A Deblocking Filter Hardware Architecture for the High Efficiency Video Coding Standard

Cláudio Machado Diniz[1], *Muhammad Shafique*[2], *Felipe Vogel Dalcin*[1], *Sergio Bampi*[1], *Jörg Henkel*[2]
[1]Informatics Institute, PPGC, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
[2]Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
{cmdiniz, fvdalcin, bampi}@inf.ufrgs.br; {muhammad.shafique, henkel}@kit.edu

***Abstract*—The new deblocking filter (DF) tool of the next generation High Efficiency Video Coding (HEVC) standard is one of the most time consuming algorithms in video decoding. In order to achieve real-time performance at low-power consumption, we developed a hardware accelerator for this filter. This paper proposes a high throughput hardware architecture for HEVC deblocking filter employing hardware reuse to accelerate filtering decision units with a low area cost. Our architecture achieves either higher or equivalent throughput (4096x2048 @ 60 fps) with 5X-6X lower area compared to state-of-the-art deblocking filter architectures.**

Keywords—HEVC coding; Deblocking Filter; Hardware Architecture.

I. Introduction and Motivation

The new High Efficient Video Coding (HEVC) standard [1] provides approximately 50% bit-rate reduction compared to the last version of H.264/AVC (Advanced Video Coding) standard [2] with similar subjective video quality [3]. To achieve such compression efficiency, especially focusing on high and ultra-high resolution videos (beyond HD 1920x1080 pixels), HEVC employs larger block sizes (up to 64x64 pixels) for prediction. A new quadtree structure splits those blocks hierarchically down to 4x4-pixel elementary blocks. A large set of new advanced coding tools was introduced in HEVC. The higher compression efficiency achieved with HEVC results in an increase of the computational complexity when compared to H.264/AVC [4].

The new Deblocking Filter (DF) [5] is included in both HEVC video encoder and decoder to reduce the blocking artifacts present in a reconstructed video, which are introduced by the inherent block partitioning and strong quantization in video encoding. DF contributes to up to 6% bit-rate reduction (1.3%-3.3% bit-rate reduction on average) for the same video quality [5]. Although DF is an optional feature in video encoder (and thus may not be used in video decoder), it is often employed because of the high bit-rate reduction.

DF is one of the most time consuming tools of HEVC video decoder. We have profiled the HEVC reference decoder software (HM 10.0) [6] to identify how much of the execution time belongs to the DF process. The profiling (using GNU gprof [7]) was performed by decoding 3 seconds of four ultra-high resolution (with 2560x1600 pixels) and five high resolution (with 1920x1080 pixels) encoded video sequences. Encoded video sequences were generated with the following encoder configuration: *(i)* Random Access (RA) configuration[1] with Group of Pictures (GOP) equal to 8 *(ii)* Intra period[2] for each video sequence is defined as in [8] depending upon the specific frame rate of the video sequence, e.g. 24, 30, 50 or 60 frames per second (fps); *(iii)* each sequence is encoded with four different Quantization Parameter (QP) values *QP={22,27,32,37}* as defined in the HEVC Common Test Conditions [8]. Fig. 1 shows the accumulated execution time (in % of total decoding time) of all functions included in C++ class *TComLoopFilter* that implement the DF in HEVC decoder software. DF contributes to up to 5%-18% to the total decoding time, depending on video sequence and QP. This percentage represents up to 11 seconds spent only in DF process to decode 3 seconds of 2560x1600 resolution video. We used Intel Core i7-3770 CPU @ 3.40GHz with 16 GB RAM to run this experiment.

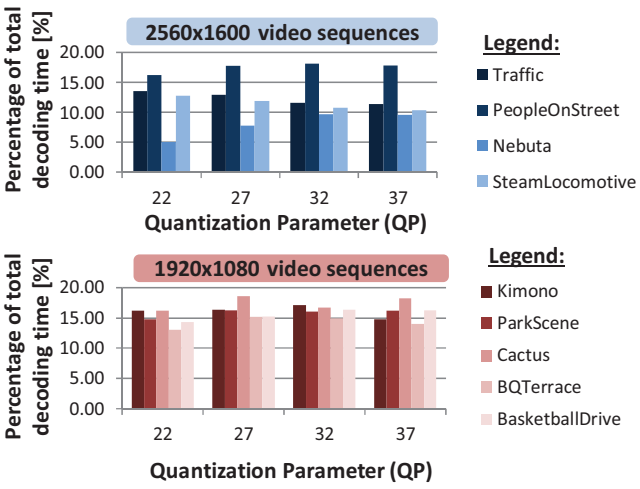


Fig. 1 Percentage of total decoding time spent in HEVC Deblocking Filter.

With the high computational complexity of DF (i.e. 11 seconds is spent in DF to decode 3 seconds of ultra-high resolution video), it is not viable to decode video in real time, even considering a high-end general purpose computing platform. Moreover, general purpose processor consumes far more power to perform the same task compared to a specialized processor. Although DF does not spend considerable portion of the encoding time (as it is dominated

[1] For details of RA configuration please refer to [4][9].

[2] Intra period is the period (in pictures) in which a picture using only Intra prediction (i.e. an Intra picture) appears in the coded video. It is usually included at one second interval [9].

by motion estimation and decision steps), the power issue also influences the encoder design and favors a hardware acceleration solution. For this reason, *high-throughput hardware acceleration is desirable to realize real time and low power HEVC video decoding and encoding.*

A. Novel Contribution

We propose a high throughput hardware architecture for HEVC deblocking filter (DF) employing hardware reuse to accelerate filtering decision units at low area cost.

Before moving to the details of our novel contribution (section III), this work provides some background of the new HEVC DF and discusses the related work in section II.

II. Background And Related Work

A. Overview of deblocking filter in HEVC

In HEVC, a video picture is split into coding tree units of up to 64x64 samples. The coding tree unit (CTU) can be further split into coding units (CU) of size 32x32 down to 8x8 samples in a quadtree form. The CU is also split into prediction units (PU) and transform units (TU) to apply prediction and transform tools, respectively.

The deblocking filter reduces the blocking artifacts (visible discontinuities in the video) caused by block-based encoding with strong quantization. It is applied by modifying samples along horizontal and vertical boundaries of PUs and TUs of size not smaller than 8x8 samples. Filtering is applied separately in 4x4 blocks (so-called P and Q blocks), as shown in Fig. 2. Normal and strong filtering modes modify 2 and 3 luma samples along each boundary, respectively. The example in Fig. 2 shows a vertical boundary, as the horizontal boundary filtering is analogous.

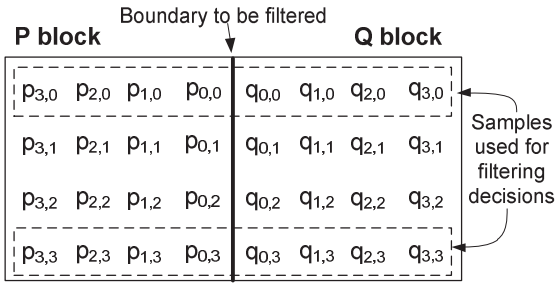
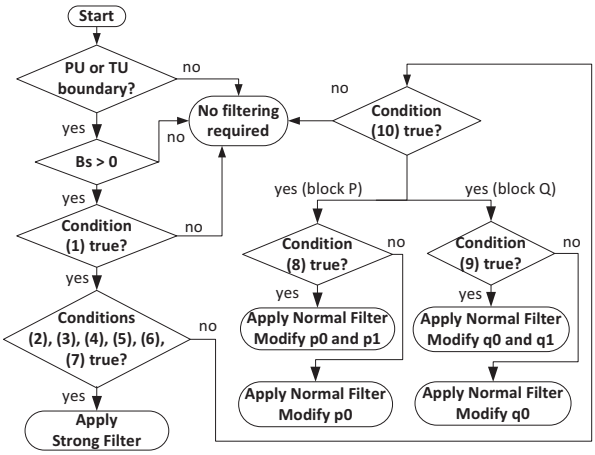


Fig. 2 Boundary of a 4x4 block (blocks P and Q).

Before the boundary filtering, many filtering decisions have to be assessed to decide whether or not the boundary should be filtered, and also to determine which filtering modes, i.e. normal or strong, must be applied on a boundary. Only the samples in the first and last rows of P and Q blocks are used for filtering decisions (see Fig. 2). Fig. 3 shows the complete algorithmic flow of the deblocking filter, along with the filtering decision equations. With the help of filtering decisions, DF avoids filtering real video boundaries and it filters only those which are artificially generated by the coding process. Filtering decisions depend upon various parameters, such as block type, QP, and video content. β and t_c value are determined by a lookup table with QP as input. Block types affect boundary strength (Bs) [1]. Chroma filter is only performed when Bs is equal to 2 and no further decisions need to be evaluated. Only the samples closer to the block boundary are modified. Equations governing filtering operations equations are shown in section III. Further details of HEVC deblocking filter are detailed in [1][5].

B. Related work

Some hardware architectures for HEVC DF were already proposed in [10]-[12]. Ozcan et al. [10] introduced an architecture with 2 datapaths in parallel. Each datapath is configurable to implement all decision and edge filter operations. Operating at 108 MHz clock frequency it is able to encode videos with the 1920x1080 pixels resolution at 30 fps. Shen et al. [11] proposed a four-stage pipeline architecture. They focus on a new filtering order, but do not provide enough details of the internal architecture of datapaths for filtering decisions and operations. Operating at 28 MHz clock frequency it is able to encode 4Kx2K video resolution at 30 fps. Shen et al. [12] extended the architecture in [11] to include the sample adaptive offset (SAO) filter in a five-stage pipeline architecture.



Filtering decisions (conditions):

$$| p_{2,0} - 2p_{1,0} + p_{0,0} | + | p_{2,3} - 2p_{1,3} + p_{0,3} | + | q_{2,0} - 2q_{1,0} + q_{0,0} | + | q_{2,3} - 2q_{1,3} + q_{0,3} | > \beta \quad (1)$$

$$| p_{2,i} - 2p_{1,i} + p_{0,i} | + | q_{2,i} - 2q_{1,i} + q_{0,i} | < \beta/8 \quad i=0\ (2),\ i=3\ (3)$$

$$| p_{3,i} - p_{0,i} | + | q_{0,i} - q_{3,i} | < \beta/8 \quad i=0\ (4),\ i=3\ (5)$$

$$| p_{0,i} - q_{0,i} | < 2.5t_c \quad i=0\ (6),\ i=3\ (7)$$

$$| p_{2,0} - 2p_{1,0} + p_{0,0} | + | p_{2,3} - 2p_{1,3} + p_{0,3} | < 3/16\beta \quad (8)$$

$$| q_{2,0} - 2q_{1,0} + q_{0,0} | + | q_{2,3} - 2q_{1,3} + q_{0,3} | < 3/16\beta \quad (9)$$

$$| \delta_0 | < 10tc \quad i=0..3 \quad (10)$$

Fig. 3 Deblocking filter flow and filtering decisions (conditions).

III. Our Deblocking Filter Hardware Architecture

Our methodology to design optimized hardware datapaths for HEVC deblocking filter is presented in Fig. 4. First, DF equations from the HEVC standard [1] are subject to architecture independent optimizations, such as equations

reformulations to facilitate the reuse of many operations using the same operators, and replacing multiplications by add/shift operations. The result of this step goes to hardware-specific optimizations, such as determining the sizing of registers and operators, the number of datapaths in parallel, scheduling, balancing of operating stages, etc.

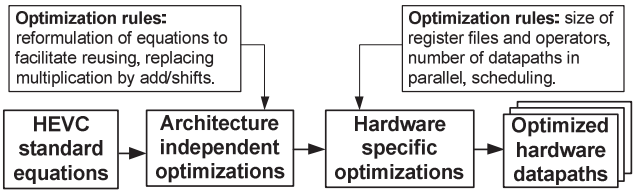


Fig. 4 Our methodology to design optimized hardware datapaths.

Following this methodology, we propose a HEVC DF hardware architecture, whose system diagram is depicted in Fig. 5. Our architecture receives as input the samples of the 2 neighboring 4x4 blocks (P and Q, see Fig. 2). P and Q blocks belong to different adjacent 8x8 blocks in PU or TU boundary to be filtered, which is determined by PU and TU flags. Our architecture decides whether filtering is required or not and the strength of the filtering to be applied if this is the case. It depends on input samples and also on Bs, β and t_c calculated over sample values, P and Q block types, and QP, which are also given as input to our architecture. Our architecture also has some simple control signals to establish a handshake between a master device (e.g. a CPU that runs the whole HEVC encoder/decoder application) and our architecture (which plays the slave role).

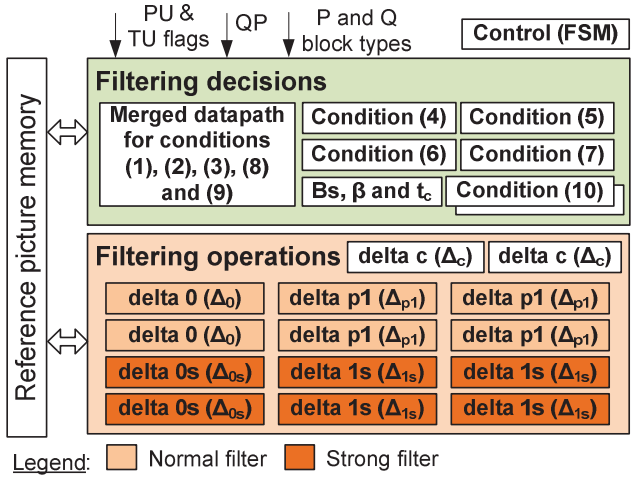


Fig. 5 System diagram of our deblocking filter hardware architecture.

Samples from P and Q blocks to be filtered are first stored into reference picture memory. As each input sample has 8 bits, our architecture supports a 256-bit wide input memory channel. Therefore, 32 samples (both 4x4 blocks, P and Q) are transmitted in one clock cycle. Filtered samples are stored back into reference picture memory after the filtering operations.

Our architecture has three main units: *(1) filtering decisions; (2) filtering operations;* and *(3) control unit*. The *filtering decisions* unit calculates the conditions to decide whether the boundary must be filtered or not and the strength of the filter. It is composed by some datapaths in parallel, whose internal architecture is detailed in section III.A. If filtering is required, the *filtering operations* unit calculates the filtered samples. Datapaths of filtering operations are detailed in section III.B. The *control unit* implements the control flow shown in Fig. 3 and it establishes the handshake with a master device, as detailed in III.C.

A. Filtering decisions

This unit determines the need of filtering two given 4x4 blocks. For input samples convention, please refer to Fig. 2. By examining the equations in Fig. 3, it can be noted that conditions (1), (2), (3), (8) and (9) share similar equations and same input samples. Partial results from conditions (2) and (3) and used for conditions (1), (8) and (9). Hence, we employ hardware reuse to design a *merged datapath* for those conditions, as depicted in Fig. 6. The condition equations require some multiplication by constants. We have replaced the multiplications by adders and shift operations to use less hardware resources. We have shorten conditions (1), (2), (3), (8) and (9) as *c1*, *c2*, *c3*, *c8* and *c9*, respectively. With that, those five conditions are generated in only one clock cycle.

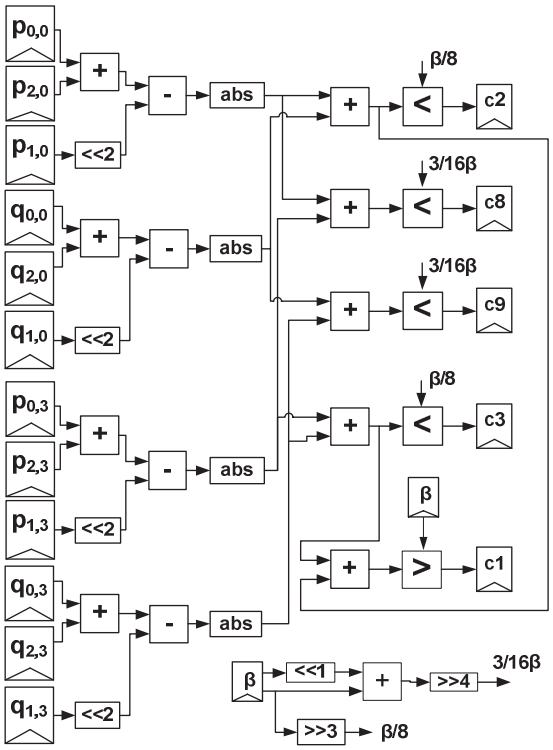


Fig. 6 Merged datapath for conditions (1), (2), (3), (8) and (9).

Datapaths for the remaining conditions are depicted in Fig. 7. Datapaths for conditions (4) and (5) are equal, only differing by the input samples (condition 4 for the first row and condition 5 for the last row of 4x4 blocks). We have included two instances of this datapath to compute both conditions (*c4*

and *c5*) in the same clock cycle. The same was made for conditions (6) and (7). Condition (10) is an additional filtering decision applied to δ_0 for the four rows of 4x4 blocks after normal filtering operation (see more details in section III.B). Our architecture includes two instances of this datapath in the design to compute *c10* for the four rows in two clock cycles. Each instance computes two rows of samples. Additional datapaths for β and t_c multiplications, needed to compute all the conditions, are also shown in Fig. 7. β and t_c values are generated by a lookup table with QP value as input index.

B. Filtering operations

After computing the filtering decisions, this unit computes the filtering operations for normal and strong filters (for luma) and chroma filter. Datapaths for normal and strong filter operations are shown in Fig. 8.

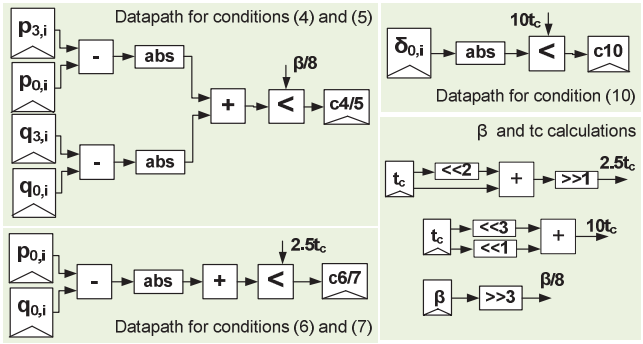


Fig. 7 Datapaths for conditions (4), (5), (6), (7) and (10).

Normal filter modifies 1 or 2 samples along the block boundary. It computes the delta values (i.e. Δ_0, Δ_{p1}, Δ_{p2}) which are offsets that must be added to the original samples (p_0, p_1, q_0 and q_1) in order to generate the final filtered sample (p_0', p_1', q_0' and q_1'). They are applied to the four rows of samples of 4x4 blocks P and Q. Delta 0 operation is always computed when the normal filter is selected in filtering decision process and modifies the p_0 and q_0 samples that are close to the boundary. Delta p1 and delta q1 operations modify also p_1 and q_1 samples. Please refer to DF flow in Fig. 3 for details. In our architecture, delta p1 and delta q1 values are computed anyway in the same cycle to achieve high throughput and to simplify control. The final sample modification depends upon the result of conditions (8) and (9). This is done by the control unit.

In the design of this unit we have also replaced the multiplications by a sequence of adders and shift operations. Our architectural design includes two instances of each datapath shown in Fig. 8 (please refer to Fig. 4). This way, we can compute the normal filtering operations in two cycles. Each instance computes two rows of samples.

Strong filter always modify 3 samples along the block boundary. It computes the delta values (i.e. Δ_{0s}, Δ_{1s}, Δ_{2s}) which are offsets that must be added to the original samples (p_0, p_1, p_2, q_0, q_1 and q_2) in order to generate the final filtered sample (p_0', p_1', p_2', q_0', q_1' and q_2'), as shown in Fig. 8. They are also applied to the four rows of samples of 4x4 blocks P and Q. Similar to normal filter datapaths, we have also replaced the multiplications by adders and shift operations and we have included two instances of each datapath in the architecture. Each instance computes two rows of samples.

Our architecture also includes two instances of the datapath for the chroma filtering, which is shown in Fig. 9. Chroma filter is only applied when Bs is equal to 2 and does not require further decisions. It modifies only p_0 and q_0 samples.

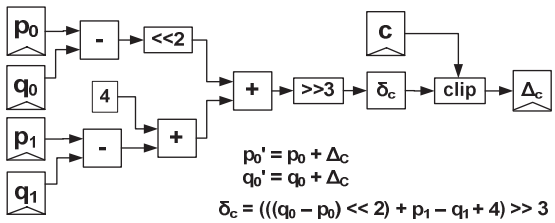


Fig. 9 Datapath for chroma filter operation.

Clipping operation, included in all the datapaths, is present in DF to prevent excessive blurriness. It keeps the final result inside a range that depends on the QP value, block type and filtering strength. The value *c* is calculated in the first cycle with the decision process and it is stored in register to be used by the filtering operation datapaths.

After generating the delta values, they are added with the original samples to produce the final filtered samples, depending on the results of conditions calculated in the *filtering decisions* unit. One multiplexer is included for each

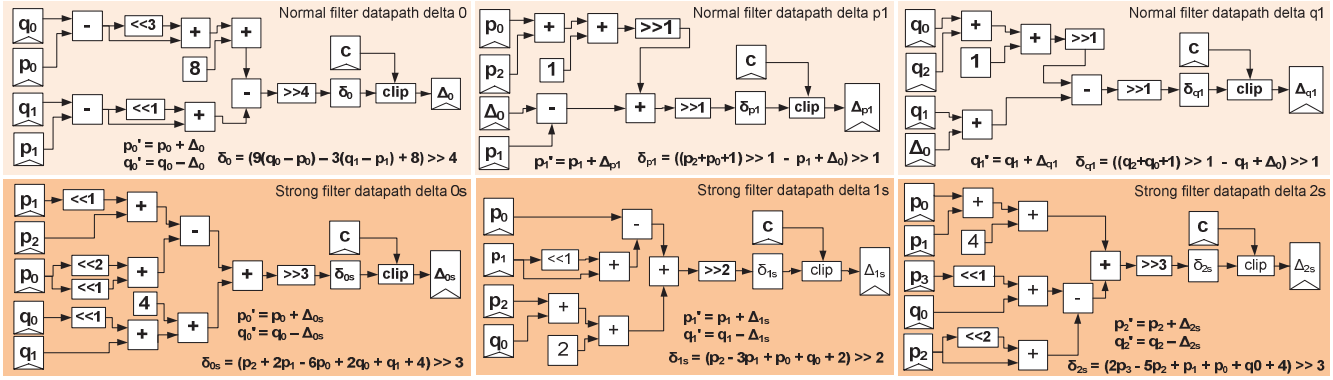


Fig. 8 Datapaths for normal and strong filtering operations.

output.

C. Control

Due to the distinct dataflow nature of the DF, the control unit of the architecture is relatively simple. A finite state machine (FSM) handles the handshake protocol between master and slave and selects the correct filtered output samples based on the filtering conditions unit. The FSM has 4 states, as shown in Fig. 10. The output values are described in a table in Fig. 10.

The master starts a transmission by signalizing with 'data_in_valid' signal. Then, the DF architecture reads the input data port from memory. At the next clock cycle, if filtering of the given input is required, the filtered samples are available at the output data port and 'data_out_valid' signal changes to '1'. The deblocking filter architecture expects to receive the next 2 rows of samples to be filtered. At the next clock cycle, the filtered samples are available at the output data port. This process can be repeated, resulting in new blocks at every 2 clock cycles or no filtered block (when filtering is not necessary) at every clock cycle. The master can stop transmitting data at the end of the second clock cycle, by signalizing 'data_in_valid' with '0'.

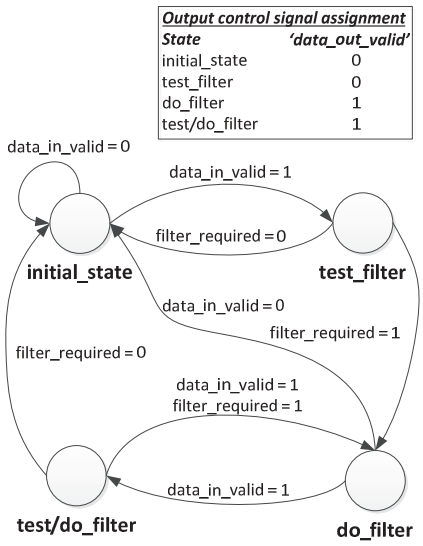


Fig. 10 State diagram of FSM.

An example of the processing scheduling of our architecture is shown in Fig. 11. We show in this example the normal filter operation which needs to modify two samples along the boundary. This is the worst case condition for our architecture, because it needs more clock cycles to complete. Other situations, e.g. no filtering, strong filtering, or even normal filtering modifying only one sample along the boundary require less clock cycles to complete the calculation (please refer to the flow in Fig. 3 and the control state machine in Fig. 10). Fig. 11 shows the scheduling of processing block boundaries of 4 samples to be filtered (P and Q blocks). The numbers inside the boxes are the boundaries' numbers.

Our architecture has an initial latency of 5 clock cycles to complete normal filter (modifying 2 samples along the boundary) for the first four-sample boundary. After the initial latency, it delivers a new normal filtered boundary at each 2 clock cycles, due to our pipelined architecture. We have considered a worst-case condition for our architecture as follows: when a CTU (64x64-pixel blocks) of video is all partitioned into 8x8-pixel blocks and all boundaries between those blocks need to be filtered by the normal filtering that modifies 2 samples along the boundary. This is a very worst case condition, since not always the CTU is all partitioned into 8x8-pixel blocks and not always it need to be filtered by the worst case filtering mode. However, this analysis is useful to compare the architecture with state of the art. Hence, considering the worst case condition, our architecture requires 1,027 clock cycles to filter one CTU.

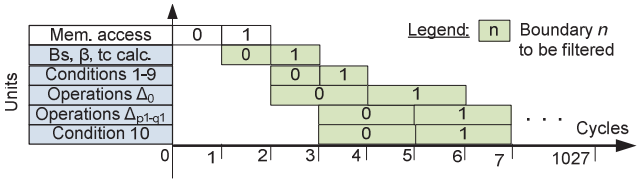


Fig. 11 Processing schedule of normal filter (worst case).

IV. Results and Evaluation

The DF architecture was implemented in VHDL and synthesized to Application Specific Integrated Circuit (ASIC) and Field-Programmable Gate Array (FPGA). FPGA synthesis and mapping is performed using Xilinx ISE design suite [13] for Xilinx Virtex-6 XC6VLX130T-ff1156-3 FPGA device [14]. ASIC synthesis is performed using Cadence RTL Compiler tool [15] for FreePDK 45nm CMOS standard-cell library [16] under a constraint of 200 MHz clock frequency. Area results for the target clock frequency for both FPGA and ASIC are shown in Table I. Area results for FPGA are shown in terms of Slice Look-up Tables (LUTs) and slice registers. ASIC area results are shown in terms of gate count (NAND-2 area equivalent) to enable comparisons with related works which are implemented in different CMOS technologies. It can be noted that our hardware architecture requires very low hardware resources due to our architecture-independent and hardware-specific optimizations to create optimized datapaths for HEVC deblocking filter.

TABLE I. Synthesis Results for FPGA and ASIC

Hardware units	FPGA results		ASIC results
	Slice LUTs	Slice Registers	Gate count (NAND2)
Conditions	383	168	980
Operations	770	265	1,931
Control (FSM)	245	8	370
Total	1,398	441	3,281
Frequency (MHz)	140		200

Table II shows the comparison of our architecture to state of the art [10][11]. The work in [12] presents very similar DF

architecture solution with the work in [11], so we have considered only the work in [11] for comparison. Compared to the work in [10], the ASIC implementation of our architecture reduces gate count in approximately 5X, while providing 4X maximum throughput in terms of real-time video processing for certain video resolution and frame rate. The FPGA implementation reduces LUTs and slice registers in approx. 4X compared to [10], while providing high throughput with a maximum frequency of 140 MHz. We have synthesized our design for the same FPGA device of [10] to enable direct comparison in terms of slice LUTs and slice registers. The throughput of our architecture is higher than [10] mainly by the reduced number of cycles/CTU (in the worst case) of our architecture. Compared to [10] we reduce the cycles/CTU in more than 7X. Area is reduced in our design through our architecture independent and hardware specific optimizations, i.e. reusing of operations, replacing multiplications by add/shift operations.

The work in [11] has a more efficient design than [10] at the cost of a larger hardware area. It does not include FPGA implementation, so comparisons are possible only against our ASIC results. Compared to [11] the ASIC implementation of our architecture reduces gate count in more than 6X, while providing a similar throughput. Another comparison is conducted in terms of the architecture (and implementation-independent) parameters: cycles/CTU (worst case) and minimum frequency to process two video resolutions in real time: 1920x1080 @ 30 fps; and 4096x2048 @ 60 fps. Compared to [11], our work requires 2X the minimum frequency to process videos at a certain frame rate (15.6 MHz vs. 7 MHz, and 124.8 MHz vs. 56 MHz) than [11]. However, our architecture achieves 6X gate count reduction compared to [11]. By increasing operating frequency to only 124.8 MHz, our design achieves throughput to process 4096x2048@60 fps. Both FPGA and ASIC implementations of our architecture reach this operating frequency, as shown in the results in Table II.

TABLE II. COMPARISON WITH STATE OF THE ART

	[10]	[11]	Our
Architecture parameters			
Cycles/CTU (worst case)	7,680	440	1,027
Minimum frequency (MHz) to process 1920x1080@30fps	108	7	15.6
Minimum frequency (MHz) to process 4096x2048@60fps	-	56	124.8
ASIC implementation			
CMOS technology (nm)	90	130	45
Maximum frequency (MHz)	108	200	200
Gate count (NAND2 equiv.)	16.4k	21k	3.3k
Maximum throughput (resolution @ frame rate)	1920x1080 @ 30 fps	4096x2048 @ 60 fps	4096x2048 @ 60 fps
FPGA implementation			
Device	Virtex-6	-	Virtex-6
Frequency (MHz)	108	-	140
Slice LUTs	5,236	-	1,398
Slice registers	1,547	-	441
Maximum throughput (resolution @ frame rate)	1920x1080 @ 30 fps	-	4096x2048 @ 60 fps

V. CONCLUSIONS

This paper proposed a high throughput and low area hardware architecture for HEVC deblocking filter. We have employed a methodology to design datapaths highly optimized for area using architecture-independent and hardware-specific optimizations from the original HEVC standard equations. ASIC and FPGA implementations of our architecture achieved higher or similar throughput than the state of the art; both are able to process video resolution of 4096x2048 pixels at 60 fps. ASIC implementation achieved 5X-6X reduction in gate count compared to state of the art. This is an exceptional result in view of the area-constrained and high throughput implementations required for HEVC codecs.

REFERENCES

[1] ITU-T and ISO/IEC, High efficiency video coding, ITU-T Recommendation H.265 and ISO/IEC 23008-2:2013, 2013

[2] ITU-T and ISO/IEC JTC 1, Advanced video coding for generic audiovisual services, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), 2011.

[3] G. J. Sullivan, J. Ohm, W.-J. Han, T. Wiegand,, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp.1649-1668

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis", IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp. 1685-1696

[5] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van der Auwera, "HEVC Deblocking Filter," IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, 2012, pp.1746-1754, Dec. 2012

[6] HEVC Test Model (HM) version 10.0. http://hevc.hhi.fraunhofer.de/.

[7] J. Fenlason and R. Stallman. GNU gprof—The GNU Profiler, Free Software Foundation, Inc., 2000.

[8] F. Bossen, "Common Test Conditions and Software Reference Configurations", JCTVC-L1100, JCT-VC, Geneva, Jan. 2013.

[9] J. Vanne, M. Viitanen, T.D. Hamalainen, and A. Hallapuro, "Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs," IEEE Transactions on Circuits and Systems for Video Technology, vol.22, no.12, pp.1885-1898, Dec. 2012.

[10] E. Ozcan, Y. Adibelli, and I. Hamzaoglu, "A high performance deblocking filter hardware for High Efficiency Video Coding," International Conference on Field Programmable Logic and Applications (FPL), 2013, pp.1-4

[11] W. Shen, Q. Shang, S. Shen, Y. Fan, and X. Zeng, "A high-throughput VLSI architecture for deblocking filter in HEVC," IEEE International Symposium on Circuits and Systems (ISCAS), 2013, pp. 673-676

[12] S. Shen, W. Shen, Y. Fana, and X. Zeng, "A pipelined VLSI architecture for Sample Adaptive Offset (SAO) filter and deblocking filter of HEVC", IEICE Electronics Express, v.10, n.11, pp. 1–11

[13] Xilinx ISE Design Suite version 14.4, http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm.

[14] Xilinx Virtex-6 Family Overview, DS150 (v2.4) http://www.xilinx.com/support/documentation/data_sheets/ds150, Jan. 2012

[15] Cadence Encounter RTL Compiler, http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx

[16] FreePDK 45nm standard-cell library, Oklahoma State University, http://vlsiarch.ecen.okstate.edu/flow.