Mux5)-(Mux8) at the output of the storage RAM units. Similarly, in the next 4 cycles, block “n+2” is stored at “set 1” address locations where as the block “n+1” is read from “set 2” and so on. Figure 11(a) depicts the the control signal of Mux1 to Mux4 during cycles c1 to c4 and the re-arranged data stored in the RAM units. The read address for the RAM units along with the corresponding data read is depicted in Figure 11(b). The control signals for multiplexers (Mux5)-(Mux8) and output of the transpose unit are also depicted in Figure 11(b). During BS computation phase, multiplexers (Mux1)-(Mux8) controls are set such that no re-positioning of the pixel data is done at both ends of the RAM units.

VI. Vertical/Horizontal Filter Unit Design

This section describes the internal architecture of the VEF/HEF unit and the corresponding optimizations employed at this level. Since all filtering operations are carried out in these units, therefore, it occupies more than two third of the area of the hardware accelerator. The architecture of this unit and choices made for its implementation play an important role in determining the throughput and over all area requirements for the deblock filter hardware accelerator.

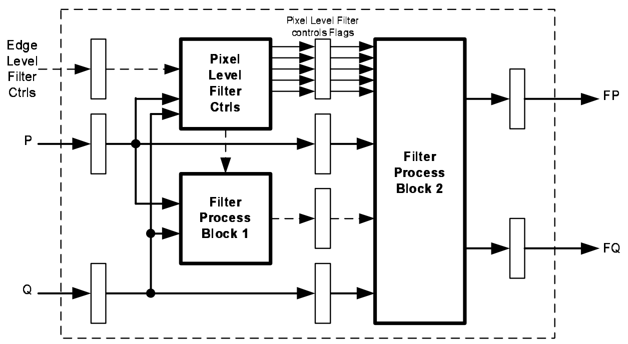


Fig. 12. Block diagram of Vertical, Horizontal Filter units

The block diagram of this unit is depicted in Figure 12. The architecture of this unit is based on pipeline processing; Pixel Level Filter Controls block and Filter Process Block 1 constitute the first pipeline stage whereas Filter Process Block 2 is the second pipeline stage of this unit.

Efficient Pipeline Design: The intermediate results storage registers between pipeline stages cost a significant amount of area. To minimize this overhead, we divided the optimized filter Eqs. (19)-(43) into two sets of equations. The first set, Eqs. (32)-(37), consists of common operations between both the filter modes and is implemented in Filter Process Block 1, The remaining filter operations are part of the second set of equations and are implemented in Filter Process Block 2. This results in only 6 intermediate results (u1, u2 and u3 in Eqs. (32)-(37)) for both the filter modes. This number is further reduced to 3 intermediate results due to the fact that both the filter modes (strong and weak filter modes) are mutually exclusive as depicted in Figure 6(a)(b)(c).

Similarly, the Pixel Level Filter Controls block computes the flags (FSF, SFF_P, SFF_Q, WFF_P and WFF_Q in Fig 4(a)), based on the current pixel values and the edge level filter thresholds (α, β) provided by the BS unit. We design the pipeline stages such that all the processing for determining the pixel level filter controls(Eqs. (1)-(5) in Figure 4(a)) is in first pipeline stage. This design choice requires only 5 one-bit flags, instead of 5, 10-bits intermediate storage registers (Fig 4a) between the two pipeline stages. Please note that this design choice results in a longer processing chain for computation of filter control flags and is on the critical path in our design. Therefore, the maximum operating frequency for our hardware accelerator is determined by the choice of implementation of pixel level filter control block. Implementation level optimization for critical path, explained below, enabled us to achieve 166 MHz operating frequency.

Moreover, we also employed the intermediate results re-use strategy in parallel computational blocks (in same pipeline stage). For instance, ($u0 = q0 - p0$) is computed in Pixel Level Filter Control block to determine the FSF (Eq. (1)) and is also in the Filter Process Block1 to compute “delta” (Eq. (27)) for filtering process in weak filter mode.

Hence the efficient pipeline design, along with our proposed decomposition, not only reduces the area required for combinational logic because of less number of operations. It also reduces the sequential logic area, at the same time, as only 3

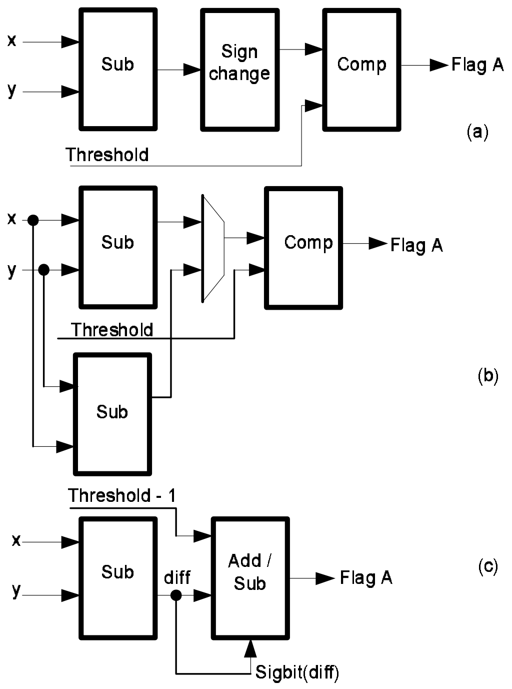


Fig. 13. Implementation of basic building block in Pixel Level Filter Control block

registers and 5 one-bit flags are required to be stored between the two pipelines stages.

Critical Path Optimization: The computation of filter control flags is on the critical path for our deblock filter hardware design. As part of optimizations for speed, we merged the processing blocks on the critical path to reduce the number of sequential operations and therefore were able a achieve a significantly higher throughput that meet processing requirement of all the levels in H.264/AVC. Optimization employed to reduce the critical is explained in the following paragraphs.

The equations to derive the values for filter control flags are provided in Figure 4a. The common operations in the computation these flags can be written as follows

Flag A = Abs(x $-$ y) $<$ Threshold ? 1 : 0.

where x and y represent the pixel values.

The straightforward implementation is depicted in Figure 13(a) and consists of following operations;

1) compute difference.
2) change sign of difference result, if negative.
3) and finally determine the flag A by comparing the absolute difference with threshold.

This choice of implementation is composed of three sequential operations. One of the possible modification shall be to compute the two difference results (x-y) and (y-x) in parallel, as depicted in (Figure 13(b)), and then select the positive difference as an absolute difference result using a multiplexer, for the subsequent comparison operation. This implementation shall help to achieve a higher clock frequency in comparison to the previous case, as the multiplex operation is more cost effective than sign change operation in terms of speed, but off course on the cost of additional "Sub" component.

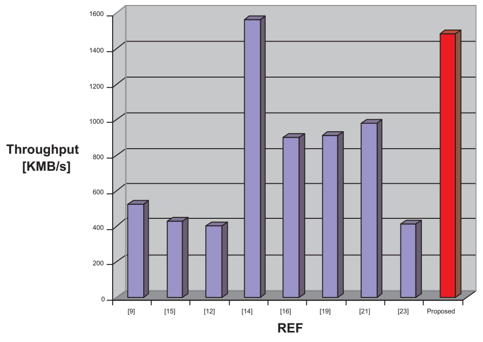


Fig. 14. Throughput comparison to related work

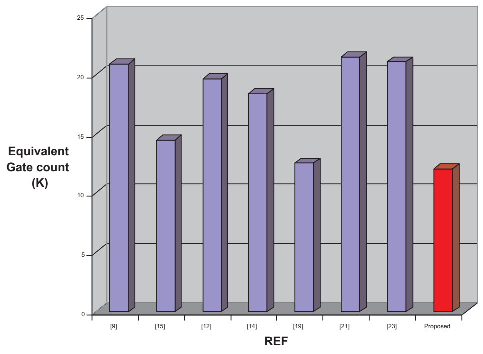


Fig. 15. Area comparisons to related work

In our implementation, we removed the absolute difference computation stage and merged it into the comparison stage by using the "Add/Sub" component (Figure 13(c)). The sign bit of the difference result determines the Add/Sub control. Subsequently, the sign bit of the result of "Add/Sub" component determines the status of the flag. This results in a further reduction in the sequential operations without any additional operation in parallel with the first stage and, therefore, helps to achieve rather even higher clock frequency (166 MHz). One requirement for such an implementation is that now we need to provide threshold value that is one less than the actual value. Since these threshold values are loaded from the ROM table and are only used for the computation filter control flags, therefore, we can either store these modified threshold values in the ROM table or this operation of modification can be implemented in the BS unit. In either case, the threshold modification shall not be on the critical path anymore, resulting in less sequential operations and therefore a higher clock frequency.

VII. Experimental Results

The deblock filter hardware accelerator presented in this paper was implemented in VHDL and synthesized by Synopsys Design Compiler (version v2002, rev 05), for a maximum clock frequency of 166 MHz with 0.18 μm CMOS standard cell technology (v1.5).

TABLE I
COMPARISON TO RELATED WORK

Ref.	Process [μm]	Filtering cycle/MB	Freq. [MHz]	SRAM requirements (bits)	Thr. MB/s	Equivalent gate count	Throughput Improvement [%]	Gate count reduction [%]
[11]	0.18	336	100	80 x 32	298K	9.16K*	397	-
[20]	0.18	342	100	96 x 32	292K	11.8K*	407	-
[10]	0.18	192	230	160 x 32	1198K	9.57K*	24	-
[9]	0.18	192	100	(128 + 1.5 x FW) x 32	521K	20.9K	184	42
[15]	0.18	236	100	(160 + 2 x FW) x 32	424K	14.5K	250	17
[12]	0.18	250	100	(96 + 2 x FW) x 32	400K	19.64K	271	39
[14]	0.18	128	200	28 x 32	1562K	18.4K	-5	34
[16]	0.18	222	200	64 x 32	901K	18.7K	64	35
[19]	0.18	110	100	64 x 32	909K	12.60K	63	4
[21]	0.18	204	200	2 x 96 x 32, 2 x FW x 32	980K	21.49K	51	44
[23]	0.18	243	100	2 x 96 x 32, 2(FW + 12) x 32	412K	21.1K	260	43
Prop.	0.18	112	166	64 x 32	1482K	12.06K	-	-

** Gate count w/o Boundary strength computation. FW : represents the frame width in pixels.*

The maximum throughput compared to recent related work is provided in Figure 14. The area requirement compared in terms of equivalent gate count with that of other state-of-the-art proposals, for the same technology, is provided in Figure 15. The comparison with respect to some of the other important features like SRAM requirements, filtering cycles per MB and maximum clock frequency is provided in Table I. The last two columns in Table I provide the throughput improvement (in %) and area reduction (in %) in terms of gate count of our proposal compared to each hardware accelerators for deblocking filter.

Figures 14 and 15, and Table I demonstrate that our accelerator provides much higher throughput on one hand and requires significantly less area on the other.

The comparisons with [19] and [14] need some clarification. The design by [19] requires almost the same area as ours, however, we provide 63% higher throughput. While on the other hand, reference [14] provides almost the same throughput as we do, we require 35% less area in terms of equivalent gates. The higher throughput of our design is achieved by merging the processing elements on the critical path and making it shorter. The area requirement reduction, on the other hand, is mainly achieved because of:

- Algorithm level optimization - through decomposition of the filter kernel, inter-filter-mode optimization and overlapped data paths, we reduce the number of additions by 51% and therefore reduce the combinational logic for the implementation;
- Architecture level optimization - through efficient pipeline stage design we reduce the number of registers required for intermediate results for the next pipeline stage, reuse the same hardware resource for the realization of transpose units by identification of mutual exclusive operations in processing chain.

From the comparison with other architectures presented in the Table I, we suggest that the proposed hardware accelerator requires 17-44% less area in terms of equivalent gate count on one hand and provides upto 271% higher throughput on the other. The designs by [10], [11], [20], though require less area in terms of equivalent gate count. However, this is not a fair comparison as these three designs do not include the logic for the boundary strength computation, whereas our design includes the boundary strength computation unit. As far as the throughput is concerned, the proposed hardware accelerator is 24% better when compared with [10] having highest throughput among [10], [11], [20].

VIII. CONCLUSIONS

This paper presents a high-throughput, area-efficient, hardware accelerator for the deblocking filter in H.264/AVC and proposes a decomposition of the filter kernels, which reduces the number of operations by more than 51%. The optimization techniques employed enable the proposed architecture to provide a significantly higher throughput (more than 63% when compared to the closest one in area requirement) and a reduced area requirement (around 35% when compared with the one having similar throughput). It can easily provide real-time filtering operation for the HDTV video format (4096×2304, 16:9) at 30 fps and meet the throughput requirements of all levels (level 1-5.1) in H.264/AVC video coding standard.

REFERENCES

[1] ISO/IEC JTC1/SC29/WG11, "Report of the Formal Verification Tests on AVC/H.264", doc. N6231, Waikoloa, USA, 2003.

[2] G. Sullivan, P. Topiwala and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions", SPIE Conf. On Apps. Of digital Image Processing, vol. 5558, pp. 454-474, 2004.

[3] J. Ostermann, J. Bormans, P. List, D. Maroe, M. Narroschke, F. Pereira, T. Stockhammer and T. Wedi, "Video Coding with H.264/AVC: Tools, Performance and Complexity", IEEE Circuit and Systems Magazine, vol 4, no. 1, pp. 7-28, 2004.

[4] Joint Video Team of IT-T VEG 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, 2003.

[5] Y. L. Lee and H. W. Park, "Loop Filtering and Post-filtering for Low-bitrates Moving Picture Coding", Signal Processing: Image Communication, vol 16, pp. 871-890, 2001.

[6] P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz, "Adaptive Deblocking Filter", IEEE Transactions on Circuits and Systems for Video Technology, vol 13, no. 7, pp. 614-619, 2003.

[7] T. Wiegand, G. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H.264/AVC Video Coding standard", IEEE Transcations on Circuits and Systems for Video technology. Vol. 13, no. 7, pp. 560-576, 2003.

[8] 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, no. 7, pp. 704-716, 2003.

[9] S. Y. Shih, C. R. Chang and Y. L. Lin, “A Near Optimal Deblocking filter for H.264 Advanced video Coding”, Asia and South Pacific Conference on Design Automation, pp 170-175, 2006.

[10] L. Li, S.Goto and T. Ikenaga, “A Highly Parallel Architecture for Deblocking Filter in H.264/AVC”, IEICE Transactions on Information and Systems, vol. E88-D, no. 7, pp. 1623-1629, 2005.

[11] C. C. Cheng and T. S. Chang, “An Hardware Efficient Deblocking Filter for H.264/AVC”, IEEE International Conference on Consumer Electronics, pp.235-236, 2005.

[12] T. M. Liu, W. P. Lee, T. A Lin and C. Y Lee, “A Memory-efficient Deblocking Filter for H.264/AVC Video Coding”, IEEE ISCS, vol. 3, pp. 2140-2143, 2005.

[13] C. C. Cheng, T. S. Chang and K. B. Lee, “An In-place Architecture for the Deblocking Filter in H.264/AVC”, IEEE Transcations on Circuits and Systems II, vol. 53, no. 7, pp.530-534, 2006.

[14] C. M. Cheng and C. H. Chen, “A Memory Efficient Architecture for Deblocking Filter in H.264 Using Vertical Processing Order”, in Proc. IEEE Int. Conf. Intell. Sensors, Sensor Networks. Inf. Process., Dec. 2005, pp. 361-366.

[15] G. Zheng and L. Yu, “An Efficient Architecture Design for Deblocking Loop Filter”, Picture Coding Symposium, 2004.

[16] Q. Chen, W. Zheng, J. Fang, K. Luo, B. Shi, M. Zhang, X. Zhang, “A Pipelined Hardware Architecture of Deblocking Filter in H.264/AVC”, International Conference on Communications and Networking in China, pp.815-819, 2008.

[17] C. M. Cheng, C. Ho Chen, “Configurable VLSI Architecture for Deblocking Filter in H.264/AVC”, IEEE Transactions on VLSI Systems, vol. 16, issue 8, pp. 1072-1082, 2008.

[18] V. Venkatraman, S. Krishnan and N. Ling, “Architecture for De-blocking Filter in H.264”, Picture Coding Symposium, 2004

[19] F. Tobajas, G. M. Callico, P. A. Perez, V. d. Armas and R. Sarmiento,“An Efficient Double-filter Hardware Architecture for H.264/AVC Deblocking Filtering”, IEEE Transcations on Consumer Electronics, vol. 54, no. 1, pp. 131-139, 2008.

[20] S. C. Chang, W. H. Peng, S. H. Wang and T. Chiang, “A Platform Based Bus-interleaved Architecture for Deblocking Filter in H.264/MPEG-4 AVC”, IEEE Transcations on Consumer Electronics, vol. 51, no. 1, pp. 249-255, 2005.

[21] K. Xu and C. S. Choy, “A Five-stage Pipeline, 204 cycles/mb, Single-port SRAM Based Deblocking Filter for H.264/AVC”, IEEE Transcations on Circuitsand Systems for Video Technology, vol. 18, no. 3, pp. 363-371, 2008.

[22] J. Webb, Texas Instruments Incorporated, Dallas TX,USA.“Video Deblocking Filter”, 0184549, Feb 07, 2003.

[23] T. M. Liu, W. P. Lee, and C. Y Lee, “An In/Post-loop deblocking Filter with Hybrid Filtering Schedule”, IEEE Transcations on Circuits and Systems for Video Technology, vol. 17, no. 7, pp. 937-943, 2007.

[24] M. Shafique, L. Bauer, and J. Henkel, “An Optimized Application Architecture of the H.264 Video Encoder for Application Specific Platforms”, In Proc. of Workshop on Embedded System Real-time Multimedia, pp. 119-124, 2007.