Waseda University Doctoral Dissertation

Fast Algorithm and VLSI Architecture of HEVC Mode Decision and Reconstruction Loop Based on Data Reuse and Reordering

Heming SUN

Graduate School of Information, Production and Systems

Waseda University

February 2017

Abstract

With the development of the information society, multimedia contents are widely used. Video data occupies the majority of multimedia data and it will dramatically grow when high definition (HD) and ultra-HD video applications are popularized in the near future. In order to relieve the burden of video storage and transmission, video compression technique has been widely used. By encoding, large raw video data is compressed to small binary data. By decoding, the compressed data is decompressed for display. High efficiency video coding (HEVC) is the latest video compression standard which doubles the compression ratio as its predecessor H.264/AVC. However, to reach such high compression ability, many new coding features are adopted. As a result, the encoding/decoding complexity becomes 5.2x/2.1x higher than H.264. So low-complexity algorithms and architectures for HEVC are extremely desired.

Mode decision and reconstruction loop are two indispensable components in the video coding. Mode decision is used to select the best mode which has the smallest rate-distortion (R-D) cost. After choosing the best mode, the reconstruction loop is conducted to generate the reconstructed pixels for the mode decision afterwards. In the mode decision, the residual of the original and predicted picture passes through forward transform and quantization to reduce the data volume. Rate represents the requiring bits for coding quantized transformed residual and the best mode information. After that, de-quantization and inverse transform are conducted to recover the residual and generate the reconstructed picture. Since quantization is lossy, the reconstructed picture is different from the original picture and distortion is used to reflect the degree of difference.

In HEVC, mode decision and reconstruction loop become much more complex and important due to two reasons. The first reason comes from the adoption of large transform in HEVC. The largest transform size in HEVC is 32×32 which is 16x larger than H.264, which will lead to huge hardware consumption. In the state-of-the-art HEVC intra encoder [Pastuszak, TCSVT 2015], transform consumes about 53% of

the overall gate counts. Therefore, the area-efficient designs for transform are highly required. Moreover, due to the large transform size, many high-frequency quantized transformed coefficients will be zero. The processing for the zero elements can be optimized. The second reason is that there are many more modes in HEVC than H.264. For intra prediction, 5 prediction units (PU) and 35 prediction modes are supported. Overall, 175 modes are provided in HEVC. However, there are only 2 PUs and 9 prediction modes for intra prediction in H.264. The complexity of calculating the R-D costs for all the modes is high. Therefore, reducing the number of modes is extremely necessary. In this thesis, low-complexity algorithms and architectures for three research topics of the mode decision and reconstruction loop are proposed. Because the simple parallelization will still suffer from high hardware cost such as large area and power consumption, the acceleration based on data reuse and reordering is studied.

The thesis is composed of five chapters.

In Chapter 1, the video encoding diagram is described at first. And then, the position of mode decision and reconstruction loop in the video coding and their relationship are presented. Finally, the motivations of choosing three research topics are given.

In Chapter 2, an area-efficient architecture for transform is presented. A complete transform is composed of row and column transform which require the logical computational part. In addition, a transpose buffer is required to store the results of row transform. For the logical computational part, Chen's algorithm is adopted so that the symmetric property of the transform matrix can be utilized to reduce the number of multiplications and additions. In addition, a reordered parallel-in serial-out (RPISO) scheme is proposed so that the inputs of the butterfly structure could be reused in each clock cycle. As a result, 25% normalized gate counts are saved compared with [Shen, IEICE 2013]. For the transpose buffer part, static random-access memory (SRAM) instead of register is adopted in order to reduce the area consumption. The storing positions in the SRAM are reordered so that it can achieve 100% I/O utilization of SRAM. In addition, two data mapping methods are proposed so that write and read

operation can be operated in parallel. As a result, about 62% area consumption can be reduced compared with the SRAM-based transpose buffer in [Zhu, ISCAS 2013].

In Chapter 3, a low-cost system design of the de-quantization and inverse transform is presented. The system can be used to generate the reconstructed pixels in both encoder and decoder. For the de-quantization, the input coefficients are multiplied with scaling parameters. In order to reduce the number of multiplications, the input coefficients are decomposed to base part (baseLevel) and remaining part. The value of base part is not greater than 3, thus the multiplication of baseLevel and scaling parameter can be replaced by look-up-tables (LUTs). For the remaining part, the number of positions with non-zero value is usually not greater than four in one 4×4 block. Therefore, only four multipliers are provided for processing one 4×4 block. In the system, there are three memory operations: read operation of the buffer between de-quantization and inverse transform, write and read operation of the transpose buffer of inverse transform. A specific path is created to detect zero elements by reusing the pixel data. After the detection, the memory operations are skipped for the zero elements. As a result, for the de-quantization, 77% normalized area consumption is reduced compared with [Tikekar, ICIP 2014]. For the memory parts in the system, 29%-86% power consumption is saved compared with not skipping the memory operations for the zero elements.

In Chapter 4, a prediction unit (PU) depth and prediction mode selection algorithm for intra prediction is proposed. At first, a fast preprocessing stage based on a proposed cost model is presented. After estimating the costs for 8×8 PU, the results are reused to predict the costs for larger PUs. Based on the estimated costs of all the PUs, 2 neighboring PU depths out of 5 are selected to do the R-D cost calculation. Still based on the preprocessing results, a prediction mode selection scheme eliminates the necessity to perform fine Hadamard cost calculation in the original HM. A 32×32 PU compensation scheme is also exploited to alleviate the mismatch problem between proposed simplified cost and R-D cost due to the lack of large transform size in proposed cost model. The compensation scheme is able to effectively improve

coding performance for high-resolution sequences. In comparison with HM (version 7.0), the proposed algorithm achieves about 52% encoding time reduction, with the corresponding 1.87% BD-bitrate increment. Compared with [Xiong, ISPACS 2012], the coding efficiency becomes worse by 0.62% in terms of BD-bitrate. However, the encoding time reduction is increased by 14% on average and 23% in the best case. Compared with [Zhang, VCIP 2012], the encoding time is increased by 5%. However, the coding efficiency becomes better by 3.23% in terms of BD-bitrate.

In Chapter 5, the conclusion and future work are given. The methods in Chapter 2 and 3 are designed for mode decision and reconstruction loop and contribute to both intra and inter prediction. The proposals in Chapter 4 intend to reduce the complexity of mode decision for intra prediction. In the future, the other components will be designed and the overall system pipeline schedule will be designed.

Acknowledgement

I would like to thank my supervisor, Professor Shinji Kimura, for his guidance in my research work. Professor Kimura talks with me about my research topic, which broadens my views and helps me to improve my research. In addition, when I write the piece of paper or dissertation, Professor Kimura reviews it very seriously and always gives many comments and suggestions in detail, from which I benefit a lot. When I do the seminar, he asks many questions about my research and gives me suggestions. Professor Kimura also leaves me sufficient freedom to do the research I am interested. I enjoyed and appreciated all of it very much.

I would also like to thank Professor Satoshi Goto, who also gives a lot of guidance on my research. He creates a good platform and provides good conditions for my research work. I gain a lot from him about how to do the research as a doctor candidate. I would also like to thank Professor Takahiro Watanabe and Professor Takeshi Yoshimura for their advice in improving my dissertation.

I would also like to thank Professor Takeshi Ikenaga for his comments to my dissertation. The comments are really helpful for improving the quality of my dissertation.

In addition, I am grateful to Dr. Dajiang Zhou, Dr. Jinjia Zhou, Mr. Jiayi Zhu, Mr. Shuping Zhang, Miss. Li Guo, Mr. Shihao Wang, Mr. Jianbin Zhou, Miss. Landan Hu, Miss. Zhengxue Cheng and other lab mates. They work together with me on video coding. Only through the team work, I can achieve good research results.

Finally, I should thank the “Graduate Program for Embodiment Informatics” initiated by the Ministry of Education, Culture, Sports, Science and Technology, Japan, for its support.

Contents

Index of Figures

Index of Tables

1. Introduction

1.1 Background

With the development of the information society, multimedia contents are widely used. Video data occupies the majority of multimedia data and it is forecasted to dramatically grow with high definition (HD) and ultra-high definition (UHD) video applications being popularized in the near future. According to the report in [1], it is forecasted that video traffic will consume about 82% of the internet traffic by 2020. In order to relieve the burden of storage and transmission for the video, the video compression techniques are widely used. In the past 20 years, there have been many video compression standards such as MPEG-2 and H.264. The development trend of standards is to achieve the higher compression ratio to meet the increase of raw video data. Recently, 8K (7680x4320) UHD has come out to further improve the visual effect. However, the storage and transmission of 8K UHD videos become a critical problem because of 16x more pixels compared with the 1080p (1920x1080) HD. In addition, the frame rate for 8K UHD can be as fast as 120 frame per second (fps) which is 4x faster than the popular 30 fps. Therefore, a new video compression standard with a higher compression ratio is required. So High Efficiency Video Coding (HEVC) [2] comes out. HEVC can achieve 50% bit rate reduction compared with the previous standard H.264. However, in order to achieve such high compression ratio, many new coding features are adopted thus the complexity of HEVC also becomes much higher. Compared with H.264, 5.2x/2.1x higher complexities are consumed for HEVC encoding/decoding as reported in [3].

A typical encoding diagram is shown in Figure 1-1. The input of the encoding is original video data and the output is the compressed bit stream. There are two kinds of prediction. Intra prediction aims to reduce the spatial redundancy within one frame, while inter prediction aims to reduce the temporal redundancy between different

frames. There are many supported encoding modes for inter and intra prediction. So the mode decision process is required to select the best encoding mode. After the mode decision, the best mode information is transferred to context-adaptive binary arithmetic coding (CABAC) for the entropy coding. The difference between the original and predicted pixels is calculated and called residual. For the residual pixels, forward transform (T) is processed to transfer the residual from spatial domain to frequency domain. After that, quantization (Q) can further reduce the volume of the transformed data. The quantized transformed residual is transferred to CABAC for the entropy coding. In order to regenerate the residual pixel, de-quantization (DQ) and inverse transform (IT) are conducted. The summation of regenerated residual and the predicted pixel is reconstructed pixel. Because of the loss of quantization, the reconstructed pixels are not exactly equal to the original pixels, which will lead to performance loss. Finally, the reconstructed pixels with loop filtering are used for the next mode decision.

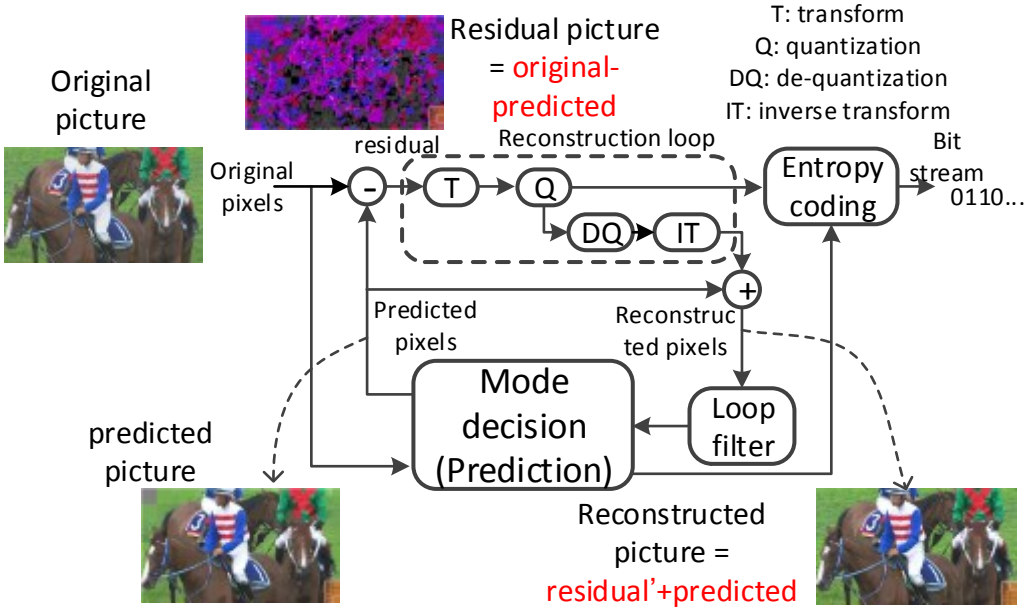

Figure 1-1 Video encoding diagram.

1.2 Mode Decision and Reconstruction Loop

The encoding diagram has been shown in Figure 1-1. We can see that mode decision and reconstruction loop are two indispensable components in the video coding. The process of mode decision and reconstruction loop and their relationship are given in this chapter.

1.2.1 Process of Mode Decision and Reconstruction Loop

The process of mode decision and reconstruction loop are introduced in this chapter. At first, three kinds of units in HEVC are introduced. The first unit is the coding unit (CU) which is corresponding to the macroblock in H.264. CU is the basic unit for coding. The CU size can be from 64x64 to 8x8. The second kind of unit is the prediction unit (PU) which is used for prediction. PU is partitioned from CU and there are eight kinds of PU partitions as shown in Figure 1-2. The third kind of unit is the transform unit (TU). The TU size can be from 32x32 to 4x4. The root of TU is the CU and a CU can be recursively split into TUs.

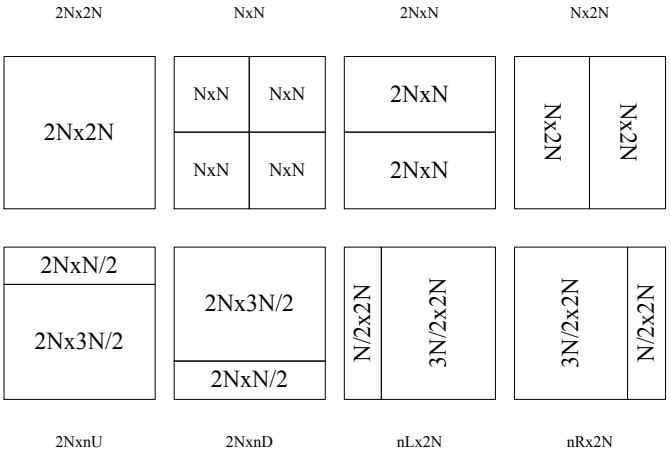

Figure 1-2 Eight PU partitions in HEVC [4].

For intra prediction, PU partition can be 2Nx2N and NxN. NxN is only adopted

for the smallest CU. Therefore, the PU can be from 64x64 to 4x4 in default as shown in Figure 1-3. For each PU, there are 35 prediction modes as shown in Figure 1-4. Mode 2 to 34 are 33 directional modes. Mode 0 (Intra_Planar) and mode 1 (Intra_DC) are two non-directional modes. Mode 35 (Intra_FromLuma) is the chroma mode which is to derive from the best luma mode.

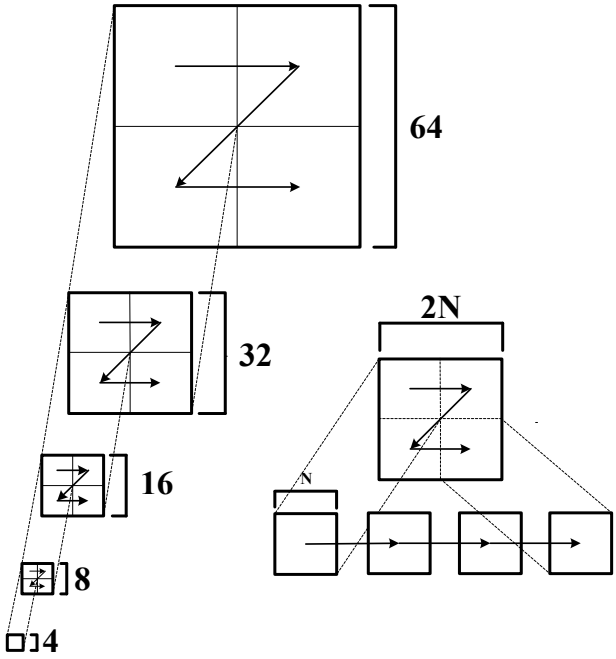

Figure 1-3 5 intra prediction units for HEVC.

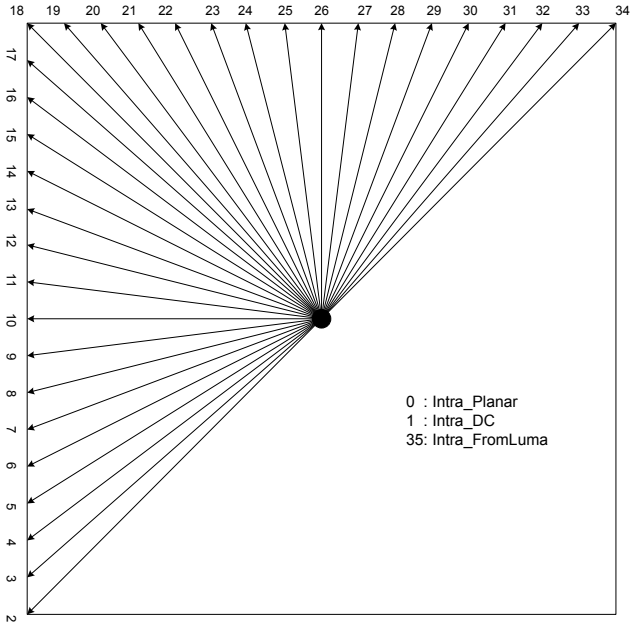

Figure 1-4 35 intra prediction modes for HEVC.

For inter prediction, if the CU is not the smallest, NxN is not supported. Otherwise, all the eight PU partitions can be supported. For each PU partition, the best motion vector is decided through the motion estimation stage. In addition, HEVC also supports the merge mode which derives the motion vector information from spatially or temporally neighboring blocks so that the motion estimation is not required for merge mode.

In order to explain clearly how to do the prediction, an example of intra prediction is shown in Figure 1-5. The prediction unit is 4x4 in this example. When using the vertical directional mode (26 in Figure 1-4), the four reconstructed pixels above the block are used to do the prediction. So the values of predicted pixels and the corresponding residual pixels are shown in the figure. When using the horizontal directional mode (10 in Figure 1-4), the four reconstructed pixels in the left side of the block are used for the prediction. In this case, the residual pixels are different from the results of the vertical directional mode.

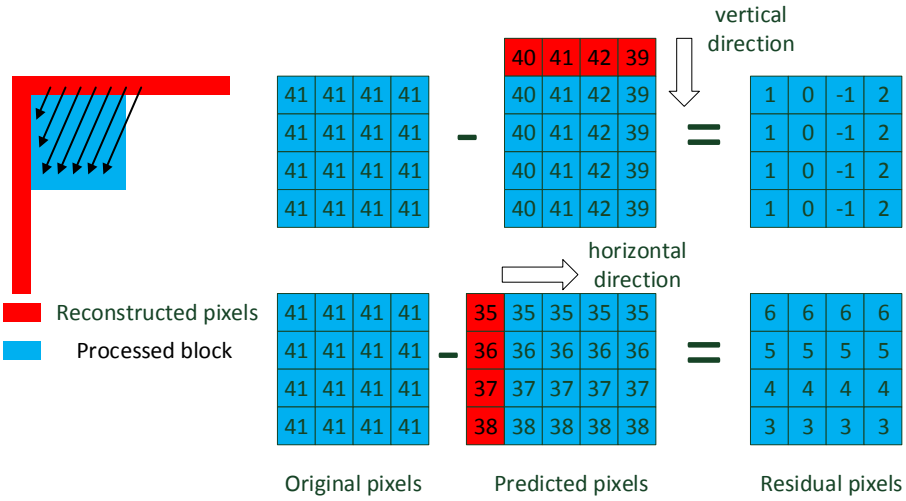

Figure 1-5 An example of intra prediction.

For each CU, the residual pixels can be generated by the prediction. After that, the residual should be coded based on different TU combination. All the TU combination will be traversed to find the best one. Therefore, for each CU 2Nx2N, we go through all supported combinations of PU, prediction mode and TU to select the best

mode. After selecting the best mode for each CU 2Nx2N, the cost of CU 2Nx2N and the summation of the costs of four CU NxN are compared to decide whether CU 2Nx2N is split or not. By doing so recursively from bottom to up, the best mode for the largest CU 64x64 can be decided.

Figure 1-6 gives an example for the mode decision result for the luma part of a 64x64 block. The CU splitting is shown by the black line. The 64x64 block is split to two 32x32 blocks, four 16x16 blocks and sixteen 8x8 blocks. In this example, for the CU8x8, the PU partition NxN is not selected. For the right top and right bottom 32x32 PUs, the prediction modes are mode 1. The prediction modes for other PUs are also shown as the number inside the block. The TU splitting is shown in red line. Two CU 16x16 is split to four 8x8 TUs. For the other CU, the best TU is the same as its attributive CU.

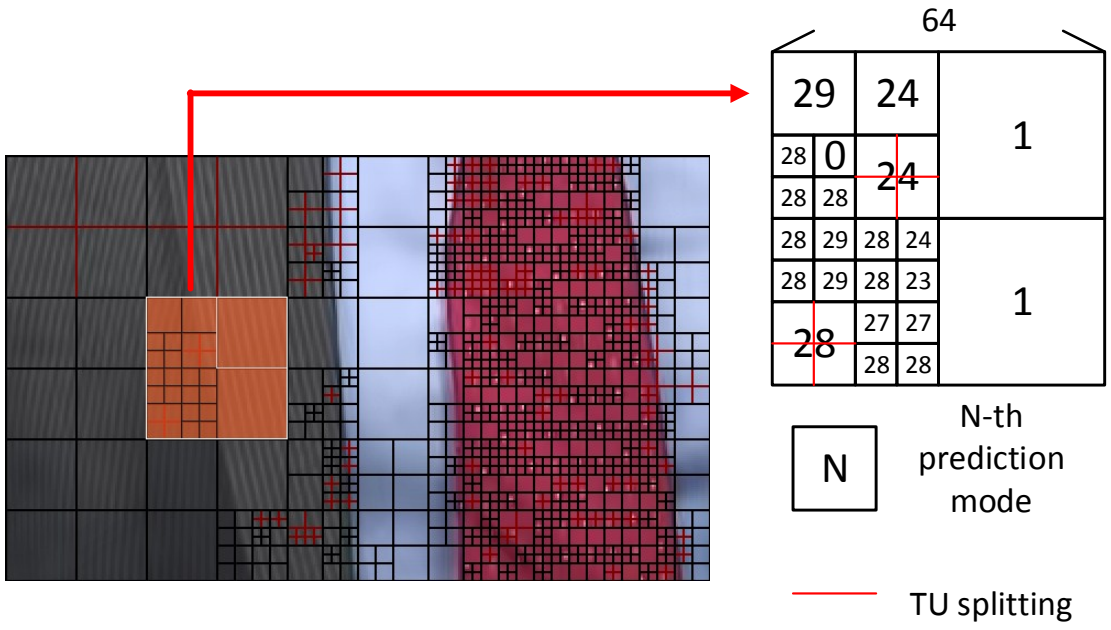

Figure 1-6 An example for the mode decision result of a 64x64 block.

However, as shown in Figure 1-6, for different 64x64 blocks, the best modes are different. The mode decision process is shown in Figure 1-7. There are two steps in the mode decision. The first step is the prediction and the second step is the

rate-distortion-optimization (RDO) process including the fast RDO and full RDO process. For the first step, the prediction is conducted for all the supported PU depths and modes. After that, fast RDO is also conducted for all the supported PU depths and modes in order to select some candidate modes for full RDO since the complexity of full RDO is very high.

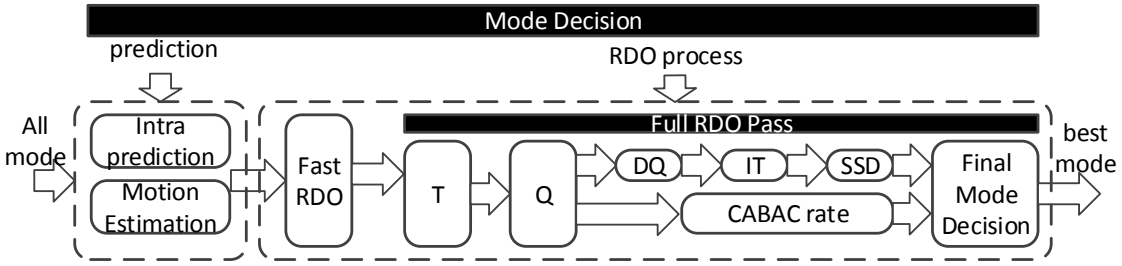

Figure 1-7 Mode decision process [5].

For fast RDO, some low-complexity costs such as sum of absolute difference (SAD) cost and sum of absolute transformed difference (SATD) cost are used. For full RDO, in order to ensure the coding performance, an optimal cost named after rate-distortion (R-D) cost is used for the referee. The calculation function for R-D cost is shown in the following equation.

$$J_{RDO} = SSD+\lambda*R \qquad (1\text{-}1)$$

where SSD is the distortion given by the sum of the squared differences between the original and reconstructed blocks. Transform, quantization, de-quantization and inverse transform are conducted to calculate the reconstructed pixels. R is the estimated encoded bits including the information for header and residual. In order to reduce the complexity, a look-up-table based CABAC rather than real CABAC is used to calculate R. λ is a coefficient related to QP, which indicates the ratio in significance between the distortion and rate.

By comparing the R-D cost, the mode with the smallest cost is selected as the best one. After that, the reconstruction loop process is used to generate the reconstructed pixels for the best mode. The process is shown in Figure 1-8. Transform, quantization, de-quantization and inverse transform are processed.

The input of reconstruction loop is the residual of the best mode. The output of reconstruction loop is added with the predicted pixels to generate the reconstructed pixels. The equation is shown in the following equations where P_{resi} is the input of reconstruction loop and $P_{resi}{}'$ is the output of reconstruction loop. We can see that if P_{resi} is equal to $P_{resi}{}'$, the reconstructed pixels are equal to the original pixels.

$$P_{resi} = P_{orig} - P_{pred} \quad (1\text{-}2)$$

$$P_{reco} = P'_{resi} + P_{pred} \quad (1\text{-}3)$$

However, since the input of quantization and the output of de-quantization are different, the reconstructed pixels are different from the original pixels and the degree of difference becomes larger with larger quantization step.

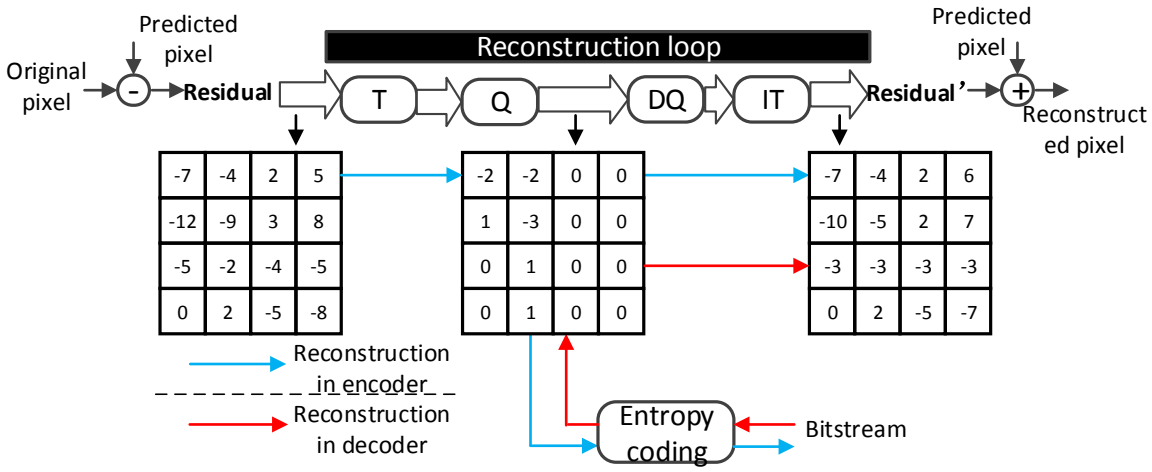

Figure 1-8 Reconstruction loop process.

1.2.2 Relationship between Mode Decision and Reconstruction Loop

In this chapter, the relationship between mode decision and reconstruction loop will be described. At first, the functional relationship is shown in Figure 1-9. The best mode is selected in mode decision and then the reconstruction loop is conducted to

generate the reconstructed pixels for the best mode. After that, the reconstructed pixels are used for the mode decision of the next block. An example is shown in Figure 1-10, there are four neighboring PUs and the block A is processed at first. After finding the best mode for A, the reconstructed pixels of block A can be calculated. The four reconstructed pixels of the rightest column of block A are required for the mode decision of block B. Similarly, the four reconstruction pixels of the bottom row of block B are required for the mode decision of block C. The four reconstructed pixels of the rightest column of block C are required for the mode decision of block D.

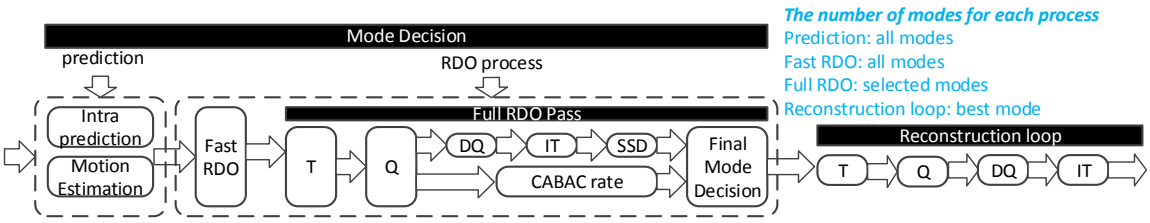

Figure 1-9 Mode decision and reconstruction loop.

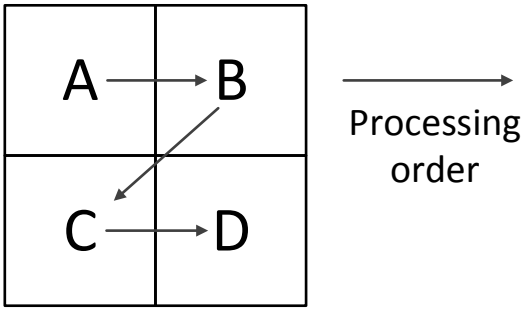

Figure 1-10 Four neighboring PUs.

Secondly, mode decision and reconstruction loop can share the hardware consumption in the real implementation. Because the full RDO process includes the reconstruction loop, these two parts can share the resource of T, Q, DQ and IT in the real implementation. For example, in the latest HEVC encoders [3][6], the reconstruction loop and RDO process share the resource of T and Q.

1.3 Motivations of Choosing Three Research Topics

The processes and relationship of mode decision and reconstruction loop have been introduced in the last chapter. In this chapter, the motivations of choosing the research topics are presented.

In HEVC, the complexity of mode decision and reconstruction loop has become much higher than H.264 because many new features are adopted. The main differences between H.264 and HEVC are shown in Table 1-1. For the basic unit, the largest in H.264 is only 16x16 while the largest can be up to 64x64 in HEVC. For the prediction methods, 26 more intra directions (prediction modes) are supported in HEVC. For the inter prediction, the merge mode and advanced motion vector prediction (AMVP) are new features in HEVC. In addition, four asymmetric prediction unit partitions are supported in HEVC. For the transform, the largest size increases from 8x8 in H.264 to 32x32 in HEVC. Sample adaptive offset is also a new in-loop filters in HEVC. For the entropy coding, only CABAC is supported in HEVC while context-adaptive variable-length coding (CAVLC) is also provided in H.264.

Table 1-1 Differences between H.264 and HEVC

Standard	H.264	**HEVC**
Basic unit	Macro block (MB) 16x16 & 4x4	**Coding Unit (CU)** **From 8x8 to 64x64**
Prediction methods	Intra prediction: 9 directions Inter prediction: Motion vector prediction Prediction unit partition: Four kinds	**Intra prediction: 35 directions** **Inter prediction: Advanced motion vector prediction & merge mode** **Prediction unit partition: Eight kinds**
Transform	Integer DCT (8x8)	**Integer DCT (32x32)**
In-loop filters	Deblocking filter	**Deblocking filter & Sample adaptive offset**
Entropy coding	CABAC and CAVLC	**Improved CABAC**

In Table 1-1, all the differences for basic unit, prediction methods and transform

are related with the mode decision process. The difference in transform size is related with reconstruction loop process. Because of the difference, the designs for mode decision and reconstruction loop in H.264 cannot be adopted in HEVC directly. Therefore, the low-complexity designs for HEVC mode decision and reconstruction loop are highly desired. In this thesis, three research topics for mode decision and reconstruction loop are selected. The reasons for choosing the three research topics are presented in the following.

Firstly, the reason for doing research on transform in Chapter 2 is given. [6] is the state-of-the-art HEVC intra-encoder. The area consumption distribution is shown in Table 1-2. Among all the components, forward transform and inverse transform take the majority of the hardware consumption. Forward transform consumes 326.2K and 42.7K gate counts, and inverse transform consumes 174.9K and 36.2K gate counts. Overall 580K gate counts are consumed on transform, which is about 53% of the overall gate counts.

Table 1-2 Hardware consumption of H.265/HEVC intra encoder [6].

Module	Main Loop (K gate)	4x4 Loop (K gate)
Intra predictor	105.2	22.1
Compensator & Orig. Pix. Buffer	7.1	2.0
Residual Buffer & Rank Lists	11.4	-
Forward Transform	**326.2**	**42.7**
Quantization	69.4	29.8
Dequantization	40.7	18.9
Inverse Transform	**174.9**	**36.2**
Reconstruction & Pred. Buffer	3.2	2.6
Rate Estimator	39.8	13.3
Dist. Estimator	11.1	4.5
Mode Decision & Coeff.Buffer	17.3	2.5
Total Loop	802.3	172.4
Main Controller	13.5	
Entropy Coder	94.3	
Total	1086.5	

We can see that the computation of transform is very high in HEVC since the

largest transform size is 32x32 which is much larger than 8x8 in H.264. For a complete 32x32 transform, 65536 multiplications and 63488 additions are required without any optimization. In H.264, the largest transform size is 8x8 thus only 1024 multiplications and 896 additions are required. Many more operations are required for HEVC transform compared with H.264. Therefore, my first research topic is to design an area-efficient transform architecture.

Secondly, the reason for designing the system for the de-quantization and inverse transform in Chapter 3 is presented. After designing the architecture for transform, the system design around transform is also required. As shown in Figure 1-9, the output of quantization is the input of de-quantization. Because of large transform size, there are many small high-frequency transformed coefficients which will be quantized to zero. Therefore, there are many zero inputs for the de-quantization. The processing for the zero elements can be optimized for area and power consumption. So there is much sufficient optimization space for the system of de-quantization and inverse transform. Moreover, the system design of de-quantization and inverse transform is important because it is required to generate the reconstructed pixels for both encoder and decoder. Therefore, the second research topic is to design the system of de-quantization and inverse transform.

Finally, the reason for reducing the number of PUs and prediction modes for intra prediction in Chapter 4 is given. In H.264, SATD cost is usually used for mode decision because it spends lower cost than R-D cost and the performance loss by using SATD cost is acceptable. However, in HEVC, replacing R-D cost by some fast cost models will lead to 10-15% rate increase for intra-frame encoding and more than 40% rate increase for inter-frame encoding as reported in [5]. As a result, we have to employ R-D cost for the mode decision in HEVC encoders such as [3], [5], [6] and [7]. The hardware consumption of [6] for each component is shown in Figure 1-11. In [6], full RDO includes forward transform, quantization, rate estimator and distortion estimator. Fast RDO is corresponding to the item of “Residual Buffer & Rank List”. For

the reconstruction loop, since transform and quantization are already categorized in full RDO, so only the de-quantization and inverse transform are categorized in the reconstruction loop. The entropy coder is corresponding to CABAC.

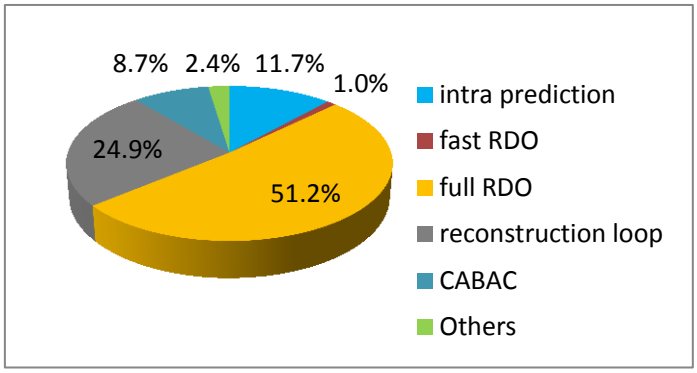

Figure 1-11 Hardware consumption distribution for each component [6].

We can see that full RDO consumes the majority. The high complexity of full RDO process comes from two reasons. The first reason is that transform is required which leads to large area consumption which has been addressed in the above. It is noted that transform consumes more hardware consumption than rate and distortion estimation. In fact, many efficient methods for rate and distortion estimation such as [9]-[11] have been developed. The second reason is that there are many modes requiring full RDO process in HEVC. For the intra prediction, there are five PU depths in HEVC as given in Figure 1-3. The largest PU is 64x64 and the smallest is 4x4. For each PU, 35 prediction modes are supported as shown in Figure 1-4. As a result, 175 modes require the full RDO process, so the number of modes has to be reduced. Reducing the number of modes requiring full RDO process is corresponding to fast RDO in Figure 1-7.

It is noted that the reason for doing fast RDO for intra rather than inter is that the full RDO process consumes the majority of area consumption for intra prediction. However, in the inter-frame encoding, the motion estimation consumes a comparable hardware resource compared with full RDO process as shown in Table 1-3. In [8], the gate counts for the motion estimation are 778.7K. In [6], the gate counts for the full

RDO process are 556.6K. Therefore, motion estimation is more important than full RDO process in inter-frame encoding. In fact, for the motion estimation, HEVC is based on the results of SATD and SAD cost which is also used in H.264. Therefore, many low-complexity H.264 designs can be friendly implemented for HEVC.

Table 1-3 Hardware consumption comparison of HEVC motion estimation with full RDO

Module	Gate counts (K)	On-chip memory (KB)
Motion estimation [8]	778.7	17.4 KB
Full RDO process [6]	556.6	-

Therefore, the research of fast RDO for intra prediction is developed. Different from the Chapter 2 and 3, fast algorithms are required before the VLSI implementation in Chapter 4. The algorithms can be classified to software-oriented algorithms and hardware-oriented algorithms. Since the final target is the hardware implementation, so the research target is the hardware-oriented algorithm.

1.4 Dissertation Organization

The rest of the thesis is organized as follows. The relationship between the chapters is illustrated in Figure 1-12.

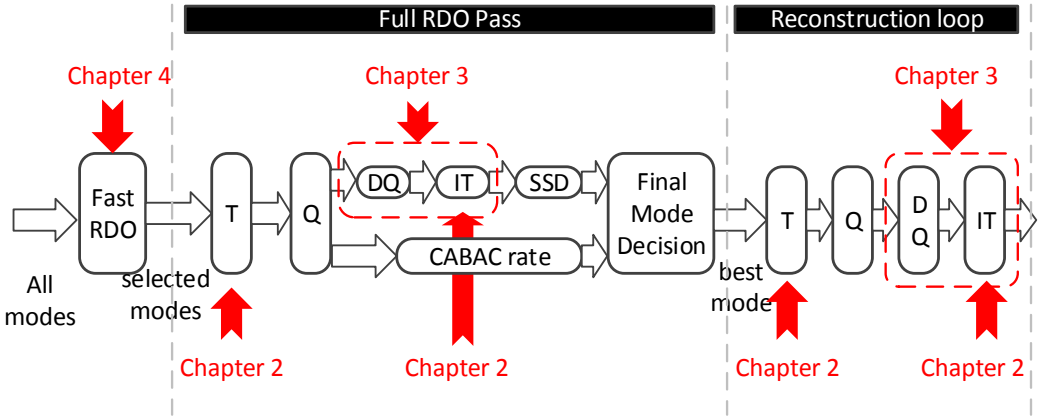

Figure 1-12 The relationship between the three chapters of proposals.

Chapter 2 gives an area-efficient transform architecture. By using Chen's algorithm, the transform can be decomposed to even and odd part. The results of even and odd parts can be shared by a butterfly structure. For the outputs in each clock cycle, the requiring outputs are reordered so that the inputs of the butterfly structure can be reused. The requiring outputs are reordered for 8/16/32-point IDCT. After presenting the method for the individual 8/16/32-point IDCT, a unified 8/16/32-point IDCT architecture is proposed. 2N-point IDCT reuse the results of N-point IDCT. As a result, about 25% area consumption can be reduced for the logical computational part compared with previous works. For the transpose buffer part, SRAM is used to store the results of row transform. The storing address in SRAM is reordered so that the I/O utilization of SRAM can achieve 100% for both writing and reading operation. Two data mapping methods are developed so that writing and reading operation can be executed in parallel. Finally, an overall pipeline schedule is proposed to avoid the writing and reading address conflict problem. As a result, 62% area consumption can be saved compared with other SRAM-based memory mapping methods. The designs in Chapter 2 can be used in the mode decision and the reconstruction loop.

Chapter 3 gives a low-cost architecture for the system of de-quantization and inverse transform. The input of the de-quantization is decomposed to base part (baseLevel) and remaining part. For the baseLevel, the value is not greater than 3. Therefore, the multiplication with the scaling parameter could be avoided. For the remaining part, the number of positions with non-zero values is not greater than 4 in many cases. Therefore, the number of multipliers in the de-quantization is reduced from 16 to 4 and four multipliers are reused in the different clock cycles. To ensure that the proposed design can meet the throughput of 8K@120fps, a complete analysis for the throughput of the frame-level and sequence-level is given. In the overall system, memories are used to transfer the pixel data. In addition, the pixel data is reused to detect the zero elements. After the detection, for each zero 4x1-row, the read operation of the memory between de-quantization and inverse transform is disabled. For

each zero row, the write operation of the transpose memory of the inverse transform is disabled. For the zero elements in each column, the read operation of the transpose memory of the inverse transform is disabled. As a result, overall, 68% normalized area consumption can be saved for the logical part, and 56% normalized power consumption can be saved for the overall system compared with the previous work. For the de-quantization, the proposed architecture can save the area consumption by 77% compared with previous works. For the zero skipping method of the memory part, 29%-86% power consumption can be saved compared with not skipping memory operations for zero elements. Chapter 3 can be used for both mode decision and reconstruction loop.

In Chapter 4, a fast RDO for the intra prediction is given. Different from most of the previous works, the proposal is based on the proposed low-complexity cost model rather than the edge and gradient information. There are two schemes in the proposed cost model. One is that original pixels rather than reconstructed pixels are utilized as the neighboring pixels for doing the prediction. The other is that only the estimated costs for 8x8 are calculated and the costs for larger PUs are estimated by reusing the results of 8x8 PUs. Based on the proposed cost model, the costs of neighboring PUs are compared and two PUs are selected. In order to improve the coding efficiency, the 32x32 PU compensation method is proposed thus some additional 32x32 PUs require the R-D cost calculation. Still reusing the results of cost models, the Hadamard cost calculation in the original HM can be completely removed. As a result, 52% encoding time reduction can be achieved with about 1.87% BD-bitrate compared with original HM. Compared with previous works, better coding efficiency or more encoding time reduction can be achieved. This chapter works for the mode decision of intra prediction.

In Chapter 5, the overall conclusion and future work are presented.

2. An Area-Efficient Transform Architecture Design

In this chapter[1], an area-efficient transform architecture will be presented. The position of this chapter in the mode decision and reconstruction loop is shown in Figure 2-1. In the mode decision, the forward transform and inverse transform are required in the full RDO process. In addition, the transforms are also required in the reconstruction loop. There are two major concepts of the idea in this chapter. For the logical computational part, not only the architecture of N-point transform is reused for 2N-point transform, but also the inputs of the butterfly structure are reused. For the transpose buffer in transform, the writing positions in SRAM are reordered in order to achieve 100% I/O utilization of SRAM. This chapter is related with the publication [2] and [8] in Page 122.

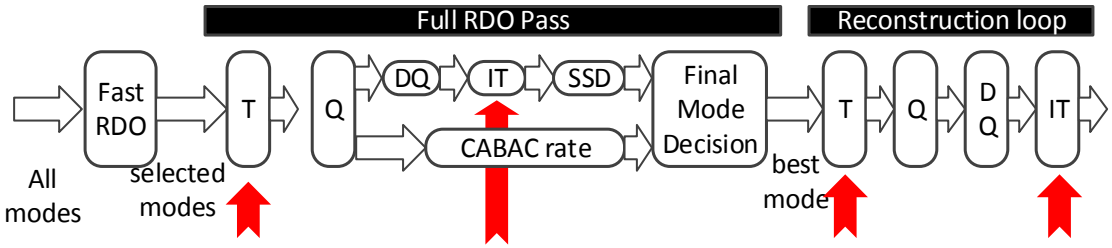

Figure 2-1 The position of chapter 2 in the mode decision and reconstruction loop.

[1] This chapter is related with the publication [2] and [8] in Page 122.

2.1 Introduction

2.1.1 Overview of HEVC Transform

Since the development of H.264, the discrete cosine transform (DCT) has been widely employed to perform the transform operation since it can provide energy compaction. To reduce the computational complexity and solve the mismatch problem between forward and inverse transforms, integer DCT has been adopted instead of floating DCT. Forward transform is used to transfer the data from spatial domain to frequency domain, and inverse transform is adopted to transfer the data from frequency domain back to spatial domain.

A complete transform in the video coding is two-dimensional as shown in Figure 2-2. Two 1-D transforms are row transform and column transform. Row transform is processed row by row and the results are stored in the transpose buffer. After that, the results in the transpose buffer will be fetched column by column to do the column transform.

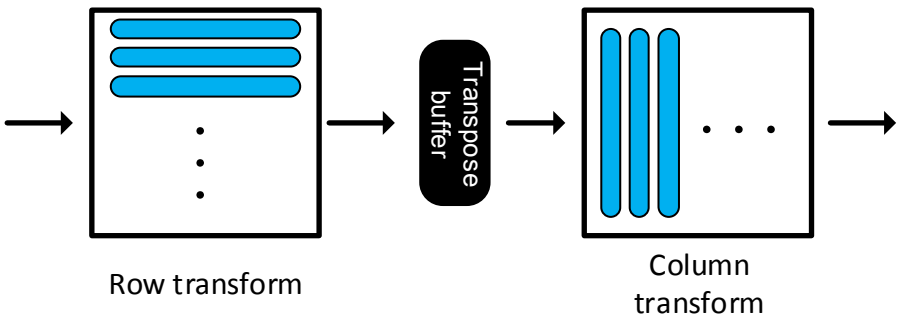

Figure 2-2 A two-dimensional transform.

The largest DCT size provided in H.264 is 8x8, while larger DCTs including 16x16 and 32x32 are provided in HEVC since a larger transform size can contribute to higher compression ratios. Because of larger transform size, there are two problems. One is the large hardware cost for logical computational part. In order to get one re-

sult of 32x32 transform, it requires 32 multiplications and 31 additions. The second problem is the large hardware cost for transpose buffer part. In order to store the results of row transform for 32x32, it requires more than 100K gate counts.

In order to reduce the number of operations for the logical computational part, HM use Chen's algorithm [17]. The inverse transform is taken as an example and the analysis for Chen's algorithm is given in the below.

A complete 2D IDCT can be decomposed to two 1D IDCTs, and the decomposition method is shown in the following equation.

$$[X] = [T_{2N}^T] * [Y] * [T_{2N}] = (([Y] * [T_{2N}])^T * [T_{2N}])^T \quad (2\text{-}1)$$

where T_{2N} is a 2N × 2N transform matrix defined by HEVC, Y is the 2N × 2N matrix before the IDCT, and X is the result after the 2D IDCT operation. The first 1D IDCT (IT1) is called a row IDCT, and the second one (IT2) is called a column IDCT. For a 2N-point input $[Y_0, Y_1, \ldots Y_{2N-1}]$, which is defined as Y_{2N}, a 1D IDCT operation $[Y_{2N}]*[T_{2N}]$ requires $(2N)^2$ multiplications and 2N*(2N-1) additions.

In fact, the transform matrix T_{2N} has the property of symmetry so that it can be decomposed. For example, T_8 is given by Eq. (2-2). We can see that the matrix has both symmetric and anti-symmetric properties, as shown in Eq. (2-3).

$$[T_8] = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 \\ 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\ 75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 \\ 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\ 50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 \\ 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\ 18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 \end{bmatrix} \quad (2\text{-}2)$$

$$\begin{aligned} T_8(i,j) &= T_8(i,7-j) \quad (i = 0,2,4,6) \\ T_8(i,j) &= -T_8(i,7-j) \quad (i = 1,3,5,7) \end{aligned} \quad (2\text{-}3)$$

The matrices for each transform size can be found in [2]. Using the symmetry, a fast algorithm was developed to decompose the transform matrix [17]. Using Chen's algorithm, the decomposition method for the HEVC transform matrix is shown in the following equation.

$$[T_{2N}] = [P_{2N}] * \begin{bmatrix} T_N & 0 \\ 0 & O_N \end{bmatrix} * [B_{2N}] \quad (2\text{-}4)$$

where T_{2N} is a 2N × 2N transform matrix, T_N is an N × N transform matrix, O_N is an odd part matrix, P_{2N} is a permutation matrix, and B_{2N} is a 2N-point butterfly structure. Detailed definitions for P_{2N}, B_{2N}, T_N, and O_N are given as follows.

$$
\begin{aligned}
P_{2N}(i,j) &= \begin{cases} 1, & i = 2*j \ \ or \ \ i = (j-N)*2+1 \\ 0, & otherwise \end{cases} \\
B_{2N}(i,j) &= \begin{cases} 1, & (i = j \ and \ i < N) \ \ or \ \ (i+j = 2N-1) \\ -1, & i = j \ and \ i \geq N \\ 0, & otherwise \end{cases} \\
T_N(i,j) &= T_{2N}(2*i, j) \\
O_N(i,j) &= T_{2N}(2*i+1, N\text{-}1\text{-}j)
\end{aligned}
\tag{2-5}
$$

By using the decomposition, $[Y_{2N}]*[T_{2N}]$ can be expressed as the following equation.

$$
\begin{aligned}
[Y_{2N}]*[T_{2N}] &= [Y_{2N}]*[P_{2N}]*\begin{bmatrix} T_N & 0 \\ 0 & O_N \end{bmatrix}*[B_{2N}] \\
&= \begin{bmatrix} [Y_N^{Even}]*[T_N] & 0 \\ 0 & [Y_N^{Odd}]*[O_N] \end{bmatrix}*[B_{2N}]
\end{aligned}
\tag{2-6}
$$

$$
\begin{aligned}
[Y_N^{Even}] &= [Y_0, Y_2, \ldots Y_{2N-2}] \\
[Y_N^{Odd}] &= [Y_1, Y_3, \ldots Y_{2N-1}]
\end{aligned}
\tag{2-7}
$$

where $[Y_N{}^{Even}]*[T_N]$ and $[Y_N{}^{Odd}]*[O_N]$ are defined as the results of the even and odd parts, respectively. The implementation of $[Y_N{}^{Even}]*[T_N]$ and $[Y_N{}^{Odd}]*[O_N]$ cost N^2 multiplications and N*(N-1) additions, respectively, while B_{2N} requires 2N additions. Therefore, a total of $2*N^2$ multiplications and $2*N^2$ additions are required. Compared with the original method before decomposition, the required number of computations is significantly reduced. In fact, T_N can be decomposed into the $T_{N/2}$ and $O_{N/2}$ in the same way to further reduce the number of computations for the even part.

Figure 2-3 illustrates the corresponding overall architecture for a 1D 8-point inverse discrete cosine transform (IDCT) based on Chen's algorithm. Y_n represents the 8-point inputs and X_n represents the 8-point outputs of IDCT. As shown in Eq. (2-6), the transform can be divided into even and odd parts. To generate 8-point results, four results are required from the even and odd parts, respectively. The results from the even part are E_n (n = 0, 1, 2, 3) and the four results from the odd part are O_n (n = 0, 1, 2, 3). Based on E_n and O_n, X_n can be obtained after an 8-point butterfly structure. The relationship is shown in the following equation.

$$X_n = E_n + O_n, X_{7-n} = E_n - O_n \quad (2\text{-}8)$$

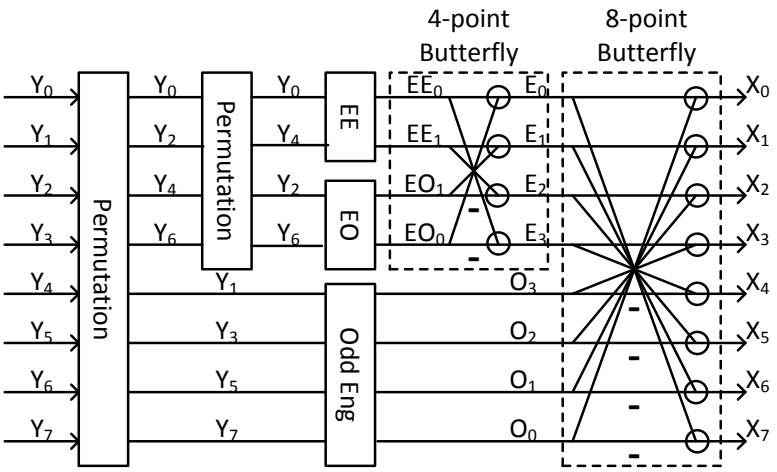

Figure 2-3 Original overall architecture for 8-point 1D IDCT. Y represents 8-point samples before IDCT, X represents 8-point samples after IDCT. The symbol "o" means addition and "-" means subtraction.

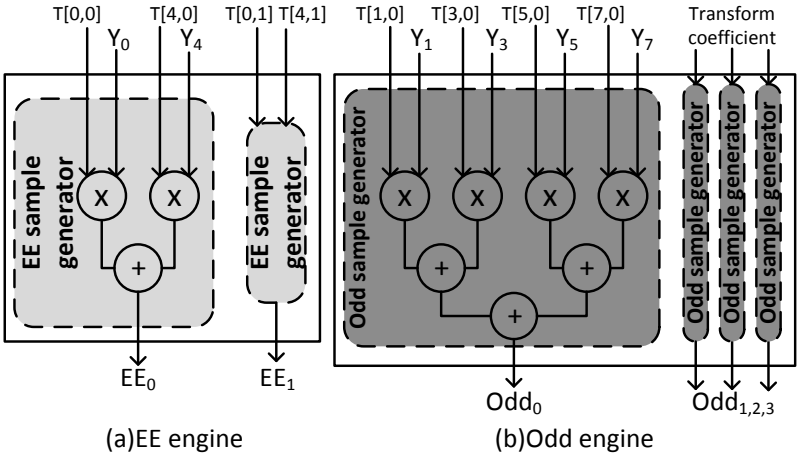

Figure 2-4 Original architecture for EE and odd engines in the 8-point IDCT (a) The EE engine is composed of two EE sample generators (b) The odd engine is composed of four odd sample generators.

The even part may be further divided into two parts, namely the EE (Even-Even) part and the EO (Even-Odd) part. Each part is required to generate two results. A detailed architecture for the EE engine is shown in Figure 2-4(a). The architecture for the EO engine is similar to that of the EE engine. Based on EE_n and EO_n, E_n can be calculated as follows.

$$E_n = EE_n + EO_n, X_{3-n} = EE_n - EO_n \quad (2\text{-}9)$$

For the odd part, four results are generated, and details of the architecture for the odd engine are shown in Figure 2-4(b).

2.1.2 Previous Works

The researches on the area-efficient architectures for transform have been conducted from H.264. In [12], the authors proposed an architecture for a multi-standard inverse transform; two circuit share strategies, a factor share, and adder share strategies were applied to reduce the required circuit resource. A low-cost hardware sharing architecture was proposed in [13] by adding the offset computations and pipelined design. In [14], Wang et al. designed a reconfigurable architecture that provided the fusing strategies that generate constant multipliers in the matrix calculation blocks. In addition, an adder-sharing strategy was adopted to save circuit area. A high-performance inverse transform circuit was also developed in [15] based on the butterfly architecture, and this work shares the resources efficiently by exploiting the similarities between transforms. In [16], extensive mathematical analysis and decomposition were performed for the H.264 transform and quantization so that all multiplication and division operations were avoided effectively. The above low-cost architectures were all designed for the 4/8-point IDCT in H.264, and they cannot be directly used in HEVC due to the differences in the transform size. The large transform size in HEVC will cause huge area consumptions from two perspectives. One is the logical computational part of the transform; a larger transform leads to a greater number of multipliers and adders causing the required amount of hardware resources to increase. The other problem is the transpose memory required to store more intermediate results as the transpose architecture consumes more area. In order to solve the above two problems, several recently developed low-cost architectures have been reported for the HEVC transform.

Based on Chen's algorithm, many literatures have been presented. Shen et al. [18] utilize the Chen's algorithm to reuse the small transform in the larger transform architecture, the author used the multiple constant multiplication (MCM) for 4/8-point transform and share the regular multipliers for 16/32-point transform. In [19], the multiplications of 16/32-point transform are implemented with input-muxed constant multipliers. Still based on Chen's algorithm, a fully pipelined transform architecture was pro-posed for an HEVC codec in [20]. The authors in [21] proposed a 16/32-point architecture that had no multiplications in the design. The similarity property of transform matrices was utilized in [22], so a calculation unit can be shared for different transform mode. In [23], the authors presented a unified forward/inverse transform architecture for HEVC; the unified architecture makes use of symmetrical properties that exist in HEVC transform matrices to achieve hardware sharing. In [24], the authors turned the multiplications by constant into shift and sum operations to achieve a multiplierless HEVC transform architecture. In [25], the N-point 1D transform was performed using an (N/2)-point 1D transform unit and a constant matrix multiplication recursively. Meher et al. [26] developed two efficient DCT architectures for HEVC, one is based on Chen's algorithm and implemented without multipliers. Moreover, the other one uses pruning scheme to truncate some transformed bits in order to save the complexity. The pruned architecture cannot be used in the decoder since the transformed bits are truncated. Do et al. [27] minimized the critical computation path of transform by replacing all the multiplications with additions and shifts.

All the previous works utilize Chen's algorithm. However, when the pixel parallelism is smaller than the transform size, the symmetric property in transform cannot be utilized completely. For example, when the pixel parallelism is 4 and the transform size is 8x8, two cycles are required to generate the eight results X_n (n = 0, 1, 2,…, 7). $\{X_0, X_1, X_2, X_3\}$ are generated in the first cycle, and $\{X_4, X_5, X_6, X_7\}$ are generated in the second cycle. In this way, according to Eq. (2-8), all the E_n and O_n results are required in the two clock cycles. To generate all of the E_n and O_n results in each cycle,

the architecture for the odd and EE engines is the same as shown in Figure 2-4. The problem is that although E_n and O_n can be used to generate all the eight results of X_n, only four of them are required in each clock cycle. Therefore, the calculation of all the E_n and O_n in one clock cycle is not required.

The above literatures aim to reduce the computation of the logical computational part. For a complete transform, a transpose buffer is required to store the results of row transform. To reduce the area of the transpose memory, SRAM instead of register is utilized in some previous works. Shen et al. [18] used 4 single-port 8 × 512-bit SRAM. Zhu et al. [20] used 32 dual-port 32 × 16-bit SRAM. The total bits of SRAM are 16384 bits in both designs. The data mapping method in [18] is shown in Figure 2-5, the width of one word is 512-bit. In the first clock cycle, four results of a0, b0, c0 and d0 are generated and they are stored in SRAM 0, 1, 2 and 3, respectively. After that, all the a_n (n=0,1,…,31) in the first column are written in the SRAM 0. By doing so, the results of first column can be feteched in one clock cycle.

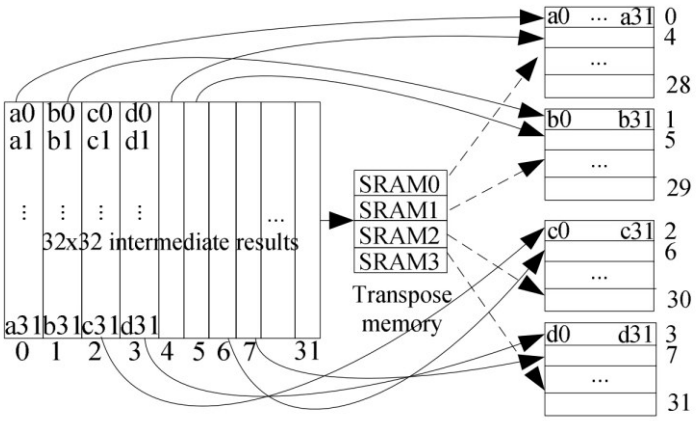

Figure 2-5 Data mapping method in [18]

The above works are focused on the low-cost implementation of accurate transform, there are also many works focused on the approximate transform. By using the approximate computing, the hardware cost can be saved with the performance loss. Lee et al. [28] designed a low-complexity integer DCT for the rate and distortion estimation. The Hadamard transform was used to substitute for the discrete cosine

transform to reduce the complexity in Zhu et al. [29]. Potluri et al. [30] used only 14 additions for the 8-point approximate DCT. Jridi et al. [31] presented a generalized recursive algorithm to get the orthogonal approximation of DCT where the larger DCT can be obtained from the smaller DCT. Cintra et al. [32] proposed a transformation matrix with only zero-element and one-element thus there are no multiplications and shifts in the computation. Bouguezel et al. [33] introduced some zeros in the 8x8 matrix which can reduce the number of arithmetic operations by about 25%. Kalali et al. [34] only calculated some low-frequency coefficients by the pre-determined methods. Jridi et al. [35] conducted the approximation based on a new 4-point DCT kernel which can increase the coding efficiency.

2.1.3 Research Target

In the previous chapter, low-cost architectures for transform are introduced. For the work on approximate computing, the coding efficiency loss is large. In [29], although the BD-bitrate is only 1.27%, the simplified R-D cost model (Hadamard cost) is only applied in the PU mode decision. When using SATD cost for all the mode decision process, the performance loss will become much larger than 1.27%.

In fact, on the encoder side, the accurate transform without approximation is adopted in some real implementations such as [3], [5] and [6]. It is because the performance loss caused by approximate transform is very large. On the decoder side, accurate transform is also required. Therefore, my research target is focused on reducing the hardware consumption of the original transform in HEVC.

For the logical computational part, when the pixel parallelism is smaller than the transform size, the outputs are reordered so that the inputs of butterfly structure can be reused. For the transpose buffer part, although some SRAM-based designs have been introduced in [18] and [20], the data width of SRAM is larger than the pixel parallelism, so the I/O utilization of each SRAM cannot reach 100%. For the design in [18],

only 16-bit is used for writing in each clock cycle. Therefore, the I/O utilization of each SRAM is only 3.125%. In [20], the pixel parallelism is different for various transform sizes. When processing 32-point transform, the pixel parallelism is 32 thus 32 16-bit results are written in the SRAM. In this case, the I/O utilization can achieve 100%. However, when processing 4-point transform, the pixel parallelism is 4 thus only four 16-bit results are written in the SRAM in one clock cycle. In this case, the I/O utilization is only 12.5%. Therefore, my research target is to reorder the writing positions of row transform results in SRAM so that it can achieve 100% I/O utilization of SRAM.

2.2 Proposed Architecture for Logical Computational Part

2.2.1 Proposed Reordered Parallel-in Serial-out (RPISO) Scheme

In Eq. (2-8), we can see that by using an 8-point butterfly structure, X_n and X_{7-n} are generated by the same E_n and O_n. If X_n and X_{7-n} can be generated within one cycle, they can share E_n and O_n. Similarly, using a 4-point butterfly structure, E_n and E_{3-n} are calculated by the same EE_n and EO_n. Likewise, if E_n and E_{3-n} can be generated within one cycle, they can share EE_n and EO_n.

In order to reduce the redundant inputs of butterfly and the required calculations for butterfly inputs in each cycle, the outputs are reordered. After the reordering, in the first cycle, $\{X_0, X_3, X_4, X_7\}$ are generated. In the second cycle, $\{X_1, X_2, X_5, X_6\}$ are generated. To calculate $X_{0,3,4,7}$ in the first cycle, $O_{0,3}$ and $E_{0,3}$ are required, while to obtain $E_{0,3}$, EE_0 is required. In the second cycle, to calculate $X_{1,2,5,6}$, $O_{1,2}$ and $E_{1,2}$ are

required, while to obtain $E_{1,2}$, EE_1 is required. Within a single cycle, only one result is generated from the EE part and two results are generated from Odd part. It should be noted that the EO engine is similar to the EE engine; within each cycle, only one result is generated from the EO engine.

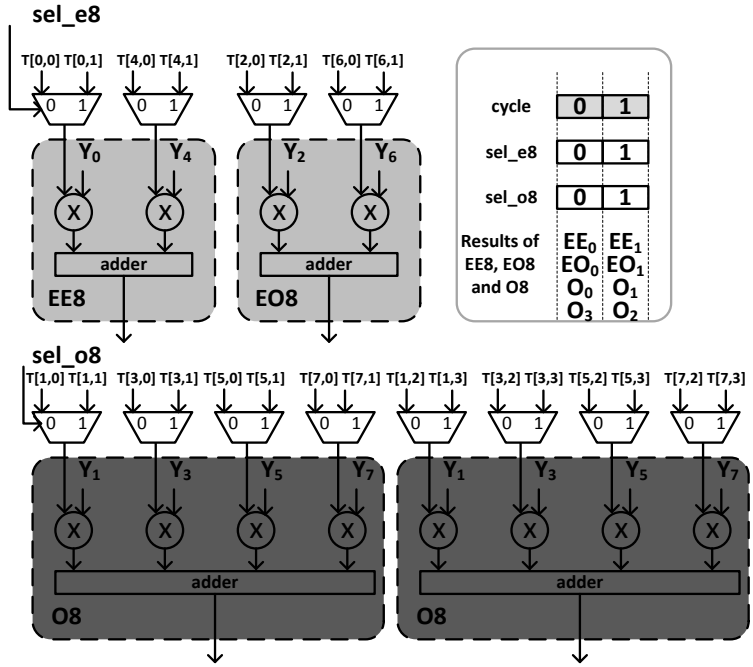

Figure 2-6 Proposed architecture for EE8, EO8 and O8 engines by using the RPISO scheme in the 8-point IDCT.

Using the RPISO scheme, the architecture for the EE engine is shown in Figure 2-6. In the first cycle, EE_0 is selected for the calculation, and the multiplexers select T[0,0] and T[4,0]. In the second cycle, EE_1 is required, thus the multiplexers select T[0,1] and T[4,1]. Compared with the original EE engine in Figure 2-4(a), the number of EE sample generators is reduced by one half with some extra multiplexers. The architecture for the odd engine is shown in Figure 2-6. The multiplexers are used to select the transform coefficients. In the first cycle, the transform coefficients for $O_{0,3}$ are selected, while in the second cycle, the transform coefficients for $O_{1,2}$ are selected. Compared with the original architecture shown in Figure 2-4(b), one half of the multipliers and adders can be reduced for the odd engine. The proposed architecture for an 8-point 1D IDCT is shown in Figure 2-7(A) where Y_n represents 8-point inputs, the

circuits for EE8, EO8 and O8 are shown in Figure 2-6. The values of sel_e8 and sel_o8 and the reordered outputs in each cycle are shown in Figure 2-7(B).

It has to be noted that the pixel parallelism is not necessary to be 4. When the parallelism is 8 or 16, the RPISO scheme can still reduce the redundant inputs of butterfly and the required calculations for butterfly inputs in each cycle.

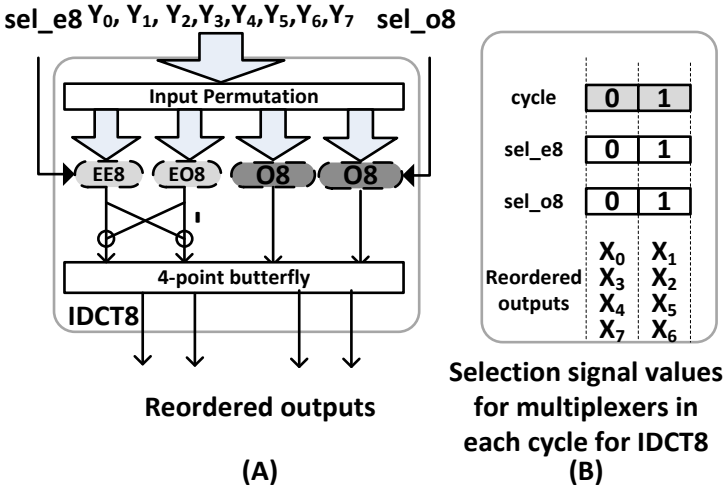

Figure 2-7 Proposed architecture for 8-point 1D IDCT.

2.2.2 Unified 8/16/32-point IDCT Architecture Based on RPISO

A 16-point 1D IDCT is similar with 8-point IDCT. The overall flowchart is shown in Figure 2-8. It is composed of permutation part, even part, odd part and butterfly structure. By using a 16-point butterfly structure, X_n and X_{15-n} are calculated by the same E_n and O_n as shown in the following equation.

$$X_n = E_n + O_n, X_{15-n} = E_n - O_n \quad (2\text{-}10)$$

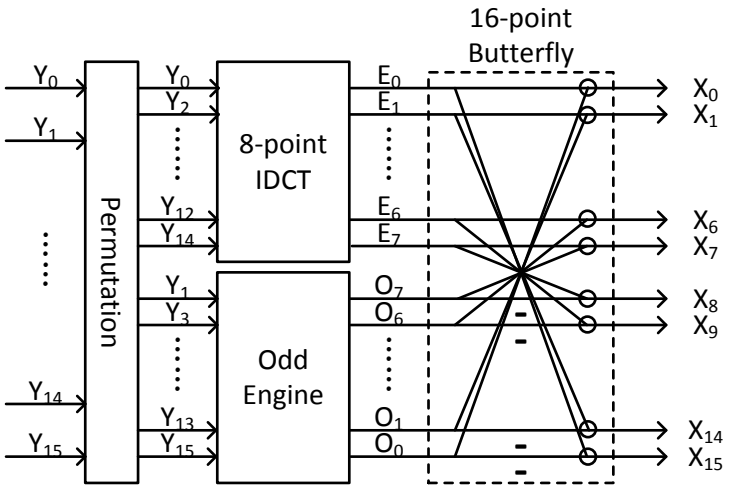

Figure 2-8 Overall flowchart for 16-point 1D IDCT based on Chen's algorithm.

By using the proposed RPISO scheme, X_n and X_{15-n} can share $\{E_n,O_n\}$ in the same cycle. To generate four outputs in one cycle, two even results and two odd results are calculated in one cycle. The reordered outputs in each cycle are stated in Figure 2-9(A). Four cycles are required to generate 16-point results.

For the even part of 16-point IDCT, E_n is equal to the result of 8-point IDCT according to [17]. So the architecture in Figure 2-7(A) can be reused as the even engine for 16-point IDCT. Within each cycle, four results are generated from 8-point IDCT. A multiplexer is required to select two as the results for even part. The selection signal is sel_e16. The selection signal sel_o8 and sel_e8 can select four results generated by 8-point IDCT. The values for the selection signals in each cycle are given in Figure 2-9(A). For example, in the first cycle, sel_o8 and sel_e8 are 0, $\{E_0,E_3,E_4,E_7\}$ are generated by 8-point IDCT, sel_e16 is 0 so that $\{E_0,E_7\}$ are selected as two even results for 16-point IDCT.

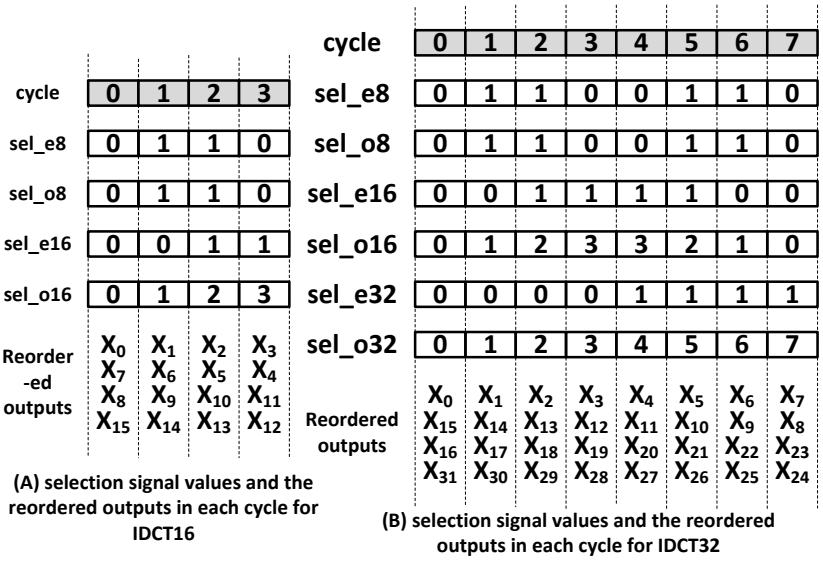

Figure 2-9 Selection signal values for multiplexers and the reordered outputs in each cycle for IDCT16 and IDCT32.

For the odd part of 16-point IDCT, O_n is generated by multiplication and addition. An odd engine for 16-point IDCT (O16) can generate one O_n result. As shown in Figure 2-10, O16 is composed of 8 multiple constant multiplications (MCM) and 7 adders. For each MCM in O16, one input is the odd-index input Y_n, and the other is constant coefficient. The constants are selected by a Look Up Table (LUT16) and the selection signal is sel_o16. For example, in the first cycle, sel_o16 is 0, the corresponding constants for O_0 and O_7 are selected so that $\{O_0,O_7\}$ are generated as the odd results for 16-point IDCT.

Similarly, the 32-point IDCT can reuse the 16-point IDCT as the even engine. A multiplexer with the selection signal sel_e32 is used to select two results from 16-point IDCT. sel_e16 and sel_o16 can decide the outputs from 16-point IDCT. For the odd part of 32-point IDCT, an O32 can generate one result. O32 is composed of 16 MCMs and 15 adders, as shown in Figure 2-10. sel_o32 is used to select the constant coefficients from LUT32. The values for all the selection signals in each cycle are shown in Figure 2-9(B). Eight cycles are required to generate 32-point results. The overall architecture for 8/16/32-point IDCT is shown in Figure 2-10.

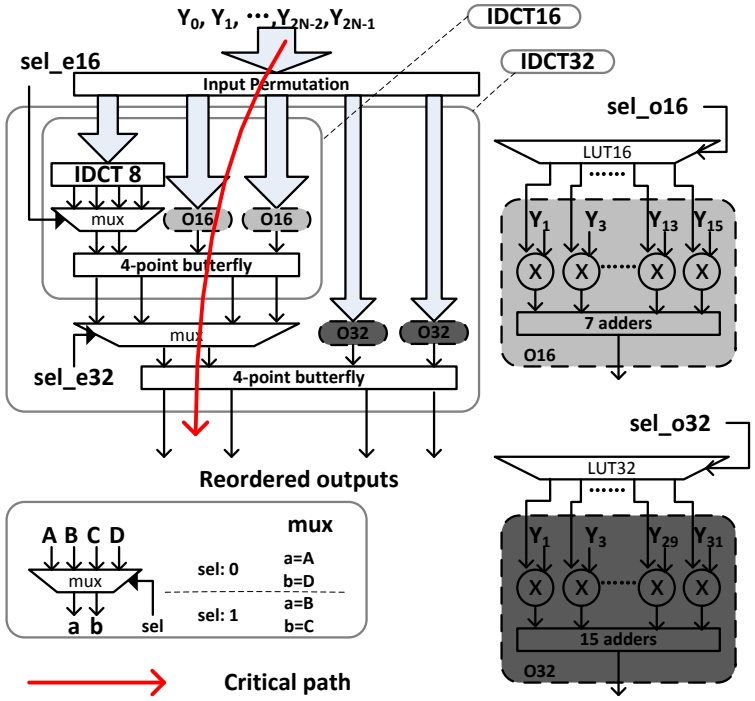

Figure 2-10 Proposed architecture for unified 8/16/32-point 1D IDCT.

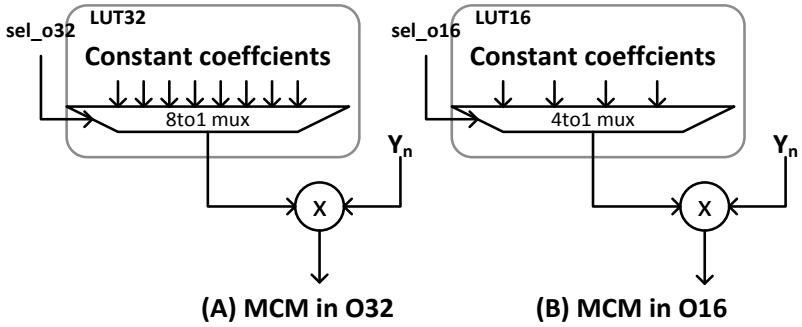

Figure 2-11 Inputs of MCM in O32 and O16.

It has to be noted that in the RTL level, the multiplication is implemented directly (* operator in Verilog). However, the multiplier in odd engine is multiple constant multiplier (MCM) which can be implemented as the adder tree by the Design Complier tool. Figure 2-11(A) gives the inputs of MCM in O32, one constant input is selected from an 8to1 multiplexer and the variable is odd-index input of 32-point IDCT. Similarly, for the multiplier in O16, one constant input is selected from a 4to1 multiplexer, and the variable is odd-index input of 16-point IDCT. Moreover, the circuit of

O32 is shown in Figure 2-10. 16 multipliers are processed in parallel. The number of multipliers does not influence the critical path. According to the timing report by Design Complier, the critical path is shown in Figure 2-10. We can see that the critical path does not include the O32.

It should be noted that in my design, within each cycle, four final outputs of IDCT can be generated regardless of the TU size. The register is not used to store the intermediate results, and the 4-point IDCT does not adopt the RPISO scheme since it may be generated within one cycle; thus, there is no need to reorder the outputs. The proposed unified architecture only supports the 8/16/32-point IDCT. Because the 4-point IDCT incurs a small hardware cost, the 4-point IDCT is implemented individually. The reason for doing so is that the overhead required to embed a 4-point IDCT into the unified architecture is higher than the direct implementation of a 4-point IDCT.

The architecture for the IDCT has been given in the above. It is noted that for the DCT, the processing order is reverse so that the butterfly structure is at the first step and the permutation is at the last step. Therefore, there is no need to reorder the outputs. For 8-point DCT, two clock cycles are required. In the first clock cycle, $\{X_0,X_1,X_2,X_3\}$ are output. In the second clock cycle, $\{X_4,X_5,X_6,X_7\}$ are output. For 16-point DCT, four clock cycles are required. The four outputs in four clock cycles are $\{X_{4*n},X_{4*n+1},X_{4*n+2},X_{4*n+3}\}$ where n is from 0 to 3 in four clock cycles. For the 32-point DCT, eight clock cycles are required. The four outputs in four clock cycles are $\{X_{4*n},X_{4*n+1},X_{4*n+2},X_{4*n+3}\}$ where n is from 0 to 7 in eight clock cycles. For the calculation part in forward transform, MCM is also used which is the same as inverse transform.

2.3 Proposed Architecture for Transpose Buffer Part

For the transpose memory part, it is the same for forward transform and inverse

transform. The inverse transform is taken as an example. As shown in Eq. (2-1), a complete 2D IDCT is composed of two 1D IDCT. The row IDCT is IT1 and the column IDCT is IT2. IT2 can only start after getting a complete column of IT1 results. However, the IT1 results are obtained row by row. Therefore, we require transpose memory stores the IT1 results.

2.3.1 Reordered Data Mapping Scheme for SRAM-based Transpose Buffer

To reduce the hardware cost, SRAM is used to realize the transpose memory. The IT1 result is stored in the SRAM. The parallelism in my design is 4 pixels/cycle, so four SRAMs were used. In order to show the features of the proposed reordered method, the method without reordering is shown at first. Taking TU 32 × 32 as an example, the mapping scheme without reordering is shown in Figure 2-12. In Figure 2-12, (m, n) indicates the IT1 result $X_{m,n}$ (m, n = 0, 1, 2,…, 31). The samples in the different columns are marked using various colors. The column 4*k (k = 0, 1, 2,…, 7) is shaded in white, the column 4*k+1 is shaded in light gray, the column 4*k+2 is shaded in dark gray, and the column 4*k+3 is shaded in black. In each clock cycle, four IT1 results of one row are written into SRAMs. In the first clock cycle, $X_{0,0}$, $X_{0,1}$, $X_{0,2}$ and $X_{0,3}$ are written in four SRAMs, respectively. For the other IT1 results, without reordering, the results of the column 4*k are stored in the SRAM 0, the results of the column 4*k+1 are stored in the SRAM 1, the results of column 4*k+2 are stored in the SRAM 2 and the results of column 4*k+3 are stored in the SRAM 3. However, by using this non-reordered data mapping, four IT1 results of one column cannot be fetched in one clock cycle. For example, $X_{0,0}$, $X_{1,0}$, $X_{2,0}$ and $X_{3,0}$ are all stored in the SRAM 0 so they cannot be fetched in one clock cycle. Therefore, the reading operation cannot achieve the pixel parallelism of 4.

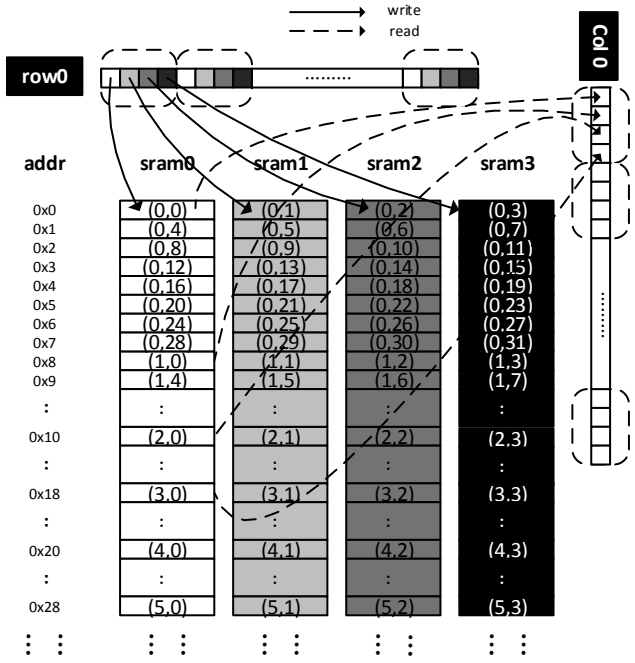

Figure 2-12 Data mapping scheme for the SRAM without reordering.

In order to make the reading operation achieve the pixel parallelism of 4, the storing position is reordered. The proposed data mapping method is shown in Figure 2-13. $X_{4*m,n}$, $X_{4*m+1,n}$, $X_{4*m+2,n}$, and $X_{4*m+3,n}$ are written in four SRAMs so that they can be read out in one cycle. For example, $X_{0,0}$, $X_{1,0}$, $X_{2,0}$, and $X_{3,0}$ are written in four SRAMs. Therefore, these four samples can be read within one cycle. Similarly, $X_{4*m,0}$, $X_{4*m+1,0}$, $X_{4*m+2,0}$, and $X_{4*m+3,0}$ are written in four SRAMs. Four IT1 results of the first column can be read within one cycle. After eight cycles, the IT1 results of the first column can be fetched. After the IT1 results of the first column are obtained, IT2 can begin. Every eight cycles, the IT1 results of one complete column are read out for IT2.

To make full use of the transpose memory, two data mapping modes are proposed. When the transpose memory is full of the IT1 results of the same transform block N, the IT1 results of the next transform block N+1 are written in from the next cycle. Meanwhile, the data mapping mode interchanges. In the other mapping scheme mode, X'n,m (the IT1 results of block N+1) is stored in the same address as Xm,n of block

N. The detailed mapping scheme for mode 1 is shown in Figure 2-14. Similarly, IT1 results of one complete column can be read within eight cycles.

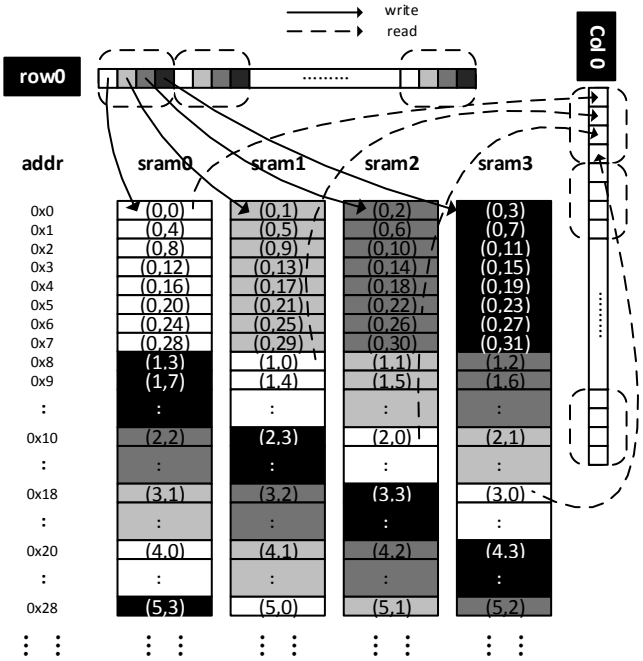

Figure 2-13 Proposed data mapping scheme for the SRAM (mode 0).

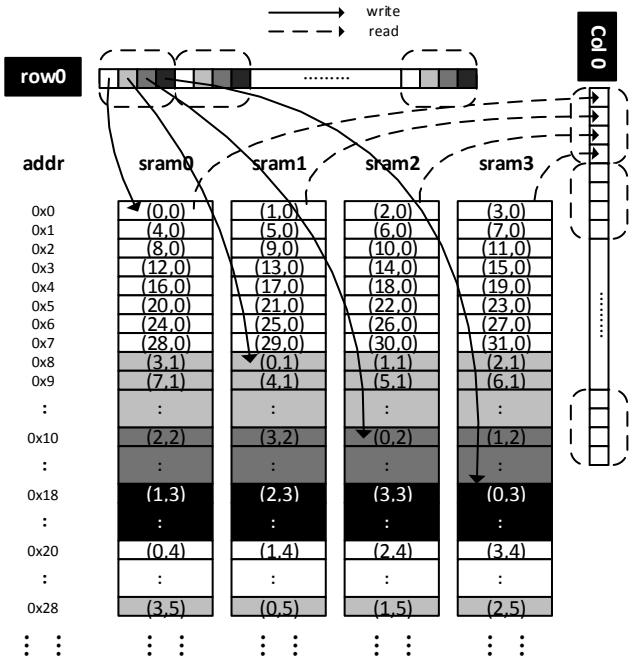

Figure 2-14 Proposed data mapping scheme for the SRAM (mode 1).

The target is to design a transpose memory that can store the IT1 results for vari-

ous transform sizes on the decoder side. The data mapping scheme used to store the 32 × 32 IT1 results was given in Figure 2-13 and Figure 2-14. To store the 32 × 32 IT1 results in four SRAMs, the depth of each SRAM is 256 (32 × 32/4). If the transform size is smaller, the IT1 results are stored in the corresponding address in the SRAM. For example, when writing the 16 × 16 IT1 results, $X_{m,n}$ (m, n = 0, 1, 2,…,15) is written in the corresponding position as shown in Figure 2-13 and Figure 2-14. One quarter of the transpose memory is used for transform block 16 × 16. When storing the 8 × 8 and 4 × 4 IT1 results, the proportion of the transpose memory that is used is only 1/16 and 1/64, respectively.

By using the proposed data mapping scheme, within each cycle, all the input/output ports of SRAM are used for writing/reading. Therefore, 100% I/O utilization of each SRAM can be achieved. In addition, the data width of SRAM is significantly reduced compared with previous work, so the hardware area can be reduced. The detail comparison result is shown in the experimental results.

The data mapping scheme has already been given when the pixel parallelism is 4. The method can be also extended to the larger pixel parallelism such as 16. Take TU8 as an example, IT1 results of two columns are read in one clock cycle, so they should be stored in the different banks of SRAMs. The detail data mapping method is shown in Figure 2-15 where [r,c] represents the IT1 result at the r-th row and c-th column. In the first clock cycle, the results of first two rows (orange in Figure 2-15) are stored. For the 0-th row, [0,0] is stored in the SRAM#0, and the rest 7 results are stored in the column order. For the 1-th row, [1,0] is stored in the SRAM#8, and the rest 7 results are also stored in the column order. In the second clock cycle, the results of second two rows (yellow in Figure 2-15) are stored. Because [2,0] will be read in the same clock cycle with [0,0] and [0,1], the bank in which it is stored has to be different with that of [0,0] and [0,1]. Therefore, [2,0] is stored in the SRAM#2. Similarly, the IT1 results of the first two columns are stored in different banks of SRAMs as shown in Figure 2-15.

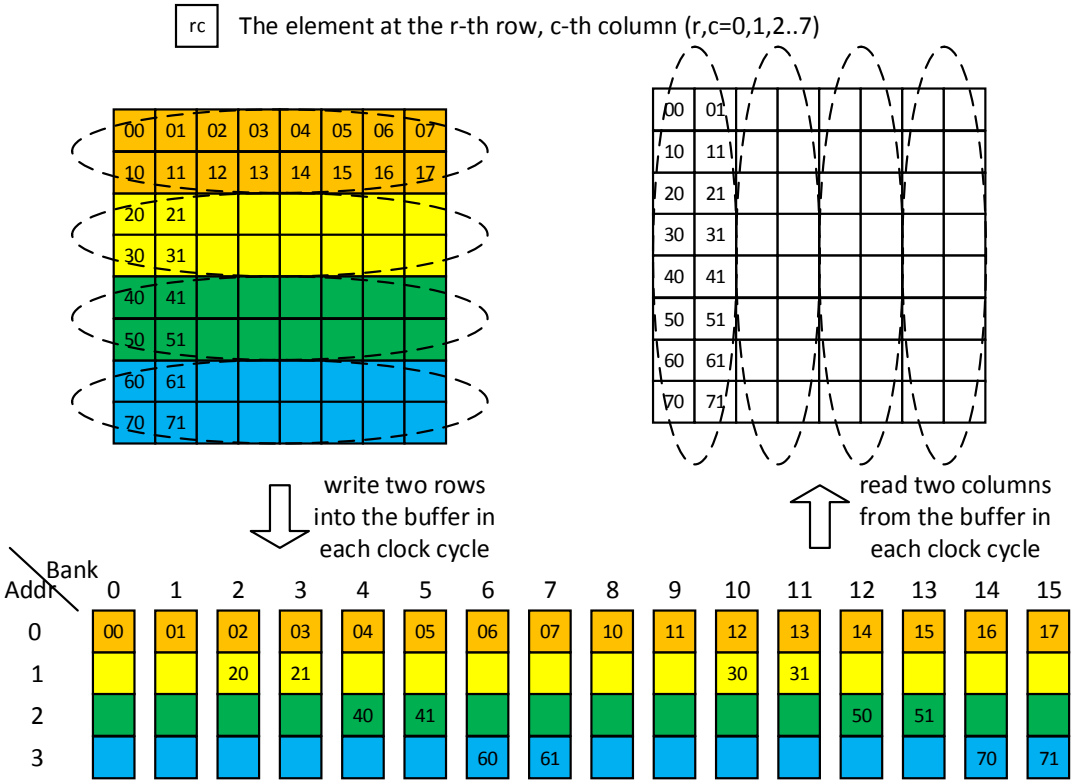

Figure 2-15 Data mapping example for TU8x8.

Table 2-1 Data mapping method for the transpose memory.

Input: TU size, the position of coefficient. c and r represent the c-th column and r-th row, respectively. The range of c and r is [0, TU size-1].
Output: the corresponding bank of SRAM and the address in each bank for storing the coefficient
switch (TU size) case 4x4: Bank = c + r*4 Addr = 0 case 8x8: Bank = (c + (r%2) * 8 + (r/2) * 2) % 16 Addr = r/2 case 16x16: Bank = (c + r) % 16 Addr = r case 32x32: Bank = (c + r) %16 Addr = r * 2+ c/16 endswitch

For each IT1 result at the r-th row and c-th column, the bank of SRAM and the address in each bank is shown in the pseudo code in Table 2-1. For a specific position of four kinds of TUs, the detail bank of SRAM and the address for writing can be obtained by the pseudo code in Table 2-1. By following the above method, IT2 can read the IT1 results of one block/two columns/one column/half column for TU4/8/16/32 in each clock cycle.

2.3.2 Pipelining Schedule for Write and Read Operation

The type of SRAM does not influence the proposed data mapping scheme in Chapter 2.3.1. No matter a single-port or two-port SRAM is adopted, the I/O utilization of SRAM can always achieve 100% by using my proposed data mapping scheme.

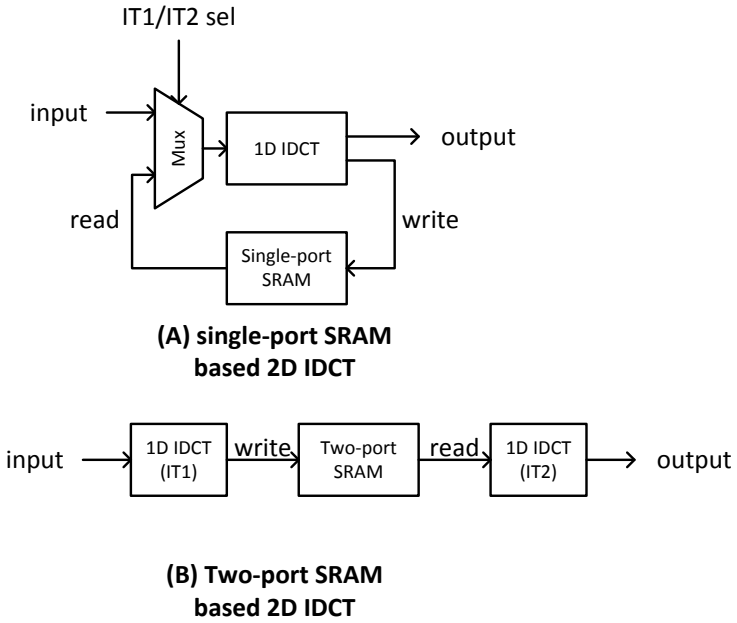

Figure 2-16 Overall 2D IDCT architecture.

If a single-port SRAM is used as a transpose memory, write and read cannot work in parallel. IT1 is performed at first and the results are written in the memory. After that, the IT1 results are read out and IT2 is performed. IT1 and IT2 can share one 1D

IDCT architecture. The overall architecture is shown in Figure 2-16(A).

To make a fully pipelined 2D IDCT architecture, two-port (1r1w) SRAM is used in my design. Writing IT1 results in memory and reading IT1 results from memory can perform in parallel. IT1 and IT2 require the 1D IDCT hardware resource, respectively. The overall architecture is shown in Figure 2-16(B).

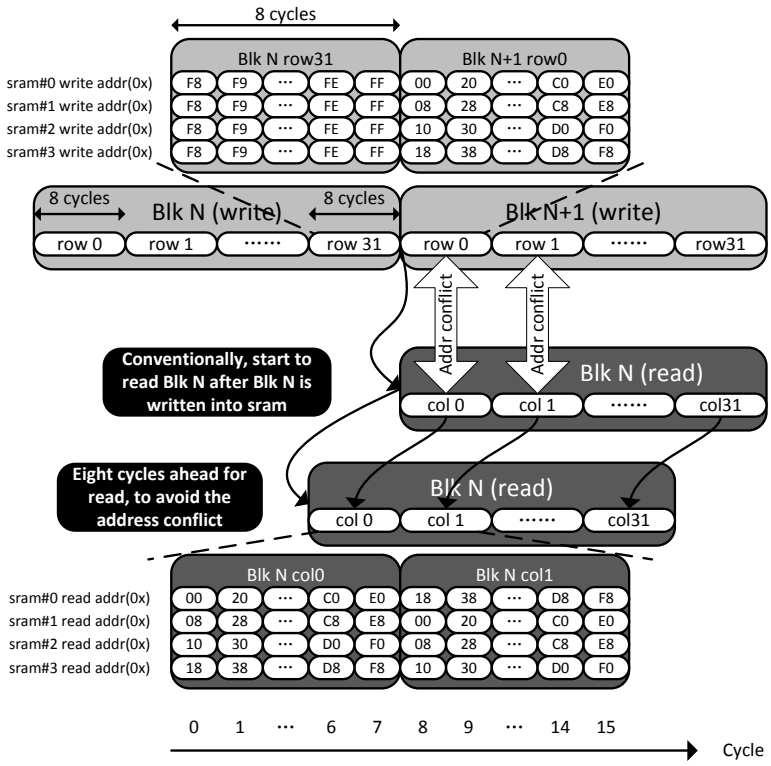

Figure 2-17 Proposed pipelining schedule for the read (IT2) and write (IT1).

The processes for IT1 and IT2 are pipelined in my design. In this way, the writing of the IT1 results and reading results for IT2 can be done in parallel. The read and write address for each SRAM should be different in each cycle. To avoid this address conflict, a pipelining schedule for a two-port SRAM is proposed as shown in Figure 2-17.

Taking TU 32 × 32 as an example, from cycles 0 to 7, the IT1 results of row 31 in block N (blkN, row31) are written in the SRAM. Assume that the data mapping

scheme for writing block N is mode 0. Within each cycle, $X_{31,4*k}$, $X_{31,4*k+1}$, $X_{31,4*k+2}$, and $X_{31,4*k+3}$ are written. The corresponding data mapping scheme is shown in Figure 2-13. Then, the IT1 results of the next block N+1 are written. The data mapping mode interchanges to mode 1, and the corresponding data mapping scheme is interchanged as shown in Figure 2-14. The IT1 results of row 0 in block N+1 (blkN+1, row0) are written from cycles 8 to 15. Conventionally, after all the IT1 results of block N are written in, the IT1 results of block N begin to be fetched. In this way, the IT1 results of column 0 in block N (blkN, col0) are read from cycles 8 to 15. However, We can see in Figure 2-17 that the address for writing (blkN+1, row0) and reading (blkN, col0) is the same, which will lead to an address conflict. To solve this address conflict problem, reading (blkN, col0) is done eight cycles in advance.

In my pipelining schedule for SRAM, from cycles 0 to 7, the IT1 results of (blkN, col0) are read out. Within each cycle, $X_{4*k,0}$, $X_{4*k+1,0}$, $X_{4*k+2,0}$, and $X_{4*k+3,0}$ are read out. It should be noted that when reading the last sample of the first column $X_{31,0}$ in the cycle 7, this sample has already been written into the SRAM in cycle 0. Thus, it could be fetched in cycle 7. From cycles 8 to 15, the IT1 results of (blkN, col1) are read out, while the IT1 results of (blkN+1, row0) are written in. We can see that there is also no address conflict between them. After that, the IT1 results of (blkN+1, rowK) are written in and (blkN, colK+1) are read out for IT2; using this schedule, the address conflict between reading and writing is avoided.

2.4 Experimental Results

Verilog HDL is used to implement the proposed design. The design was synthesized with a TSMC 90nm cell library. As mentioned previously, there are two critical problems that lead to the significant hardware cost. One is the logical computational part, and the other is the transpose memory part. The results are analyzed from these two perspectives.

2.4.1 Experimental Results of Logical Computational Part

In my design, a 4-point IDCT is not included in the unified architecture. The reason is explained at the end of Section 2.2.2. The individual gate count for a 4-point IDCT is 2.4 k, and the gate count for the 8/16/32-point unified IDCT is 63.9 k. Therefore, an overall gate count of 66.3 k is required for a 1D IDCT design that can support the entire transform size on the decode side. The concept of the normalized area (NA) is introduced to obtain a fair performance comparison, and normalized area by maximum throughput is defined by the following equation.

$$\begin{aligned} &\textit{Normalized area by maximum throughput} \\ &= \frac{Gate}{MaxThroughput} = \frac{Gate}{MaxFreq \times Tp} \end{aligned} \tag{2-11}$$

where MaxFreq is the maximum frequency of the design and Tp is the throughput within each cycle, which is equal to the pixel parallelism. “Gate” refers to the gate count for the logical calculation part for the IDCT excluding the transpose memory.

In order to demonstrate the effectiveness of my proposed design, the comparison is performed with those in references [18]-[22], which describe the low-cost designs for the HEVC IDCT. The comparison result is shown in Table 2-2. It should be noted that the throughput in [19], [20], and [22] is different for various 2N-point IDCTs. On the decoder side, the worst-case scenario should be considered. For example, in [20], the authors achieve 4/8/16/32 pixels/cycle for a 4/8/16/32-point IDCT, respectively. In the worst case, when the TU size is always 4 × 4 on the decoder side, the parallelism is 4 pixels/cycle. The smallest parallelism is used to calculate the normalized area. For the logical IDCT computation part, compared with [18]-[22], the gate count is reduced by more than 25%.

For the 1D DCT architecture in [26], the pixel parallelism is 16 pixels/cycle. In fact, for the same function, the architecture with higher pixel parallelism can share more computations and hardware resources. So the architecture with higher pixel parallelism usually has less normalized area than one with lower pixel parallelism. The

normalized area by parallelism is defined as follows.

$$Normalized\ area\ by\ parallelism = \frac{Gate}{pixel\ parallelism} \quad (2\text{-}12)$$

Table 2-2 Comparison with other HEVC 2D IDCT architectures.

Design	Tech-nology (nm)	Max Speed (MHz)	Pixel Par-allelism	Gate Count (k)[5]	Normalized Area by Eq. (2-11)[6]	Normalized Area by Eq. (2-12)[7]	Supporting Video Format
Shen [18]	130	350	4	109.2 x 2	156	54.6	4Kx2K 30fps
Shen [19]	130	191	4~8[2]	54.1 x 2	141.62	27.05	4Kx2K 30fps
Zhu [20]	90	311	4~32[3]	117.7 x 2	189.23	58.85	4Kx2K 60fps
Park [21][1]	180	300	2.12	52.3 x 2	164.46	49.34	4Kx2K 30fps
Chiang [22]	90	270	1.38~2.67[4]	63.8 x 2	342.46	92.46	4Kx2K 30fps
Meher[26]	90	N/A[8]	16	131 x 2	N/A	16.38	8Kx4K 60fps
Proposed 4x	**90**	**312**	**4**	**66.3 x 2**	**106.25**	**33.15**	**4Kx2K 60fps**
Proposed 8x	**90**	**312**	**8**	**88.2 x 2**	**70.67**	**22.05**	**8Kx4K 30fps**
Proposed 16x	**90**	**312**	**16**	**113 x 2**	**45.27**	**14.13**	**8Kx4K 60fps**

1) Park [21] only supports 16/32-point IDCT.

2) The parallelism for 4/8/16/32-point IDCT is 4 / 8 / 4 / 4 pixels/cycle, respectively.

3) The parallelism for 4/8/16/32-point IDCT is 4 / 8 / 16 / 32 pixels/cycle, respectively.

4) The parallelism for 4/8/16/32-point IDCT is 2 / 2.67 / 2 / 1.38 pixels/cycle, respectively.

5) Gate count for 2D IDCT.

6) The unit for Normalized Area by max. throughput is 1/(MHz*pixel/cycle). If the parallelism is different for various IDCT size, the least parallelism is used to calculate in Eq. (2-11).

7) The unit for Normalized Area by parallelism is 1K/(pixel/cycle). If the parallelism is different for various IDCT size, the least parallelism is used to calculate in Eq. (2-12).

8) Reference [26] synthesize the architecture with the desired timing constraint for 8Kx4K 60fps. The corresponding operating frequency is 187MHz.

To have a fair comparison with [26], the architecture with higher parallelism such

as 8 pixels/cycle and 16 pixels/cycle is designed. Proposed RPISO scheme is still adopted in these architectures. When the pixel parallelism is 8 pixels/cycle, 16/32-point IDCT can take advantage of RPISO to reduce the redundant inputs and corresponding calculations of 16/32-point butterfly structure. Similarly, when the pixel parallelism is 16 pixels/cycle, the 32-point IDCT can take advantage of RPISO scheme to save the hardware cost.

The results are shown in Table 2-2. We can see that the design with higher pixel parallelism consumes much less normalized area is than the design with lower pixel parallelism. Compared with [26], my IDCT architecture with parallelism of 16pixels/cycle consumes less normalized area by parallelism than [26]. The reason is partially because that RPISO scheme is used to reduce the calculations of butterfly inputs for 32-point IDCT.

It has to be noted that in [26], the author synthesize the 1D architecture with the desired timing constraint for 8Kx4K 60fps. The corresponding frequency is 187MHz (7680x4320x60x1.5/16). Therefore, 187MHz mentioned in [26] is considered as the operating frequency. The results in Table 2-2 show a comparison of my results with other unified architectures for HEVC IDCT. Using the RPISO scheme, it is possible to realize savings in area. Each transform block is also implemented individually. The results are shown in Table 2-3. In [19], the author presented the hardware area for each transform block. Compared with [19], savings in the required hardware resources can be realized for each transform block by using the RPISO scheme.

Table 2-3 Comparison of the hardware cost for each transform block.

	Gate Count (k)			
Transform size	4-point	8-point	16-point	32-point
Shen [19]	7.2	19.6	31.1	54.1
Proposed	**2.4**	**8.6**	**24.3**	**47.8**

In addition, it is obvious that the unification method proposed in Section 2.2.2 can result in reduced hardware overhead. The overall gate count for an individual transform block is 83.1 k (2.4 + 8.6 + 24.3 + 47.8), which is greater than the 66.3 K of the proposed unified architecture.

2.4.2 Experimental Results of Transpose Buffer Part

For the transpose memory part, four SRAM blocks are used since the parallelism is 4 pixels/cycle. To store the IT1 results of a 32 × 32 transform block, the width of each SRAM is 16 bit (one IT1 result) and the depth is 256 (32 × 32/4). A total of 16384 SRAM bits were used. To store the IT1 results of the smaller transform, a part of the transpose memory is used. The detailed data mapping scheme was given in Chapter 2.3.

In [18] and [20], the authors also used SRAM to realize the transpose architecture, and the total SRAM bit is the same as my design. However, much smaller area can be achieved since the width of my SRAM is significantly reduced compared with [18] and [20]. To perform a comparison with various transpose memory designs, a memory complier is implemented using the same process TSMC 90nm. A single-port SRAM is used in [18]. However, a 1D IDCT architecture in [18] can also be adopted in a fully pipelined 2D IDCT architecture when using a two-port SRAM. To have a fair comparison of all the SRAM-based transpose memory design, a two-port memory complier is used to generate the area. The total area for the SRAMs is shown in Table 2-4. Using my proposed narrower and deeper SRAMs, the total SRAM area was 80988 um^2. Therefore, a savings of at least 62% in the SRAM area can be realized compared with [18] and [20]. In [21] and [22], the authors used registers to implement a transpose memory based on the traditional method, which is not so area-efficient. The author in [19] did not develop the transpose architecture.

Table 2-4 Comparison with other SRAM-based transpose memory.

Memory design		Shen [18] **	Zhu [20]	**Proposed**
Total Bits		16384bits		
Number of SRAMs		4	32	**4**
Each SRAM	Width	512bits	16bits	**16bits**
	Depth	8	32	**256**
	Area(um^2)	68431	6590	**20247**
Total SRAM Area (um^2) *		273724 (4*68431)	210880 (32*6590)	**80988 (4*20247)**

* The areas are generated by the same process TSMC 90nm.

** The type of SRAM in [18] is single-port. A two-port memory complier is used to generate its area for a fair comparison of a fully pipelined 2D IDCT.

2.5 Chapter Summary

In this chapter, an area-efficient multi-size transform architecture for HEVC is presented. To reduce the required amount of hardware resources, the RPISO scheme is proposed. In this scheme, the final outputs are reordered so that the redundant inputs of butterfly and the required calculations for butterfly inputs in each cycle are reduced. Based on the RPISO scheme, a unified architecture for an 8/16/32-point transform is presented. Using Chen's algorithm, the N-point architecture is reused in the 2N-point architecture. An SRAM instead of a register is used to realize the transpose memory. The data mapping scheme is proposed in which the storing positions in SRAM are reordered. In addition, the pipelining schedule for the SRAM is proposed so that the write and read operation can be conducted in parallel. The results show that the savings of about 25% in the area for the logical computational part, and 62% in the area for the transpose memory part can be achieved compared with previous works.

3. A Low-Cost System Design for De-quantization and Inverse Transform

The low-cost architecture for transform has been introduced in the last chapter, and this chapter[2] will introduce the proposals for the system of de-quantization (DQ) and inverse transform (IT). The position of this chapter in the mode decision and reconstruction loop is shown in Figure 3-1. In the mode decision, DQ and IT are required in the full RDO pass. In addition, these two components are also required in the reconstruction loop. There are two major concepts of ideas in this chapter. For the de-quantization, the idea is to reduce the number of multipliers and then reuse four multipliers in different clock cycles. For the system of de-quantization and inverse transform, the concept is to reuse the pixel data to detect the zero elements and then skip the read and write operations of SRAM for the zero elements. This chapter is related with the publication [1] in Page 122.

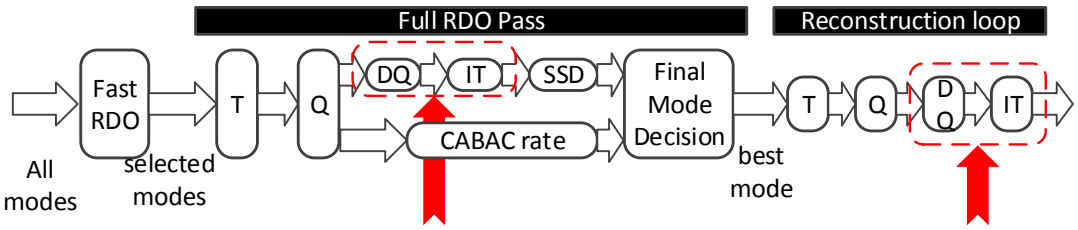

Figure 3-1 The position of chapter 3 in the mode decision and reconstruction loop.

[2] This chapter is related with the publication [1] in Page 122.

3.1 Introduction

3.1.1 Overview of the System of De-quantization and Inverse Transform

The system of de-quantization and inverse transform is shown in Figure 3-2. In the de-quantization, the operation is multiplication, addition and shift. The process of inverse transform has been shown in Chapter 2. Between de-quantization and inverse transform, since the data format is 4x4 in the de-quantization and row by row in the inverse transform in the decoding, so a memory is required between the de-quantization and inverse transform.

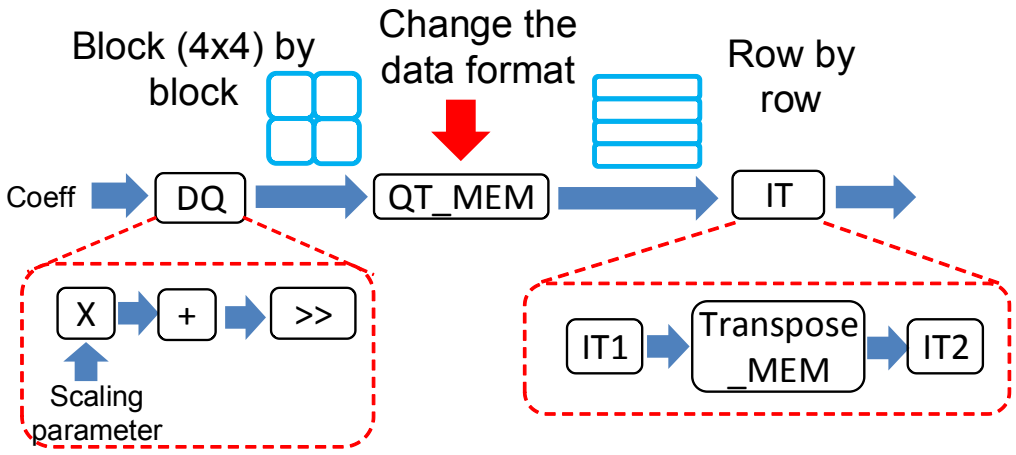

Figure 3-2 System of de-quantization and inverse transform.

In the encoding, the inputs of de-quantization are the results of quantized coefficients. In the decoding, the inputs of de-quantization are generated by the results of syntax elements. Each coefficient can be represented by five syntax elements. *sig_coeff_flag*, *coeff_abs_level_greater1_flag*, *coeff_abs_level_greater2_flag* indicate whether the coefficient is larger than 0/1/2 or not, respectively. *coeff_abs_level_remaining* indicates the remaining absolute value. *coeff_sign_flag*

means the sign of each coefficient. For each TU, the SE's are coded in the format of 4x4 according to the HEVC standard [2]. Within each 4x4 block, 16 coefficients are coded in the scan order which is selected from up-right diagonal, horizontal and vertical order. For each specific coefficient, *TransCoeffLevel* can be calculated as the pseudo code in Table 3-1 where *numSigCoeff* is the number of non-zero coefficients already coded. *lastGreater1ScanPos* is the first coefficient whose *coeff_abs_level_greater1_flag* is 1 in the scan order.

Table 3-1 Calculation of TransCoeffLevel.

Input: sig_coeff_flag, coeff_abs_level_greater1_flag, coeff_abs_level_greater2_flag, coeff_abs_level_remaining, coeff_sign_flag
Output: TransCoeffLevel

for each coefficient in the 4x4 block
if(sig_coeff_flag==0)
TransCoeffLevel = 0
else
baseLevel = 1+ coeff_abs_level_greater1_flag+
coeff_abs_level_greater2_flag
if(baseLevel==((numSigCoeff<8) ? ((the coefficient is at the lastGreater1ScanPos) ?
3 :2): 1)))
TransCoeffLevel = coeff_abs_level_remaining + baseLevel
else
TransCoeffLevel = baseLevel
endif
TransCoeffLevel = TransCoeffLevel *(1-2*coeff_sign_flag)
endif
endfor

An example of input and output is given in Figure 3-3, if *sig_coeff_flag* is 0, *TransCoeffLevel* is 0. Otherwise, *TransCoeffLevel* is based on *baseLevel* and *coeff_abs_level_remaining*. *coeff_abs_level_remaining* is only required under the following cases. When *numSigCoeff* is not smaller than 8 (n=0,1,2…6 in Figure 3-3), *coeff_abs_level_remaining* is required. When *numSigCoeff* is smaller than 8, for the coefficient at the *lastGreater1ScanPos* (n=11 in Figure 3-3), *co-*

eff_abs_level_remaining is required if *coeff_abs_level_greater2_flag* is true; for the coefficient not at the *lastGreater1ScanPos*, *coeff_abs_level_remaining* is required if *coeff_abs_level_greater1_flag* is true (n=7,9 in Figure 3-3). After getting the absolute value of *TransCoeffLevel*, *coeff_sign_flag* is used to indicate the sign for *TransCoeffLevel*. After that, the DQ results can be calculated by the addition and shift.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
coeff_sign_flag[n]	0	0	1	0	1	0	1	1	1	0	1	1	0	0	0	0
sig_coeff_flag[n]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
coeff_abs_level_greater1_flag[n]	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
coeff_abs_level_greater2_flag[n]	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
coeff_abs_level_remaining[n]	20	3	15	8	2	6	1	3	0	14	0	1	0	0	0	0
TransCoeffLevel[n]	21	4	-16	9	-3	7	-2	-5	-1	16	-1	-4	1	1	1	0

Figure 3-3 An example of the input and output for the pseudo coded shown in Table 3-1.

For the de-quantization, there are two problems. One problem is that in the real implementation, the results of *coeff_abs_level_remaining* do not align with the coefficients because of two reasons. One reason is that the *coeff_abs_level_remaining* is coded from the last coefficient to the first coefficient, and the other reason is that it is not coded for all the coefficients. An example is given in Figure 3-4 where C[n] means the original result of *coeff_abs_level_remaining* before the alignment, R[n] indicates whether the coefficient requires the results of *coeff_abs_level_remaining* or not, and AC[n] indicate the result of *coeff_abs_level_remaining* after the alignment, and it align with R[n]. In the example, only three coefficients at position of 0,7,15 need the remaining results. So R[0], R[7], R[15] are 1. AC[0], AC[7] and AC[15] are C[2], C[1] and C[0], respectively.

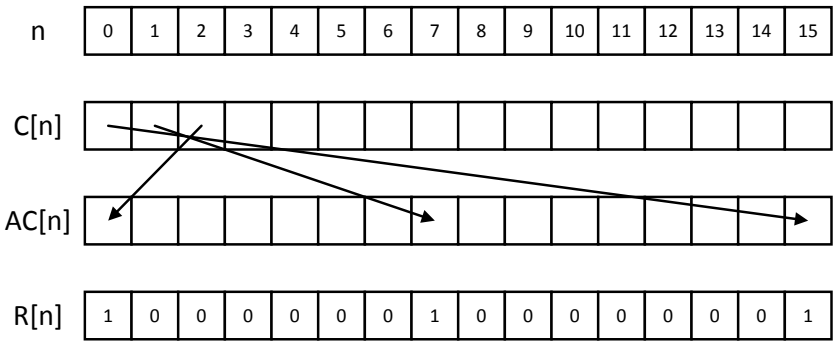

Figure 3-4 An example for the unaligned *coeff_abs_level_remaining*.

The second problem is that the multiplications are required in the de-quantization. Although de-quantization does not consume as many as hardware resources compared with inverse transform, multiplications in the de-quantization are still hardware consuming. For the de-quantization, there are no specific previous works on reducing the hardware consumption up to now.

For the whole system, the problem is that there are many zero elements in the system as shown in Figure 3-5. For the zero elements, the processing can be optimized. In addition, since the data format is different for the de-quantization and inverse transform, so a data mapping method is required for the buffer between the de-quantization and inverse transform.

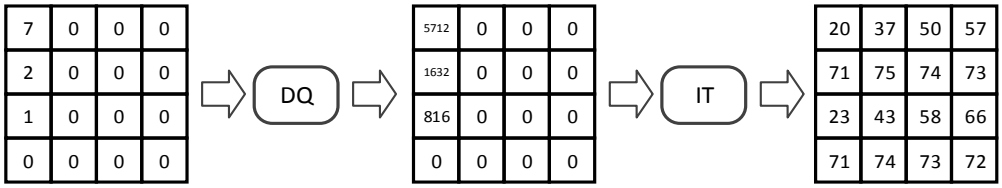

Figure 3-5 Many zero elements in the system.

3.1.2 Previous Works

For the de-quantization, there are no specific previous works on reducing the hardware consumption as far as the author knows. For the data exchange buffer, [36] gave a very delicate mapping method as shown in Figure 3-6. In the case of TU4x4, the

half row of 0-0, 0-1, 0-2 and 0-3 are written in the SRAM. Therefore, the I/O utilization of SRAM is only 50%. For the other TU sizes, the utilization is 100%.

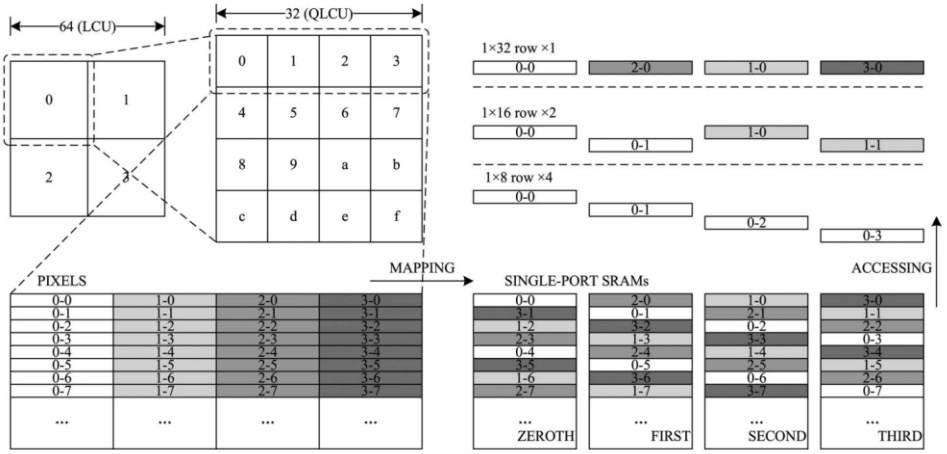

Figure 3-6 The exchange buffer method around DCT/IDCT [36].

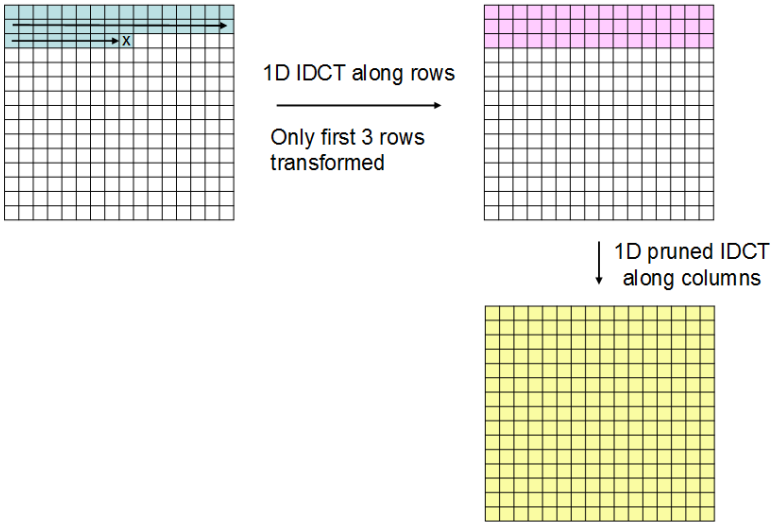

Figure 3-7 IDCT pruning when transform block is scanned horizontally. Colored regions are non-zero regions.

For the optimization of the zero element processing, [37] used the method in [38] as shown in Figure 3-7. In the case that the scan type is horizontal, the row IDCT is processed before the column IDCT. By doing so, the last non-zero row is detected and the logical computation for all the zero rows can be saved for the row IDCT. For the example in Figure 3-7, the last non-zero element is in the third row thus only the first three rows need the process of row IDCT.

3.1.3 Research Target

For the de-quantization, since there are no previous works, so the remaining problems are that the results of *coeff_abs_level_remaining* do not align with the coefficients and the multiplications are required. Therefore, my research target is to design a low-delay alignment mapping architecture for *coeff_abs_level_remaining* and reduce the number of multipliers. For the system of de-quantization and inverse transform, there are two remaining problems. One problem is that when processing transform unit 4x4, the I/O utilization of SRAM cannot reach 100% in [36]. The other problem is that [37] only adopted the zero skipping for the logical computational part while they did not present the methods to optimize the zero element processing for the memory part. Therefore, the research target is to design a SRAM-based buffer between de-quantization and inverse transform which can achieve 100% I/O utilization. For the zero elements in the system, the target is to prune the read/write operations to save the power consumption.

3.2 VLSI Architecture for De-quantization

3.2.1 Low-Delay Alignment Mapping Architecture

As described in Chapter 3.1.1, the results of *coeff_abs_level_remaining* do not align with the coefficients. So the first step is to align *coeff_abs_level_remaining* with the coefficient. As shown in Table 3-1, *numSigCoeff* and *lastGreater1ScanPos* are required in the judgment. So these two elements are calculated first. The architecture is shown in Figure 3-8.

The equivalent pseudo code with Table 1 for R[n]

```
for each position n in the 4x4 block
  if(numSigCoeff>=8)
     R[n] = sig_coeff_flag[n]
  else if(lastGreater1ScanPos[n])
     R[n] =coeff_abs_level_greater2_flag[n]
  else
     R[n] = coeff_abs_level_greater1_flag[n]
  endif
endfor
```

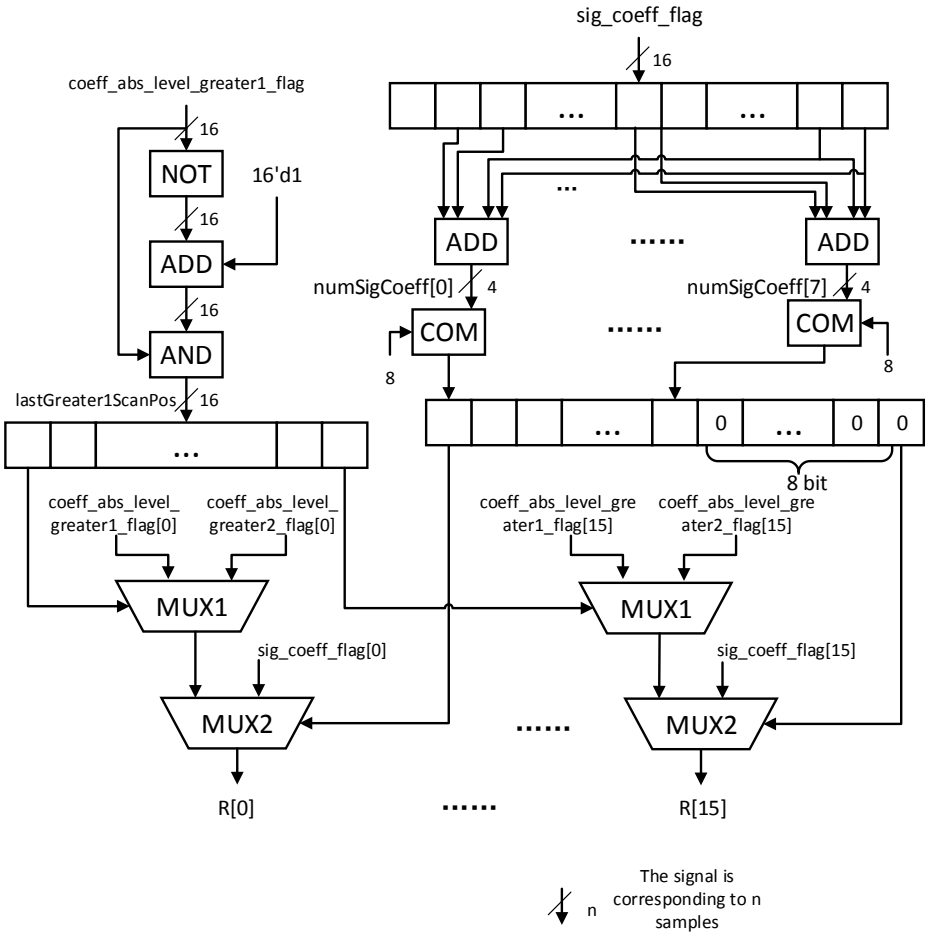

Figure 3-8 The architecture to generate R[n] which indicates whether each coefficient requires *coeff_abs_level_remaining* or not.

The pseudo code for R[n] is shown in Figure 3-8. *lastGreater1ScanPos* is detected in parallel by a leading-one circuit, the result of the bitwise AND operation of the true code and the complement code of *coeff_abs_level_greater1_flag* is a one-hot

code in which the single high bit is corresponding to the position of *lastGreater1ScanPos*. *numSigCoeff* is equal to the summation of *sig_coeff_flag*. For the eight rightmost positions, *numSigCoeff* is not calculated because it is impossible to be larger than 8. After getting the results of *lastGreater1ScanPos* and *numSigCoeff*, the positions of coefficients with *coeff_abs_level_remaining* can be detected by the pseudo code in Table 3-1. The circuit and the equivalent pseudo code are shown in Figure 3-8. A multiplexer (MUX2 in Figure 3-8) is used and the select signal is the comparison result of *numSigCoeff* and 8. If *numSigCoeff* is not smaller than 8, *sig_coeff_flag* is used to indicate R[n]. If *numSigCoeff* is smaller than 8, another multiplexer (MUX1 in Figure 3-8) is used. If the position is *lastGreater1ScanPos*, *coeff_abs_level_greater2_flag* is used to indicate R[n]. Otherwise, *coeff_abs_level_greater1_flag* is used to indicate R[n].

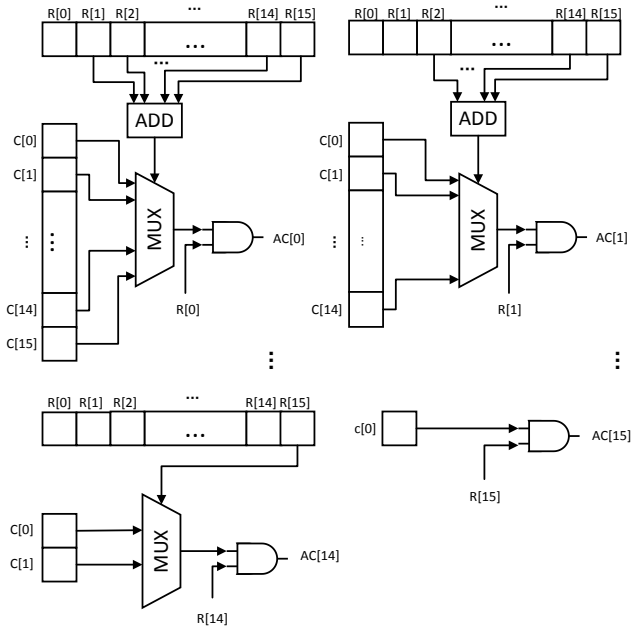

Figure 3-9 The architecture to generate AC[n] which indicates the result of *coeff_abs_level_remaining* after the alignment.

After obtaining R[n], AC[n] can be selected by multiplexers. AC[n] is 0 if R[n] is 0. Therefore, the AND operation is used for generating AC[n] and one input is R[n]. For the non-zero AC[n], the value can be obtained by the multiplexers. Start from

AC[15], it has to be C[0]. For AC[14], it may be equal to C[0] or C[1], and it depends on the R[15]. If R[15] is 1, AC[14] is C[1]; otherwise, AC[14] is C[0]. Similarly, for the other AC[n] (n<15), the values depend on the number of R[m] (m>n) that is 1. The corresponding circuit is shown in Figure 3-9.

3.2.2 Four-multiplier-based Multiplication with Scaling Parameters

After finding the aligned remaining values for the corresponding coefficients, *TransCoeffLevel* can be obtained by the summation of *baselevel* and remaining values, as shown in Table 3-1, and 16 multipliers are used to calculate the multiplication of *TransCoeffLevel* and scaling parameter, which will lead to large hardware cost. *TransCoeffLevel* is composed of two parts, one part is *baselevel*, and the other part is remaining value. For the first part, the largest value is three, so the multiplication results can be obtained by look up tables instead of multipliers. For the second part, according to the experiments, the probability of the 4x4 block with not greater than 4 remaining values is very high. Therefore, only 4 results of *coeff_abs_level _remaining* are input in each clock cycle. By doing so, there are two merits. The first merit is that the number of multipliers can be reduced to 4 so that the hardware cost can be reduced. In almost all the cases, the 4x4 block with less than 4 remaining values can be processed within one clock cycle. For the 4x4 blocks with more than 4 remaining values, four multipliers are exploited in different clock cycles. For example, for the 4x4 block with 7 remaining values, four remaining values are multiplied by the scaling parameter in the first clock cycle. The rest three remaining values are multiplied by the scaling parameter in the second clock cycle. In addition to the merit of reducing the hardware cost, 75% reduction for the data width of *coeff_abs_level_remaining* can be reduced.

3.2.3 Pipeline Schedule for De-quantization

Two stages are designed for DQ. In the first stage, four remaining values are multiplied by the scaling parameter. In the first cycle, C[n] (n=0,1,2,3) are input and they are multiplied by the scaling parameter. The multiplication results of C[n] and the scaling parameter are defined as MC[n]. Because C[n] (n=4,5,....15) are not input in the first cycle, MC[n] (n=4,5,…15) are set as zero. If there are more than four remaining values, C[n] (n=4,5,6,7) are input in the second cycle. MC[n] (n=4,5,6,7) are the multiplication results of input C[n] and the scaling parameter. MC[n] (n=0,1,2,3,8,9,…15) are set as zero. If the third and fourth clock cycles are required, the zero compensation methods are shown in Figure 3-10.

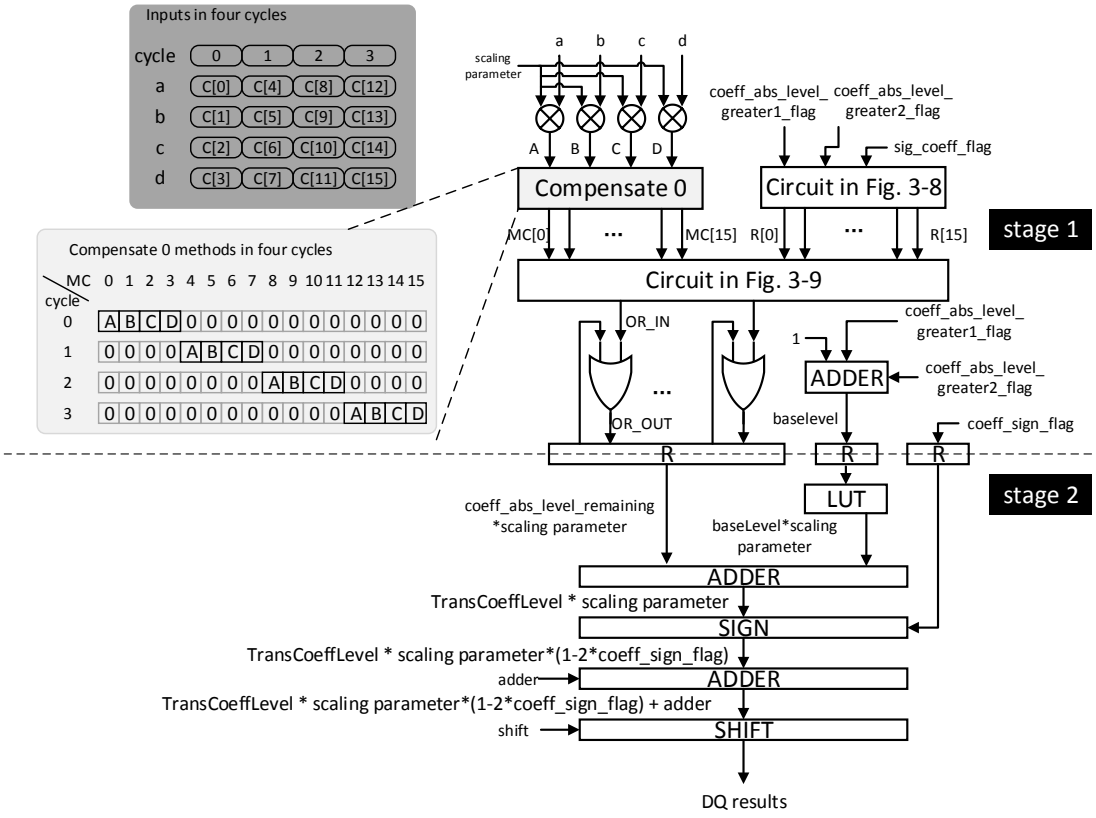

Figure 3-10 The architecture of two stages for DQ.

In each clock cycle, there are only 4 valid MC[n] corresponding to 4 C[n], while the other 12 MC[n] are compensated by 0. In order to integrate the 16 MC[n] calcu-

lated in more than one clock cycle, the OR gate is used. An example is given to explain the meaning of the OR gate in Figure 3-11. There are 6 non-zero C[n] (A-F in Figure 3-11), so 2 clock cycles are required. In the first cycle, the leftmost four C[n] (n=0,1,2,3) are input, thus one input of the OR gate is corresponding to MC[n] (n=0,1,2,3). The other input of the OR gate is initialized as 0 in the first cycle of each 4x4 block, so the output of the OR gate is corresponding to MC[n] (n=0,1,2,3). In the second cycle, the second leftmost four C[n] (n=4,5,6,7) are input, thus one input of the OR gate is corresponding to MC[n] (n=4,5,6,7). The other input of the OR gate is the output of the last clock cycle, so the output of the OR gate is corresponding to the MC[n] (n=0,1,2…,7). We can see that by using the OR operation, MC[n] calculated in different clock cycles can be integrated. In addition to MC[n], the results of baselevel and coeff_sign_flag are also stored in the pipeline registers. In the second stage, the steps are shown in Figure 3-10. At first, the multiplication results of baselevel and scaling parameter are obtained by a LUT and the multiplication results of TransCoeffLevel and scaling parameter can be generated by the addition. After that, coeff_sign_flag is used to add the sign bit. After the addition and shift, the final results of DQ can be calculated and reordered to the horizontal order for the IDCT.

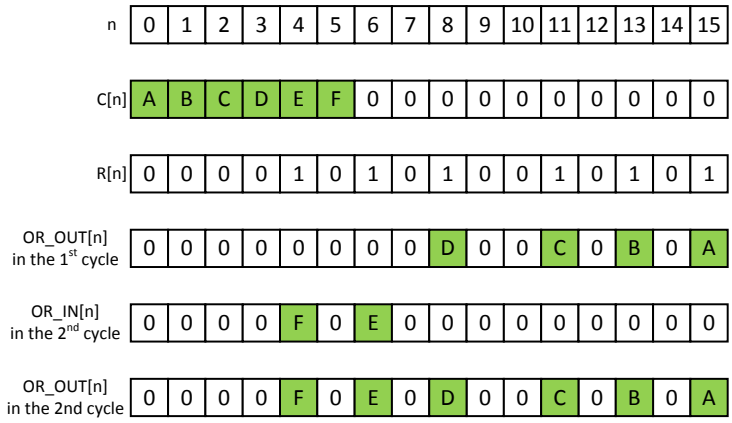

Figure 3-11 An example for the usage of the OR gate in Figure 3-10.

3.3 System Architecture of De-quantization and Inverse Transform

The overall architecture is shown in Figure 3-12. DQ is processed in serial for luma and chroma because there are many all-zero 4x4 blocks which are skipped in the DQ process. According to the experiments, many blocks are all-zero ones, so the serial processing of luma and chroma will not influence the throughput in an actual conditions. IT is processed in parallel for luma and chroma. So there are two paths for IT in Figure 3-12.

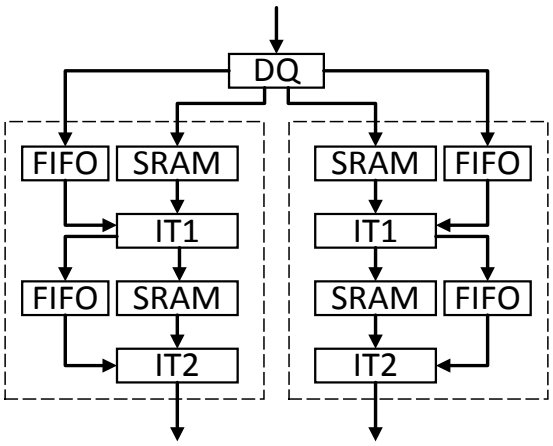

Figure 3-12 Architecture for the whole system.

Between DQ and IT, a buffer (QT buffer) is required to collect all the non-zero 4x4 blocks for each TU. After all the DQ results of non-zero 4x4 blocks are calculated and written in the QT buffer, IT1 fetches the DQ data row by row. The results of IT1 are transposed before entering IT2. The results of IT2 are the final results of the system.

The FIFO between DQ and IT1 transfers two kinds of information. One kind is the zero-element flags, the other kind is the TU information such as TU size. The FIFO between IT1 and IT2 also stores two kinds of information including zero-element flags and TU information. Because IT is processed in parallel for luma and

chroma, there are two FIFOs between DQ and IT1 and two FIFOs between IT1 and IT2.

3.3.1 Reordered Data Mapping Scheme for QT Buffer

The QT buffer is required to store the 4x4 blocks, so the write of QT buffer is in the format of 4x4. However, the format of IT1 process is row by row. So if the read of QT buffer is in the format of 4x4, an additional register-based buffer is required to change the data format. The registers will consume about 80K and 30K gate counts for luma and chroma, respectively. In order to eliminate the additional registers to change the data format, SRAM is used for the QT buffer. Take TU8 as an example, without reordering, the data mapping method for SRAM is shown in Figure 3-13.

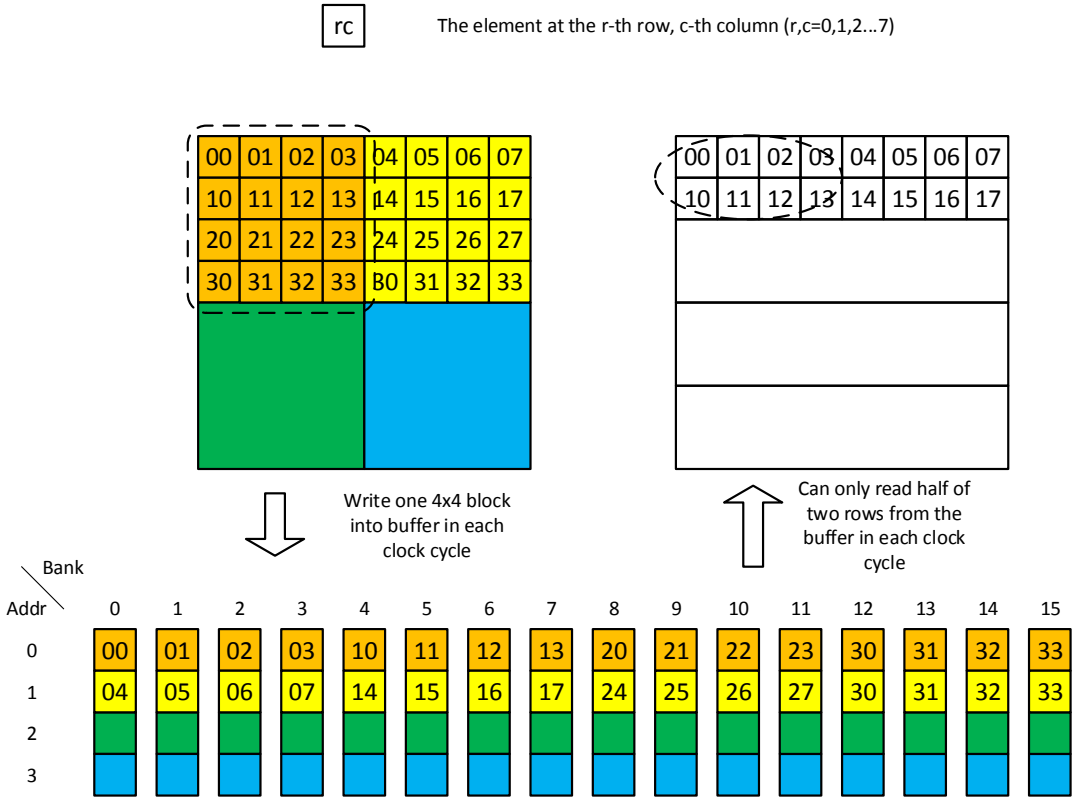

Figure 3-13 Data mapping example for TU8x8 for the QT buffer without reordering.

The coefficients of one 4x4 block are written orderly in 16 banks of SRAMs. However, the results of two rows are in two words of one SRAM so that they cannot

be fetched in one clock cycle.

Table 3-2 Data mapping method for the QT memory.

Input: TU size, sblk_x, sblk_y and n (the position of the coefficient within a 4x4 block). The range of sblk_x and sblk_y is $[0,\frac{TU\ size}{4}-1]$

Output: the corresponding bank of SRAM and the address in each bank for writing.

```
switch(TU size)
 case 4x4:
      Bank = n
 case 8x8:
      switch (sblk_x)
           case 0: Bank = n
           case 1: Bank = (n+8) % 16
              endswitch
 case 16x16:
      Bank = (n+sblk_x*4)%16
 case 32x32:
      Bank = (n+ (sblk_x%4)*4) % 16
endswitch

switch(TU size)
 case 4x4:
      Addr = 0;
 case 8x8:
      Addr = n/8 + sblk_y*2
 case 16x16:
      Addr = n/4 + sblk_y*4
 case 32x32:
           Addr = n/4 + (sblk_x/4)*4 + sblk_y*8
endswitch
```

Therefore, the writing positions are reordered to fetch the results of two rows in one clock cycle. The mapping method is shown in the pseudo code in Table 3-2. (sblk_x, sblk_y) are used to represent the position of the 4x4 block within the whole TU, and they can be obtained by the Eq. (3-1).

$$sblk\_x = x \gg 2 \quad sblk\_y = y \gg 2 \qquad (3\text{-}1)$$

where x,y are the horizontal and vertical axes of any coefficient of the current 4x4 block. For the TU 2Nx2N, the range of sblk_x and sblk_y is between 0 and 2N/4 – 1.

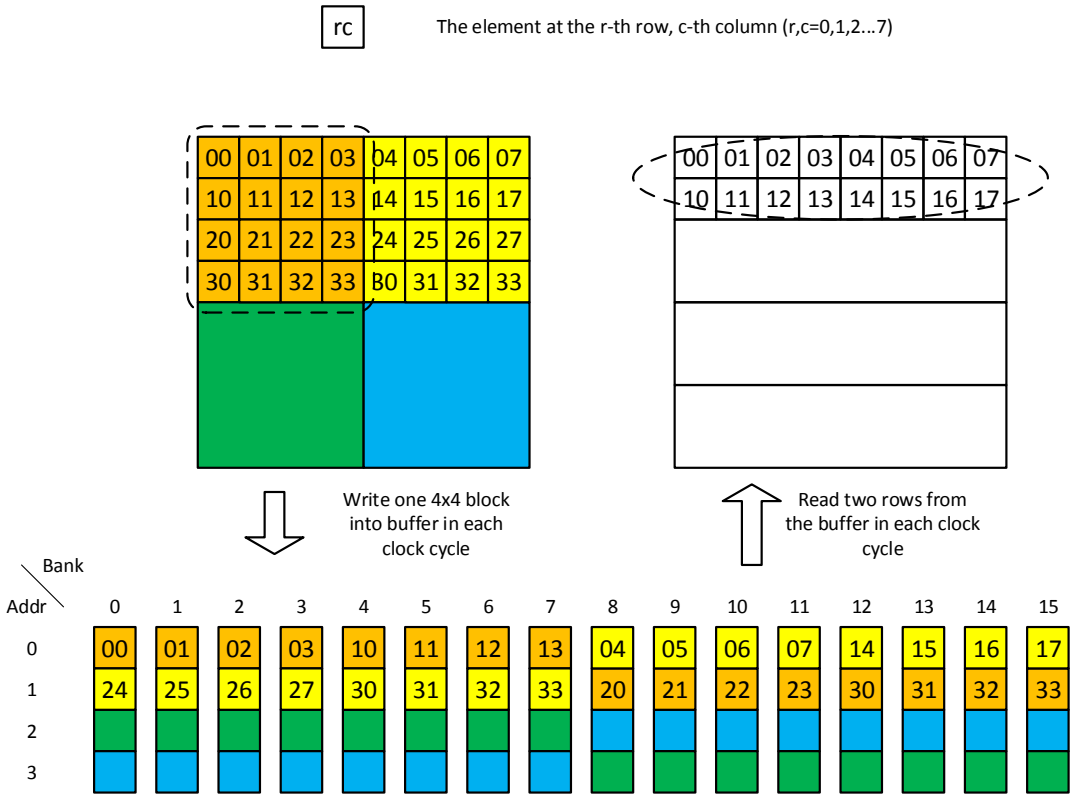

Figure 3-14 Reordered data mapping example for TU8x8 for the QT buffer.

The proposed reordered data mapping method is shown in Figure 3-14. For the 4x4 block at (0,0), DQ[n] (n=0,1,…7) belong to the first two rows of TU8, and they are written in the first eight banks of SRAMs. For the 4x4 block at (1,0), DQ[n] (n=0,1,…7) also belong to the first two rows of TU8, so they are written in the second eight banks of SRAMs. By doing so, 16 DQ results of the first two rows are written in different banks of SRAMs, thus they can be fetched in one clock cycle. For the 4x4 block at (0,1), the SRAM mapping method is the same with that for the 4x4 block at (0,0). The only difference is that the writing address of each SRAM is two larger than that for the 4x4 block at (0,0). For the 4x4 block at the specific (sblk_x,sblk_y), the SRAM and address mapping method for writing is shown in the pseudo code in Table 3-2.

By following the above method, IT1 can read the DQ results of one block/two rows/one row/half row for TU4/8/16/32 in each clock cycle. It is noted that for TU32, only half row is fetched in one cycle. So it takes two clock cycles to fetch the results of one complete row. Registers are used to store the DQ results of half row in the first cycle.

3.3.2 Multiple-Shape Inverse Transform Architecture

The architecture of 4-pixel-parallelsim has been introduced in Chapter 2.2. Since the pixel parallelism for DQ is 16, the pixel parallelism for IT is also 16 in the system. This chapter will give a multiple-shape transform architecture when the pixel parallelism is 16 for all the TU sizes. The 2N-point 1D IT architecture can process one row for TU 2Nx2N. Since the pixel parallelism is 16, four/two/one/half rows are processed for TU4/8/16/32, respectively in each clock cycle. So four/two/one/one copies of 4/8/16/32-point 1D IT architecture are required to support all the TU sizes. By using Chen's algorithm, the architecture for 2N-point 1D IT can be decomposed of the odd part and even part. The architecture of the even part is same as the N-point 1D IT architecture. So we can reuse the even part in 2N-point 1D IT architecture for the N-point 1D IT architecture.

The proposed overall architecture is shown in Figure 3-15 where IT_2N is the architecture for 2N-point 1D IT. In the figure, O8, O16 and O32 mean the odd part of IT_8, IT_16 and IT_32, respectively. The odd parts can be implemented by multiplications and adders, and the detail circuits are shown in the previous chapters. In my proposal, only one copy of IT_32, IT_8 and two copies of IT_4 are required. For TU4, the IT_4 within the IT_32 can process one row, the IT_4 within IT_8 can process another row, and the other two rows can be processed by two individual copies of IT_4. For TU8, the IT_8 within IT_32 can process one row, and the other row can be processed by the individual copy of the IT_8. For TU16, the IT_16 within IT32 can pro-

cess one row. For TU32, IT_32 can process half row in each clock cycle.

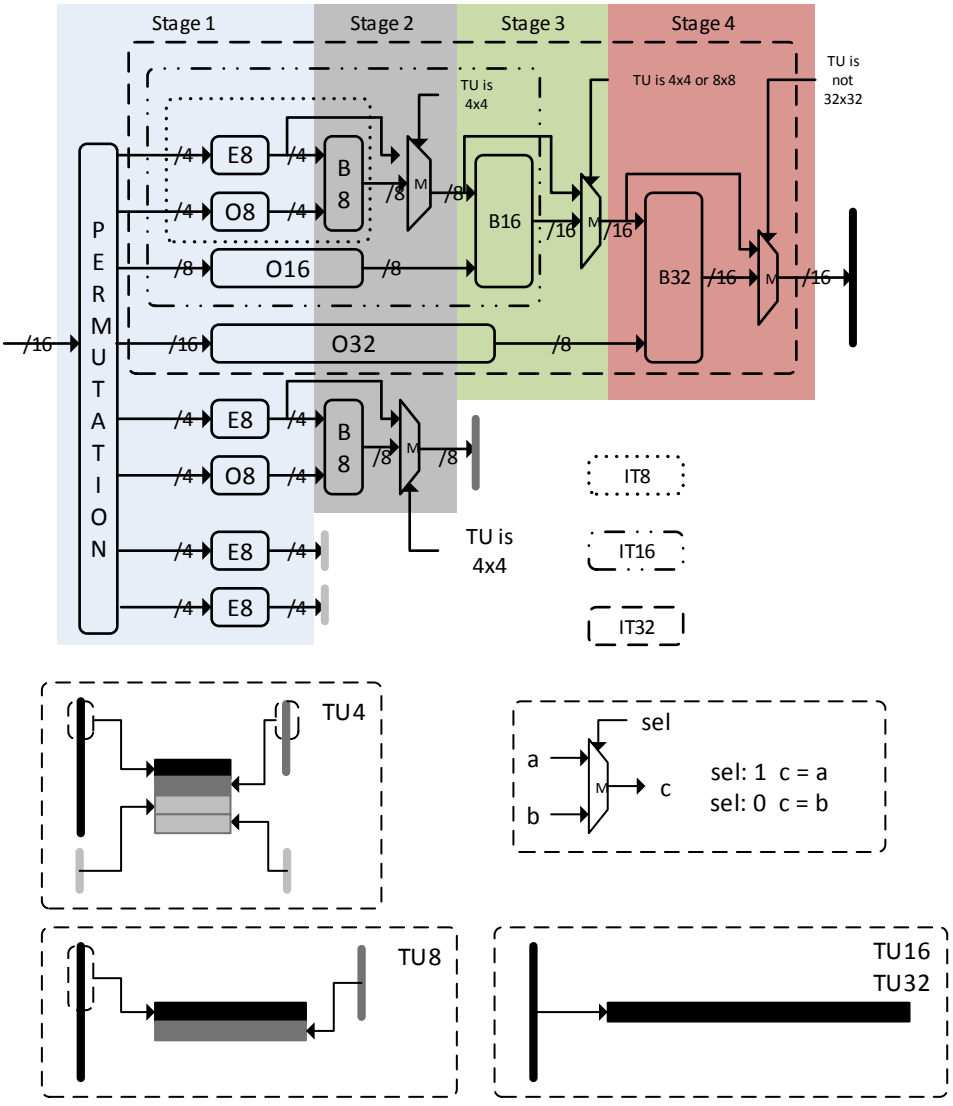

Figure 3-15 Architecture for 16-pixel-parallelism multiple-shape transform.

There are four stages in the design. When the TU size is 4x4, in the first stage, the results of IT_4 can be calculated. In the next three stages, the results of IT_4 are selected by the multiplexer and stored in the registers. It is noted that in order to save the hardware cost of registers, the registers for transferring the results of IT_4 can be shared with O32. It is because the bit-depth in O32 is much larger than IT_4 and O32 is not used when TU size is 4x4. So the registers in O32 can be shared to transfer the results to the fourth stage. When the TU size is 8x8, it needs two stages to generate the results of IT_8. We need some registers to transfer the results to the final stage.

Similarly, the registers in O32 are used to transfer since O32 is not used when TU size is 8x8. When the TU size is 16x16, it needs three stages to generate the results. In the fourth stage, a multiplexer is used to select the results. When TU size is 32x32, it takes four stages to generate the results and output.

It is noted that in the case of intra prediction and luma processing, Discrete Sine Transform (DST) is used for TU 4x4, so the architecture for DST of TU 4x4 is also individually supported.

3.4 Zero Skipping Method for SRAM-based Buffer

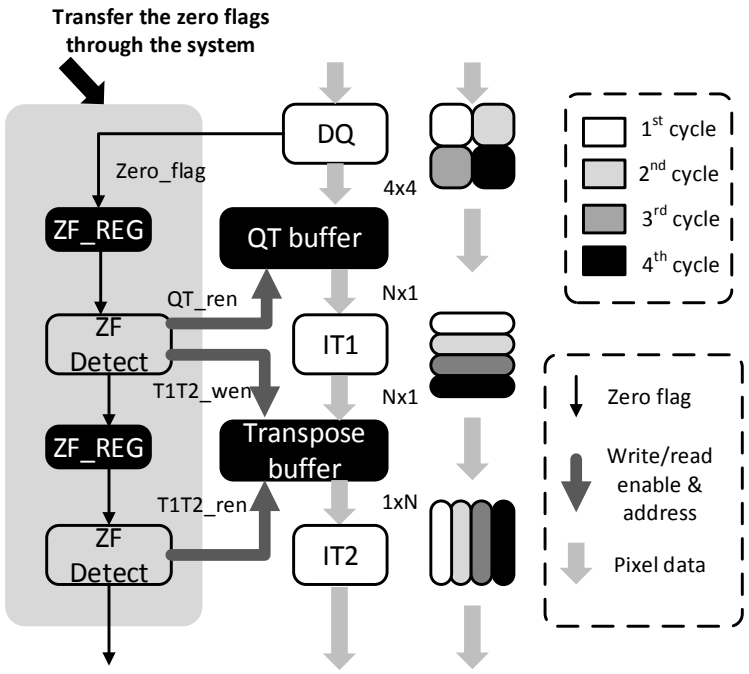

Figure 3-16 Proposed system with zero skipping method.

The structure for the system has been shown in the previous chapter. There are many zero elements in the system. So the methods for detecting zero elements and skipping the write/read operation are presented in this chapter. The proposed system with zero skipping method is shown in Figure 3-16. Originally, there is only one path to transfer the pixel data through the system. DQ writes the pixel data into QT buffer. And then, IT1 reads the pixel data from QT buffer. After IT1 is processed, the results are written

in the transpose buffer. Finally, IT2 fetches the data from the transpose buffer. In the proposed system, one path is created to detect the zero flag by reusing the pixel data, as shown in the left path of Figure 3-16. ZF_REG means the registers to store the results of zero flags in pipelines. ZF Detect aims to detect the zero elements. However, for the different memory operations, the zero skipping patterns are different. The detail zero skipping patterns are presented in the following.

3.4.1 4x1-Row-based Zero Skipping Method for QT Buffer

This section gives zero skipping method for QT buffer. Because the smallest row size is 4 for TU4x4, the zero pattern is defined as 4x1 that is one row of TU4x4. When writing the DQ results of the 4x4 block in the QT buffer, four flag bits are stored in four registers, respectively. Each flag bit can indicate whether each 4x1 row is all-zero or not. The flag bits are stored in the registers. And the size of registers is the same as the depth of the SRAM. So in each clock cycle, the writing address of the 4x1 row is the same as that of the flag bit. An example of the timing diagram is shown in Figure 3-17.

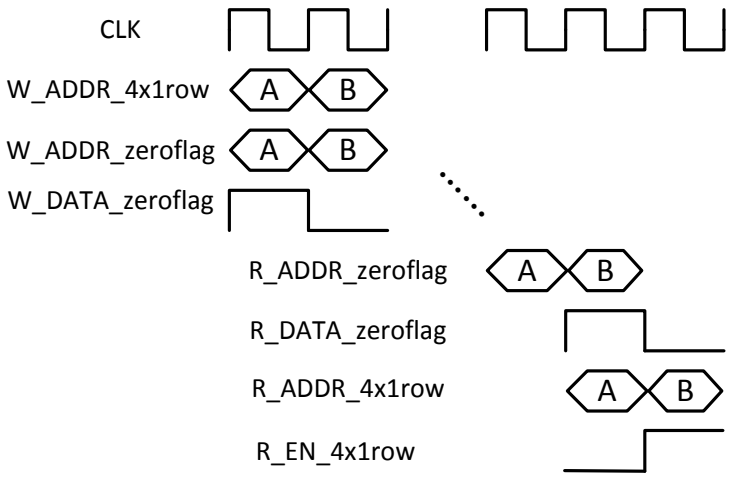

Figure 3-17 An example of the timing diagram for skipping the read operation.

The 4x1 row “A” is all-zero so that a high bit is stored in the corresponding register, while the 4x1 row “B” is not all-zero so that a low bit is stored. When reading

the DQ results from the QT buffer, the corresponding flag bit is read one clock earlier before. The reading address of the flag bit is the same as that of the corresponding 4x1 row. If the flag bit is high which means the corresponding 4x1 row are all-zero, the read for the 4x1 row is disabled. By doing so, the read operation of QT memory can be skipped.

3.4.2 Row-based Zero Skipping Method for Transpose Buffer

For the write operation of the transpose buffer, the all-zero input rows can be detected based on the results of four flag bits. For example, for TU8, two flag bits stand for one 8x1 row. If the two flag bits are both high, the corresponding 8x1 row is all-zero. Since IT1 is processed row by row, if the input row of IT1 is all-zero, the corresponding output row is certainly all-zero. Therefore, the zero flag of the IT1 input row is transmitted to the last stage of the IT1, and disable the corresponding write operation to the transpose memory.

For the read operation of the transpose buffer, the information of all-zero rows cannot be directly utilized because IT1 results are fetched column by column. However, when reading the IT1 results of each column, the read for some banks of SRAMs can still be disabled by the following scheme. Once writing the non-zero IT1 results into the transpose memory, the row number is stored in the register. And the last non-zero row number is written in the FIFO between IT1 and IT2. Take TU16 as an example. If the IT1 results of the below 12 rows are all zero. When reading the IT1 results of one column in each clock cycle, 12 banks of SRAM can be prevented from reading. Only the non-zero data are fetched from 4 banks. In this case, 75% reading activities can be saved.

In addition to saving the power of the transpose memory, the power of the pipeline registers in IT1 can also be saved. When the input row of IT1 is all-zero, there is no need to do the 1D IDCT computations. Considering there are many registers in the

1D IDCT computation especially for large TU, this scheme can save the switching power consumption.

3.4.3 An Example of Zero Skipping Method

This chapter gives an example of zero skipping method for 8x8 in Figure 3-18. For 8x8, four clock cycles are required. The zero flag pattern for QT buffer is 4x1, and four zero flags for four 4x1 rows are detected in the same clock cycle when writing the pixel data in the QT buffer. For the read operation of QT buffer, the pixel data is in the unit of 8x2. In the first clock cycle, the first two rows are read. The corresponding 4x1 zero flags are all 0 so that the four 4x1 rows are all read from the QT buffer. In the second clock cycle, the 3rd-4th rows are read. There are three 4x1 rows that are all-zero. Therefore, the read operations of the three 4x1 rows are disabled. In the third clock cycle, the 5th-6th rows are read. All the four 4x1 rows are all-zero, so all the read operations are disabled. In the fourth clock cycle, the final two rows are read. All the four 4x1 rows are all-zero, so all the read operations are disabled.

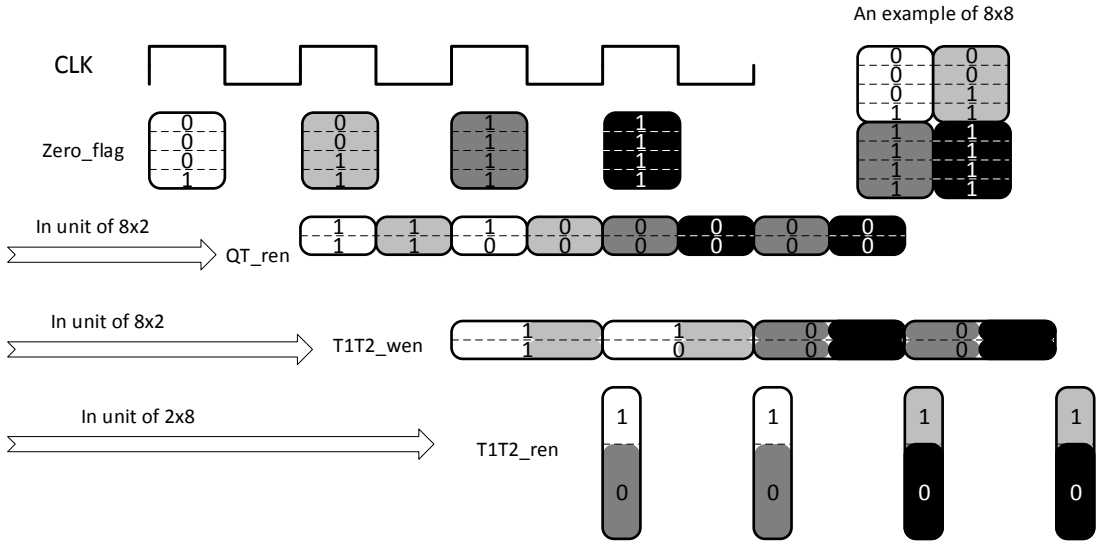

Figure 3-18 An example for zero skipping method of 8x8.

For the write operation of transpose buffer, the pixel data is in the unit of 8x2.

According to four flags of 4x1 rows, the zero 8x1 row can be detected in each clock cycle. In the first clock cycle, the first two rows are written in the transpose buffer. For the first two rows, because both 4x1 rows are non-zero, the write operation is abled. In the second clock cycle, the 3rd-4th rows are written. For the 3rd row, one 4x1 row is zero and the other 4x1 row is non-zero. Therefore, the 3rd row is non-zero and the corresponding write operation is enabled. For the 4th row, both 4x1 rows are zero so it is a zero row. So the write operation is disabled. In the third clock cycle, the 5th-6th rows are zero so the write operation is disabled, so as the final two rows in the final clock cycle.

For the read operation of transpose buffer, the pixel data is in the unit of 2x8. According to the results of zero flags, we can know the final non-zero 8x1 row. In the example, the third row is the final non-zero 8x1 row. From the 4th row, all the 4x1 rows are zero which means that all the rows are zero. Therefore, when reading the data from transpose buffer in the unit of 2x8, the pixels below the 3rd row are not read in each clock cycle. It is noted that for the read operation of QT buffer and the write operation of transpose buffer, the 16 memory operations are all enabled or disabled in one clock cycle. However, for the read operation of transpose buffer, the 16 memory operations are selectively enabled or disabled in one clock cycle.

3.5 Frame-Level Worst Case

3.5.1 Required Clock Frequency for One Frame

The overall proposed system is shown in Figure 3-12. DQ writes the output to the QT buffer from which IT reads the input data. In my design, 16 pixels are processed for IT in one clock cycle, while 16 pixels (a SBLK) are processed for DQ in at most four clock cycles. So the number of processing clock cycles of DQ may be larger than that of IT, thus the pipeline stall may arise. The number of clock cycles will directly in-

fluence the required frequency, so the number of clock cycles of the pipeline stall is calculated at first.

DQ is processed in units of SBLK. For each zero SBLK, the process can be skipped. For each non-zero SBLK, the number of clock cycles can be calculated by the following equation where M is the number of non-zero remaining values in the SBLK.

$$\text{cyc(SBLK)} = \begin{cases} 1, & M \leq 4 \\ 2, & 4 < M \leq 8 \\ 3, & 8 < M \leq 12 \\ 4, & 12 < M \leq 16 \end{cases} \quad \text{(3-2)}$$

After getting the number of clock cycles for each SBLK, the number of clock cycles of the pipeline stall is calculated based on a read/write model. In the model, the read/write of the QT buffer is executed in units of 32x32 which is the largest TU size. For each 32x32 block, the number of clock cycles for IT is 64 (32x32/16), and the number of clock cycles for DQ is the summation of cyc(SBLK) for all the luma and chroma SBLKs because the luma and chroma are processed in serial for DQ, as shown in the equation (3-3).

$$cyc(b32) = \sum cyc(SBLK) \quad \text{(3-3)}$$

Because QT buffer can store three 32x32 blocks, when reading the DQ results of one 32x32 block, the DQ results of the next 32x32 block can also be written simultaneously. A pipeline example is shown in Figure 3-19 where writing the DQ results of block N and reading the DQ results of block N-1 are simultaneous. For the block N, if the number of clock cycles of DQ is not larger than 64, IT can read the DQ results without any pipeline stalls.

In fact, even if the number of clock cycles is larger than 64, the pipeline stall might still be avoided in some cases. Because the QT buffer can store three 32x32 blocks, when DQ results of one 32x32 block are read, DQ results of the next two 32x32 blocks can be stored. Therefore, for each 32x32 block, DQ results can start to be stored in the buffer right after its previous block. So if the summation of the num-

ber of clock cycles for the current and previous 32x32 blocks is not larger than 128, IT can read the DQ results without any pipeline stalls for the current block. A pipeline example is illustrated in Figure 3-20 where cyc(b32) for the block N is larger than 64 while the summation of cyc(b32) for the block N-1 and block N is not larger than 128. In this example, for the block N, IT can read the DQ results without any pipeline stalls.

However, if the summation of cyc(b32) for the block N-1 and block N is larger than 128, the pipeline stall will arise. A pipeline example is illustrated in Figure 3-21. In this example, for the block N, there is a pipeline stall for reading. The number of clock cycles of the pipeline stall is calculated by the equation (3-4).

$$Delay\big(b32(N)\big) = cyc(b32(N)) + cyc(b32(N-1)) - 128 \quad (3\text{-}4)$$

After that, the number of clock cycles of the pipeline stall for one frame can be calculated by the summation of Delay(b32) for all the 32x32 blocks, as shown in the equation (3-5). So the number of required clock cycles for decoding one frame with the pipeline stall is given in the equation (3-6) where PicSizeInSamplesY is the number of luma samples in one frame.

$$Delay(f) = \sum_{all\ the\ 32x32\ block} Delay(b32) \quad (3\text{-}5)$$

$$cyc(f) = \frac{PicSizeInSamplesY}{16} + Delay(f) \quad (3\text{-}6)$$

If one frame is decoded without a pipeline stall, the number of required clock cycles is $PicSizeInSamplesY/16$ and the required frequency is 250MHz for supporting real-time decoding of 8K@120fps. If one frame is decoded with the pipeline stall, the required frequency becomes larger and is proportional to the number of required clock cycles, as shown in the equation (3-7).

$$Freq(f) = \frac{cyc(f)}{\frac{PicSizeInSamplesY}{16}} \times 250MHz \quad (3\text{-}7)$$

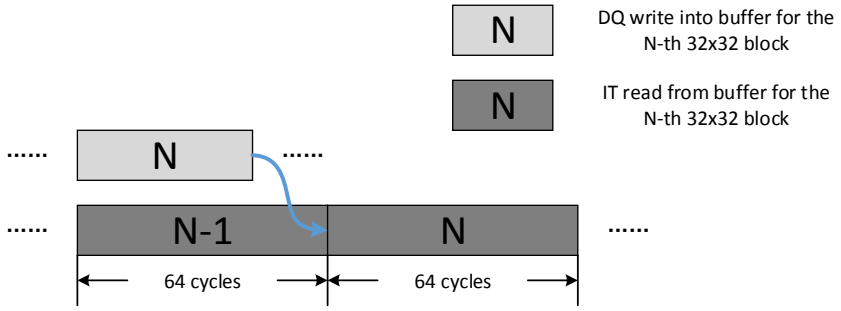

Figure 3-19 A pipeline example without a pipeline stall.

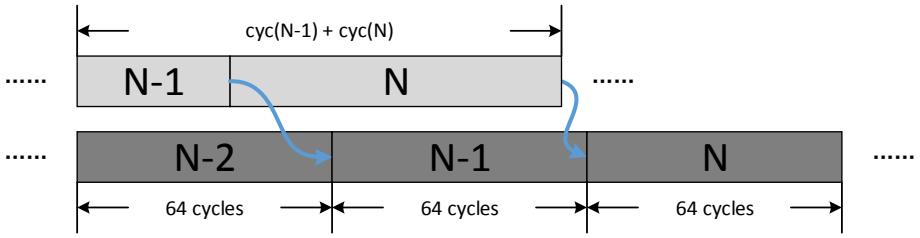

Figure 3-20 A pipeline example without a pipeline stall.

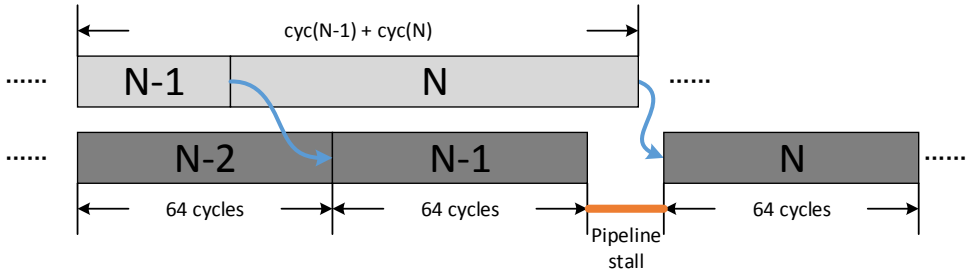

Figure 3-21 A pipeline example with a pipeline stall.

3.5.2 Frame-Level Worst Case at MinCR

For each frame, if more bits are coded, cyc(f) becomes larger and a higher clock frequency is required. So the frame with the most bits is considered as the worst case. I-frame usually consumes much more bits than B-frame, so the first frame (I-frame) with the largest possible number of bits is checked.

HEVC standard [1] defines NumBytesInVclNalUnits which can be regarded as

the number of bytes for each frame. For the first frame, it is defined in the equation (3-8) according to [39].

$$\text{NumBytesInVclNalUnits} \leq \frac{1.5}{MinCR} * (MAX(PicSizeInSamplesY, \frac{MaxLumaSr}{300})) \quad (3\text{-}8)$$

where MinCR is the minimal compression ratio defined in [1], and it is 6 for 8K@120fps. PicSizeInSamplesY is the number of luma samples in one frame, and MaxLumaSr is the maximum luma sample rate. Therefore, the maximum bits for the first I-frame is equal to $\frac{1.5*8}{6} * (MAX(PicSizeInSamplesY, \frac{MaxLumaSr}{300}))$.

In my experiments, 13 test sequences including five 4K and eight 8K sequences are used. For each sequence, the first frames for all the QPs are encoded, and get the first frame whose bits are close to the maximum bits. For decoding each frame with MinCR, the required frequency can be calculated by the method described in the last section.

Assuming that the required frequencies of different sequences are subject to a normal distribution, the required frequencies of 13 test sequences are used as the samples to estimate the mean and the standard deviation of the normal distribution. The normfit function in MATLAB is used to do the estimation and the distribution is shown in Figure 3-22. The estimation results show that the probability for the required frequency being less than 400MHz is well larger than 99.99%. Considering that the samples are the required frequencies in case of MinCR, 400MHz can be regarded as the maximum frequency for decoding one frame in the worst case. In fact, the maximum frequency of my design can achieve 400MHz, while the working frequency is set as 300MHz because it can support the real-time decoding of 8K@120fps at the sequence level in all practical cases. The analysis will be given in the next section.

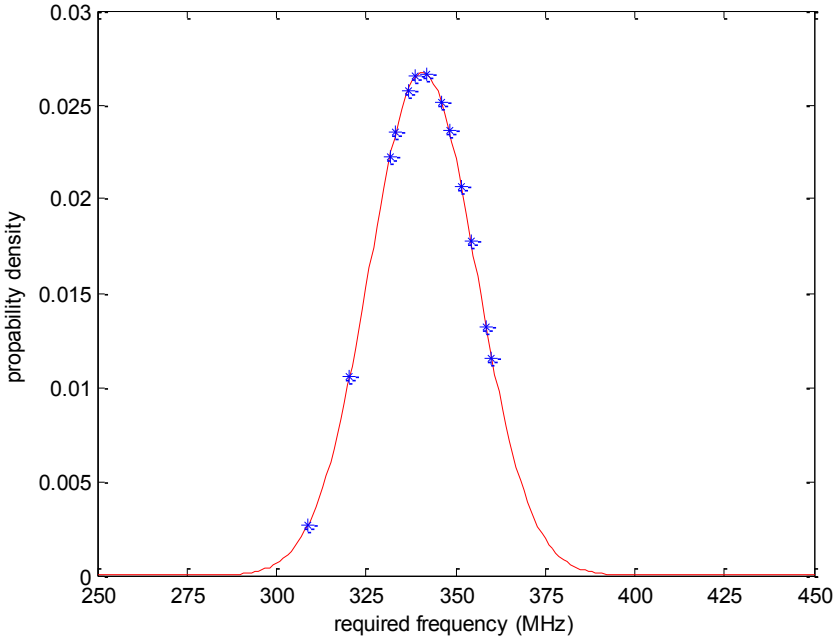

Figure 3-22 Normalized distribution estimation for the required frequency in case of MinCR.

3.6 Sequence-Level Worst Case

The last section analyzes the frame-level worst case in which the frame is compressed in MinCR. However, for each bit stream, the bitrate cannot exceed the maximum bitrate (MaxBR) defined in [1]. So the number of frames with MinCR is limited. The sequence-level worst case in which the bit rate is close to MaxBR will be analyzed in this section.

3.6.1 Artificial Worst-Case Analysis

At first, an artificial worst-case for decoding 120 frames of 8K is shown in Figure 3-23. For the 8K sequence, if one frame is compressed in MinCR, 66355.2k (7680x4320x1.5x8/6) bits are coded. According to the HEVC standard [1], the maximum bitrate (MaxBR) for the level 6.2 (8K@120fps) is 240000 kbps, so there are at most three frames with MinCR in 120 frames. For decoding one frame with MinCR, the required frequency is 400MHz while the working frequency is 300MHz, so

11.11ms ($\frac{1s}{120} \times \frac{400}{300}$) is required. Compared with 8.33ms ($\frac{1s}{120}$) by 400MHz, 2.78ms is delayed for storing the decoded frame into a decoded picture buffer (DPB). In the worst case, the three frames with MinCR are consecutive in the decoding order so that the delay for storing the decoded frames into the DPB becomes the largest. Therefore, at most 8.34ms are delayed for storing the decoded frames into the DPB. For the remaining 117 frames, the overall bit consumptions are 40934.5 (240000-66355.2x3) kbits. The average bit consumptions for each frame are only 349.87 kbits which are quite small. So the required clock frequency will be close to 250MHz. Therefore, 1.39ms ($\frac{1s}{120} \times \frac{300-250}{300}$) can be advanced for storing one decoded frame in the DPB. As a result, after 6 frames, 8.34ms can be advanced and there will be no delay for the subsequent frames. So the system can support real-time decoding at the sequence level even in the artificial worst case, despite a temporary delay of less than 10ms, which can be overcome by a frame buffer between the decoder and display.

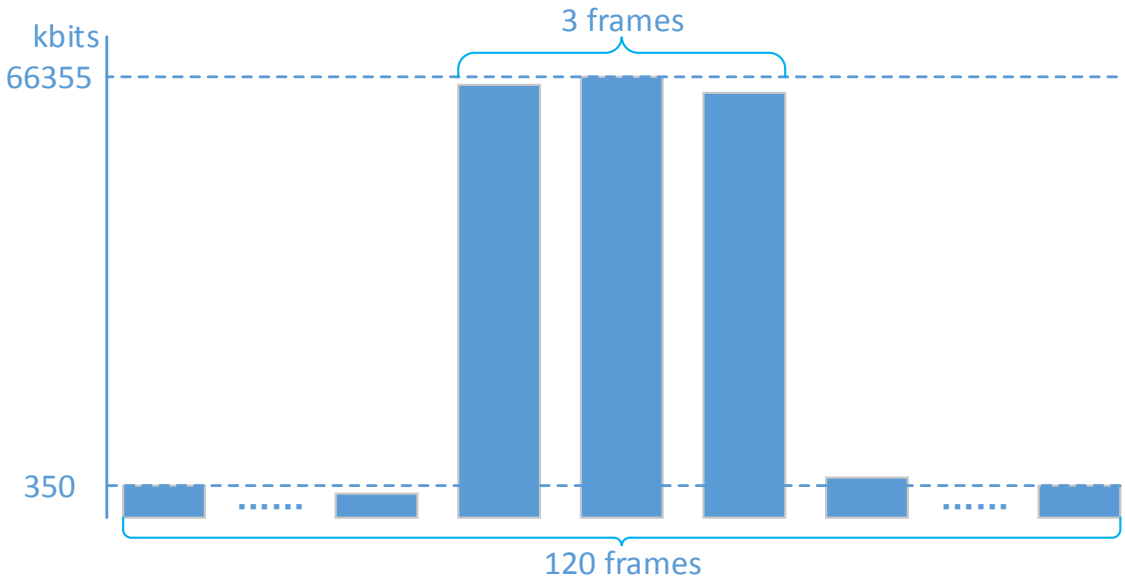

Figure 3-23 An artificial worst-case for decoding 120 frames of 8K.

In addition to the above artificial worst case for normal sequences, the case for white-noise image sequences is also analyzed. Because there is almost no temporal correlation in white-noise image sequences, B-frame can consume as many bits as I-frame, which is similar to the all intra configuration. 240000k bits will be consumed on the 120 frames nearly on average. Therefore, about 2000 kbits are consumed for each frame, thus the compression ratio is about 200. For the frame with such large compression ratio (note the analyzed frame-level worst case was based on MinCR=6), 300MHz is enough for real-time decoding. Therefore, white noise image sequences can also be decoded.

3.6.2 Practical Worst-Case Analysis

In fact, when the bit stream achieves MaxBR, each frame is rarely compressed with MinCR in the practical case. 120 frames for 13 test sequences are encoded under the configuration “Random-access”. For each sequence, all the QPs are traversed to find the bit stream whose bitrate is close to 240000 kbps. It is noted that for the test sequence whose resolution is smaller than 8K or frame rate is lower than 120fps. The normalized bitrate is calculated by the equation (3-9).

$$\text{Normalized bitrate} = \text{bitrate} \times \frac{8K}{resolution} \times \frac{120}{frame\ rate} \quad (3\text{-}9)$$

For the first frame (usually also the frame with the largest number of bits) of the sequence with MaxBR, the compression ratio is shown in Figure 3-24. We can see that the compression ratios are much larger than 6 for all the cases. Therefore, it is believed that in the practical worst case with MaxBR, the frames with MinCR rarely occur. So the proposal can support real-time decoding at the throughput of 8K@120fps in all the practical cases.

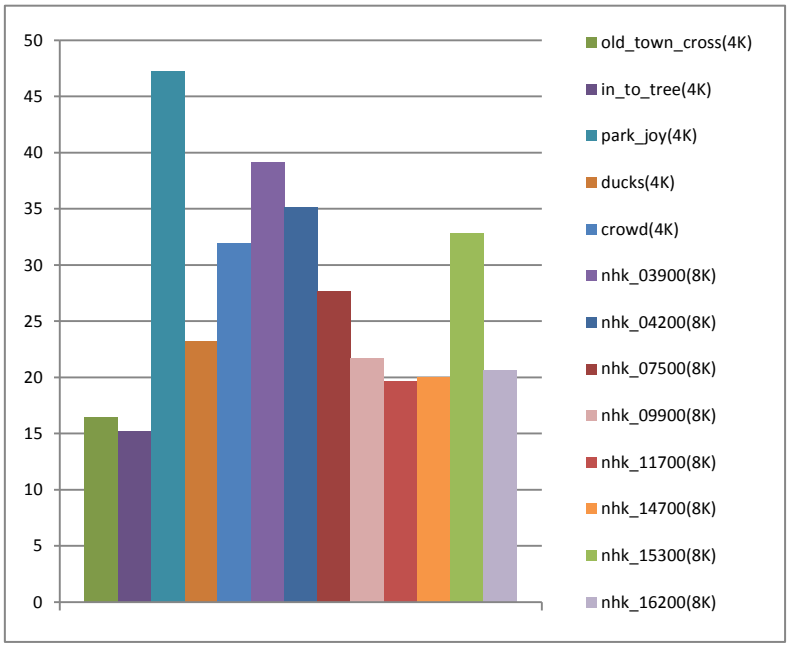

Figure 3-24 The compression ratios of the first I-frames for the bit streams with maximum bi-trate.

3.7 Experimental results

Verilog-HDL is used to implement the proposed design. The design is synthesized with a SMIC 40nm cell library. 16 pixels are processed in each clock cycle for all the components in the system. The operating frequency is 300MHz because it is sufficient for processing 8K videos at 120fps. The results for the DQ, IT and system design are given, respectively.

3.7.1 Experimental Results of Area Consumption

For DQ, 24K gate counts are consumed for the first stage, and 26K gate counts are consumed for the second stage. The comparison of area consumption is shown in Table 3-3. The DQ design in [37] consumed 27.7K gate counts. Since the throughputs of the proposed design and [37] are different, so the normalized area consumption is introduced to do a fair comparison. The normalized area is calculated by the Eq. (3-10).

Normalized area = Gate/(Throughput) (3-10)

Table 3-3 Comparison of area consumption for DQ.

Design	Throughput	Gate counts (K)	Normalized Area by Eq. (3-10) **
[37]	4K@60fps*	27.7	37.11
Proposed	**8K@120fps**	**50**	**8.37**

*The throughput of the whole system in [37] is 4K@30fps, while DQ in [37] can achieve 4K@60fps.

** The unit of the normalized area is 1/(M pixel/s).

In Chapter 3.2.2, a method is given to reduce the number of multipliers from 16 to 4 in each clock cycle. Without this scheme, 67K gate counts are consumed. With all the schemes, 50K gate counts are consumed. Compared with [37], about 77% normalized area consumption can be reduced. The area consumption of the other components is shown in Table 3-4. 176K gate counts are consumed for the 1D IDCT. By using my proposal, one copy of 16-point 1D IDCT architecture, one copy of 8-point 1D IDCT architecture and two copies of 4-point 1D IDCT architectures can be saved. About 20% area consumption can be saved by this method. In my system, IT1 and IT2 do not share the architecture, so two copies of 1D IDCT architecture are required. For luma, IT1 and IT2 consume 352K (176K x 2) gate counts. The rest 71K gate counts are consumed for memory address generator, FIFOs, control parts and pipeline registers. For chroma, IT1 and IT2 consume fewer gate counts compared with luma. Only 120K (60K x 2) gate counts are consumed. It is because TU32 is not supported for chroma. The rest 49K gate counts are also consumed for the same purpose as luma. Totally, 642K (592K + 50K) gate counts are cost for the logical part of the whole system. Compared with [37], about 68% area consumption can be saved for the whole system of the de-quantization and inverse transform, as shown in Table 3-5.

Table 3-4 Gate counts for the other logical components.

	1D DCT x 2	Memory address generator	FIFOs	Pipeline registers and controls	Total
Luma	176x2	16	10	45	423
chroma	60x2	16	6	27	169
Total	**472**	**32**	**16**	**72**	**592**

Table 3-5 Comparison of area consumption for all the logical parts in the system.

Design	Throughput	Gate counts (K)	Normalized Area by Eq. (3-10)
[37]	4K@30fps	125.8	337.04
Proposal	**8K@120fps**	**642**	**107.50**

For the memory part, two buffers are required for one path of luma or chroma. For luma, each buffer is composed of 16 banks of SRAMs, and the depth is set as the capacity of storing three complete TU32 so that the system pipeline will not be stalled. Thus the depth is 192. The widths of each SRAM are 16 bits that can store one coefficient. For chroma, the depth is set as the capacity of storing three complete TU16 thus the depth is 96. The two-port (1R1W) SRAM is used in my design. The memory complier is used to generate the area. For the luma part, the total bits are 49152 bits, and the area consumption is 88921.462 um^2. If registers are used to store the same bits, 235339.78 um^2 are consumed. About 63% area consumption is reduced compared with using registers. For the chroma part, the total bits are 24576 bits, and the area consumption is 56756.32 um^2. If registers are used to store the same bits, 117669.89 um^2 are consumed. About 52% area consumption is reduced compared with using registers.

3.7.2 Experimental Results of Power Consumption

For the memory part, three write/read operations can be skipped. One is to prevent the read operation from the QT memory. Another is to prevent the write operation to the transpose memory. The other is to prevent the read operation from the transpose memory. By using the three schemes, the results are shown in the following four tables. The power consumption reduction is dependent on the input bit stream. The results of different kinds of input bit streams are given. Table 3-6 and Table 3-7 give the results under the configuration of “Random-access” and common QPs. We can see

that about 86% power consumption can be reduced. Table 3-8 and Table 3-9 give the results of the first frames when the compression ratio is 6. We can see that there are only 29% and 34% power reduction because the ratio of zero elements is small in the frames compressed by a small compression ratio. It is noted that for the logical part, the switching power can also be reduced by preventing the switching activities of the registers in the pipelines.

Table 3-6 Power consumption of memory part for *Cactus* under the "Random-access" and QP 27.

Scheme	w/o skipping	Q_T_C	Q_T_C + T1_T2_P	**With all the skipping**
Luma	45.08	31.81	18.24	**6.47**
Chroma	21.85	15.51	8.82	**2.81**
Power (mW)	66.93	47.32	27.06	**9.28**

Table 3-7 Power consumption of memory part for *PeopleOnStreet* under the "Random-access" and QP 32.

Scheme	w/o skipping	Q_T_C	Q_T_C + T1_T2_P	**With all the skipping**
Luma	45.54	32.06	18.27	**6.60**
Chroma	22.27	15.88	9.09	**2.85**
Power (mW)	67.81	47.94	27.36	**9.45**

Table 3-8 Power consumption of memory part for *Cactus* of the first frame with compression ratio 6.

Scheme	w/o skipping	Q_T_C	Q_T_C + T1_T2_P	**With all the skipping**
Luma	42.06	38.31	35.82	**33.98**
Chroma	18.88	15.59	12.29	**9.36**
Power (mW)	60.94	53.9	48.11	**43.34**

Table 3-9 Power consumption of memory part for *PeopleOnStreet* of the first frame with compression ratio 6.

Scheme	w/o skipping	Q_T_C	Q_T_C + T1_T2_P	**With all the skipping**
Luma	40.40	35.73	32.49	**29.75**
Chroma	19.10	15.86	12.73	**9.93**
Power (mW)	59.5	51.59	45.22	**39.68**

* Q_T_C is the skip for the read operation of the QT buffer.

T1_T2_P is the skip for write operation of the transpose buffer.

Table 3-10 Power consumption for the components.

Component	DQ	IT_luma	IT_chroma	Memory	Overall
Power (mW)	4.90	35.49	14.46	9.37	54.85

Table 3-11 Power consumption comparison with previous work.

Design	Throughput	Power (mW)	Normalized power by Eq. (3-11)
[37]	4K@30fps	7.8	31.35
Proposal	**8K@120fps**	**54.85**	**13.77**

The power consumption for the other logical parts is shown in Table 3-10. For the memory part, the power consumption is the average value of the power consumptions in the common test condition such as “random-access”. Overall, the power consumption is 54.85 mW. The normalized power consumption is obtained by the following equation. Compared with reference [37], the normalized power consumption can be reduced by 56% as shown in Table 3-11.

$$\text{Normalized power} = \text{power} * 10^9 / (\text{Throughput}) \quad (3\text{-}11)$$

3.8 Chapter Summary

In this chapter, a low-cost system architecture of de-quantization and inverse transform is presented. Firstly, for the de-quantization, the coefficient is decomposed to two parts. One part is base part (baseLevel) whose value is not greater than 3 thus the multiplication with scaling parameter can be replaced by LUT. The other part is remaining part and the number of positions with non-zero remaining values is usually not greater than 4 within one 4x4 block. So the number of required multipliers is reduced from 16 to 4. Four multipliers can be reused in different clock cycles. Secondly, a system with zero skipping method is created. The zero elements are detected by reusing the pixel data. After detecting the zero elements, the read/write memory operation is skipped in order to save the power consumption. As a result, overall, for the logical part, 68% normalized area consumption can be reduced compared with the

previous work. For the whole system, 56% normalized power consumption can be reduced compared with the previous work. For the de-quantization, the proposed architecture can save 77% area consumption compared with previous works. For the memory part, 29%-86% power consumption can be saved compared with not using the zero skipping method.

4. Fast Prediction Unit Depth and Prediction Mode Selection Algorithm for HEVC Intra Prediction

This chapter[3] will reduce the number of intra modes requiring R-D cost calculation. The position of this chapter is corresponding to the fast RDO in mode decision as shown in Figure 4-1. The algorithms can be classified to software-oriented and hardware-oriented. Since my final target is the hardware implementation, so my research target is hardware-oriented. The major concept of the idea in this chapter is to create a simplified cost model and reuse the results of calculated costs. This chapter is related with the publication [3] and [12] in Page 122. Most of the previous hardware-oriented works are based on the sequence features such as edge and gradient information. The conceptual difference of my method is based on a proposed a low-complexity cost model rather than sequence features.

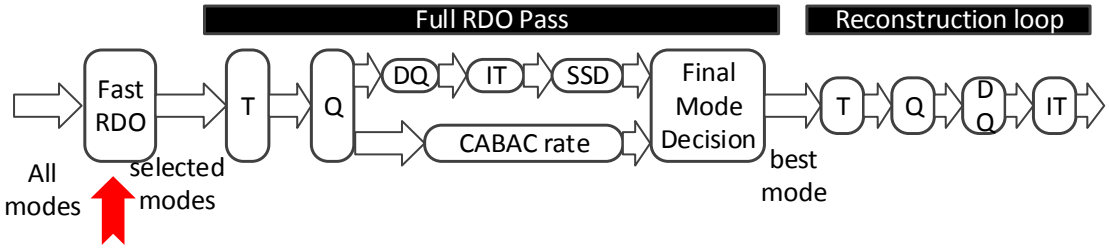

Figure 4-1 The position of Chapter 4 in the mode decision.

[3] This chapter is related with the publication [3] and [12] in Page 122.

4.1 Introduction

4.1.1 Overview of Mode Decision for Intra Prediction in HM

HEVC intra PU and prediction modes are ready shown in Figure 1-3 and Figure 1-4. There are five PU depths and 35 prediction modes. In HEVC Test Model (HM), for the intra prediction, not all the prediction modes require the R-D cost calculation. Some prediction modes are filtered by a fast RDO. The numbers of selected prediction modes are different for various PUs as shown in Table 4-1. For the PUs larger than 8x8, 3 prediction modes are selected. For the other PUs, 8 prediction modes are selected. The selection is based on SATD cost.

Table 4-1 Number of intra modes and candidate modes supported for various PUs.

PU depth	# of intra prediction modes	# of prediction modes requiring R-D cost
4x4	35	8
8x8	35	8
16x16	35	3
32x32	35	3
64x64	35	3

In addition, the most probable mode (MPM) derived from the intra modes of the above and left neighboring blocks will also be added into the candidate modes. After selecting the best prediction by comparing the R-D cost, the residual pixels can be generated. For the residual, all the TU combinations will be traversed to code the residual. However, the TU splitting will not contribute too much to the coding efficiency for the intra prediction. According to [40], when the residual quadtree (RQT) is turned off, the average performance loss is only 0.4%. Therefore, in the intra mode decision, the most important issue is to choose the best combination of PU and prediction mode rather than the RQT. Although HM already reduces the number of PUs

and prediction modes requiring R-D cost calculation, the remaining number is still excessive. So there are many algorithms in order to reduce the number of PU depths or prediction modes requiring R-D cost calculation.

4.1.2 Previous Works

From H.264, there are extensive researches focused on fast RDO. For example, a directional field based approach was reported by Pan, et al. [41] in which prediction modes are reduced according to an edge direction texture histogram. Based on that, Wei, et al. [42] further proposed a method to reduce the computation overhead for edge detection. Kim, et al. [43] suggested a method based on a multi-stage sequential mode decision process to filter out unlikely candidate modes. Huang, et al. [44] proposed a variance-based algorithm for block size decision and an improved filter-based algorithm for prediction mode decision. Tian, et al. [45] proposed a fast block type decision algorithm based on the entropy feature.

However, most of the above algorithms for fast mode and block size decision were designed for H.264/AVC, which cannot be directly applied in HEVC due to the distinctions in block types and angular prediction modes. Several latest proposals have been reported to reduce the complexity of HEVC intra prediction. [7] developed the relationship between texture and R-D cost for depth selection and use hadamard transform rather than DCT for PU mode selection. Chen et al. [51] executed chroma prediction in advance to generate a depth map, and thereby reduced the complexity of luma prediction. Wallendael et al. [52] exploited the neighboring intra modes to obtain a prioritization of the different modes. Similarly, Zhao et al. [53] made use of neighboring prediction information to eliminate unlikely angular modes. Jiang et al. [54] calculated the gradient directions and generated a gradient-mode histogram to do the fast mode decision. Motra et al. [55] used direction information of the co-located neighboring block of previous frame along with neighboring blocks of current frame

to speed up intra mode decision. Silva et al. [56] calculated the predominant orientation of the edge of the current PU pixels to reduce the number of evaluated angular modes. Zhang et al. [57] presented a gradient-based fast decision algorithm for prediction unit size selection and mode selection. Xiong et al. [58] used the non-normalized histogram of oriented gradient (n-HOG) to select CU size. Gweon et al. [59] can reduce the computational complexity by using the signaling information coded_block_flag (cbf) in the HEVC syntax. If cbf is zero after encoding a PU partition, the next PU process of encoding the CU can be eliminated. Gweon et al. [59] can only be used in P or B slices. And it has already been included in HEVC Test Model (HM) 7.0. Yao et al. [60] used the bit number difference between the most probable mode (MPM) and non-MPM to do the early decision for the intra mode selection. Wang et al. [61] exploited the Otsu's method to calculate the texture complexity of the largest coding unit and then filter some coding units. Liu et al. [62] selected some coding units based on the texture complexity, and then selected the modes for each unit based on the texture direction. Park et al. [63] used the temporal correlation among frames to skip some unlikely units. In [64], all the modes are clustered in K groups for the mode selection. Shang et al. [65] utilized the depth information of neighboring units and the mode information of larger units to do the unit and mode selection. Lim et al. [66], Kim et al. [67] and Cho et al [68] presented a fast coding unit partitioning algorithm based on Bayesian decision rule. In [69], some homogenous coding units are terminated in the first step, and then the linear support vector machines (SVMs) are used to do the early coding unit split and termination decision. Liu et al. [70]-[71] and Yu et al. [72] are creative works which used the convolution neural network to do the mode decision. Min et al. [73] calculated the global and local edge complexities in four directions to decide the coding unit partition. Chen et al. [74] presented a fast mode and depth decision algorithm based on edge detection and reconfiguration. Zhao et al. [75] detected the dominate direction of the coding unit by calculating the discrete cross differences and then select some modes. Gweon et al.

[76] terminated the PU partitions when there exists a PU with coded block flag (CBF) being 0. Hu et al. [77] changed the intra mode selection problem to a Bayesian decision problem. Shen et al. [78] used the information in spatially nearby coding units and different depth levels to skip some specific depths. Tseng et al. [79] studied the statistical relationships of coding unit to the standard deviation of the pixels in the largest coding unit and R-D cost and then conducts the depth and prediction mode selection. Chen et al. [80] developed two kinds of classifiers. One is to categorize the depth to split, indistinct and non-split. After that, for the indistinct category, another classifier is used to distinguish the split or non-split. Hu et al. [81] terminated the splitting by a logistic regression classifier. In order to obtain the coefficients for the logistic regression classifier, an offline training scheme is developed. Kim et al. [82] exploited some key points in the picture such as the difference of Gaussian and then do the CU depth decision. Geuder et al. [83] developed a descriptor to investigate the pixel pair differences and then the number of differences is compared with the threshold to do the block size decision. Shang et al. [84] used the depth information of neighboring CUs to early terminate the CU splitting and used the correlations between the prediction mode of higher depth and current depth to terminate some prediction modes. Radosavljević et al. [85] detected the local image characteristics and use the histogram matching of two neighboring CU depths to do the split decision. Du et al. [86] utilized the random forests to decide whether the current CU depth should be skipped or terminated. Rhee et al. [87] developed the relationship between the forward and backward directional prediction and skip the backward prediction when the minimal cost of forward directional prediction and merge mode is smaller than a threshold. Goswami et al. [88] terminated the unit splitting based on the results of R-D costs of the higher and current levels. Shen et al. [89] extracted some features and did the CU level decision based on the Bayesian decision rule. Shen et al. [90] filtered some depths which are rarely used in the previous frame and neighboring units. Huang et al. [91] filtered some depths by utilizing the depth information of spa-

tial and temporal units. Different from the other works, Jiang et al. [92] took the motion consistency into consideration for the CU depth decision.

Many good works on fast RDO have been introduced in the above. In fact, they can be classified into software-oriented algorithm and hardware-oriented algorithm. For the hardware-oriented algorithms, there are two properties. One is to achieve the block-level complexity reduction. In HEVC, the largest coding block is 64x64, so a stable complexity reduction for 64x64 should be achieved. For example, [68] required updating the parameters periodically. For the frames with parameter updating phase, the encoding is conducted as origin. Therefore, the complexity reduction cannot be ensured for each 64x64 block. The second property is that the algorithm should be convenient to be implemented in hardware. Some intricate mathematical models such as Bayesian decision rule in [66]-[68] can contribute to the precise estimation accuracy for the rough mode decision. However, Bayesian decision rule is not friendly for the hardware implementation due to the high computation. In the previous works, [7], [56]-[58], [70], [72], [74] and [75] satisfy the above two conditions for being the hardware-oriented methods and they are very representative works. [57], [70] and [72] mainly focused on the reducing the number of R-D cost calculations by filtering the PU depths. [56], [74] and [75] reduced the complexity by reducing the number of prediction modes within each depth.

4.1.3 Research Target

My research target is the hardware-oriented algorithm. In the previous hardware-oriented works, there are still several problems. In [57] and [75], the performance loss is so large that the BD-bitrate is more than 3%. [70]-[72] are the latest works focusing on reducing the number of depths. The works are quite creative and full of novelty. However, the computation of convolutional neural network is still a little bit high. [74] outperformed [56] in terms of both coding efficiency and encoding

time reduction. However, the PU depth selection's contribution to the complexity reduction is a little bit limited. In my method, the target is to reduce the number of PU depths need full RDO. The difference with the previous work is that all the previous works use the image information such as edge, gradient and texture while my method is based on a proposed simplified cost model.

4.2 Proposed Algorithm for PU Depth Selection

4.2.1 Proposed Low-Complexity Cost Model

Although the R-D cost is nearly optimal, intra mode decision involve huge computational complexity, making it unsuitable for pre-processing. It is therefore useful to start with a simpler Hadamard cost function and reduce its complexity by using two schemes.

Firstly, original pixels rather than reconstructed pixels can be utilized to compute the sum of the absolute transformed differences (SATD). In the pre-processing part, the mode decision part has not been finished, so the reconstructed pixels cannot be used. By using the original pixels, the estimated SATD cost can be calculated to support the proposed PU size and mode selection algorithm, which is beneficial to the complexity reduction. Using original pixels can also eliminate the reconstruction process and reduces critical data dependency in the reconstruction loop, which makes implementation of parallel design for hardware or software easier.

The HEVC behavior uses the reconstructed pixels of the neighboring PUs as the neighboring pixels to compute the SATD for the current PU. In my proposal, after the pre-processing, some PUs (depths) and modes can be filtered. For the remained PU depths and modes, the reconstructed pixels are also used to do the fine-processing including the R-D cost calculation.

An SATD cost vector (CV_{SATD}) is employed to calculate the SATD of each PU.

Each CV_{SATD} is composed of at most 35 elements and each element is the estimated SATD for each mode. After it obtains the CV_{SATD} of a PU, the minimum element estimates that PU's minimum SATD.

The second scheme involves reducing the number of SATD calculations. The CV_{SATD} of only the 8 × 8 PU are obtained by means of complete calculations. After this, rather than calculating the CV_{SATD} for the larger PUs, the existing 8 × 8 CV_{SATD} is used to estimate these values. For instance, the CV_{SATD} of a 2N × 2N (N = 8, 16) PU can be estimated by summation of the CV_{SATD} of the leaf 8 × 8 PUs. Eq. (4-1) and (4-2) show the estimation method, j means the specific larger 2N × 2N (N = 8, 16) PU while i represents the corresponding specific leaf 8 × 8 PU. Considering that the 64 × 64 depth cannot bring any gain, this depth is split without any processing. So there is no need to compute CV_{SATD} for PU 64 × 64.

$$\overrightarrow{CV_{SATD32}^{J}} = \sum_{i=0}^{15} \overrightarrow{CV_{SATD8}^{i+16*J}} (j = 0,1,2,3) \quad (4\text{-}1)$$

$$\overrightarrow{CV_{SATD16}^{J}} = \sum_{i=0}^{3} \overrightarrow{CV_{SATD8}^{i+4*J}} (j = 0,1,2,\dots,15) \quad (4\text{-}2)$$

After applying the estimation, the SATD cost can be rewritten as:

$$J_{HAD}=SATD'+\Delta E +\lambda \cdot R \quad (4\text{-}3)$$

Here SATD' is the estimation to SATD with ΔE being the estimation error. Note that SATD' is usually smaller than SATD since the former is from the prediction based on small blocks which utilizes reference pixels nearer and therefore more likely to be closer to the current pixels. As a result, ΔE tends to be positive. In additional, ΔE increases with PU size 2N, since the estimation of SATD' is performed from lower 8x8 depth to higher 2Nx2N depths, while the error also accumulates.

In (4-3), λ is closely related to *QP*. In deciding the mode between PU depths, R indicates coding bits that involve information for PUs of size 2N, as larger PUs involve less mode information to be written to the bit rate**.**

Considering the strong relations between ΔE and 2N, and between $\lambda \cdot R$ and 2N together with QP, ($\Delta E + \lambda \cdot R$) can be estimated as a function of 2N and QP, so that:

$$J_{HAD} \approx SATD' + f(QP, 2N) \quad (4\text{-}4)$$

In HM, whether a 2Nx2N is divided into four NxN or not is determined by the cost of adjacent PU depths recursively. Based on approximate Hadamard based cost shown in equation (4-4), the cost of split or not for a 2Nx2N is shown as follows:

$$J_{non-split} \approx SATD'_{2N} + f_{2N}(QP, 2N) \quad (4\text{-}5)$$

$$J_{split} \approx \sum\{SATD'_N + f_N(QP, N)\} \quad (4\text{-}6)$$

where $\sum$ means the sum of $J_{HADAMARD}$ for individual split NxN. $SATD_{2N}' - \sum SATD_N'$ is defined as ΔC_{2N}, and then the determination condition for splitting 2Nx2N is shown as follow:

$$\begin{cases} nonsplit\ 2Nx2N, & \Delta C_{2N} < \sum f_N(QP,N) - f_{2N}(QP,2N) \triangleq T(QP,2N) \\ split\ 2Nx2N, & \Delta C_{2N} > \sum f_N(QP,N) - f_{2N}(QP,2N) \triangleq T(QP,2N) \end{cases} \quad (4\text{-}7)$$

where T(QP,2N) is considered as a threshold, which is a function of QP and PU size 2N.

4.2.2 Training Method for Obtaining Thresholds

Theoretically, the optimal threshold can be obtained by training as shown in Figure 4-2. The threshold combination with best coding efficiency should be the optimal one. However, for each round of the training process, a complete encoding process has to be gone through which is computationally intensive. In addition, a total of 2 thresholds are involved, which results in huge number of combinations of thresholds. This may not be reasonable in practical use.

Here a fast training method is presented. The original criterion for the optimal thresholds is to minimize the rate distortion. Alternatively, the target is at the closest decision results as HM. By taking HM's decision on whether one 2Nx2N depth is split or not as the "correct" choice, the aim is to get a set of thresholds that minimize the "error" rate. For this purpose, a data statistic component is incorporated into HM for offline training of ΔC_{2N} for the split determination of each 2Nx2N depth, and the

corresponding determination results. Figure 4-3 plots the normalized probabilities (NP) of split and non-split CUs under different ΔC_{2N}. Normalized probability (NP) is plot for each ΔC_{2N} value by using:

$$\mathrm{NP}_{\mathrm{split}}(\mathrm{delta\_cost})=\frac{M_{split}(\mathrm{delta\_cost})}{\sum_{i=0}^{\infty}M_{split}(i)} \tag{4-8}$$

$$\mathrm{NP}_{\mathrm{nonsplit}}(\mathrm{delta\_cost})=\frac{M_{nonsplit}(\mathrm{delta\_cost})}{\sum_{i=0}^{\infty}M_{nonsplit}(i)} \tag{4-9}$$

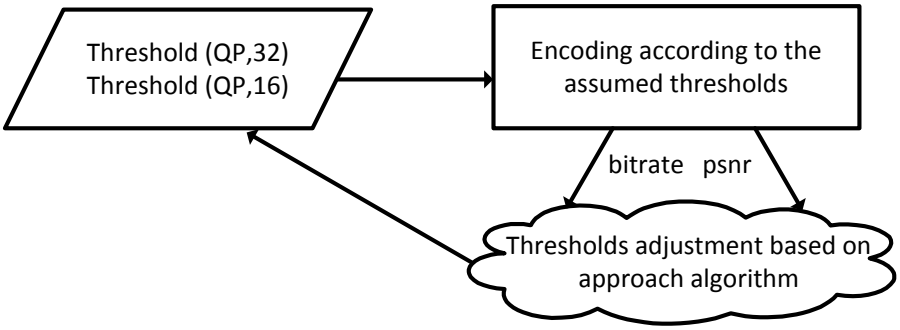

Figure 4-2 Theoretical method to get the best thresholds.

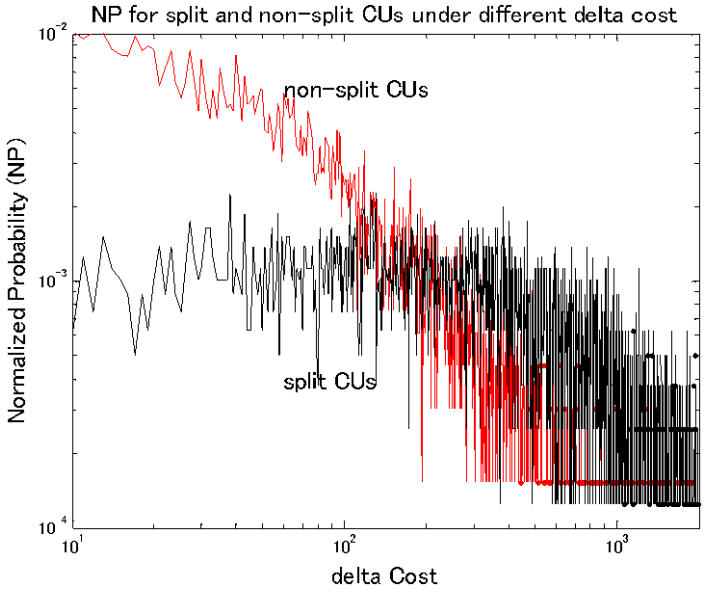

Figure 4-3 NP curve for sequence *BQMall (QP=32,2N=16,"intra main")*.

M_{split} (i) is the number of 2Nx2N PUs which has ΔC_{2N} equal to i and takes the "split" decision in HM. Correspondingly, $M_{nonsplit}$ (i) is the number of 2Nx2N PUs

which has ΔC_{2N} equal to i and takes the "nonsplit" decision. When ΔC_{2N} is small, there are many PUs sampled for each ΔC_{2N} value, so the statistical dispersion of NP_{split} and $NP_{nonsplit}$ is small, which results in a relatively smooth interval of the curve. On the contrast, when ΔC_{2N} grows higher, the number of PUs for each ΔC_{2N} becomes much smaller. It is the lack of statistical samples that leads to the severe variations of the curves. In spite, the trends of the curves are still quite clear to support the proposed model.

From Figure 4-3, it can be seen that CU is more likely to be non-split for small ΔC_{2N}, while the NP of being split increases as ΔC_{2N} increases. These trends conform to the assumed decision mechanism described in (4-7). The combined error rate of the conditional probabilities P(split|ΔC_{2N}) and P(non-split|ΔC_{2N}) can be defined as:

$$E(T) = \int_{\Delta C_{2N}=0}^{T} P(split \mid \Delta C_{2N}) + \int_{\Delta C_{2N}=T}^{\infty} P(non-split \mid \Delta C_{2N}) \qquad (4\text{-}10)$$

where E denotes error rate and T is the threshold.

Therefore, the threshold which can satisfied with $\frac{dE}{dT} = 0$ and $\frac{d^2E}{dT^2} > 0$ is the best threshold which can also be regarded as the intersection of two curves.

Then for each 2Nx2N PU, if ΔC_{2N} is larger than the off-lined trained threshold, it is considered that it should split to NxNs. Otherwise, 2Nx2N PU size is suitable for encoding this 2Nx2N depth.

Following this approach, for each sequence and QP value in the training set, only one encoding iteration is required to get a whole set of thresholds. Although this may not ensure the optimum results as the method in Figure 4-2 provides, the complexity is much more reasonable.

4.2.3 Summary of Proposed PU Depth Selection Algorithm

Generally, there are 4 CU depths from 64x64 to 8x8. For depth of 8x8, there are two kinds of PUs which are 8x8 and 4x4. The filtering judgment for each depth can be

obtained according to the decision mechanism in (4-7), based on ΔC_{2N} acquired in the fast pre-processing (FP) stage together with already offline trained T(QP,2N). However, the FP system is based on fast Hadamard cost, while the optimal criterion is R-D cost. Therefore, only some unfiltered PUs in FP system are required to calculate R-D cost to make the final decision of coding structure.

If 2Nx2N depth is not filtered in FP, 2Nx2N PU is likely to be suitable for predicting the current CU depth in R-D cost based criterion, so that it is required to calculate the R-D cost of this 2Nx2N PU. Considering the risk of missing the best coding structure generated by smaller PUs, the R-D cost generated by the smaller PUs is used to compare with the R-D cost of 2Nx2N PU. The R-D cost by the smaller PUs is defined as the sum of R-D cost of four split NxN PUs. By the comparison of the R-D costs of the neighboring 2 PU depths, the branch structure of the current 2Nx2N is constructed as either itself or the four splits of it. This situation is shown in the right case in the Figure 4-4.

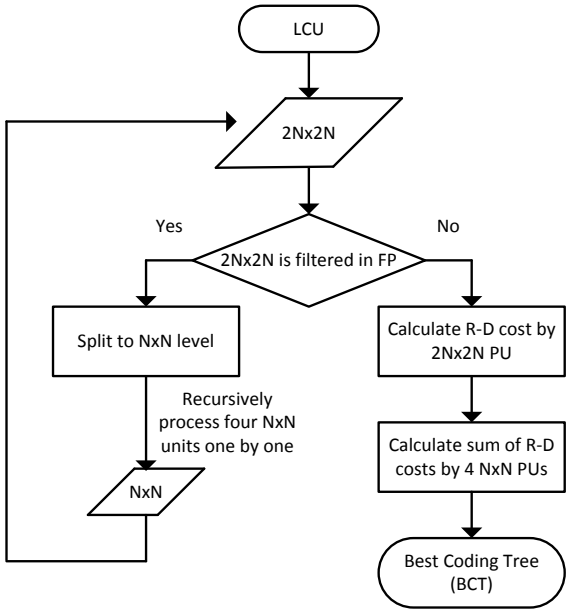

Figure 4-4 Prediction Depth selection algorithm.

Otherwise, 2Nx2N PU is decided to be filtered in FP, the fine processing including R-D cost for this PU is eliminated and directly split to the lower NxN and process four NxN PUs recursively, which is shown in the left case of the Figure 4-4.

For one LCU, all the filter-flag for each depth can be achieved by the FP. Then the process is from largest CU to smallest CU. Starting from LCU (64x64), each 2Nx2N depth is processed recursively as Figure 4-4 shows. The final structure by this PU size selection scheme is still based on the generic quadtree. However, only two neighboring depths need detail R-D cost calculation to get each leaf, while all the five PU depths has to be gone through in HM.

4.3 Proposed Algorithm for Prediction Mode Selection

After doing the PU depth selection, some unlikely R-D cost calculations can be filtered. However, for some remaining PU depths such as 32x32, 16x16 and 8x8, all the 35 intra modes have to be gone through by calculating Hadamard-based cost to select the candidate modes, as shown in Table 4-1. It will cost large computation. Therefore, the target is to reduce the number of modes supported for Hadamard-based cost computation.

In the process of the pre-processing, CV_{SATD} can be obtained for each 2Nx2N PU (N=4, 8, 16) according to the equation (4-1) and (4-2). Despite the element of CV_{SATD} represents the SATD' by original pixels quickly, it can still be utilized to reflect the relative value of precise Hadamard cost in the case of different modes. The modes having a smaller SATD' are mostly likely to incur smaller Hadamard cost. In order to reduce the complexity of this approach, the modes with the smallest SATD' are used as candidates. In this way, the precise Hadamard calculation stage can be eliminated. The number of candidate modes left by the mode selection is 8, 3 and 3 for 8x8, 16x16 and 32x32 PU, respectively. Since no CV is calculated for 4x4 PU in preprocessing, the mode selection process does not apply to 4x4 PUs.

It has to be noted that the most probable mode (MPM) is not ignored in the proposals. After 3 or 8 modes are selected at first by the mode filtering algorithm, MPMs are also added for the final R-D cost evaluation. The flowchart of this proposed

method is shown in Figure 4-5.

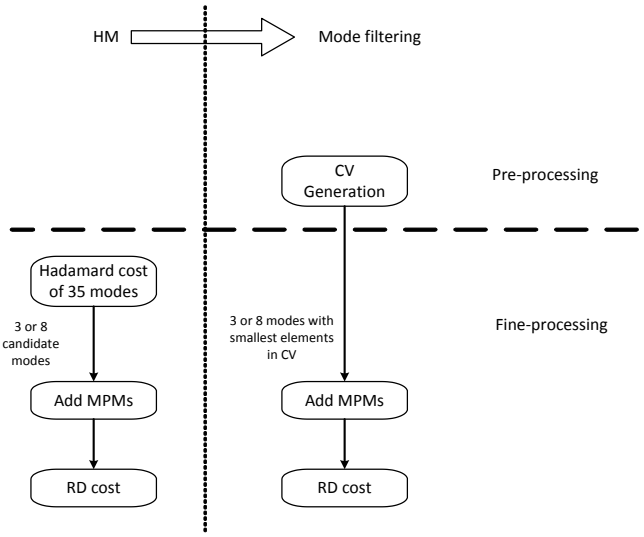

Figure 4-5 Mode selection algorithm.

4.4 Proposed 32x32 PU Compensation Strategy

The largest TU size supported in HEVC can be 32x32. By using large TUs, a large PU (32x32 PU) can benefit a lot in terms of R-D cost, especially in high resolution sequences. However, this advantage will not be reflected in fast pre-processing (FP), which always estimates the 8x8 SATD costs, since the largest SATD supported by HM is 8x8. Therefore, this 32x32 PU compensation strategy is proposed to compensate some 32x32 PUs.

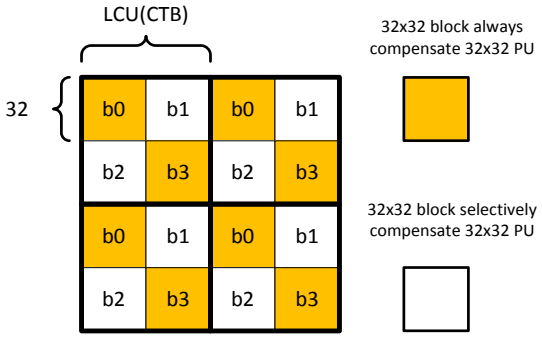

Figure 4-6 Four 32x32 blocks in one CTB.

One LCU (CTB) can be divided to four 32x32 CUs as shown in Figure 4-6. To

solve the above problem, a 32x32 PU compensation strategy can be applied to b0 and b3. For the b1 and b2, some filtered 32x32 PUs in FP can then be selectively compensated based on the spatial features of adjacent b0 and b3.

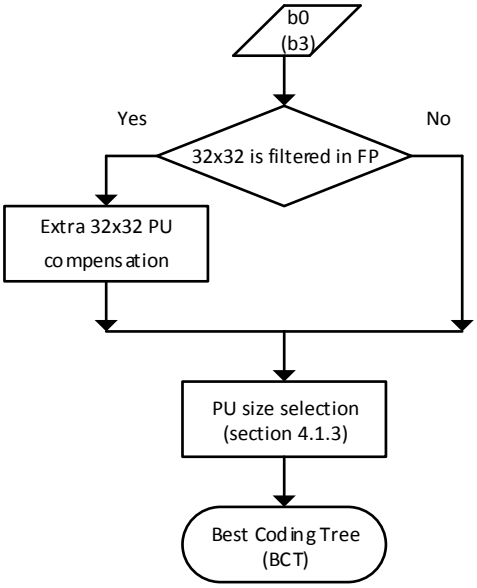

Figure 4-7 32x32 PU compensation strategy for b0 and b3.

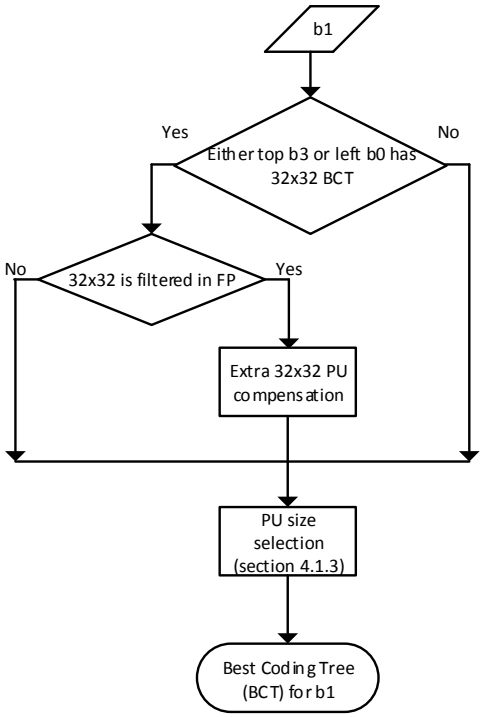

Figure 4-8 32x32 PU compensation strategy for b1 and b2.

Figure 4-7 shows the 32x32 PU compensation scheme for b0 and b3. Even if the 32x32 PU is filtered by the FP system, this PU will be fine-processed additionally. In this manner, it can be always determined whether 32x32 PU is the best coding structure for each b0 and b3 by comparing the R-D cost calculation.

For b1 or b2, if its 32x32 PU is filtered in pre-processing, the compensation strategy will be taken into account. The compensation strategies for b1 and b2 are the same, as shown in Figure 4-8.

When processing each b1 or b2, their left and top neighboring 32x32 CUs have already been processed. So it can refer to the best coding tree of the left and top neighboring CU. If one of its neighboring CU (left or top) selects the best coding structure as 32x32, then the 32x32 PU should calculate R-D cost for this current CU (b1 or b2). The reason is that the current CU is more likely to select 32x32 as the best coding structure if its neighboring CU has already chosen 32x32 as the best coding structure. This case is shown in the middle path of Figure 4-8. In the other cases (left and right path), it will follow the depth selection method in Section 4.2.3.

4.5 Experimental Results

4.5.1 Threshold Training Results

Based on test conditions recommended for HM, the test sequences were divided into five classes according to the resolution. For each class, one sequence was selected to perform the fast threshold training detailed in Section 4.2.2. These training sequences are listed in Table 4-2, and the rest sequences are used as testing sequences. For each sequence, four QP values were used: 22, 27, 32 and 37. As shown in (4-7), the threshold was based on both the CU size (2N) and the QP value. After obtaining the T(QP,2N) combinations from all the training sequences, the average T(QP,2N) of two low resolution (LR) training sequences (*BQMall* and *RaceHorses*) are calculated ac-

cording to the corresponding QP and 2N. After that, the polynomial fitting (degree is one) is done for all the QP values by MATLAB to generate the threshold combinations for LR (Class C&D). The method to gain the threshold combination for HR (Class A&B&E) is the same while the training sequences are *Traffic*, *Cactus* and *KristenAndSara*. T(QP,32) and T(QP,16) finally adopted are listed in Table 4-3 and Table 4-4, respectively. We can see that the T(QP,32) adopted for low resolution is small. Thus, with smaller T(QP,32), more 32x32 will be split for the low resolution. This training result is quite reasonable considering that the 32x32 is always split for the low resolution in original HM.

Table 4-2 Training and testing sequences.

Training sequences	Traffic, Cactus, BQMall, RaceHorses, KristenAndSara
Testing sequences	PeopleOnStreet, Nebuta, SteamLocomotive, Kimono, ParkScene, BQTerrace, BasketballDrive, RaceHorses, PartyScene, BasketballDrill, BQSquare, BlowingBubbles, BasketballPass, FourPeople, Johnny

Table 4-3 T(QP,32) finally adopted with various QPs.

QP	High Resolution	Low Resolution
22	285	102
27	293	111
32	301	119
37	309	128

Table 4-4 T(QP,16) finally adopted with various QPs.

QP	High Resolution	Low Resolution
22	200	101
27	213	134
32	226	167
37	240	200

4.5.2 Coding Performance of Proposed Algorithm

The proposed algorithms were integrated with HM 7.0 [93]. The common test condition "Intra, main" recommended in [94] is adopted. The simulation was performed on a six-core, 3-GHz Intel Xeon-based server. In the experiment, entire sequences for all five classes are coded. The results of these assessments, in which the bit-rate were measured for QPs of 22, 27, 32, and 37 using Bjontegaard's method [95], are listed in Table 4-5. The time reduction is evaluated as follows:

$$\Delta\text{Time}=\frac{Encoding\ time_{proposed}-Encoding\ time_{HM}}{Encoding\ time_{HM}}\times 100\% \qquad (4\text{-}11)$$

Table 4-5 Performance comparison by all the proposed algorithms.

		Proposed Algorithm		
Class	Sequence	ΔTime [%]	BD-bitrate [%]	BD-psnr [dB]
Class A (4k)	PeopleOnStreet	-47.85%	2.30	-0.1298
	Nebuta	-41.99%	1.54	-0.1124
	SteamLocomotive	-48.35%	0.70	-0.0424
Class B (1080p)	Kimono	-59.39%	0.87	-0.0313
	ParkScene	-54.98%	2.06	-0.0881
	BQTerrace	-51.47%	1.54	-0.0952
	BasketballDrive	-62.53%	2.59	-0.0619
Class C (WVGA)	RaceHorses	-53.32%	1.82	-0.1159
	PartyScene	-48.92%	1.30	-0.1021
	BasketballDrill	-55.96%	3.12	-0.1479
Class D (WQVGA)	BQSquare	-48.72%	1.16	-0.1023
	BlowingBubbles	-46.29%	1.20	-0.0717
	BasketballPass	-53.72%	2.28	-0.1303
Class E (720p)	FourPeople	-49.95%	2.43	-0.1407
	Johnny	-56.64%	3.09	-0.1256
Average		**-52.01%**	**1.87**	**-0.0998**

As shown in Table 4-5, the proposed algorithm achieves an average of 52% encoding time saving with the performance loss being less than 1.9%, which can be considered acceptable. For different sequences, stable complexity reduction can be

achieved. We can see that at least 41% complexity can be reduced, with sequence *Nebuta*. In addition, a quite stable complexity reduction can be achieved for different sequences within each resolution.

The effect of mode selection scheme is shown in Table 4-6. Without the mode selection, about 48% time reduction can be achieved with BD-bitrate about 1.8%. Mode filtering is not as effective as PU depth selection in terms of the complexity reduction. The reason is that mode filtering can reduce the complexity of Hadamard cost while the PU depth filtering can reduce the number of PU depths with R-D cost calculation. R-D cost takes the majority of encoding complexity. Even so, the mode filtering can still achieve 5% more complexity reduction with almost no performance loss.

Table 4-6 The effect of the mode selection.

		Proposed Algorithm **w/o** Mode Selection		
Class	Sequence	ΔTime [%]	BD-bitrate [%]	BD-psnr [dB]
Class A (4k)	PeopleOnStreet	-43.50%	2.20	-0.1237
	Nebuta	-38.18%	1.53	-0.1112
	SteamLocomotive	-43.59%	0.67	-0.0406
Class B (1080p)	Kimono	-53.59%	0.76	-0.0268
	ParkScene	-50.07%	2.16	-0.0925
	BQTerrace	-47.24%	1.55	-0.0959
	BasketballDrive	-57.11%	2.15	-0.0515
Class C (WVGA)	RaceHorses	-49.34%	1.79	-0.1131
	PartyScene	-45.95%	1.35	-0.1067
	BasketballDrill	-51.61%	3.13	-0.1488
Class D (WQVGA)	BQSquare	-45.42%	1.15	-0.1016
	BlowingBubbles	-43.07%	1.12	-0.0673
	BasketballPass	-49.29%	2.19	-0.1245
Class E (720p)	FourPeople	-45.25%	2.30	-0.1335
	Johnny	-51.43%	2.65	-0.1074
Average		**-47.64%**	**1.78**	**-0.0963**

By using the PU size selection method presented in section 4.2.3, only two PU depths need fine-processing. After adopting the 32x32 compensation strategy pro-

posed in section 4.4, only two or three PU depths need fine-processing including R-D cost calculation. To further verify the effectiveness of my algorithms, my proposals are compared with the fixed two or three PU depths. The comparison result is shown in Table 4-7. Almost all the combination of fixed two or three PU depths cannot get the better coding efficiency than my algorithms. The fixed three PU depths (16&8&4) can achieve almost the same coding efficiency as my algorithms and it can achieve the BD-bitrate of 2.12%. However, my algorithm is much more attractive in terms of encoding time reduction (**-52.01%** vs. -16.49%).

Table 4-7 Performance comparison with the fixed two and three PU depths.

Fixed two and three PU depths	ΔTime[%]	BD-bitrate[%]
64&32	-66.75%	16.18
32&16	-62.37%	10.52
16&8	-51.71%	5.48
8&4	-34.77%	8.90
64&32&16	-48.46%	10.42
32&16&8	-35.16%	3.31
16&8&4	-16.49%	2.12
Proposed	**-52.01%**	**1.87**

Table 4-8 shows the performance comparison with all the hardware-oriented previous works. ΔT_c means the encoding time reduction from the coding depth selection and ΔT_p means the encoding time reduction from the prediction mode selection. ΔT is the overall encoding time reduction. From the table, we can see that [56] and [74] are focused on prediction mode selection. Compared with [74], 10% more encoding time reduction can be achieved. However, the BD-bitrate becomes 1.2% larger. Compared with [56], 32% more encoding time reduction can be achieved while the BD-bitrate becomes 1.0% larger. The rest of the references in Table 4-8 are focused on depth decision. Compared with the references of [7], [57], [70], [72] and [75], the coding efficiency is better while they can achieve 5%-15% more encoding time reduction. Com-

pared with [58], about 15% more encoding time reduction can be achieved with almost the same coding efficiency.

About the feature of my design, most of the previous works calculate edge or gradient for the depth or mode decision. A low-complexity SATD-based cost model is created and the fast depth decision is conducted based on proposed cost model rather than edge or gradient information.

Table 4-8 Performance comparison with previous work.

Proposed Algorithm	ΔT_c	ΔT_p	ΔT	Maximum BD-bitrate [%]	Average BD-bitrate[%]
[7]	62	0	62	6.73	4.53
[56]	0	20	20	2.96	0.9
[57]	52	5	57	7.00	5.10
[58]	38	0	38	2.28	1.25
[74]	14	28	42	0.9	0.66
[75]	0	67	67	8.35	3.92
[70]*	61	0	61	5.01	2.67
[72]*	61	0	61	5.72	3.39
Proposal	**47**	**5**	**52**	**3.12**	**1.87**

* After the publication of my proposal

To compare with [59] (HM7.0), the experiments are done based on the common test condition “lowdelay, main” and “randomaccess, main” which is recommended in the common test condition. And the function (Cbf fast mode setting) for [59] is turned on in HM7.0 during this experiment. Since my algorithm aims to the early termination for intra prediction, the results are compared with HM7.0 in terms of encoding time for intra prediction and bit-rates.

The results are shown in Table 4-9. By using the common condition “lowdelay, main”, average 56% intra encoding time can be reduced and the average BD-bitrate is only 0.88%. Under the common condition “randomaccess, main”, about 51% intra encoding time can be saved while the performance loss is only 1.44% in average.

For the common test sequences in HM, the effect to high resolution (Class A&B&E) is shown in Table 4-10. The sequence *Nebuta*, *SteamLocomotive*, *Kimono*

and *Johnny* can benefit a lot from this compensation. This was because using a large PU (32x32 PU) contributed significantly to the coding efficiency of these sequences. It can be concluded that the proposed strategy is really effective to improve performance considerably at high resolution.

Table 4-9 Performance comparison with [59].

	Lowdelay, main		Randomaccess, main	
Class	ΔIntra Time	BD-bitrate [%]	ΔIntra Time	BD-bitrate [%]
A	-47.24%	0.89	-46.18%	1.54
B	-48.73%	0.86	-52.55%	1.24
C	-58.81%	0.99	-58.48%	1.61
D	-66.47%	0.48	-44.33%	0.98
E	-57.70%	1.31	-50.18%	2.12
Average*	-56.24%	0.88	-50.94%	1.44

* the average is the average value of all the test sequences given in the common test condition

Table 4-10 The effect of 32x32 PU compensation strategy to high resolution sequences.

	Proposed Algorithm		Proposed Algorithm w/o 32x32 PU Compensation	
High resolution	BD-bitrate[%]	BD-psnr[dB]	BD-bitrate[%]	BD-psnr[dB]
PeopleOnStreet	2.30	-0.1298	2.79	-0.1569
Nebuta	**1.54**	**-0.1124**	**9.20**	**-0.6427**
SteamLocomotive	**0.70**	**-0.0424**	**9.78**	**-0.5695**
Kimono	**0.87**	**-0.0313**	**8.38**	**-0.2952**
ParkScene	2.06	-0.0881	2.94	-0.1252
BQTerrace	1.54	-0.0952	1.82	-0.1121
BasketballDrive	2.59	-0.0619	3.39	-0.0814
FourPeople	2.43	-0.1407	3.04	-0.1755
Johnny	**3.09**	**-0.1256**	**6.41**	**-0.2571**
Average	**1.90**	**-0.0919**	**5.31**	**-0.2684**

4.5.3 Analysis for the Coding Performance

The overall results are shown in Table 4-5. We can see that for some specific se-

quences such as BasketballDrive and Johnny, the performance loss is large. The reason will be analyzed in this chapter.

Figure 4-9 The 1st frame of Traffic.

Figure 4-10 The 1st frame of Cactus.

Figure 4-11 The 1st frame of BasketballDrive.

Figure 4-12 The 1st frame of Johnny.

As shown in Eq. (4-7), the threshold is used to do the depth selection and it is a function of QP and 2N. The function includes the bit part thus the threshold is used to partially compensate the bit part. However, for different sequences, the required bits are different. If the bits for the training sequences are obviously different from the

testing sequences, the trained threshold may not be suitable for the testing sequences. For the high resolution, the training sequences are Traffic, Cactus and KristenAndSara. The 1st frame of Traffic and Cactus are shown in Figure 4-9 and Figure 4-10. We can see that Traffic and Cactus are the sequences with complex textures since there are no large backgrounds and there are many objects. Therefore, more bits are required. Compression ratio is adopted to reflect the number of coded bits and it is about 28 for Traffic and 24 for Cactus. Among the test sequences, BasketballDrive and Johnny are the sequences with simple textures. The 1st frame of BasketballDrive and Johnny are shown in Figure 4-11 and Figure 4-12. We can see that there are many simple textures in the background of picture. For BasketballDrive, the wall and ground have simple textures. For Johnny, the blue board has simple textures. The encoding results show that the compression ratio reaches 60 for the two sequences. As a result, for the sequence with simple textures, the trained thresholds may not be appropriate for the depth decision and lead to large coding performance loss.

In order to prove my assumption, the coding results of Johnny and BQTerrace are analyzed. The BD-bitrate of Johnny is as large as 3.09% while the BD-bitrate of BQTerrace is only 1.54%. The CU partition result of the Johnny is shown in Figure 4-13, we can see that there are many large CUs. For Johnny, more than 50% picture area is coded in the large CUs such as 32x32 and 64x64. The CU partition result of BQTerrace is shown in Figure 4-14. We can see that there are many small CUs. There are only about 30% picture area coded in the large CUs such as 32x32 and 64x64. If the picture is simple, more large CUs will be selected. Therefore, the coding performance difference among various sequences is resulted from the different texture complexity of sequences. In this thesis, the training sequences are with the complex texture, thus, for the sequences with simple texture, the coding efficiency will become a little bit worse.

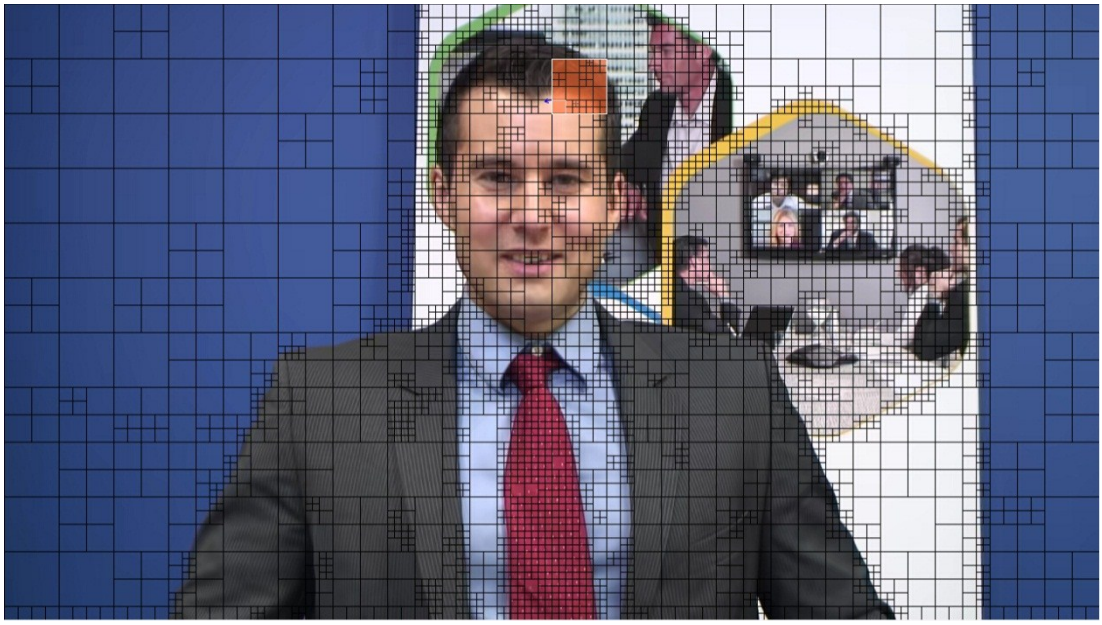

Figure 4-13 The CU partition results of Johnny.

Figure 4-14 The CU partition results of BQTerrace.

4.5.4 Stable Complexity of Proposal

As mentioned before, my proposals can achieve a stable reduction for various sequences. Moreover, a stable time reduction for different QPs and various resolutions

can be ensured. Table 4-11 shows the time reduction of different QPs, [58] could gain more time reduction under larger QPs. By using my proposals, a stable and considerable time reduction can be achieved under different QPs, which indicates the stability of my algorithms. In Table 4-12, we can see that [58] can achieve about 52% reduction for Class E while only 27% reduction can be achieved for Class D. However, the proposal can ensure a stable reduction for various resolutions (Class).

Table 4-11 Time reduction of different QPs.

QP	Proposed Algorithm	[58]
22	-48.08%	-30.85%
27	-49.41%	-30.13%
32	-51.49%	-43.03%
37	-53.34%	-47.47%

Table 4-12 Time reduction of different Classes.

Class	Proposed Algorithm	[58]
A	-46.07%	-37.36%
B	-57.09%	-41.28%
C	-52.74%	-32.03%
D	-49.58%	-26.84%
E	-53.30%	-51.83%

4.6 Chapter Summary

A low-complexity algorithm for HEVC intra prediction is proposed in this chapter. By using the fast PU depth selection scheme, the number of PUs that requires full RDO is reduced from five to two. To supply this PU size decision, a fast off-line training method is also designed. Based on the benefit of using a large size transform, the 32x32 PU is selectively compensated and processed. For the selected PU depths, the proposed fast prediction mode selection scheme reuses the information in the pre-processing in order to avoid the precise Hadamard cost calculation stage in the

original HM. As a result, the proposed method achieves 52% reduction in encoding time relative to HM 7.0, while its corresponding bit rate increase is about 1.87%. Compared with [56], [58] and [74], 10%-32% more encoding time reduction can be achieved with 0.62%-1.21% more coding efficiency loss. Compared with [7], [57] and [75], the proposal can achieve 2.05%-3.23% coding efficiency improvement with 5%-15% less encoding time reduction. Compared with [72], the proposal can achieve 1.52% better coding efficiency with 9% less encoding time reduction. Compared with the latest work [70], the proposal can achieve 0.8% better coding efficiency with 9% less encoding time reduction.

5. Conclusion and Future Work

In this thesis, the low-complexity algorithms and architectures for HEVC mode decision and reconstruction loop are presented. Chapter 2 gives an area-efficient transform architecture. There are two major contributions. Firstly, the requiring outputs are reordered in each clock cycle to reuse the inputs of the butterfly structure. By doing so, about 25% area consumption can be saved compared with previous works. Secondly, the transpose memory is implemented by SRAM instead of registers. A data mapping scheme is designed to reorder the writing positions in the SRAM so that the I/O utilization of SRAM can achieve 100%. The results show that about 62% area consumption can be saved compared with previous works. This chapter can be used in the mode decision and reconstruction loop for both intra and inter prediction.

Chapter 3 gives a low-cost architecture for the system of de-quantization and inverse transform. There are two major contributions. Firstly, for the de-quantization, the coefficient is decomposed to two parts. One part is baseLevel whose value is not greater than 3 thus the multiplication with scaling parameter can be replaced by LUT. The other part is remaining and the number of non-zero remaining values is usually not greater than 4 within one 4x4 block. So the number of required multipliers is reduced from 16 to 4. Four multipliers can be reused in different clock cycles if there are more than 4 non-zero remaining values. Secondly, a system with zero skipping method is created. The zero elements are detected by reusing the pixel data. After detecting the zero elements, the read/write memory operation is skipped in order to save the power consumption. As a result, overall, for the logical part, 68% normalized area consumption can be reduced compared with the previous work. For the whole system, 56% normalized power consumption can be reduced compared with the previous work. For the de-quantization, the proposed architecture can save 77% area consumption compared with previous works. For the memory part, 29%-86% power consump-

tion can be saved compared with not using the zero skipping method. This chapter can be applied in the mode decision and reconstruction loop for both intra and inter prediction.

In Chapter 4, a fast PU depth and prediction mode decision method for the intra prediction is given. There are two major contributions. At first, a pre-processing method based on original pixels rather than reconstructed pixels is given. Only the costs for 8x8 are calculated and then the results are reused to estimate the cost for larger PUs. Two PU depths are selected for R-D cost calculation based on the results of pre-processing. Secondly, by reusing the results of pre-processing, the Hadamard calculation for all the prediction modes in original HM can be eliminated. As a result, about 52% encoding complexity can be reduced with 1.87% BD-bitrate compared with HM. Compared with previous works, more encoding time reduction or better coding performance could be achieved as a trade-off result. This work can be used for intra prediction in the mode decision.

About the future work, there are four plans. The first plan is to polish the algorithm for the intra fast RDO and do the hardware implementation. In this thesis, the number of PUs requiring R-D cost calculation has been reduced. For each PU, the number of prediction modes requiring R-D cost calculation is the same as origin. For the PU depth 4x4 or 8x8, 8 modes are required to calculate the R-D cost. In fact, based on the proposed estimated cost, some prediction modes can be filtered for the R-D cost calculation. Secondly, the rate estimator will be designed. Although there are already some existing architectures for the rate estimator, the performance is still not high enough. Thirdly, the target is focusing on the fast RDO for the inter prediction. Although the full RDO does not consume the majority in the inter-frame encoding, it is still a very interesting and meaningful work. Finally, after designing the individual components, the individual components will be combined and the overall pipeline will be developed.

Bibliography

[1] Cisco, White paper: Cisco VNI Forecast and Methodology, 2015-2020

[2] “High efficiency video coding,” ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), Jan. 2013.

[3] C.-C. Ju, et al. “A 0.5 nJ/Pixel 4 K H.265/HEVC Codec LSI for Multi-Format Smartphone Applications”, IEEE Journal on solid-state circuits, vol. 51, no. 1, pp. 56-67, Jan. 2016.

[4] 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.

[5] V. Sze, M. Budagavi, G.J. Sullivan, “High Efficiency Video Coding (HEVC): Algorithms and Architectures”, Chapter 11.

[6] G. Pastuszak, and A. Abramowski, “Algorithm and Architecture Design of the H.265/HEVC Intra Encoder,” IEEE Transactions on circuits and systems for video technology, vol. 26, no. 1, pp. 210-222, May 2015.

[7] J. Zhu, Z. Liu, D. Wang, Q. Han, and Y. Song, "HDTV1080p HEVC Intra encoder with source texture based CU/PU mode pre-decision," in Proc. of IEEE Asia and South Pacific Design Automation Conference, pp. 367-372, Jan. 2014.

[8] S.-Y. Jou, S.-J. Chang, and T.-S. Chang, “Fast motion estimation algorithm and design for real time QFHD High Efficiency Video Coding”, IEEE Transactions on circuits and systems for video technology, vol. 25, no. 9, pp. 1533-1544, Sept. 2015.

[9] Z. Sheng, D. Zhou, H. Sun and S. Goto, “Low-Complexity Rate-Distortion Optimization Algorithms for HEVC Intra Prediction,” in Proc. of MultiMedia Modeling, pp. 541-552, Jan. 2014.

[10] L. Hu, H. Sun, D. Zhou and S. Kimura, “Hardware-oriented rate-distortion optimization algorithm for HEVC intra-frame encoder,” in Proc. of IEEE International Conference in Multimedia & Expo Workshops (ICMEW), pp. 1-6, June 2015.

[11] W. Shen, Y. Fan, L. Huang, J. Li, “A Hardware-Friendly Method for Rate-Distortion Optimization of HEVC Intra Coding,” in Proc. of International

Symp. on VLSI Design Automation and Test, pp. 1-4, Apr. 2014.

[12]H. Qi, Q. Huang and W. Gao, “A low-cost very large scale integration architecture for multistandard inverse transform,” IEEE Transactions on Circuits and Syst. II: Express Briefs, vol. 57, no. 7, pp. 551-555, Jul. 2010.

[13]G.-A. Su and C.-P. Fan, “Low-cost hardware-sharing architecture of fast 1-D inverse transforms for H.264/AVC and AVS applications,” IEEE Trans. On Circuits and Syst. II: Express Briefs, vol. 55, no. 12, pp. 1249-1253, Dec. 2008.

[14]K. Wang et al., "A reconfigurable multi-transform VLSI architecture supporting video codec design," IEEE Trans. on Circuits and Syst. II: Express Briefs, vol. 58, no. 7, pp. 432-436, Jul. 2011.

[15]H. Chang and K. Cho, “High-performance inverse transform circuit based on butterfly architecture for H. 264 high profile decoder,” In Proc. of IEEE Asia Pacific Conf. on Circ. and Sys., pp. 394-397, Dec. 2010.

[16]C. Peng, D. Yu, X. Cao and S. Sheng, “A new high throughput VLSI architecture for H. 264 transform and quantization,” In Proc. of Int. Conf. on ASIC, pp. 950-953, Oct. 2007.

[17]W. H. Chen, C. H. Smith, and S. C. Fralick, “A fast computational algorithm for the discrete cosine transform,” IEEE Trans. Commun., vol. COM-25, no. 9, pp. 1004-1009, Sep. 1977.

[18]S. Shen, W. Shen, Y. Fan and X. Zeng, “A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards,” In Proc. of IEEE Int. Conf. on Multimedia and Expo, pp. 788-793, Jul. 2012.

[19]S. Shen, W. Shen, Y. Fan and X. Zeng, “A unified forward/inverse transform architecture for multi-standard video codec design,” IEICE Trans. on Fundam. of Electron., Commun. and Comp. Sci., vol. E96-A, no. 7, pp. 1534-1542, Jul. 2013.

[20]J. Zhu, Z. Liu, and D. Wang, “Fully pipelined DCT/IDCT/ Hadamard unified transform architecture for HEVC Codec,” In Proc. of IEEE Int. Symp. on Circuits and Syst., pp. 677-680, May 2013.

[21]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),” J. Semicond. Technol. and Sci., vol. 12, no. 2, pp. 203-211, June 2012.

[22]P. T. Chiang, and T. S. Chang, “A reconfigurable inverse transform architecture design for HEVC decoder,” In Proc. of IEEE Int. Symp. on Circuits and Syst., pp.

1006-1009, May 2013.

[23]M. Budagavi, and V. Sze, “Unified forward+ inverse transform architecture for HEVC,” in proc. of IEEE Int. Conf. on Image Processing, pp. 209-212, Sep. 2012.

[24]W. Zhao, T. Onoye, and T. Song, “High-performance multiplierless transform architecture for HEVC,” In Proc. of IEEE Int. Symp. On Circuits and Syst., pp. 1668-1671, May 2013.

[25]S. Y. Park, and P. K. Meher, “Flexible integer DCT architectures for HEVC,” In Proc. of IEEE Int. Symp. on Circuits and Syst., pp. 1376-1379, May 2013.

[26]P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim and C. Yeo, “Efficient Integer DCT Architectures for HEVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 24, no. 1, pp. 168–178, Jan. 2014.

[27]T. Do, Y. Tan, and C. Yeo, "High-throughput and low-cost hardware-oriented integer transforms for HEVC," In Proc. of IEEE International Conference on Image Processing, pp. 2105-2109, Oct. 2014.

[28]B. Lee, and M. Kim, "A CU-Level Rate and Distortion Estimation Scheme for RDO of Hardware-Friendly HEVC Encoders Using Low-Complexity Integer DCTs," In Proc. of IEEE Transactions on Image Processing, pp. 3787-3800, August 2016.

[29]J. Zhu, Z. Liu, D. Wang, Q. Han, Y. Song, “Fast Prediction Mode Decision with Hadamard Transform Based Rate-Distortion Cost Estimation for HEVC Intra Coding,” in Proc. of IEEE International Conference on Image Processing, pp. 1977-1981, Sep. 2013.

[30]U. Potluri, et al. "Improved 8-point approximate DCT for image and video compression requiring only 14 additions," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 61, no. 6, pp. 1727-1740, June 2014.

[31]M. Jridi, A. Alfalou, and P. K. Meher, "A generalized algorithm and reconfigurable architecture for efficient and scalable orthogonal approximation of DCT," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62, no. 2, pp. 449-457, Feb. 2015.

[32]R. Cintra, and Fábio M. Bayer, "A DCT approximation for image compression," IEEE Signal Processing Letters, vol. 18, no. 10, 579-582, Oct. 2010.

[33]S. Bouguezel, M. Omair Ahmad, and M. N. S. Swamy, "Low-complexity 8× 8

transform for image compression," Electronics Letters, vol. 44, no. 21, pp. 1249-1250, Oct. 2008.

[34]E. Kalali, A. C. Mert, and I. Hamzaoglu, "A computation and energy reduction technique for HEVC Discrete Cosine Transform," IEEE Transactions on Consumer Electronics, vol. 62, no. 2, pp. 166-174, May 2016.

[35]M. Jridi, and P. K. Meher, “A Scalable Approximate DCT Architectures for Efficient HEVC Compliant Video Coding,” IEEE Transactions on Circuits and Systems for Video Technology, accepted.

[36]Y. Fan, L. Huang, Y. Bai, and X. Zeng, “A parallel-access mapping method for the data exchange buffers around DCT/IDCT in HEVC encoders based on single-port SRAMs,” IEEE Trans. Circuits Syst. II, vol. 62, no. 12, pp. 1139-1143, Dec. 2015.

[37]M. Tikekar, C.T Huang, V. Sze, A. Chandrakasan, “Energy and Area-Efficient Hardware Implementation of HEVC Inverse Transform and Dequantization,” in Proc. of IEEE International Conference on Image Processing, pp. 2100-2104, Oct. 2014.

[38]M. Budagavi and V. Sze, “IDCT pruning and scan dependent transform order,” Joint Collaborative Team on Video Coding (JCT-VC), 2011.

[39]Y. H. Chen, and V. Sze, “A deeply pipelined CABAC decoder for HEVC supporting level 6.2 high-tier applications,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 5, pp. 856-868, May 2015.

[40]X. Huang, H. Jia, B. Cai, C. Zhu, J. Liu, M. Yang, D. Xie, and W. Gao, “Fast algorithms and VLSI architecture design for HEVC intra-mode decision,” Journal of Real-Time Image Processing, vol. 12, no. 2, pp. 285-302, Aug. 2016.

[41]F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, D. Wu, and S. Wu, “Fast mode decision algorithm for intra prediction in H.264/AVC video coding,” IEEE Transactions on circuits and systems for video technology, vol. 15, no. 7, pp. 813–822, Jul. 2005.

[42]Y. C. Wei, and J. F. Yang, “Transformed-domain intra mode decision in H.264/AVC encoder,” in Proc. of IEEE Region 10 Int. Tech. Conf., pp. 1-4, Nov. 2006.

[43]C. Kim, and C. C. J. Kuo, “Feature-based intra-/interceding mode selection for H.264/AVC,” IEEE Transactions on circuits and systems for video technology,

vol. 17, no. 4, pp. 441-453, Apr. 2007.

[44] Y.-H. Huang, T.-S. Ou, and H. Chen, “Fast Decision of Block Size, Prediction Mode, and Intra Block for H.264 Intra Prediction,” IEEE Transactions on circuits and systems for video technology, vol. 20, no. 8, pp. 1122-1132, Aug. 2010.

[45] G. Tian, T. Zhang, X. Wei, T. Ikenaga, and S. Goto, “A block type decision algorithm for H.264/AVC intra prediction based on entropy feature,” in Proc. of IEEE Asia Pacific Circuits and Systems, pp. 1348-1351, Nov. 2008.

[46] Y. Lee, and Y. Lin, "Zero-block mode decision algorithm for H.264/AVC," IEEE Transactions on Image Processing, vol. 18, no. 3, pp. 524-533, March 2009.

[47] H. Zeng, K. Ma and C. Cai, "Fast mode decision for multiview video coding using mode correlation," IEEE Transactions on circuits and systems for video technology, vol. 21, no. 11, pp. 1659-1666, Apr. 2011.

[48] H. Wang, S. Kwong and C. Kok, "An efficient mode decision algorithm for H.264/AVC encoding optimization," IEEE Transactions on Multimedia, vol. 9, no. 4, pp. 882-888, May 2007.

[49] H. Zeng, K. Ma and C. Cai, "Hierarchical intra mode decision for H.264/AVC," IEEE Transactions on circuits and systems for video technology, vol. 20, no. 6, pp. 907-912, March 2010.

[50] Y. Sun, and J. Wang, "Fast mode decision for H.264/AVC based on rate-distortion clustering," IEEE Transactions on Multimedia, vol. 14, no. 3, pp. 693-702, Feb. 2012.

[51] W. Chen, J. Su, B. Li, and T. Ikenaga, “Reversed intra prediction based on chroma extraction in HEVC,” in Proc. of Int. Symp. Intell. Signal Process. Comn. Syst., pp. 1-5, Dec. 2011.

[52] G. V. Wallendael, S. V. Leuven, J. D. Cock, P. Lambert, R. V. Walle, J. Barbarien, A. Munteanu, “Improved intra mode signaling for HEVC,” in Proc. of IEEE Int. Conf. Multimedia Expo, pp. 1-6, Jul. 2011.

[53] L. Zhao, L. Zhang, S. Ma, and D. Zhao, “Fast mode decision algorithm for intra prediction in HEVC,” in Proc. of IEEE Vis. Comm. Image Pro., pp. 1-4, Nov. 2011.

[54] W. Jiang, H. Ma, and Y. Chen, “Gradient based fast mode decision algorithm for intra prediction in HEVC,” in Proc. of International Conf. on CECNet, pp. 1836-1840, Apr. 2012.

[55] A. S. Motra, A. Gupta, M. Shukla, P. Bansal, and V. Bansal, “Fast intra mode decision for HEVC video encoder,” in Proc. of International Conf. on SoftCOM, pp. 1-5, Sept. 2012.

[56] T. L. Silva, L. V. Agosini, and L. A. S. Cruz, “Fast HEVC intra prediction mode decision based on edge direction information,” in Proc. of European Signal Processing Conference, pp. 1214-1218, Aug. 2012.

[57] Y. Zhang, Z. Li, and B. Li, “Gradient-based fast decision for intra prediction in HEVC,” in Proc. of Visual Communications and Image Processing, pp. 1-6, Nov. 2012.

[58] J. Xiong, and H. Li, “Fast and efficient prediction unit size selection for HEVC intra prediction,” in Proc. of Intelligent Signal Processing and Communications Systems, pp. 366-369, Nov. 2012.

[59] R.-H. Gweon, Y.-L. lee, “Early termination of CU encoding to reduce HEVC complexity,” IEICE Trans. Fundamentals, vol.E95-A, no.7, pp. 1215-1218, July 2012.

[60] F. Yao, X. Zhang, Z. Gao, and B. Yang, "Fast mode and depth decision algorithm for HEVC intra coding based on characteristics of coding bits," in Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp. 1-4, June 2016.

[61] X. Wang, and Y. Xue, "Fast HEVC intra coding algorithm based on Otsu's method and gradient," In Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp. 1-5, June 2016.

[62] X. Liu, Y. Liu, P. Wang, C.-F. Lai, and H.-C. Chao, "An Adaptive Mode Decision Algorithm Based on Video Texture Characteristics for HEVC Intra Prediction," IEEE Transactions on circuits and systems for video technology, accepted.

[63] S. Park, S. Lee, H. Xu, and E. S. Jang, "Temporal correlation-based fast encoding algorithm in HEVC intra frame coding," in Proc. of IEEE International Conference on Consumer Electronics-Berlin, pp. 113-115. Sep. 2015.

[64] S. Jaballah, K. Rouis, and J. B. Tahar, "Clustering-based fast intra prediction mode algorithm for HEVC," In Proc. of European Signal Processing Conference, pp. 1850-1854, Aug. 2015.

[65] X. Shang, G. Wang, T. Fan, and Y. Li, "Fast CU size decision and PU mode decision algorithm in HEVC intra coding," in Proc. of IEEE International Conference on Image Processing, pp. 1593-1597, Sep. 2015.

[66] K. Lim, J. Lee, S. Kim, and S. Lee, "Fast PU skip and split termination algorithm for HEVC intra prediction," IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 8, pp. 1335-1346, Aug. 2015.

[67] H.-S. Kim, and R.-H. Park, "Fast CU Partitioning Algorithm for HEVC Using an Online-Learning-Based Bayesian Decision Rule," IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 1, pp. 130-138, Jan. 2016.

[68] S. Cho, and M. Kim, "Fast CU Splitting and Pruning for Suboptimal CU Partitioning in HEVC Intra Coding", IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 9, pp. 1555-1564, Sep. 2013.

[69] T. Zhang, M.-T. Sun, D. Zhao, and W. Gao, "Fast Intra Mode and CU Size Decision for HEVC," IEEE Transactions on Circuits and Systems for Video Technology, accepted.

[70] Z. Liu, X. Yu, Y. Gao, S. Chen, X. Ji, and D. Wang, "CU Partition Mode Decision for HEVC Hardwired Intra Encoder Using Convolution Neural Network," IEEE Transactions on Image Processing, vol. 25, no. 11, pp. 5088-5103, Nov. 2016.

[71] Z. Liu, X. Yu, S. Chen, and D. Wang, "CNN oriented fast HEVC intra CU mode decision," In Proc. of IEEE International Symposium on Circuits and Systems, pp. 2270-2273, May 2016.

[72] X. Yu, Z. Liu, J. Liu, Y. Gao, and D. Wang, "VLSI friendly fast CU/PU mode decision for HEVC intra encoding: Leveraging convolution neural network," in Proc. of IEEE Int. Conf. Image Process., pp. 1285–1289, Sep. 2015.

[73] B. Min, and R. CC Cheung, "A fast CU size decision algorithm for the HEVC intra encoder," IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 5, pp. 892-896, May 2015.

[74] G. Chen, L. Sun, Z. Liu, and T. Ikenaga, “Fast Mode and Depth Decision for HEVC Intra Prediction Based on Edge Detection and Partition Reconfiguration,” IEICE Tran. Fund., vol. E97-A, no. 11, pp. 2130-2138, Nov. 2014.

[75] W. Zhao, T. Oneye, and T. Song, “Hardware-oriented fast mode decision algorithm for intra prediction in HEVC”, in Proc. of Picture Coding Symposium, pp. 109-112, Dec. 2013.

[76] R. Gweon and Y. Lee, “Early Termination of CU Encoding to Reduce HEVC Complexity,” IEICE Tran. Fund., vol. E95-A, no.7, pp. 1215-1218, July 2012.

[77] N. Hu and E. H. Yang, "Fast Mode Selection for HEVC Intra-Frame Coding With Entropy Coding Refinement Based on a Transparent Composite Model," IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 9, pp. 1521-1532, Sept. 2015.

[78] L. Shen, Z. Zhang and P. An, "Fast CU size decision and mode decision algorithm for HEVC intra coding," IEEE Transactions on Consumer Electronics, vol. 59, no. 1, pp. 207-213, February 2013.

[79] C. F. Tseng and Y. T. Lai, "Fast coding unit decision and mode selection for intra-frame coding in high-efficiency video coding," IET Image Processing, vol. 10, no. 3, pp. 215-221, February 2016.

[80] J. Chen and L. Yu, "Effective HEVC intra coding unit size decision based on online progressive Bayesian classification," in Proc. of IEEE International Conference on Multimedia and Expo, pp. 1-6, July 2016.

[81] Q. Hu, Z. Shi, X. Zhang and Z. Gao, "Fast HEVC intra mode decision based on logistic regression classification," in Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp. 1-4, June 2016.

[82] N. Kim, S. Jeon, H. J. Shim, B. Jeon, S. C. Lim and H. Ko, "Adaptive keypoint-based CU depth decision for HEVC intra coding," in Proc. of IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp. 1-3, June 2016.

[83] W. Geuder, P. Amon and E. Steinbach, "Low-complexity block size decision for HEVC intra coding using binary image feature descriptors," in Proc. of IEEE International Conference on Image Processing, pp. 242-246, Sept. 2015.

[84] X. Shang, G. Wang, T. Fan and Y. Li, "Fast CU size decision and PU mode decision algorithm in HEVC intra coding," in Proc. of IEEE International Conference on Image Processing, pp. 1593-1597, Sept. 2015.

[85] M. Radosavljević, G. Georgakarakos, S. Lafond and D. Vukobratović, "Fast coding unit selection based on local texture characteristics for HEVC intra frame," in Proc. of IEEE Global Conference on Signal and Information Processing, pp. 1377-1381, Dec. 2015.

[86] B. Du, W. C. Siu and X. Yang, "Fast CU partition strategy for HEVC intra-frame coding using learning approach via random forests," in Proc. of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, pp. 1085-1090, Dec. 2015.

[87] C. E. Rhee, “Skipping Prediction Directions Based on the Cost Relationship between Multi-Directional Predictions for an HEVC Encoder,” IEICE Trans. Inf. and Syst., vol. E97-D, no. 9, Sep. 2014.

[88] K. Goswami, B-G. Kim, D-S Jun, S-H Jung, and J. S. Choi, “Early Coding Unit (CU) Splitting Termination Algorithm for High Efficiency Video Coding (HEVC),” ETRI Journal, vol. 36, no. 3, pp. 407-417, June 2014.

[89] X. Shen, L. Yu, and J. Chen, “Fast coding unit size selection for HEVC based on Bayesian decision rule,” in Proc. of Picture Coding Symposium, pp. 453-456, May 2012.

[90] L. Shen, Z. Liu, X. Zhang,W. Zhao, and Z. Zhang, “An effective CU size decision method for HEVC encoders,” IEEE Transactions on Multimedia, vol. 15, no. 2, pp. 465-470, Feb. 2013.

[91] X. Huang, C. Kuo, C, Mao and Y. Ciou, “Adaptive Depth Search Range for HEVC Coding Unit Size Selection,” in Proc. of Asia-Pacific Signal and Information Processing Association, pp. 1-6, Dec. 2014.

[92] X. Jiang, T. Song, W, Shi, T. Shimato, L. Wang, “High Efficiency CU Depth Pre-

diction Algorithm for High Resolution Applications of HEVC," IEICE Trans. Fundamentals, vol. E98-A, no.12, pp. 2528-2536, Dec. 2015

[93]https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/

[94]F. Bossen, "Common HM test conditions and software reference configurations," JCTVC-I1100, Jul. 2012.

[95]G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," in 13th Video Coding Experts Group (VCEG)-M33 Meeting, Austin, TX, Apr. 2001.

Publications

The publications with ○ are included in this thesis.

Transactions:

○[1] **Heming Sun**, Dajiang Zhou, Shuping Zhang, and Shinji Kimura, "A Low-Power VLSI Architecture for HEVC De-quantization and Inverse Transform", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E99.A, No. 12, pp. 2375-2387, December 2016.

○[2] **Heming Sun**, Dajiang Zhou, Peilin Liu, and Satoshi Goto, "A Low-Cost VLSI Architecture of Multiple-Size IDCT for H.265/HEVC", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E97.A, No. 12, pp. 2467-2476, December 2014.

○[3] **Heming Sun**, Dajiang Zhou, Peilin Liu, and Satoshi Goto, "Fast Prediction Unit Selection and Mode Selection for HEVC Intra Prediction", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E97.A, No. 2, pp. 510-519, February 2014.

International Conferences:

[4] Dajiang Zhou, Shihao Wang, **Heming Sun**, Jianbin Zhou, Jiayi Zhu, Yijin Zhao, Jinjia Zhou, Shuping Zhang, Shinji Kimura, Yoshimura Takeshi, and Satoshi Goto, "A 4Gpixel/s 8/10b H.265/HEVC video decoder chip for 8K Ultra HD applications ", IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, America, pp. 266-268, February 2016.

[5] Zhengxue Cheng, **Heming Sun**, Dajiang Zhou and Shinji Kimura, "Merge mode based fast inter prediction for HEVC", IEEE International Conference on Visual Communications and Image Processing (VCIP), Singapore, Singapore, pp. 1-4, December 2015.

[6] Landan Hu, **Heming Sun**, Dajiang Zhou and Shinji Kimura, "Hardware-oriented rate-distortion optimization algorithm for HEVC intra-frame encoder", IEEE International Conference on Multimedia and Expo Workshops (ICMEW), Turin, Italy, pp.

1-6, June 2015.

[7] Zhengxue Cheng, **Heming Sun**, Landan Hu, and Shinji Kimura, "A fast level filtering algorithm for inter prediction in HEVC encoder", International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Seoul, Korea, pp. 404-407, June 2015.

○[8] **Heming Sun**, Dajiang Zhou, Jiayi Zhu, Shinji Kimura and Satoshi Goto, "An area-efficient 4/8/16/32-point inverse DCT architecture for UHDTV HEVC decoder", IEEE International Conference on Visual Communications and Image Processing (VCIP), Valletta, Malta, pp. 197-200, December 2014.

[9] Jianbin Zhou, Dajiang Zhou, **Heming Sun**, and Satoshi Goto, "VLSI architecture of HEVC intra prediction for 8K UHDTV applications", IEEE International Conference on Image Processing (ICIP), Paris, France, pp. 1273-1277, October 2014.

[10] **Heming Sun**, and Satoshi Goto, "A fast mode selection algorithm for HEVC intra prediction", International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Phuket, Thailand, pp. 449-451, July 2014.

[11] Zhe Sheng, Dajiang Zhou, **Heming Sun**, and Satoshi Goto, "Low-complexity rate-distortion optimization algorithms for HEVC intra prediction", International Conference on Multimedia Modeling (MMM), Dublin, Ireland, pp. 541-552, January 2014.

○[12] **Heming Sun**, Dajiang Zhou, and Satoshi Goto, "A low-complexity HEVC intra prediction algorithm based on level and mode filtering", IEEE International Conference on Multimedia and Expo (ICME), Melbourne, Australia, pp. 1085-1090, July 2012.

Appendix

"Graduate Program for Embodiment Informatics" supports me during the doctor course. Therefore, I would like to appreciate this program and explain my understanding about the "Embodiment Informatics" in the appendix.

Literally speaking, Embodiment Informatics is composed of two words. Informatics is a study about information science. In the ancient, the amount of information is quite limited. However, in the modern life, the amount of information has been exploding thus informatics is developed to manage massive information systematically. Embodiment means the implementation which can turn something from the idea-level to product-level. Nowadays, with the development of many fields, the ideas are not limited to only one research field. For example, in the past, the car is only related with the mechanical system. There was no relationship between car manufacture and electronic engineering. However, recently, the auto car based on the electronic system becomes a hot topic. Therefore, the combination of different research fields is highly desired. After the implementation of the ideas, the product will come out and be used in daily life. Therefore, the target of Embodiment Informatics is to combine the concepts of different research topics and develop some real products which are really useful for the human being's daily life.

For example, my research topic is the algorithm and architecture design for the video compression. The latest video compression standard is High Efficiency Video Coding (HEVC) which can double the compression ratio compared with the former standard H.264. By HEVC encoding, the storage and transmission burden for the ultra high definition videos can be relieved. In fact, my research can also be collaborated with the research topics in other fields such as robotics and wireless communication. After the fabrication, the encoding chip is small enough to be placed on a robot. This robot with video encoding chip can capture the video and compress the data in the

real-time. By developing an ultra-low-power encoder, the robot is able to have a long-life battery. By using the wireless communication, it can support the real-time communication between the robot and control center. A stable wireless communication system is required. The wireless communication is related with the information engineering. As a result, the product is an intelligent robot which can take the film, and then compress the captured video and transmit the compressed data in the real-time.

In order to master the knowledge of different research fields, taking the courses of different fields is highly required. For example, I belong to the department of system LSI. However, I not only learn the knowledge of my department such as digital circuits, but also take some courses in the information engineering such as machine learning in order to widen the knowledge.

In addition to knowledge, the program also encourages the students to improve several abilities. The first ability is the social ability in the company. I take some courses about business in order to know how to apply the knowledge in book to the practical marketing. Moreover, the program also sets up the studio to hold many events. In the studio, the students can gather together and have a hot discussion to improve the communication ability. The second ability is the leadership. The program not only aims to develop good researchers, but also develop good leaders who can contribute to the future of Japan. Therefore, I have tried my best to improve the leadership during the doctor course. I not only focus on my own research, but also take care of many master students on their researches. By my instruction and their hard working, they have obtained good results and published transactions and conference papers. The third ability is the language. It is obvious that English is very important for the researchers. The program also supports the students for one-month English training in U.C.Davis. Therefore, I have always kept in mind to practice English.

In short, I really appreciate this program for giving me so many chances. I will apply the knowledge learnt from this program into the future life.