DMATP: A Design Method and Architecture of TU Parallel Processing for 4K HEVC Hardware Encoder

Seiya Shibata
e-mail : s-shibata@ax.jp.nec.com

Eita Kobayashi
e-mail : e-kobayashi@fg.jp.nec.com

Noriaki Suzuki
e-mail : n-suzuki@ha.jp.nec.com

Atsufumi Shibayama
e-mail : sibayama@cd.jp.nec.com

Takeo Hosomi
e-mail : hosomi@ah.jp.nec.com

Green Platform Research Laboratories, NEC Corporation,
Kawasaki, Kanagawa 211-8666, Japan

Abstract - This paper proposes design method and architecture of parallel processing hardware for Transform Units in High Efficiency Video Coding (HEVC). HEVC is the next generation video coding standard which is expected to be used for high resolution broadcasting such as 4K UltraHD. Since HEVC introduces higher complexities and dependencies than previous standard H.264/AVC, hardware designers have to find and utilize parallelism in HEVC to realize strict real-time encoding performance especially for broadcasting purpose. We propose design method to find appropriate parallelism considering both HEVC algorithm and hardware resources focusing on the Transform Units processing, and propose architecture to bring the parallelism efficiently. With the architecture, we got a prospect of realizing 4K HEVC encoder.

I. Introduction

Emerging 4K UltraHD (3840x2160) video broadcasting brings 8 times as much pixels as current FullHD video. High Efficiency Video Coding (HEVC) [1], that is the next generation video compression standard, is a promising solution to reduce the impact of the amount of pixels. HEVC is expected to achieve double compression ratio compared to previous standard H.264/AVC [2], and it is realized by utilizing correlations among pixels in pictures with highly complex prediction and compression techniques. On the other hand, the complexity of the processing is making harder to implement HEVC encoder achieving both real time and the high compression ratio. In particular, increased dependencies among image blocks to be processed made utilizing parallelism difficult. Therefore, parallelism exploitation is becoming more important than H.264/AVC encoder design.

For the enterprise broadcasting purpose, real-time encoding with low delay and constant throughput is required and the throughput should be strictly guaranteed. Therefore hardware implementation is suitable. In the view point of hardware design, parallel execution of processes generally results in increase of costs such as area, power and energy consumption. In order to design hardware meeting required performance with minimum cost, parallelism should be sufficient and minimum.

Since recent video coding standards adopts block-by-block processing algorithms, hardware encoders generally take pipelined architecture where each pipeline stage processes a block for achieving required throughput with utilizing resources. In such architecture, each pipeline stage should finish processing of a block in a certain number of cycles.

In this paper, we focus on design of a single stage for "Intra Transform Unit (TU) loop", described later. Intra TU loop is one of the bottleneck parts which is difficult to be separated into multiple stages and therefore it determines maximum throughput of the encoder. Note that other stages, which we do not consider in this paper, can be relatively easy to design since they can be separated into multiple stages by algorithm-level modification or by other techniques.

The contribution of this paper is following two. One is a design method with automatic exploration of minimum parallel processing architecture for Intra TU loops which meets real-time requirement, and the other is architecture to realize the parallel processing. By combination of the two techniques, designers can design appropriate TU parallel processing architecture for required performance.

The remaining of this paper is organized as follows. In Section II, overview of HEVC and its Transform Unit are explained. Section III shows related works. Then Section IV shows the proposed method and architecture in detail and Section V demonstrates the proposed method with an example. Finally, Section VI concludes this paper.

II. HEVC Transform Units Processing

A. HEVC overview

In general, HEVC encoder splits input pictures into blocks and compress them one-by-one applying functions shown in Fig. 1. For a block, the encoder explores appropriate parameters for functions such as direction modes for intra-picture prediction, motion vectors for inter-picture predictions, block partitions (described later) and quantization parameters ("Parameter exploration"). After the parameter exploration, the encoder generates prediction image using "Intra prediction (IPRED)" or "Inter prediction", generate residual image by subtracting them from original image ("SUB"), and transform residual image using "DCT" and quantization ("Q"). Since IPRED require reconstructed image (in other words, local decode image) of adjacent blocks, inverse quantization ("IQ"), inverse DCT ("IDCT"), and reconstruction ("RECO") should be conducted before processing next block. Therefore, the functions from IPRED to RECO make a loop shown as "Intra TU loop" in Fig. 1.

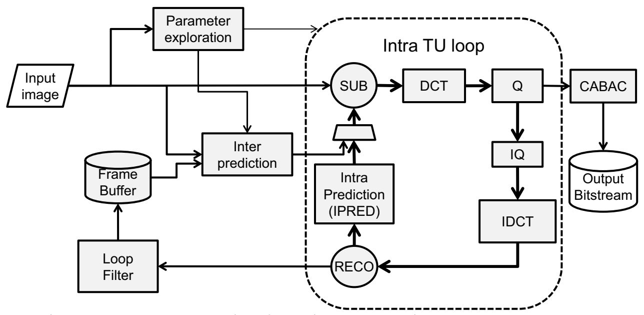


Fig. 1 : HEVC encoder functions overview

On the end, encoder compresses parameters and transformed images with arithmetic encoder (“CABAC”) and outputs HEVC bitstream.

B. Quad-tree based block partitions

One of the biggest differences between H.264/AVC and HEVC is the block size to be processed. In H.264/AVC, pictures are separated into 16x16 pixel blocks which are called MacroBlocks(MBs) and coded per MBs. Variation of transformation size is 4x4 and 8x8 and the sizes are not mixed in an MB. In HEVC, pictures are separated into large blocks (i.e. 64x64 pixel blocks) which are called CTUs (Coding Tree Units) which are processed in raster scan order and the blocks are further split into small blocks in quad-tree manner as shown in Fig. 2. First, CTU is split into Coding Units (CUs) and CUs can be further split into Prediction Units (PUs) and Transform Units (TUs). Simply speaking, CUs are the units to be encoded, PUs are used to transport block level prediction information, and TUs are used to transport residues which DCTs and quantization are applied. In the standard specification, CU size ranges from 64x64 to 8x8, and TU size ranges from 32x32 to 4x4. As shown in Fig. 2, various block sizes can be mixed in a CTU.

TUs in a CTU are processed in so called “Z-order” as shown in Fig. 3. A block depends on and requires results of adjacent former blocks in Z-order.

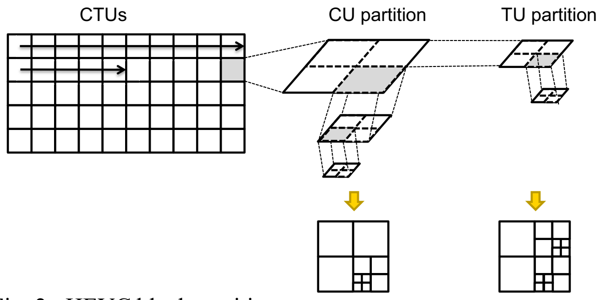


Fig. 2 : HEVC block partitions

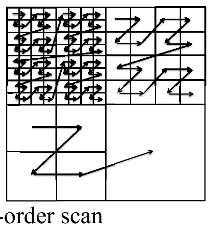


Fig. 3 : Example of Z-order scan

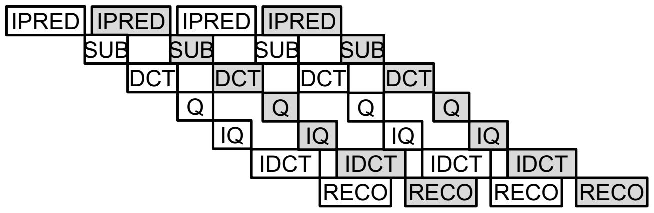


Fig. 4 : Ideal TU processing pipeline (mostly impossible)

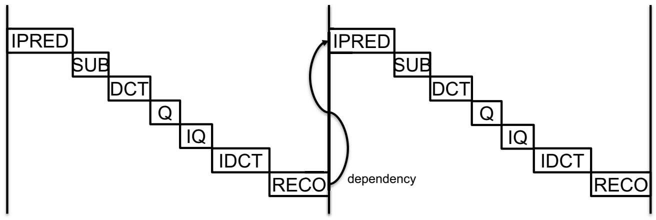


Fig. 5 : Dependency between TUs

C. Design matters on Transform Units processing

For each TU, encoder processes Intra TU loop which consists of IPRED, SUB, DCT, Q, IQ, IDCT, and RECO. Note that this structure is same to H.264/AVC.

As described before, IPRED for a TU uses reconstructed pixels of previous TUs generated by RECO. This brings two difficulties for parallel processing of Intra TU loops.

One is difficulty on pipelined processing design. If there were no dependency among TUs, functions of TUs can perform pipelined processing as shown in Fig.4 and can achieve high throughput. However, in actuality, TUs mostly cannot be processed in such pipeline manner since IPRED for a TU uses reconstructed pixels of just a previous adjacent TU in Z-order generated by RECO. Therefore most TUs should be processed in sequential manner as shown in Fig. 5.

The other is difficulty on data parallel processing design. The number of adjacent reconstructed pixels required by IPRED is larger in HEVC than in H.264/AVC. As shown in Fig.6, when a size of TU is NxN, adjacent 4N+1 pixels are used at maximum (N for top, N for top-right, N for left, N for bottom left, and 1 for top-left). The bottom part of Fig.6 illustrates executable timing of 4x4 blocks in case two 4x4 blocks can be processed in parallel, as an example. The number with prime such as “ 5’ ” represents that the blocks can be executed in parallel with the block “5”. Because of the dependency issue, only 2 pair of 4x4 blocks can be processed in parallel on HEVC and results in requiring 14 block time for 16 4x4 blocks. In contrast, IPRED of H.264/AVC requires only 3N+1 pixels and 6 pair of 4x4 blocks can be processed in parallel. Processing of 16 4x4 blocks can be finished in 10 block time.

Furthermore, analysis of the worst case execution time is another problem because of vast number of TU partition patterns including mix of TU sizes. For instance, although all 4x4 TU partition pattern will be the worst when all TUs are executed in sequential manner, the pattern is not the worst one on the architecture where two 4x4 TUs can be processed in parallel. Other partition pattern, especially where multiple TU sizes are mixed, will be the new worst one.

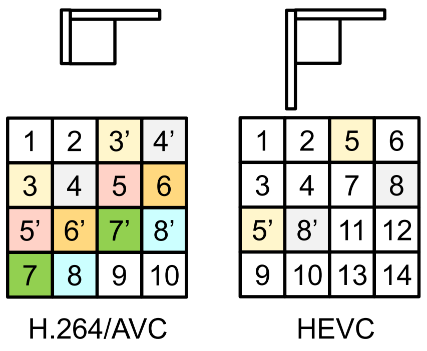


Fig. 6 : Parallelism comparison between H.264/AVC and HEVC

D. *Real-time Requirements for TU processing*

Given the target frame rate (frames per second) R, encode time spent for a picture $T_{\_Pic}$ should hold $T_{\_Pic} < 1/R$ for every pictures. For example, $T_{\_Pic}$ should be smaller than 16.6 msec. when 60 frames per second is required.

The number of cycles for a picture $C_{\_Pic}$ and that for a CTU are obtained from T_pic, clock frequency of a device F, and the number of CTUs in a picture $N_{\_CTU}$.

$$C_{\_Pic} = T_{\_Pic} * F$$
$$C_{\_CTU} = C_{\_Pic} / N_{\_CTU}$$

As a result, requirement for $C_{\_CTU}$ is defined as follows.

$$C_{\_CTU} < F / (N_{\_CTU} * R)$$

When we consider $C_{\_CTU}$ to be a throughput of HEVC encoder and processing time of Intra TU loops to be the bottleneck, the worst case number of cycles spent for Intra TU loops will be $C_{\_CTU}$. For example, if 64x64 CTU is split into 256 4x4 TUs and they are processed sequentially, $C_{\_CTU}$ is obtained by followings using the number of cycles for a 4x4 TU $C_{\_TU\_4x4}$.

$$C_{\_CTU} = 256 * C_{\_TU\_4x4}$$

$C_{\_TU\_NxN}$ is obtained by summation of the number of cycles of functions in Intra TU loop $C_{\_M\_NxN}$ for each TU size.

$$C_{\_TU\_NxN} = \Sigma_M \{ C_{\_M\_NxN} \}$$
where M={IPRED, SUB, DCT, Q, IQ, RECO}, N={4, 8, 16, 32}.

Note that if equation $C_{\_M\_2Nx2N} = 4*C_{\_M\_NxN}$ holds for all M and N, the problem is simple and impossible to solve since the four 32x32TU case become the worst one since four 32x32 TUs can not be processed in parallel because of their dependencies. Therefore, the relationships among $C_{\_TU\_NxN}$ should hold following conditions.

$$C_{\_TU\_32x32} < 4*C_{\_TU\_16x16} < 16*C_{\_TU\_8x8} < 64*C_{\_TU\_4x4}$$

The condition above can be achieved if execution cycles of some parts of intra TU loop are constant regardless of TU size. For instance, SUB, Q and IQ can be implemented to be fully parallelized for pixels in a TU and can get constant execution time.

III. Related Works

There are some works to improve performance and/or area of a certain module such as IPRED [3][4] and DCT [5][6]. However, as long as we know, there are few works which consider and optimize whole TU process of HEVC.

For the previous standard H.264/AVC, there are some works to overcome dependency among blocks in intra TU loop. In [7] and [8], processing order of 4x4 blocks are changed and some intra prediction modes of them are restricted in order to relax dependency among 4x4 blocks and increase the chance for pipeline execution. In [9], processing order of 8x8 blocks and 16x16 blocks also changed and pipelined. Although these approaches can increase throughput of intra TU loops processing, it introduces some amount of image quality loss because of restriction of intra prediction modes. Moreover, when these approaches applied to HEVC encoder, the effect will be limited because of the differences between H.264 and HEVC on block partition manner and on the number of reconstructed pixels used in intra prediction as described in Section II. Compared to these works, our method does not restrict any intra prediction mode and therefore do not introduce any image quality loss.

IV. Design Methods for TU Parallel Processing

A. *Overview*

In order to design TU processing part of HEVC encoder which meets real-time requirement with minimum and sufficient amount of parallelism, we developed automatic exploration method of TU processing parallelism and architecture for efficient parallel processing management.

Hereafter, we call functions for Intra TU loop “TU processor” as a unit, and explore parallelism of TU processing by the number of TU processors.

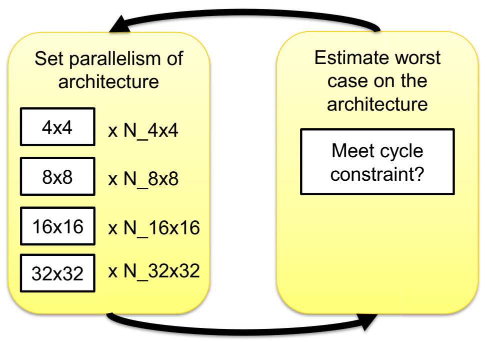
B. *Automatic parallelism exploration method*

Automatic exploration of parallelism for TU processing, in other words, the number of TU processors, is realized with performance estimation method and iterative execution of it.

The performance estimation method takes $C_{\_M\_NxN}$, the number of TU processors (N_4x4, N_8x8, N_16x16 and N_32x32), and the required number of cycles for a CTU as input and calculates the worst number of cycles spent for a CTU considering all partition patterns, dependency of TUs and availability of TU processors.

The exploration starts with a single TU processor and gradually increase the number of TU processors in its iterations until estimated $C_{\_CTU}$ become smaller than required number of cycles as shown in Fig.7.

The algorithm of performance estimation is shown in Algorithm 1. In brief, the estimation method generates all



Set parallelism of architecture
4x4
x N_4x4
8x8
x N_8x8
16x16
x N_16x16
32x32
x N_32x32
Estimate worst case on the architecture
Meet cycle constraint?

Fig. 7 : Exploration Method

```
1: Performance_estimation()
2: Begin Process
3:   M = get_modules() // available TU processors
4:   TP = generate_partitions() // TU partition patterns
5:   WCEC = 0 // worst case execution cycles
6:   WCP   = None // worst case partition
7:   For T in TP begin
8:     S = Schedule()   // empty schedule
9:     EC = 0 // execution cycles
10:    While T != empty
11:      E = get_ TU_Zorder(T)
12:      scheduleASAP(S, M, E)
13:    End While
14:    EC = get_EC(S)
15:    If(WCEC > EC) then
16:      WCEC = EC
17:      WCP   = P
18:    End if
19:  End For
20:End Process
```

Algorithm 1 : performance estimation method

partition patterns ("generate_partitions()") and estimate execution cycles for each partition pattern. In the execution cycle estimation of a partition pattern, TUs are scheduled one-by-one in Z order. The processing timings of TUs in a CTU are scheduled by ASAP manner, where a TU is scheduled to execute just after the timing when the dependent TUs are all finished and the TU processors for the TU is available.

After scheduling of all TUs, the number of execution cycles of a partition pattern is obtained as EC ("get_EC(S)"). EC is stored as worst case execution cycles (WCEC) if it is larger than others. On the end of all partition estimation, WCEC on the architecture is output.

Although the performance estimation method achieves fast estimation by its simple implementation, the number of partitions of 64x64 CTU is vast huge and reaches almost 2^{84} patterns. In order to cut off unnecessary estimation, the estimation method generates WCEC for smaller CTU size such as 32x32 in advance, and stores it as $WCEC_{\_NxN}$. If $4xWCEC_{\_NxN}$ is smaller than the requirement, the method judges that all actual partitions will meet the requirement and finishes estimation. This is because $WCEC_{\_2Nx2N} <= 4xWCEC_{\_NxN}$ always holds as long as $C_{\_TU\_2Nx2N} < C_{\_TU\_NxN}$ holds as described in Section II.

C. *Architecture for TU Parallel Processing*

In order to realize parallel TU processing explored and determined by automatic exploration method shown in above, TUs whose dependent TUs are all finished should be issued to TU processors as soon as possible. Therefore we developed Dependency Management Module (DMM) and Issuer Module (IM) placed in front of TU processors as shown in Fig. 8. DMM manages location of finished TUs and that of executable TUs. IM takes executable locations managed by DMM and drive appropriate TU processors. The numbers of TU processors are represented as $N_{\_4x4}$, $N_{\_8x8}$, $N_{\_16x16}$ and $N_{\_32x32}$.

DMM has two groups of register maps as shown in Fig. 9. One is the "End maps" and the other is "Resolved maps". Each group of maps contains maps for 4 sizes of TUs. End maps stores the location of TUs whose processing is finished. Resolved maps have the location of TUs who are able to be processed. Every time when a TU processing is started (not finished), DMM updates the End map for 4x4TU by itself, and End maps of larger sizes and Resolved maps of all sizes are updated automatically according to End map of 4x4TU. Each Resolved map outputs the register values as bit strings in Z-order. "mask4x4", "mask8x8", "mask16x16" and "mask32x32" are bit strings representing input TU partitions. By masking the outputs of Resolved maps using the masks, IM can find sizes and positions of executable TUs, and can issue them to TU processors.

The automatic update mechanism is briefly illustrated in Fig. 10. Update mechanism of End maps from NxN map to 2Nx2N map (for example 4x4TU map to 8x8TU map) is implemented using 4 input AND logics. This is simply because pixel region of four NxN TU is equal to those of 2Nx2N TU. By fixing map which is updated by DMM to 4x4TU map, automatic update logics can be uni-directional from maps for small size to those for large ones, and can be simple. The number of 4 input AND logics used in the End map is 85 in total.

Update mechanism of NxN Resolved maps from NxN End maps uses 6 input AND logic. 5 of 6 inputs are End map value of adjacent NxN blocks. If adjacent blocks are not available (blocks whose processing order in Z-scan is latter than the target position or the block position is outside of CTU), the input value is fixed to 1. Rest 1 input is NOT signal of End map value of the target position in order to make the value 1 only for the position which can be executed next (not finished). The number of 6 input AND logics used in the Resolved maps is 340 in total.

Since automatic update of maps is done in parallel to TU processing, IM can search bits of Resovled maps for executable TUs and can issue them immediately when processing of current TUs finished.

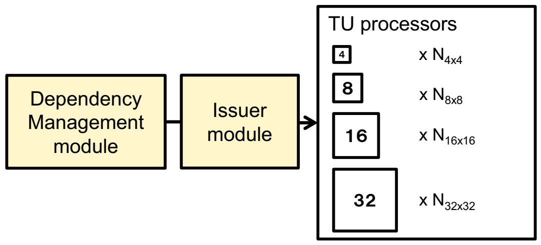


Fig. 8 : parallel TU processing architecture

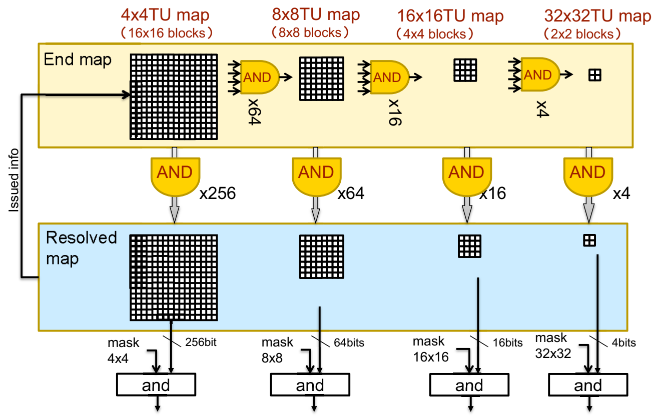


Fig. 9 : Maps in dependency management module

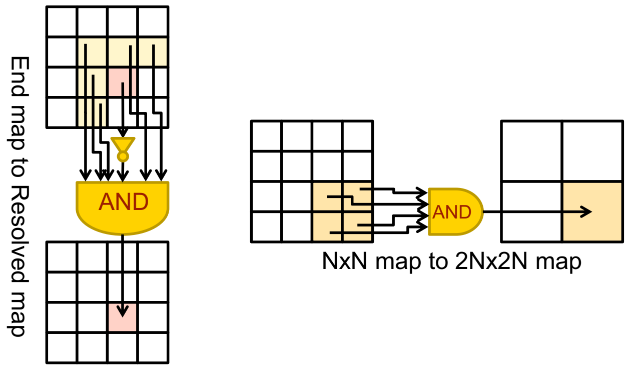


Fig. 10 : Automatic update mechanism of maps

V. Example of Proposed Design Method

Here we demonstrate an example of design. Table 1 shows $C_{\_M\_NxN}$ of each modules M and sizes NxN in this example. Basically all modules are considered to have capabilities to process 16pixels/cycle. As an exception, SUB and IQ can be executed in parallel with IPRED and Q, respectively, and therefore their latency (1 cycle) is shown as their values. With these values, we will design intra TU loop targeting 4K UltraHD (3840x2160 pixels) 60 frames per second encoder. Clock frequency F is assumed to be 200MHz. On this requirement, $N_{\_CTU}$ becomes 2025 (*=3840*2160/64/64*) and $C_{\_CTU}$ should hold $C_{\_CTU}<1646.09...$ $(=200*10^6 / (2025 * 60))$. If TUs are processed in sequential manner, all 4x4 TU case would take *1792(=7*256)* cycles and results in violating the real-time requirement.

Thus we conducted exploration of the number of TU processors using proposed automatic exploration method. As a result, we obtained a design with TU processors where

TABLE 1 : Exam`ples of $C_{\_M\_NxN}$ definition

	IPRED	SUB	DCT	Q	IQ	IDCT	RECO	TU total
4x4	1	1	1	1	1	1	1	7
8x8	4	1	4	4	1	4	4	22
16x16	16	1	16	16	1	16	16	82
32x32	64	1	64	64	1	64	64	322

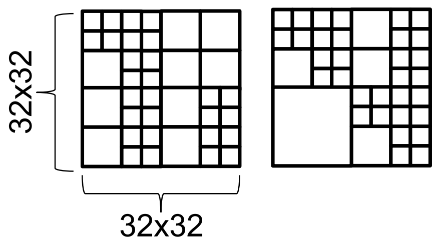


Fig. 11 : Two worst case block partition with two 4x4 TU processors on 32x32 size

$N_{\_4x4}$ = 2 and $N_{\_8x8}$ = $N_{\_16x16}$ = $N_{\_32x32}$ = 1 can meet the requirement with minimum parallelism. On the design, two worst case partitions of 32x32 CTU was obtained as shown in Fig. 11. On the left case in Fig.11, there are 28 4x4TUs and nine 8x8TUs and they cannot be executed in parallel, and it takes 394 *(=7*28+22*9)* cycles in total. On the right case, it takes also 394 *(=7*32+22*4+82)* cycles by sequential execution of 32 4x4TUs, four 8x8TUs and one 16x16TU. As a result, WCEC of 32x32 was found to be 394. Since 1576 *(=4*394)* is smaller than requirement 1646, WCEC of 64x64 CTU is also guaranteed to be smaller than the requirement, and performance estimation of 64x64 is skipped.

The performance estimation method is implemented using Python language and executed on host PC with Intel Xeon CPU 3.60 GHz clock frequency and 64GB DRAMs, and took only about 16 seconds to obtain exploration results.

VI. Summary and Conclusions

We proposed design method and architecture of parallel TU processing for HEVC encoder hardware. Intra TU loops in HEVC encoder is one of the bottleneck part because of strong dependencies among functions and among blocks. Proposed design method with performance estimation can explore parallelism of TU processing considering variety of TU partitions exhaustively and can automatically obtain minimum parallelism where the worst case execution cycles do not exceed required amount of cycles. Proposed architecture realizes to handle parallel processing of multiple TUs with small logics. We demonstrated the effect of our method using an example and could obtain a prospect of realizing 4K HEVC encoder.

References

[1] ITU-T, “H.265 High efficiency video coding,” Apr., 2013.

[2] G. J. Sullivan, J. Ohm, W. J. Han and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC)

Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol.22, no.12, pp.1649-1668, Dec. 2012.

[3] L. Fu, S. Guangming and W. Feng, "An efficient VLSI architecture for 4×4 intra prediction in the High Efficiency Video Coding (HEVC) standard," *18th IEEE International Conference on Image Processing (ICIP)*, pp.373-376, 11-14 Sept. 2011.

[4] Y. Jiang; D. Llamocca, M. Pattichis and G. Esakki, "A unified and pipelined hardware architecture for implementing intra prediction in HEVC," *2014 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI),* pp.29-32, Apr., 2014.

[5] J. S. Park, W. J. Nam, S. M. Han and S. Lee, "2-D large inverse transform (16x16, 32x32) for HEVC (high efficiency video coding)," *Journal of Semiconductor Technology and Science*, vol.*12*, no.2, pp.203-211, 2012.

[6] A. Edirisuriya, A. Madanayake, R. J. Cintra and F. M. Bayer, "A multiplication-free digital architecture for 16× 16 2-D DCT/DST transform for HEVC," *IEEE 27th Convention of Electrical & Electronics Engineers in Israel (IEEEI),* pp. 1-5, 2012.

[7] W. Lee, S. Lee and J. Kim, "Pipelined Intra Prediction Using Shuffled Encoding Order for H.264/AVC," *2006 IEEE Region 10 Conference*, pp.1-4, Nov. 2006.

[8] G. Jin and H. J. Lee, "A parallel and pipelined execution of H. 264/AVC intra prediction," *The Sixth IEEE International Conference on Computer and Information Technology 2006,* pp. 246-246, Sep., 2006.

[9] G. He, D. Zhou, J. Zhou and S. Goto, "A 1991 Mpixels/s intra prediction architecture for Super Hi-Vision H.264/AVC encoder," *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pp.1054-1058, Aug. 2012.