1. Introduction

HEVC is the next generation international video coding standard. HEVC has already reduced 50% bit rates in encoding video sequences with same picture quality compared to its predecessor H.264/AVC, in-loop filter is an important part of the video coding standard. In the main profile of HEVC, it is composed of 2 parts, de-blocking filter and SAO.

Deblocking filter is a traditional way to relieve the blocking artifacts in the video decoding process. There are some previous works on deblocking filter. In [5], 4 edge filters organized in 2 groups for simultaneously processing vertical and horizontal edges are applied; in [6], a 5-stage pipelined and resource shared dual-edge filter to process the pixels in parallel and thus one luma block can be processed in 4 cycles without any intermediate storage or circuit. It takes only four cycles to finish the deblocking filter and SAO. By these means, a high performance in-loop filter including both deblocking filter and SAO is achieved without any intermediate storage or circuit. It takes only four cycles to finish the deblocking and SAO of one 8×8 block. The implementation results show that the proposed solution can be synthesized at 240 MHz with 65 nm technology. Thus this solution can process 3.84G pixels/s at maximum. UHDTV 4320p (7680 × 4320) @ 60 fps decoding can be realized with 124.4 MHz working frequency by the proposed architecture.

key words: DBF, SAO, HEVC, pipeline
ture in details. The implementation results are illustrated in Sect. 5. Finally, Sect. 6 concludes this paper.

2. HEVC In-Loop Filter Algorithms

2.1 De-Blocking Filter Algorithms

In HEVC [1], boundaries of 8 × 8 blocks are needed to be deblocking filtered, shown in upper part of Fig. 2. Luma pixels and chroma pixels are filtered respectively. There are 2 modes for luma deblocking filtering, one is strong filtering and the other is weak filtering. As shown in Fig. 1(a) and Fig. 1(b), for strong filtering, 6 pixels among the 8 input pixels besides the edge are to be modified; for weak filtering, 2 to 4 pixels among the 8 input pixels are to be modified.

Whether the edge is strongly filtered or weakly filtered and how many pixels are to be modified in weakly filtered case depends on the decision parameters: dE, dEp and dEq. dE decides whether the luma filtering of the edge is strong or weak. dEp and dEq decide whether p1 and q1 are modified in weak filtering.

As shown in Fig. 2, the derivation of these parameters depend on not only beta and tc, but also value of the top and bottom row of two 4 × 4 luma blocks besides the edge to be filtered. The value of dE, dEp and dEq is determined by the comparison result between the sums of absolute difference of certain luma samples and threshold determined by parameter beta and tc. They are actually a measure of whether the samples are stable or dramatically variant. If they are considered as dramatically variant, then strong filtering is applied. Otherwise weak filtering or even no filtering is applied.

The fact that dE, dEp and dEq and related to the value of top and bottom row of 4 × 4 blocks (shown in Fig. 2) means that in order to filter the top row pixels, the value of bottom row pixels are necessary to know. This is called as decision process and it is new in HEVC. It proposes some limits on the hardware. One conventional deblocking method is to fetch the top row, filter it and output it. Then the second row, the third row and the bottom row are processed one by one. However, due to the decision process in HEVC deblocking filter, there exist issue to use the traditional way because the filtering of top row cannot be achieved without the information of bottom row. Our proposed VLSI architecture in Sect. 5 avoids this issue. For chroma deblocking filtering, there is only one filtering mode, 2 pixels of the 4 input pixels are modified, as shown in Fig. 1(c). The filtering process is actually some arithmetic operations on the input pixels. All of these arithmetic operations can be implemented in hardware by adder, comparator and wiring.

In [4], a HEVC and H.264 dual mode deblocking filter is proposed to implement the algorithms. In this article, only the HEVC deblocking filter is considered. This is one reason to improve the work [4].

2.2 SAO Algorithms

SAO classifies reconstructed samples into different categories, obtains an offset for each category and then add the offset to each sample of the category [3]. There are 2 types of SAO, edge offset (EO) and band offset (BO). BO implies one offset is added to all samples of the same band. As shown in Fig. 3, for 8-bit samples ranging from 0 to 255, the sample value range is equally divided into 32 bands. And the width of each band is 8. Offsets of four consecutive bands and the starting band position are derived from the
bitstream. In Fig. 3, k is the starting band position, k and offsets of band k to band k + 3 are coded in the bitstream. Offsets of the 4 bands are decoded and then added to all the samples that are within the corresponding band. That is band offset.

EO uses 4 1-D directional patterns for sample classification: horizontal, vertical, 135 diagonal and 45 diagonal, as shown in Fig. 4. Where the label “c” represents a current sample and the labels “a” and “b” represent two neighboring samples.

For a given EO class, each sample inside the CTB is classified into one of five categories. The current sample value labeled as “c” is compared with its two neighbors along the selected 1-D pattern. The relationship between the categories and the comparison results is shown in Fig. 5.

If the comparison results of a, b and c is not in the 4 categories in Fig. 5, then it belongs to category 0 and offset set is zero. Offset sets of the above 4 categories are derived from bitstream. When do the SAO, the category of each sample is detected by compare its value with its neighboring sample’s value and thus the offset of that category is added to the sample.

The process of SAO is controlled by parameters. The parameters for SAO are in the unit of CTB. That is to say, all the pixels within one CTB share the same parameters regardless of the size of CTB.

3. Data Flow for Proposed Architecture

3.1 Data Flow of One 8 × 8 Block

In this section, the data flow of deblocking filter is illustrated firstly. Then the issue incurred by the combination of deblocking filter and SAO is described. Next the drifted block based SAO is proposed, which solves the issue incurred by the combination of deblocking filter and SAO.

The data flow of deblocking filter in work [4] is adopted in this article. The shifted 8 × 8 block (SB8) shown in Fig. 6(b) instead of current 8 × 8 block (CB8) shown in Fig. 6(a) is input to the deblocking filter. This can be easily achieved with RAM in the reconstructor [4]. The advantage of this input pattern is obvious. E0-E3 in Fig. 6(b) can be deblocking filtered without referring neighboring samples. Four edges E0-E3 are deblocking filtered one by one. Finally, the filtered samples are shown in Fig. 6(c) as grey cycles.

In HEVC, the pixels after deblocking filter shall be input to SAO filter. However, how to combine SAO together with deblocking filter is a problem. That is due to the fact that SAO process requires referring the deblocked pixels of neighboring blocks. As introduced in Sect. 2.2, edge offset of SAO refers neighboring samples of current samples to obtain the offset. So if the SB8 shown in Fig. 6(b) and Fig. 6(c) is to be through SAO, the black pixels in Fig. 6(c) are referred. Since the black pixels of right column and bottom row in Fig. 6(c) belong to right or bottom blocks which have not been deblocked.

Conventional ways is to delay the start of SAO. That is to say, SAO of one block (e.g. SB8 in Fig. 6(c)) should be started after the moment when its right and bottom block have been de-blocked. But this incurs buffer between deblocking filter and SAO filter and it is complex. To solve this issue, we propose a novel data processing method in this section.

The proposed SAO data flow is based on drifted 8 × 8 block (DB8). As shown in Fig. 6(d), after the deblocking of SB8, the corresponding DB8 is to go through SAO and its surrounding pixels (black cycles in Fig. 6(d)) have already been de-blocked. Hence, a 10 × 10 upper-left toward drifted pixels block will be input SAO and the center 8 × 8 pixels are modified, as shown in Fig. 6(d). By this means, right and bottom blocks of the SB8 are not needed to be
referred. Only its upper and left 4×4 blocks (i.e. E, F, G, H and I in Fig. 7(a)) are needed to be referred.

As shown in Fig. 7(a), A, C, B and D are the shifted 4×4 blocks (SB4). In Fig. 7(b), the blocks within red square is drifted 4×4 blocks (DB4) and they are named as A’, C’, B’ and D’ respectively. For DB4 A’, it contains pixels of E, F, H and A. And similar cases exist for B’, C’ and D’. Four DB4 are input to SAO filter one by one. After the deblocking filter and SAO filter process of A’, B’, C’ and D’, the pixels of the right column of B, D and bottom row of C, D have not completed SAO filter. But all the pixels in E, F, H and A have completed SAO filter and will be output.

3.2 Data Flow between 8×8 Blocks

In HEVC, the size of CTB maybe 16×16, 32×32 or 64×64. The size of CTB decides how many SB8 to process within one CTB. As shown in Fig. 8, SB8s in one CTB (red lines) is processed in raster scan order.

As shown in Fig. 9, suppose there is a picture composed of six 16×16 CTBs (the black lines). Then in addition to the six shifted CTB (0-2, 4-6), one more column and one more row of shifted CTB (3, 7, 8-11) are required to process. But not all of the SB8 in these shifted CTB are required to process, only the most left and top SB8 which cover the picture are processed. That is to say, if the width of picture is W 8×8 blocks and the height of picture is H 8×8 blocks, then (W+1)×(H+1) SB8s are processed for this picture.

When processing the pixels near the picture or CTB boundary, SB4s within one SB8 maybe from different CTB or maybe unavailable. For example, in Fig. 9, left upper SB8 of shifted CTB 0 (the most upper left SB8 in this figure) contains only one SB4. All the other 3 SB4 are unavailable. Another example is the left upper SB8 of shifted CTB 5. It contains four SB4, which are from CTB0, CTB1, CTB4 and CTB5 respectively. This has some impact on deblocking filter and SAO.

For deblocking filter, whether the SB4 is unavailable determines whether the edge is to be deblocking filtered. Some examples are shown in Fig. 10, in which gray blocks mean that they are unavailable. The deblocking filter supports skipping filtering. If the edge is not deblocking filtered, then the output samples of the filter is same as the input samples without modification. This can be implemented easily with a multiplex.

For SAO, whether the SB4 is unavailable or whether the SB4 belongs to different CTB influence the process of SAO. If SB4 belongs to different CTB, then the SAO parameters are different with each other within one SB8. All the parameters of related CTB are required to input the SAO filter. If SB4 is unavailable, then a flag indication is input to SAO filter for filtering.

When processing the most right column SB8 along CTB boundary (e.g. 7, 15, 23, ... in Fig. 8 64×64 CTB), a storage named as left buffer is required to store the data which is referred in processing the most left column SB8 (e.g. 0, 8, 16, ... in Fig. 8 64×64 CTB) of the right CTB. Take the example in Fig. 7(a), four SB4 A, B, C and D form one SB8. After the processing of this SB8, if it is not most right column of one CTB, then no data need to be stored to left buffer. Because 4×4 blocks GBD automatically becomes the 4×4 blocks EHI when the right CTB is processed. If this SB8 is the most right column of one CTB, then 4×4 blocks GBD are stored to left buffer. And proper operation is required to read the left buffer to ensure that GBD is fetched out as EHI of the most left SB8 of the right CTB. The details
of the timing are shown in Sect. 4.6.

4. Proposed VLSI Architecture

In this section, the proposed VLSI architecture is illustrated. As shown in Fig. 11, the proposed architecture is mainly divided to DBF part and SAO part. Reconstructed pixels go into the deblocking filter firstly and then go into SAO. SAO referred the left and upper pixels to do the offset and output completed pixels.

In the proposed architecture, there are 3 features which are beneficial to construct a high performance in-loop filter architecture for HEVC which integrate both de-blocking filter and SAO.

Firstly, luma and chroma samples of each $4 \times 4$ block are hold in same data access unit and they are transmitted simultaneously. For one $4 \times 4$ block of 4:2:0, the space for luma is 16 bytes, the space for Cb and Cr are 4 bytes respectively. In addition, when one $4 \times 4$ block is go through deblocking filter or SAO, three components of the block are processed in parallel, as shown in Fig. 12.

Secondly, in both deblocking filter and SAO, calculation core is implemented in combinational logic and data storage is implemented in register groups. Calculation core keeps processing data continually, which greatly raises the utilization of DBF core and SAO core. In our design, four luma deblocking filter cores together with four chroma deblocking filter cores finish one edge in one cycle. So four cycles are needed to filter all the four edges within one $8 \times 8$ block. Similarly, one SAO core output one $4 \times 4$ block in single cycle. So it takes four cycles to filter one $8 \times 8$ block. This structure reduces the control complexity. A counter counts from 0 to 3. Multiplexes of calculation cores and registers groups selects different data source according to the counter and send them to the calculation core (deblocking filter cores or SAO core) or registers groups. This structure also raises the utilization of the calculation cores. The deblocking filter cores and SAO core keep working continually in the decoding process. Refer Sects. 4.4 and 4.5 for details.

Thirdly, task level pipeline in processing $8 \times 8$ block is employed between deblocking filter and SAO. That is to say, when the task of processing one $8 \times 8$ block is undergoing in SAO, the task of processing its following $8 \times 8$ block is undergoing in deblocking filter. Meaning while, the registers groups of deblocking filter which holds the processed data serves as the input buffer for SAO. Thus no intermediate buffer is needed to store the data between deblocking filter and SAO. Refer Sects. 4.4 to 4.6 for details.

The rest of this section describes the proposed architecture to illustrate the mentioned feature. It is organized as follows. Section 4.1 illustrates the line buffer and left buffer of the proposed architecture shown in Fig. 11. Sections 4.2 and 4.3 introduce the deblocking filter core and SAO core respectively. Sections 4.4 and 4.5 describe the architecture and working scheme of deblocking filter and SAO. Finally, Sect. 4.6 shows the overall pipeline.

4.1 Line Buffer and Left Buffer

As shown in Fig. 11, two buffers are needed to store the neighboring pixels in processing SAO. One is the line buffer which stores the upper row data. The other is left buffer which stores the left column data. In Fig. 13, the blocks are same with Fig. 7(a). As mentioned in Sect. 3, the DB8 ABCD is the input of SAO. Before SAO of the BD8 ABCD, FG are fetched as upper pixels and EHI are fetched as left pixels. After SAO of the DB8 ABCD, IC are stored as upper pixels for below $8 \times 8$ block. GBD are stored as left pixels for right $8 \times 8$ block. For $8 \times 8$ blocks which are not the most right column of one CTB, GBD are not needed to store and EHI of next $8 \times 8$ block are not needed to fetch
because GBD of current $8 \times 8$ block directly become EHI of next $8 \times 8$ block.

The volume of left buffer is only related to the CTB size of the picture. Maximum CTB size in HEVC is 64, so eight $8 \times 8$ blocks which reside in the right column of the CTB are needed to store. For each $8 \times 8$ block, GBD are needed to store, so the volume of the left buffer shall be $3 \times 24 \times 8 = 576$ Bytes.

The volume of line buffer is related to the size of the picture. For picture of large size, the data to be stored in line buffer is quite large. For 4:2:0 picture, every $4 \times 4$ block is 24 Bytes. For 1080p ($1920 \times 1080$) picture, the capacity of the line buffer should be $1920/4 \times 24 = 11.25$ KBytes. For 2160p and 4320p pictures, the capacity is 22.5 kbytes and 45 kbytes respectively. The cost to implement it in on-chip SRAM form is too high. The line buffer is implemented as a module which accesses off-chip SDRAM, as shown in Fig. 14. A SRAM of pingpong form is used to buffer the data.

From SAO side, before SAO of SB8 (shifted $8 \times 8$ block) ABCD in Fig. 13, $4 \times 4$ data blocks F and G are fetched and they are input to SAO core. After SAO of this SB8, $4 \times 4$ blocks I and C are stored for future application. There is a “ready_to_sao” signal from line buffer module to SAO module, which means the reference data is prepared and SAO can start to work. Actually, this signal decides whether de-blocking filter can start to work. Once this signal shows that reference data is ready, the de-blocking filter and SAO can begin to work continually without the interrupt due to the delay of SDRAM. From agent side, after SAO of SB8 ABCD, $4 \times 4$ blocks I and C are store to SDRAM and reference data for next SB8 is read from SDRAM. Actually, the buffer of pingpong form allows SAO and agent work in pipeline.

The minimum volume of ping and pong is two $4 \times 4$ blocks. That is $24 \times 2 = 48$ Bytes. But in order to hide the latency of SDRAM access, the volume of ping and pong is better to be larger. The most proper volume depends on the SDRAM access latency.

4.2 Deblocking Filter Cores

The deblocking filter cores are composed of combinational logic. They are composed of four luma cores, four chroma cores and one decision module. Each core is responsible for one line to be filtered. The decision module outputs decisions on luma strong or luma weak filtering mode according to pixel value of the top and bottom row of the two $4 \times 4$ blocks. So in each luma deblocking core, there are actually 2 calculation engines and the outputs of the 2 engines are selected according to decision.

In Fig. 16(a), the simplified luma strong filtering algorithm is shown. The eight white cycles above the line means the eight input samples of the luma strong filter. The left side below the line means the modified samples in the filter process. The cycles below the line represents those samples in same column above the line whose arithmetic average is the modified sample in same row. For example, $p0'$ is the average of $p2, 2*p1, 2*p0, 2*q0, q1$. So the arithmetic average of the eight samples is $p0'$.

In Fig. 16(b), the simplified strong filtering engine of luma deblocking filter core is shown. It is composed of adder and wiring. The color of multiple samples in Fig. 16(a) correspond the color of adder in Fig. 16(b) whose output is the sum of the multiple samples of same color. This is actually a resource share technique. By sharing the source to obtain modified samples, some unnecessary adders are saved. The chroma deblocking filter core and the weak filtering engine of luma deblocking filter core is similar.

4.3 SAO Core

The SAO core is composed of 16 1-sample processors.
When one $4 \times 4$ block (p11-p44 in Fig. 17(a)) and its surrounding samples input the SAO as a $6 \times 6$ block (p00-p55 in Fig. 17(a)), each sample of the $4 \times 4$ block together with its surrounding eight samples are input to the $16$ 1-sample processors respectively, as shown in Fig. 17(b).

The 1-sample processor is shown in Fig. 18. It is combinational logic. Nine samples (A-I in Fig. 18) are input to this processor and only the center sample (E) may be modified. All other eight samples keep unchanged when they are input to the 1-sample processor to do the SAO. Some SAO parameters including sao_offset_abs, sao_eo_class, sao_band_position, sao_type_idx and sao_off are input to the 1-sample processor. Most of them are defined in the HEVC specification. sao_off is a parameter which means whether the SAO function is on or off. It is derived from other parameters in HEVC specification which includes: pcm_loop_filter_disable_flag, slice_sao_luma_flag, cu_transquant_bypass_flag, SaoTypeIdx, slice_sao_chroma_flag, sample_adaptive_offset_enabled_flag. When sao_off is on, then the 1-sample processor does not modify the input sample “E”. The original value is output. The rectangle with M in Fig. 18 means multiplex. The rectangle with “==” and “>” are comparators. The rectangle with “edge logic” and “band logic” is combinational logic for edgeIdx and bandIdx. Their truth table is shown in Fig. 18.

The function of both edge offset and band offset are supported in the 1-sample processor. In Fig. 18, the upper part is the edge filter. Six multiplexes are used to select the effective neighboring samples according to the parameter sao_eo_class. There are four sources for the neighboring samples as shown in Fig. 4. The output is “a” and “b” in Fig. 4. Then “c” is compared to “a” and “b”. The result is input to “edge logic” module which derives edgeIdx as shown in Fig. 18. EdgelIdx indexes the sao_offset_abs, the result is added to original sample “E” to get the output of edge offset process edge_off.

The lower part is the structure of band filter. The parameter sao_band_position is added to 0, 1 and 2 respectively and then the sums are compared to the band of “E”. The result of the comparison is input to “band logic” module which derives bandIdx. BandIdx indexes the sao_offset_abs, which is added to original sample “E” to get the result of band offset process band_off.

### 4.4 Deblocking Filter VLSI Architecture

Deblocking filter is composed of deblocking cores and four registers groups, as shown in Fig. 19 and Fig. 20 respectively. The gray rectangles labeled as “m” in both diagrams mean multiplexes. The number on certain arrow-line in both diagrams means that when the counter whose range is 0-3 counts to the value on the arrow-line, the output of multiplex is the data which connect to that arrow-line. DBF cores in Fig. 19 are explained in Sect. 4.2. They are combinational circuit. Thus two $4 \times 4$ data blocks besides one edge can be filtered in one cycle, as shown in Fig. 15. Four edges of one shifted $8 \times 8$ block (E0-E3, shown in Fig. 6(b)) are filtered in four cycles. dr_ia, dr_ia, dr_ec and dr_d in Fig. 20 means four registers groups which hold four $4 \times 4$ data block ABCD in Fig. 7.

Text in the dotted rectangle means the data input to the multiplex. in_ia and in_ia is the input port of the deblocking filter. *_ia means $4 \times 4$ data block are input horizontally to filter the vertical edges. That is to say, each row of the $4 \times 4$ block are input to DBF cores, respectively. *_ia means $4 \times 4$ block in the left side of the filtered edge and *_ia means $4 \times 4$ block in the right side of the filtered edge.

In cycle 0, in_ia input $4 \times 4$ block A in Fig. 7(a) and in_ia input $4 \times 4$ block B. In this cycle, edge E0 in Fig. 7(a) is filtered. The output of DBF cores are input to dr_ia and dr_ia. Thus $4 \times 4$ block A and B are stored in registers groups dr_ia and dr_ia. In cycle 1, in_ia input $4 \times 4$ block C and
in_r.h input 4 × 4 block D. In this cycle edge E1 in Fig. 7(a) is filtered. The output of DBF cores are input to dr_c and dr_d. Thus 4×4 block C and D are stored in registers groups dr_c and dr_d. In cycle 2, edge E2 in Fig. 7(a) is filtered. 4 × 4 block in dr_a and dr_c are input to DBF cores vertically to filter the horizontal edge. That is to say, columns instead of rows of these 4×4 data block are input to the DBF cores respectively, which is actually a kind of transpose. The output of DBF cores return to dr_a and dr_c vertically. It is actually another transpose and thus the 4 × 4 data block is correctly stored in dr_a and dr_c. In cycle 3, everything is similar to those in cycle 2, except that the register groups are dr_b and dr_d instead of dr_a and dr_c.

This is one period to process one 8 × 8 block. The structure that DBF cores are combinational logic while data are stored in registers groups reduces the control complexity and raises the utilization of DBF cores. In all the four cycles of one period to process one 8 × 8 block, DBF cores keep working continually.

4.5 SAO VLSI Architecture

SAO is composed of a SAO core and 6 registers groups, as shown in Fig. 21. Data are input to SAO core from registers groups of deblocking filter, line buffer and left buffer. The data go through SAO core and output. The block whose pixels are all finished is output. Other blocks are stored in SAO registers groups, upper and left buffer.

The SAO core is shown in Fig. 22. Its internal structure is illustrated in Sect. 4.3. The 6 registers groups are shown in Fig. 23 and they are sf_a, sf_b, sf_c, sf_d, sf_e and sf_f.

Same as that in deblocking filter. It also takes four cycles for one DB8 to finish SAO. SAO flow is shown in Fig. 7(b), four DB4 A’, C’, B’ and D’ are input to the SAO core one by one. As shown in Fig. 7(a), nine 4×4 data blocks (A-I) are referred to do the SAO of one DB8. Each 4×4 data

Fig. 19 DBF cores, its input data multiplexes and data input in the 4 cycles to filter one 8 × 8 block.

Fig. 20 DBF registers groups and its input multiplex (only luma part in shown for neat).

Fig. 21 SAO architecture.

Fig. 22 SAO core and input data.

Fig. 23 SAO registers groups.
block is hold in one 4 × 4 register groups. So theoretically, nine 4 × 4 registers groups are needed to hold them.

But by exploiting the time division multiplexing, only six registers groups are enough. For example, as shown in Fig. 7(b), when DB4 A′ is processed, then only four 4 × 4 blocks EFHA is required, all other blocks are not needed. So at this moment, only four registers groups are needed. When A′ is finished, E can be output because it is never referred during the processing of current 8 × 8 block. Then only eight 4 × 4 blocks are remained.

As shown in Fig. 7(b), after A′ is finished, C′ instead of B′ is processed. It is because that, once C′ is finished, two 4 × 4 blocks H and I can be output because they are never referred during the processing of current 8 × 8 block. While once B′ is finished, only one 4 × 4 block F can be output. 4 × 4 block G must remain to be 4 × 4 block E of next 8 × 8 block.

In a word, six registers groups are enough to finish the SAO process by carefully adjusting the pipeline. The detail of the pipeline is illustrated in Fig. 24.

On each cycle of SAO process, a 6 × 6 data block is input to SAO core and only center 4 × 4 data block are modified, as shown in Fig. 22. The letter in the cycle in Fig. 22 and Fig. 23 represents the 4 × 4 data block in Fig. 7(a). The 6 × 6 block is divided into 4 parts, which are called sf_0 to sf_3. sf_0 is composed 4 samples, sf_1 and sf_2 is composed of 8 samples, sf_3 is composed of 16 samples. For sf_0 part, E, H, F and A are input to SAO core when counter is 0, 1, 2 and 3. The corresponding hardware holding the data is sr_a, sr_b, sr_c, sr_d (SAO registers groups). Similar cases happen for sf_1 to sf_3. Among all the hardware multiplexed to SAO core, sr_a to sr_d are 6 SAO registers groups which serve as the temporary storage. Up is the data from line buffer. dr_a and dr_c are deblocking registers groups shown in Fig. 20.

In addition to the SAO core, there are 6 registers groups together with their multiplexes in the SAO. Each registers group is composed of 4 × 4 normal registers and another 7 shadow registers (the gray ones in Fig. 23). The shadow registers are used to store the date from deblocking filter. When the data of this block is input to SAO core, the value in the normal registers is modified while that in the shadow registers keeps un-changed. The value in the shadow registers are used to hold the neighboring pixels for drifted block in SAO. e.g. the top row and left column of the 6 × 6 pixels block in center of Fig. 22 (the right column and the bottom row of the 6 × 6 pixels block are always un-modified according to the flow). In Fig. 23, all the sources of each multiplex to the SAO registers groups are illustrated. Letter in cycle means the 4 × 4 blocks to be registered. Box in its right side is the storage hardware holding the 4 × 4 blocks. Box in its left side means the data block has just been modified by SAO core. The modified pixels are from output of SAO core instead of the storage hardware mentioned above. The data flow inferred in Fig. 19 to Fig. 23 matches that in Fig. 24.

### 4.6 Overall Pipeline

The overall pipeline of deblocking filter and SAO is shown in Fig. 24. The task level pipeline in processing 8 × 8 block is employed between deblocking filter and SAO. The upper half is deblocking filter pipeline and the lower half is SAO pipeline. The tint boxes mean the registers groups receive the newly output of deblocking core or SAO core. The grey boxes mean that the deblocking cores or the SAO core is processing data.

When dbf_cnt is 0, 4 × 4 block A and B (refer (a)) are input through in_A and in_B. They are directly wiring to the input port of the deblocking filter and A_v and B_v which mean the new 4 × 4 blocks whose intermediate vertical edge (E0) has already been filtered are output from the deblocking filter core. When dbf_cnt is 1, similar cases happen to block C and D. In addition, A_v and B_v are registered in dr_a and dr_b. When dbf_cnt is 2, C_v and D_v are registered in dr_c and dr_d. Deblocking filter cores filter the horizontal edge E2 which is between A and C. The input of the deblocking filter core is from dr_a and dr_c while the output is named as A_h and C_h. When dbf_cnt is 3, similar cases happen to B and D.

When dbf_cnt is 3, A_h and C_h are registered in dr_a and dr_c, which means that 4 × 4 block A and C has already been deblocked. At this moment (sao_cnt is 0), SAO starts to work. sf_0 means the input and output of SAO core,
which corresponds to sf_0 to sf_3 in Fig. 22. Up_ra, up_rd and up_wa/d are the read address, read data, write address and write data of line buffer. Lf_ra, Lf_rd and Lf_wa/d are the read address, read data, write address and write data of left buffer. GBD in left buffer are only read when processing the drifted 8 × 8 blocks in right column of one CTB. They will be registered into sr_wa, sr� and srэ as the left reference data block of next 8 × 8 block. If it is not the drifted 8 × 8 blocks in right column of one CTB, sr_wa, sr� and srэ register the output of SAO core. Because GBD of this 8 × 8 block is EHI of next 8 × 8 block. G and F are read from line buffer in proper timing. The number after the letter means how many times this 4 × 4 block is filtered by SAO core in this period of processing one 8 × 8 block. For example, when sao_cnt is 1, sr_a registers E0. This means that 4 × 4 block E has already been filtered by SAO core for once. And for E, this processing finish all the pixels in E and it is output through ou_wa/d. Output to left buffer and line buffer is similar, through up_wa/d and Lf_wa/d.

The detail arrangement of the pipeline is in order to reduce the number of necessary registers groups. And it successfully save three registers groups.

5. Implementation Results

The implementation results are shown in Table 1. Fujitsu e-shuttle 65 nm technology is used in our implementation. The result is compared to the result of HM9. Since HEVC is a new video coding standard, till now, there do not exist many works on HEVC. H.264 is the most recent international video coding standard, and the deblocking filter of HEVC is somehow similar to that of H.264. Related works on either HEVC or H.264 are referred to compare this work in this section.

From the function perspective, [5] and [6] only supports H.264 deblocking filter function, [4] supports both HEVC and H.264 deblocking filter function. This work is the only one to support HEVC SAO, which is actually the critical part of HEVC in-loop filter.

From the decoding performance perspective, this work can process one 8 × 8 block in only 4 cycles. That is to say, 16 pixels can be processed in every cycle. It is at least 50% faster than all of the other works. To decoding 4320p (7680 × 4320) @ 60 fps, it only need to run under 124.4 MHZ. Other works need to run above 186 MHZ to decode 4320p @ 60 fps. In addition, this work can be synthesized to 240 MHz by 65 nm technology. Under this frequency, it is capable of decoding 3.84 Gpixels/s. It is the maximum throughput of this work and it is 1.8 times of [4]. For [5] and [6], even if their technology is scaled to 65 nm and synthesis frequency reaches 240 MHz, their throughput is still one-third of or lower than this work.

From the cost perspective, this work is the highest. One reason why cost of this work is highest is that only this work support HEVC SAO, which accounts for more than half of the total cost. In addition, the number of deblocking filter cores of this work is more than others, because four luma and four chroma deblocking filters are implemented in this architecture. This is why DBF cost of this work is higher than that of [5] and [6]. The deblocking filter part cost of this work is lower than [4] because [4] supports both H.264 and HEVC. In addition, resources share mentioned in Sect. 4.2 may be a contribution. If we consider both deblocking filter and SAO, the registers groups in deblocking filter are shared as the input buffer to SAO, which remove any intermediate storage between deblocking filter and SAO and save the hardware cost. The difference in RAMs between this work and [4] is the statistic scope. In this article, only RAMs holding pixels in the process of decoding is counted. RAMs holding pixels before the in-loop filter process is not counted. The reason why RAMs of this work is larger than [5] is due to the algorithms of the standard. In HEVC, CTB is 64 × 64 and in H.264, MB is 16 × 16.

In a word, although the cost of this work is higher than others, it brings the dramatic boost of performance and it is the only one to support HEVC SAO function.

6. Conclusion

In this article, we propose a high performance HEVC de-blocking and SAO architecture for UHDTV decoder. In this architecture, SAO is processed based on drifted block; the luma and chroma components are processed in parallel; the structure of both deblocking filter and SAO filter are based on combinational calculation logic and registers groups’ storage; task level pipeline is employed between de-blocking filter and SAO filter.

The proposed in-loop filter architecture takes only four cycles to finish the de-blocking filter and SAO of one 8 × 8 block. It can be synthesized to 240 MHz with Fujitsu e-shuttle 65 nm technology. Thus this solution can process 3.84G pixels/s at maximum. UHDTV 4320p (7680 × 4320) @ 60 fps decoding can be realized with only 124.4 MHz working frequency by the proposed architecture.
Acknowledgments

This research was supported by the regional innovation strategy support program of MEXT.

References

[9] [Online]. https://hevc.hhi.fraunhofer.de/svn/HEVCSoftware/tags/

Jiayi Zhu received the B.E. and M.E. degrees from Shanghai Jiao Tong University, Shanghai, China. He is currently a Researcher with Waseda University, Kitakyushu, Japan. His interests are in algorithms and VLSI architectures for multimedia and communication signal processing.

Dajiang Zhou received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China. He received the Ph.D. degree in engineering from Waseda University, Japan, in 2010, where he is currently a researcher. His interests are in algorithms and VLSI architectures for multimedia and communication signal processing.

Satoshi Goto was born on January 3rd, 1945 in Hiroshima, Japan. He received the B.E. degree and the M.E. degree in Electronics and Communication Engineering from Waseda University in 1968 and 1970 respectively. He also received the Dr. of Engineering from the same university in 1981. He is IEEE fellow, Member of Academy Engineering Society of Japan and professor of Waseda University. His research interests include LSI System and Multimedia System.