行政院國家科學委員會補助專題研究計畫 **■成果報告**
□期中進度報告

應用於數位電視的雙模視訊解碼器

計畫類別：■個別型計畫　　□整合型計畫
計畫編號：NSC97－2221－E－009－167-MY3
執行期間： 97 年 8 月　1 日至 100 年 7 月 31 日

執行機構及系所：國立交通大學 電子工程學系

計畫主持人：李鎮宜 教授
共同主持人：
計畫參與人員：李曜 蔡長宏 郭明諭 涂淑文 范期赫 劉家麟 鍾思齊

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本計畫除繳交成果報告外，另須繳交以下出國心得報告：
□赴國外出差或研習心得報告
□赴大陸地區出差或研習心得報告
□出席國際學術會議心得報告
□國際合作研究計畫國外研究報告

處理方式：除列管計畫及下列情形者外，得立即公開查詢
　　　　　□涉及專利或其他智慧財產權，□一年■二年後可公開查詢

中　華　民　國 100 年　7 月　31 日

Figure Index

Figure Index

Table Index

中文摘要

新一代的視訊解碼系統，除了必須滿足多標準和多模式的操作模式外，更重要的是如何降低功耗，以及隨著電源能量的多寡，提供行動視訊的最佳終端服務需求。在此三年 (2008/8~2011/7) 的研究計畫中，我們延續過去三年 (2005/8~2008/7) 在視訊處理器的研究成果，朝低功耗、低成本、以及多模式的視訊解碼方案進行多項關鍵技術的研究。在多模式的研究議題上，主要將 H.264/SVC 的功能需求加入現有的雙模式 (MPEG2 和 H.264) 硬體平台上，探討新的關鍵模組的實現方案，以及從系統整體行為模式和效能的考量下，提出一更好的系統硬體架構，有助於單獨系統的效能展現和以 IP 為次系統的整合效益。在低成本的研究議題上，主要考量到如何降低解碼過程所需求的記憶容量，並採用外掛的記憶體模組，尤其當動態補償所需求的高記憶體容量和頻寬時，如何充分使用有限的資源 (記憶容量和頻寬) ，達成符合標準解碼的運算需求。在低功耗的設計議題上，我們除了分析解碼行為的特徵和架構的相依性，藉以探討系統、個別模組、資料流等不同層級的低功耗設計方法，亦將奈米級製程所衍生的漏電流效應，一併考量，提供符合視訊解碼標準下的低功耗設計方案。此外，我們亦將建立 FPGA 的雛形系統展示平台，有利於關鍵模組和系統行為的呈現。

關鍵字：視訊解碼系統；多模式；多標準；低功耗；低成本；行動視訊

英文摘要

It is well understood that research efforts, for next-generation video decoding system, have to cover not only multi-standard and multi-mode operation capability, but also less power dissipation and power awareness with optimal picture quality, especially when mobile video services are taken into account. As a result, in this 3-year (2008/8~2011/7) research project proposal, we further investigate several key issues related to so-called low-power, low-cost, and multi-mode video decoder solutions. Based on our previous work on a dual-mode video (2005/8~2008/7), we leverage the available design platform and research results to further explore new design approaches. For multi-mode task, we investigate the specifications defined in H.264/SVC and add those key modules into our H.264/MPEG2 decoder platform. Not only new key modules are explored, but also system decoding behaviors are analyzed to study a better system architectural model so that a stand-alone and IP-based decoder solution can be obtained. For low-cost issue, the major problem lies in memory management and limited bus bandwidth. It is necessary to take into account available stand-alone memory modules; even SoC solutions become a must. Therefore developing a well-organized memory hierarchy and access mechanism to meet decoding requirements under limited resources (storage space and bus bandwidth) has been further explored. For low-power issue, an analysis of the decoding behavior and related hardware architecture has been conducted. Thus system exploration, module design, and data flow have been investigated to reduce power dissipation at different levels. In addition, leakage current due to nano-meter CMOS process has been considered to provide a competitive video decoder solution. Finally an FPGA prototype has already been set up to evaluate the performance of the proposed video decoder and related key modules.

Keywords:

Video Decoder, Multi-Mode, Multi-Standard, Low-Power, Low-Cost, Mobile Video

一、 計畫的緣由與目的

A. The Embedded Compressor/Decompressor

To improve the video coding efficiency, eliminating temporal redundancy between frames is a useful technique. This technique is widely used in nowadays video coding standards such as MPEG-1/2/4, H.263 and H.264. But to accomplish this method when encoding or decoding, at least one previous frame must be stored in frame memory as reference. Also, when processing motion compensation function in H.264 decoder, the rapid data accesses dominate the power consumption of whole system. For a mobile device, power is always the critical issue that people do care about. Embedded compression (EC) is a common technique to reduce the transferring of data and the size of off-chip frame memory. Moreover, if we embedded a compressor into a system with determined bandwidth, the access times can be efficiently reduced as long as the compressed unit is well-designed. Nowadays, the mobile devices become more and more powerful by their various functions, to reduce the bandwidth and resource requirement of each hardware accelerator is definitely an important topic.

Basically, compression can be divided into two types, lossy and lossless. Lossless methods are good at quality but suffered from variable data amount after compressed. Variable data amount cannot guarantee the reduction neither on the size of external frame memory nor the bandwidth. Lossy compression technique is suitable here because lossy compression with fixed compression ratio can ensure the reduction. Therefore, how to organize the lossy coding methods is important. To cover information as much as possible within limited budget is the main challenge. Several research activities about embedded compression have been proposed in [2]-[4]. Discrete cosine transform with high efficiency bit plane zonal coding have been proposed in [1], this JPEG-like methods provides good compressed quality, and the hardware is relatively smaller than JPEG. But the bit plane zonal coding is too complicate, thus its processing cycles may become too long and not suitable for being embedded with H.264 video decoder. Also, the input data of motion compensation is provided through the embedded decompressor. The packing unit of [1] is 8x8 pixel matrix, and this size will cause access redundancy for 4x4 block based motion compensation. Another kind of algorithms is DPCM-based [2]. By taking the intra prediction information of H.264 encoder to remove the spatial redundancy and combining Golomb-Rice coding, DPCM-based method achieves good quality. However, this method needs iteration several times to get the most appropriate quantization level. Thus compression cycle for each coding unit is not a constant, and leads to the complicate embedded compressor design. In [4], the authors modified Hadamard transform and combined with Golomb-Rice coding. With the shortest encoded cycles, MHT becomes the most flexible embedded compression scheme to be embedded into a video decoder.

In this report, a new transform-based lossy embedded compression scheme is proposed. Taking 4x4 pixels as a coding unit and each unit is compressed with compression ratio two, only 64 bits is needed to store in external memory. And with the simple modified bit plane zonal coding, the decoding process of a 4x4 block can be done within 4 cycles including the data fetching. This algorithm can be pipelined into two stages. A MB only needs 34 cycles to decode and has 5.29dB quality improvement compared with MHT [4].

B. *The High Profile Intra Predictior*

H.264/AVC [6]-[7] is the latest international video coding standard from MPEG and ITU-T Video Coding Experts Group. It consists of three profiles which are defined as a subset of technologies within a standard usually created for specific applications. Baseline and main profiles are intended as the applications for video conferencing/mobile and broadcast/storage, respectively. Considering the H.264-coded video in high profile, it targets the compression of high-quality and high-resolution video and becomes the mainstream of high-definition consumer products such as Blu-ray disc. However, high-profile video is more challenging in terms of implementation cost and access bandwidth since it involves extra coding engine, such as macroblock-adaptive frame field (MBAFF) coding and 8×8 intra coding, for achieving high performance compression.

In the MBAFF-coded pictures, they can be partitioned into 16x32 macroblock pairs, and both macroblocks in each macroblock-pair are coded in either frame or field mode. As compared to purely frame-coded pictures, MBAFF coding requires two times of neighboring pixels size and therefore increases implementation cost. To cope with aforementioned problem, we propose neighboring buffer memory (include upper/left/corner) to reuse the overlapped neighboring pixels of an MB pair. Furthermore, we present memory hierarchy and pixel re-ordering process to optimize the overall memory size and external access efficiency. On the other hand, H.264 additionally adopts intra 8×8 coding tools for improving coding efficiency. It involves a reference sample filtering process (RSFP) before decoding a Luma *intra_8x8* block. These filtered pixels are used to generate predicted pixels of 8×8 blocks. Hence, the additional processing latency and cost are required, and they may impact the overall performance for the real-time playback of high-definition video. In this report, we simplify the RSF process via a base-mode predictor and optimize the processing latency and buffer cost. Compared to the existing design [10] without supporting intra 8×8 coding, this design only introduces area overhead of 10% and 7.5% of gate counts and SRAM.

An architectural choice advocated for dealing with long past history of data is a memory hierarchy [12]. In the intra prediction, it utilizes the neighboring pixels to create a reliable predictor, leading to dependency on a long past history of data. This dependency can be solved by storing upper rows of pixels for predicting current pixels but is a challenging issue in implementation cost and access bandwidth. To optimize the introduced buffer cost and access efficiency, we use two internal Line SRAM1 and Line SRAM2 to store the Luma and Chroma upper line pixels, as illustrated in Figure 1. By the ping-pong mechanism, the upper neighboring pixels of current MB (or MB pair) are stored to one of them, and the other Line SRAM is used to store next MB of upper neighboring pixels. This memory hierarchy facilitates the internal Line SRAM size and the decoding pipeline schedule.

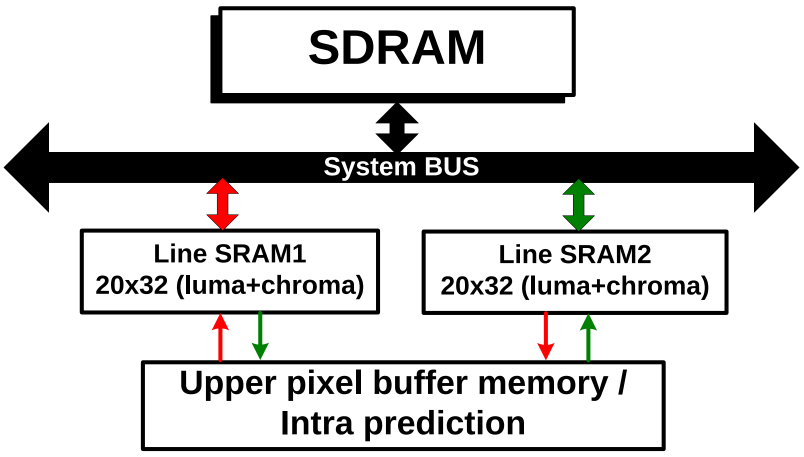


Figure 1: Memory hierarchy of high-profile intra prediction.

C. The Reduced Patterns Comparison Embedded Compressor/Decompressor (RPCC)

To improve the video coding efficiency, diminishing the data correlation of the temporal redundancy in each frame is widely used in the latest video coding standard, such as H.264/AVC [13]-[14]. However, it causes a large amount of data transmission between on-chip processing modules and external memory. In addition, the rapid and huge data access from Motion Compensation (MC) consumes the majority of system power and become serious in many portable devices. Many low power techniques have already been proposed to reduce power consumption, but data transmission still dominates huge amount of system power. Hence, reduce data access between on-chip processing modules and external memory is the critical consideration in a mobile video device. Although the mobile video devices are suffered from limited battery capability, the visual quality requirement is not as high as high resolution applications. Therefore, the embedded compression is suitable to lessen the volume of data access and the size of off-chip memory under the premise of maintaining acceptable visual quality. The mobile video devices are more and more important due to their various functions at the present time. Reducing the usage of bandwidth and the required resource of hardware in the mobile video devices is a critical topic.

In general, the compression methods are classified into two categories: lossless compression and lossy compression. It is obvious that lossless compression methods [15] completely reserve the information while truncating the size of data, so there has no quality loss. However, some problems of lossless compression are so fatal that it’s not suitable for system integration application. The lossless compression suffers from variable length of lossless compressed data that we cannot regularly control the compression ratio, frame memory size and bandwidth requirement. These disadvantages are also attributed to the needs of memory to prepare for the worst case of data access and the unknown size of data. Therefore, there exists an important characteristic of lossy compression methods [16]-[19] which differs them from lossless compression methods. The characteristic of fixed compression ratio allows us to improve the disadvantages of lossless compression methods mentioned previously. Although lossy compression algorithm will sacrifice tolerable visual quality, the reduced power consumption, memory size and bandwidth requirement are more attractive for mobile video devices.

Several lossy compression schemes have been proposed in [16]-[19]. The transform-based compression methods can convert the signal from time domain to frequency domain and move the energy to up-left corner. In human visual system, the lower frequency component is more important than the higher frequency component whose feature can be exploited to efficiently compress the amount of data, such as in [16]-[17]. In [16], both Modified Hadamard Transform (MHT) and quantization of Golomb-Rice Coding (GRC) are employed. To improve the quality loss of [16], [17] adopts Discrete Cosine Transform (DCT) and Modified Bit Plane Zonal Coding (MBPZC) instead of MHT and GRC. Although transform-based schemes provide good compressed quality, MHT and DCT are too complicated to suit for being embedded with H.264 mobile video devices. Another kind of algorithms is pattern-based [18]-[19]. [18] adopts 64 patterns to improve Bit Plane Truncation (BTC) algorithm and [19] increases extra acceptable quality loss to reduce the number of compared patterns from[18]. Both [18] and [19] are limited by BTC algorithm; the coding latency is still too long to be well- embedded into the target H.264 system. However, through [18] and [19], we find a way to utilize the patterns to reduce the coding latency and the amount of data.

In this paper, we propose a pattern-based lossy embedded compression method which adopts 4x2 pixels as coding unit and CR is fixed as two.

D. The Bitplane Truncation with Pattern Comparison Coding Embedded Compressor/Decompressor

A video coding standard achieves high compression efficiency such as H.264 [19] and so forth. For H.264, at least one previous frame is stored in frame memory to generate a predicted frame. Obviously, Motion Compensation (MC) demands a huge amount of data accesses between off-chip memory devices and the video decoder chip. However, data transfer consumes a lot of power. For mobile video devices, one major issue is the limited power supply from battery. Therefore, reducing the bandwidth requirement and size of frame memory is greatly demanded while maintaining acceptable visual quality.

In general, embedded compression methods can be categorized into two fundamental groups: lossless and lossy. Lossless compression algorithms [21] have no error propagation problem. Lossy compression algorithms, comparing with lossless compression algorithms, accomplish the fixed compression ratio (CR). Several lossy compression algorithms have been proposed such as Modified Hadamard Transform (MHT) plus quantization of Colomb-Rice Coding [16], DCT plus Modified Bit Plane Zonal Coding [17], and et al. [18] exploits forty-six patterns to improve Block Truncation Coding and [19] increases extra acceptable quality loss to reduce the number of compared patterns from [18].

Lossless compression can guarantee no quality loss, but variable length of the compressed data caused irreducible frame memory size. Therefore, existing lossless algorithms are not suitable for frame compression because their primary purpose is high coding efficiency rather than low latency, computation complexity, and high random accessibility. On the contrary, lossy compression algorithm with the fixed CR can guarantee the reduction of frame memory size. Consequently, it is important to design a lossy algorithm with the following features: 1) Low distortion visual quality, 2) Low complexity, 3) Low bandwidth requirement, and 4) Low power consumption.

E. An Area-efficiently High-accuracy Prediction-based CABAC Decoder for H.264/AVC

H.264/AVC [13] is the state of the art video compression standard in current video applications. It supports two entropy coding tools. One is Context-based Adaptive Variable Length Coding (CAVLC), and the other is Context-based Adaptive Binary Arithmetic Coding (CABAC) [13]. CABAC can achieve 9% to 14% bit-rate saving in average compared with CAVLC. However, it has notably strong data dependency to restrict throughput. Even if a DSP processor can work at 3GHz, it would be difficult to achieve the real-time CABAC decoding for HD video at 30 frames/s.

Many state-of-the-art works has been proposed for raising the throughput. In [26]-[28], multiple-bin decoding mechanism provides a significant improvement for throughput. However, it also provides extra overhead for hardware cost. Besides, previous works [25] clearly described the syntax element switch overhead (SESO) problem and presented a prediction scheme to solve it.

In this report, we proposed high-accuracy prediction scheme and area-efficient decoding architecture. Furthermore, we optimized the memory system to reduce extra overhead. Through our design, the overall CABAC decoder can be implemented by less hardware cost and still have acceptable throughput.

In the traditional CABAC decoding flow, it separates 4 pipeline stages - Context index Calculate (CC), Context model Load (CL), Arithmetic Decode (AD) and DeBinarization (DB). In Figure 2, the pipeline would be stalled when ctxIdx depend on bin from AD, or SE[i+1] depend on SE[i]. That was main critical performance lost in traditional CABAC decoder.

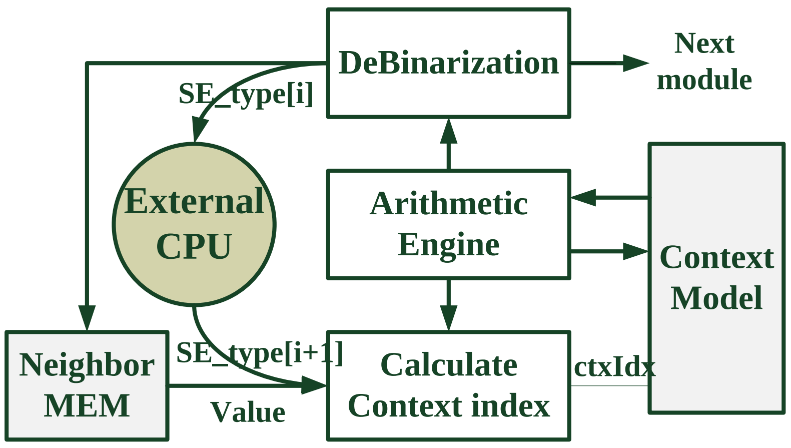


Figure 2 : Block Diagram of Traditional CABAC Decoder

In order to avoid the impact of separation parser and decoder, prediction-based CABAC decoder (just like [25]) includes a SE predictor to determine next ctxIdx by itself. Therefore, SE predictor can efficiently ease data dependency, but it wouldn't be available when it predicts miss. Actually, we can make a formula for throughput as follows:

Total Cycle = CycleR (Regular) + CycleB (Bypass) + CycleT (Terminal)

≈ CycleR (Regular) + CycleB (Bypass)

=Total Bin (1 + Regular Rate x Miss Rate) + *Idle times

*Idle times: number of stalls waits for neighbour information

Throughput = Working Frequency x (Total Bin/ Total Cycle)

≈ Working Frequency (1+ Regular Rate x Miss Rate+ Idle times/ Total Bin)-1

The working frequency depends on design's critical path or system constraints. Regular Rate and Total Bin depend on patterns, and idle times can be ignored when Total Bin >> Idle times. Therefore, we can consider the performance would degrade when high miss rate or large idle times.

F. A Predefined Bit-Plane Comparison Coding (PBPCC) for Mobile Video Applications.

Video coding standards achieve high compression efficiency such as H.264 [13] and so forth. For H.264, at least one previous frame is stored in the external memory device to generate a predicted frame. Obviously, motion compensation (MC) demands a huge amount of data accesses between off-chip memory devices and the video decoder chip. However, data transfer consumes a lot of power. For mobile video devices, one major issue is the limited power supply from the battery. It is essential to reduce the bandwidth requirement and the size of frame memory while maintaining acceptable visual quality. Therefore, embedded compression (EC) is a suitable technique to achieve the requirement.

EC methods can be categorized majorly into two types: lossless and lossy. Lossless compression algorithms [30], [31] have the advantages of no error propagation and quality loss. However, the variable lengths of the compressed data result in irreducible frame-memory size. Hence, existing lossless algorithms are not suitable for frame compression because their primary purpose is to achieve high coding efficiency rather than low latency, computation complexity, and high random accessibility.

Lossy compression algorithms, compared with lossless compression algorithms, accomplish the fixed compression ratio (CR). Several lossy compression algorithms have been proposed, such as the modified Hadamard transform (MHT) with the quantization of Colomb–Rice coding [32], discrete cosine transform with modified bit-plane zonal coding [33], and so on. Yang and Chai [34] exploit 64 patterns to improve block truncation coding, where Amarunnishad et al. try to enlarge acceptable quality loss by reducing the number of compared patterns in [34]. Lossy compression algorithm with the fixed CR can guarantee the reduction of frame-memory size. However, how to maintain an acceptable visual quality remains to be solved. Consequently, it is important to design a lossy algorithm with the following features: 1) low-distortion visual quality; 2) low complexity; 3) low bandwidth requirement; and 4) low power consumption.

In this report, a novel embedded lossy algorithm based on predefined bit-plane comparison coding (PBCC) is proposed. The CR is fixed at 2. Each 4 × 2 block can be compressed into a 32-bit segment. The proposed PBCC encodes a 4 × 2 block in 2 cycles and decodes a 4 × 2 block in 1 cycle to meet the requirement of embedded frame data processing.

二、 研究方法及成果

A. The Embedded Compressor/Decompressor

(1) *Proposed Embedded Compression Algorithm*

(i) Algotrithm

The compression is conducted on a 4x4 pixel matrix (128 bits) obtained from the output of deblocking filter, and the compression ratio is fixed at two. Each 4x4 unit will become a 64 bits package after compression. Fixed compression ratio leads to constant amount of the coded data, so this EC scheme ensures the ability of random access without extra memory to record the segment address of coded data. Moreover, the 4x4 block unit is the basic coding unit in H.264 standard, and makes the data access in an efficient way.

Figure 3 shows flowchart of the proposed algorithm. The overall compression scheme is formed by two parts: 1) two dimensions discrete cosine transform (DCT) and 2) modified bit plane zonal coding (MBPZ). Two dimensions DCT is composed by two one dimension, 4 points DCT. The discrete cosine transforms is a technique for converting a signal into elementary frequency components and it is widely used in image compression. For human visual system, human eyes are more sensitive on low frequency component of a picture and less sensitive on high frequency component. The DCT can gather the relative important low frequency signal on up left corner, and the most high frequency in down right corner. Thus the DCT combines with bit plane zonal coding with original point at up left corner can efficiently collect the information.

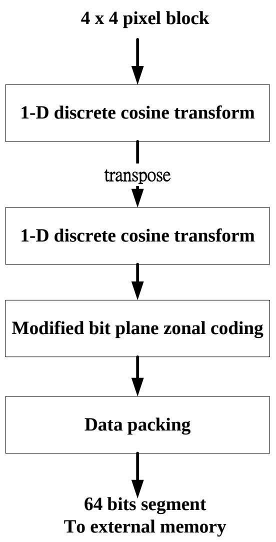

4 x 4 pixel block
1-D discrete cosine transform
transpose
1-D discrete cosine transform
Modified bit plane zonal coding
Data packing
64 bits segment
To external memory

Figure 3: Flowchart of the proposed embedded compression algorithm

The second part is to perform modified bit plane zonal coding on 16 coefficients output from DCT. First, we reverse those negative coefficients into positive value and mark a “1” at the same position of sign bit plane.

Sign bit plane can be considered as union of sign flags for each coefficient. Second, to improve the coding efficiency, we record the start plane. Search each bit planes from MSB to LSB (not including sign bit plane), and the first plane contains nonzero bits is the start plane. To avoid adding too many extra cycles and to simplify the hardware complexity in system view, we use one simple type only for recording each bit plane, though other complex types which can be more scrimping on bit using do exist. For each bit plane, we simply record the maximum row (RMAX) and column (CMAX) which have a "1" in this row or column, and then pack the bits in this zone which is enclosed by RMAX and CMAX. 4 bits are used to record RMAX/CMAX of each plane. And then, we packed the sign bit plane. Since we have only 64 bits budget for each 4x4 unit, the situation of unable to pack all the information may be happened frequently. Since not every coefficient can be packed, packing whole sign bit plane may become a waste. So we take the maximum value of RMAX and CMAX out of each packed bit plane and packing useful sign bits only by using those two boundaries. Under this method we do not waste extra bits to pack those unused sign bits, and the RMAX/CMAX of sign bit plane need not to be packed since they can be derived from the previous coded information. Finally, the end plane needs to be estimated and packed to specify when to stop.

(ii) Packing Mechanism

The overall packing scheme is introduced in this part. After doing discrete cosine transform, we get 16 coefficients from each 4x4 block. There are 15 AC coefficients and one DC coefficient. Sine DC coefficient is the average value and is the most important in transform, we reserve 8 bits budget for the DC coefficient of every 4 x 4 block. DC coefficient is always positive, so we don't have to worry about the sign bit for DC coefficient. For the rest 15 coefficients, we first packed the start plane and end plane (6 bits total). Then, we separate RMAX/CMAX and plane content of those planes which are between start plane and end plane, and connecting all the RMAX/CMAX together and all plane content together respectively. Sign bit information is inserted between the CMAX/RMAX and the plane content. Figure 4 shows the compressed segment format.

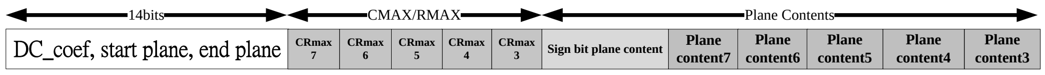


Figure 4: Flowchart of the proposed embedded compression algorithm

(2) *Hardware Design*

(i) Discrete Cosine Transform

The hardware design of DCT is referred from Lee's architecture [5]. This architecture can maintain the same performance with original DCT while reduced the number of multiplications to about half of those required by the existing efficient algorithms. This design allows us to take the advantage of DCT while not suffering from its hardware complexity.

(ii) Modified Bit Plane Zonal Coding and Data Packing

There is a combinational block dealing with coefficients to derive the RMAX/CMAX and plane content of each plane. To serialize the plane information in one cycle, we propose the content adaptive ripple architecture to solve the problem. The basic concept is shown in Figure 5. The 11 lines at left represent the 10 plane contents plus 1 sign bit plane content. Each circle represents a 16 to 1 MUX controlled by 4 bits RMAX/CMAX and is shown in Figure 6. By the ripple behavior, the wire at the end of the flow is the connected result. Notice that we embedded this compressor into our 108MHz decoder, thus one cycle time is enough to finish our ripple processing.

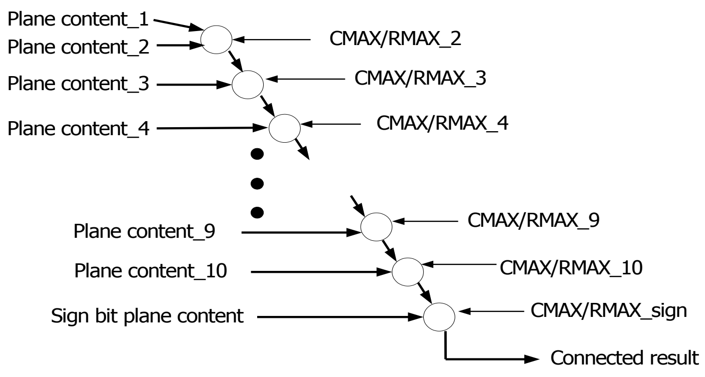


Figure 5: Context adaptive ripple architecture

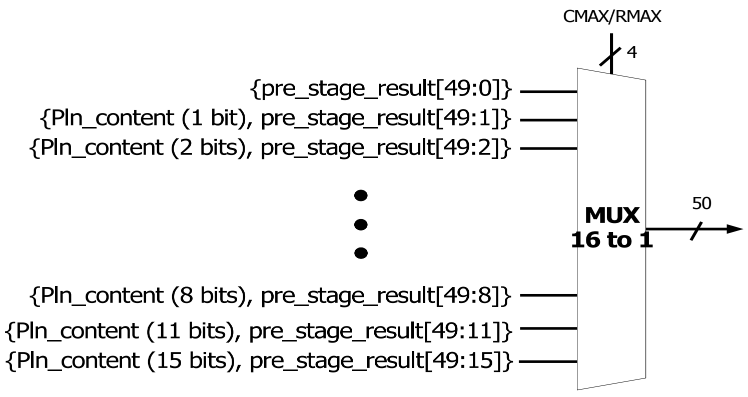

CMAX/RMAX
4
{pre_stage_result[49:0]}
{Pln_content (1 bit), pre_stage_result[49:1]}
{Pln_content (2 bits), pre_stage_result[49:2]}
{Pln_content (8 bits), pre_stage_result[49:8]}
{Pln_content (11 bits), pre_stage_result[49:11]}
{Pln_content (15 bits), pre_stage_result[49:15]}
MUX
16 to 1
50

Figure 6: The MUX structure

(iii) Encoder Design and Decoder Design

Figure 7 shows the pipeline architecture of compressor design. Since compressor has more time to handle the encoding process, we use three stages here and each stage needs 4 cycles. This design with longer cycles can shrink the gate count by reducing one dimension four points DCT units. Under this design, a MB needs 72 cycles to encode.

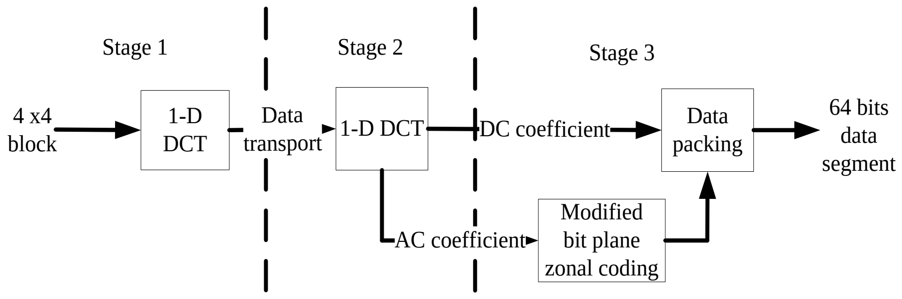


Figure 7: Encoder architecture

Figure 8 shows the architecture of decompressor. To provide data for motion compensation unit suitably, the decompressor must support higher throughput inevitably. The decompressor is divided into two stages and each stage needs 2 cycles, a 4x4 block needs 4 cycles to decode, including the data-fetching. Decoding a MB just needs 34 cycles.

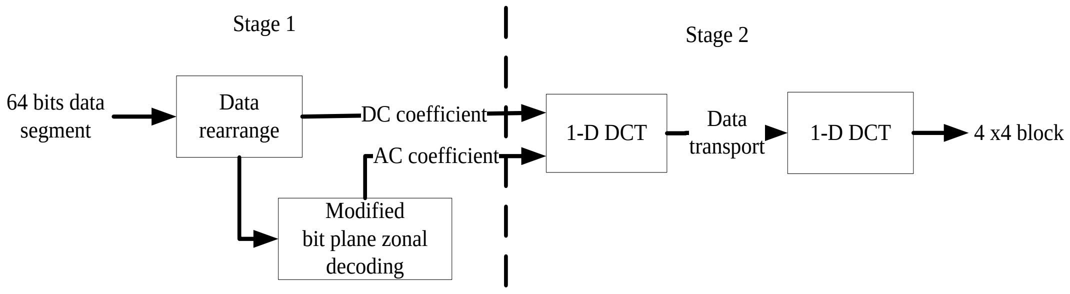


Figure 8: Decoder architecture

(3) *Integration with H.264 Decoder*

Figure 9 shows the block diagram of the H.264 video decoder containing an embedded compressor. Embedded compressor works as the interface between decoder IP and external memory. And the EC mechanism is ready to be used by adding extra address control logic. Since the compression ratio of our embedded compressor is two, each data is compressed into half of the original size. The address control logic is very easy to implement.

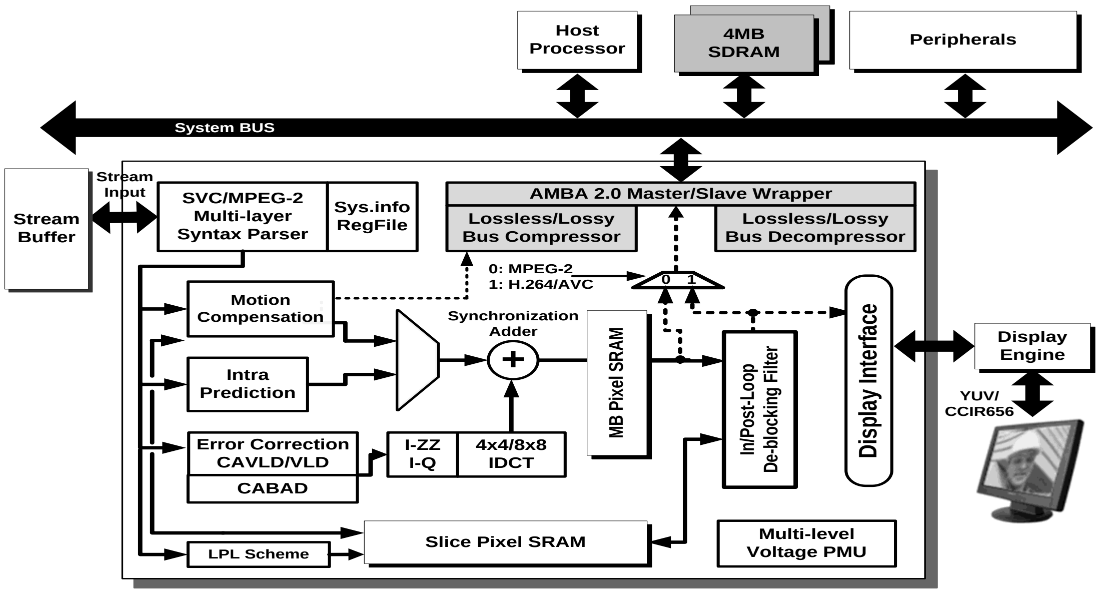


Figure 9: H.264 decoder architecture with proposed embedded compression algorithm

The H.264 decoder works at 108 MHz, performing the HD1080i@30fps. The EC compresses the data from deblocking filter, and 4x4 blocks become 64 bits segments and then stored into off-chip memory. Thus, the data access times of off-chip memory are half of the original access times for the system without an embedded compressor. The embedded decompressor decompresses data from external memory and sends them to motion compensation unit. The system bus bandwidth is default as 32 bits and the external memory is 32 bits per entry, so the original system takes 4 pixels as accessing data unit. The access behavior of motion compensation with/without embedded compressor can be analyzed as follows. If the requested 4x4 blocks are perfectly aligned with the coded 4x4 blocks, only 2 cycles are needed to fetch the 4x4 block while the original system needs 4 cycles to fetch. For the needed block not aligned with the coded blocks in only one direction, the system with embedded compressor needs to decode two blocks to derive the needed data, so 4 cycles are needed. The original system, taking 4 pixels as accessing unit, needs 4 cycles to fetch data. For the needed 4x4 block not aligned with the coded data in both vertical and horizontal directions, the system with EC needs decoding four 4x4 blocks and 8 cycles are needed for data fetching. The original system needs 8 cycles too. For the final situation, the sub pixel case, a 4x4 block needs a 9x9 pixels block to finish the motion compensation. 18 cycles is needed for EC while original system needs 27 cycles. From the analysis above, we can see that H.264 decoder with an embedded compressor does reduce the access times and can efficiently reduce the access power consumption.

(4) *Evaluation Result*

Software implementation of the proposed algorithm is integrated with JM12.4. The reference frame is compressed by the proposed algorithm and compared with the result of previous work using MHT and GR coding respectively. The test sequences are Foreman, Stefan, Mobile and Akiyo in CIF format and Station in HDTV format. All sequences are organized in one I frame follows nine P frames format. For each sequence, 100 frames are used to compute the average PSNR value reference to the original sequences.

Table 1 is the simulation result of five sequences and shows the performance of original H.264 decoder without any recompression, embedded with MHT and embedded with our proposed algorithm.

Table 1: PSNR comparison

Test Sequence	Algorithm (@CR = 2.0)	PSNR (dB) (original)	PSNR lost (dB)
Forman (CIF)	original	35.57	0
	MHT	29.48	6.08
	proposed	34.21	1.36
Stefan (CIF)	original	36.02	0
	MHT	28.64	7.38
	proposed	33.86	2.16
Mobile (CIF)	original	33.75	0
	MHT	23.10	10.65
	proposed	29.27	4.48
Akiyo (CIF)	original	39.64	0
	MHT	32.17	7.47
	proposed	38.33	1.31
Station (HDTV)	original	38.85	0
	MHT	32.97	5.88
	proposed	37.13	1.72

The proposed method maintains better quality in slow motion sequences than high motion sequences. However, performance of the proposed method in all sequences is better than previous MHT work. The average PSNR degradation of proposed method is 2.21dB while the MHT is 7.5dB. Figure 10 shows the embedded result with Station sequence in HDTV format. Although the quality drop is inevitable, the proposed method efficiently slows down the speed of decay than previous MHT work.

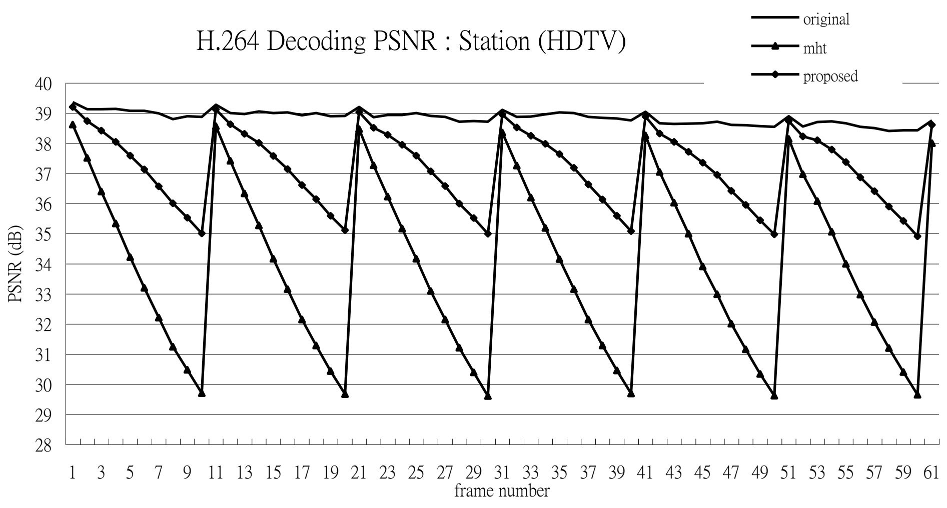


Figure 10: Simulation result of Station sequence(HDTV)

B. The High Profile Intra Predictior

(1) *Proposed High-Profile Intra Predictor*

Figure 11 shows the block diagram of the proposed high-profile intra compensation architecture. A pixel rearranging process, which is located on the bottom-left of Figure 11, is proposed to reduce the complexity of neighbor fetching when MBAFF coding is enabled. The signal, *Line SRAM1/2 data_out*, is directly connected to the intra prediction block for replacing the last set of upper buffer memory. As for 8×8 intra coding, a dedicated pixel buffer memory is used to store the filtered neighboring pixels and reuse the overlapped pixel data. According to the relations between Luma *intra_8x8* modes and numbers of filtered pixels which are needed in each mode, we minimize the number of stored pixels to 17 (i.e. 136 bits). The output of predicted pixel is interfaced to the filtered pixel buffer memory because RSFP is embedded in the intra prediction generator.

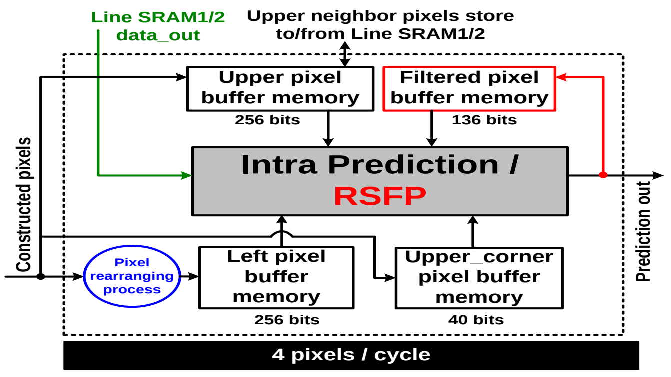

Line SRAM1/2 data_out
Upper neighbor pixels store to/from Line SRAM1/2
Upper pixel buffer memory
256 bits
Filtered pixel buffer memory
136 bits
Intra Prediction / RSFP
Constructed pixels
Prediction out
Pixel rearranging process
Left pixel buffer memory
256 bits
Upper_corner pixel buffer memory
40 bits
4 pixels / cycle

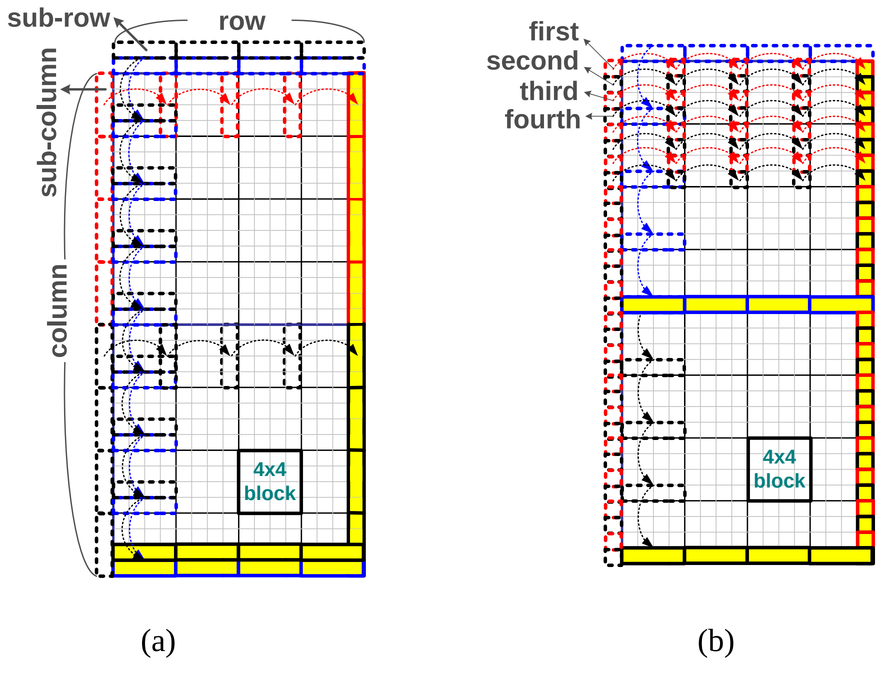
Figure 11: Block diagram of the proposed high-profile intra predictor.

(2) *MBAFF Decoding with Data Reuse Sets*

MBAFF is proposed to improve coding efficiency for interlaced video. However, it introduces longer dependency than conventional frame-coded picture. In this section, we analyze and realize it via upper, left, and corner data reuse sets (DRS) to reuse the pixels and improve the cost and access efficiency.

(i) Upper DRS

For decoding an MBAFF-coded video, upper buffer memory is used to store the constructed upper pixels of current MB pair. These upper buffers are updated with the completion of prediction process on every 4×4 block. For each updated sub-row(s), they can be reused by the underside 4×4 blocks. According to the different prediction modes of MB pair, the upper buffer will store data from different directions. If current MB pair is frame mode, it only needs to load one row of upper buffer (16 pixels) at first, and when a 4×4 block is decoded, updating the two sub-rows in two rows (8 pixels) of upper buffer from top to down at one time, as illustrated in Figure 12(a). In Figure 12 (b), a field-coded MB pair needs to load two rows of upper buffer (32 pixels), two times of frame-coded MB pairs. Then, only one sub-row of upper buffer memory will be updated when a 4×4 block is decoded. However, considering the fifth 4×4 block, it still needs a sub-row of upper buffer to predict, as shown in Figure 13. In order to reduce the upper buffer memory size, the Line SRAM *data_out* is directly used. The only penalty to this scheme is that the Line SRAM *data_out* must hold the value until fifth 4×4 block is decoded.



(a) (b)

Figure 12: The updated direction of upper/left buffer memory in (a) frame and

(b) field mode MB pair.

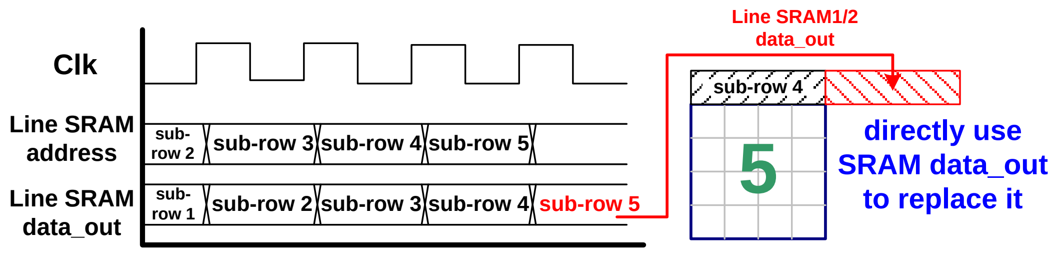


Figure 13: Line SRAM *data_out* replaces the last sub-row of upper buffer.

(ii) Left DRS Upper DRS

The updated direction of the left buffer is similar to that of the upper one. The direction ranges from left to right. When the left buffer is located on the right hand side of MB pair, the next MB pair can reuse these new pixels for the following prediction procedures. However, when the modes of current and previous MB pairs are different, the left neighbors of a 4x4 block will become complicated. To reduce computational complexity of this intra predictor, pixel rearranging process is exploited. If current MB pair is frame mode, each sub-column of left buffer will be updated when each 4x4 block is decoded. On the other hand, if current MB pair is field mode, first and third buffers in each sub-column of left buffer will be updated when each 4×4 block is decoded in the top MB. Second and fourth buffers in each sub-column of left buffer will be updated when each 4×4 block is decoded in bottom MB. Hence, we only need to consider what the mode current MB pair is instead of handling four coding modes for the combination of current and previous MB pairs, and therefore the complexity can be reduced.

(iii) Corner DRS Upper DRS

Using corner buffer memory can efficiently reuse the upper left neighboring pixels. We change the positions of corner buffer from left [5] to top. Therefore, the total corner buffer size can be reduced by 38% (i.e. 64bits → 40bits, because the MB number in horizontal is less than that of vertical MB pair). In particular, Figure 14 shows the updating directions of corner buffer. However, because the upper neighboring pixels will be either the last row or the row prior to the last row in upper MB pair, so the first corner of current MB pair has two processing states: reuse and reload. The first corner is reused when 1) the mode of current MB pair is identical to that of previous (left) MB pair or 2) before decoding the bottom MB of frame-coded MB pair. On the other hand, the first corner is reloaded when 1) the current MB pair has the different modes as previous (left) MB pair or 2) before decoding the bottom MB of field-coded MB pair. In summary, using neighboring buffer memory and their different directions of updates according to different modes of MB pair can reuse the neighboring pixels and improve the access efficiency. The associated pipeline structure of MBAFF decoding is shown in Figure 15. We can see that during a MB pair decoding process, the interaction between buffers and Line SRAM can be completed easily and efficiently, and the communication between another Line SRAM and external SDRAM can be done at the same time.

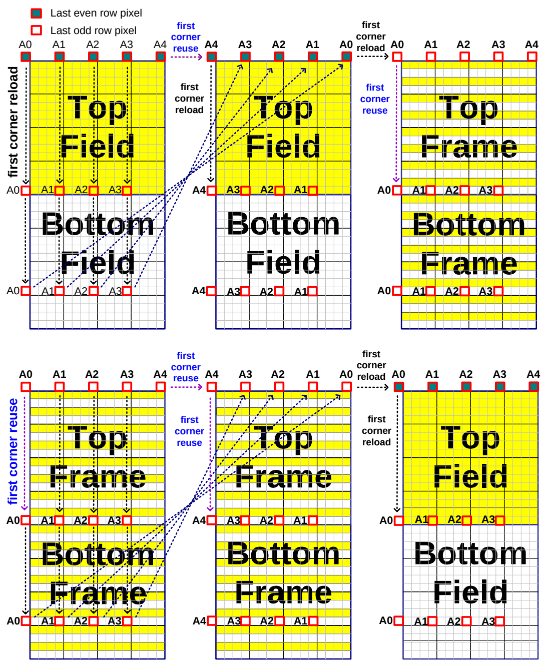


Figure 14: The updated direction of corner pixel buffers.

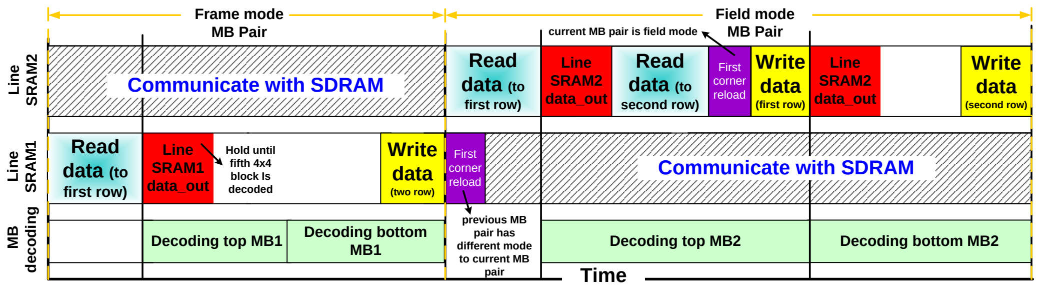


Figure 15: The pipeline scheme of MBAFF decoding

(3) *Intra 8x8 Decoding with Base-Modes*

Luma *intra_8x8* is an additional intra block type supported in H.264 high profile. Before decoding an *intra_8x8* block, there is an extra process that is different from *intra_4x4* and *intra_16x16*, which called reference sample filtering process (RSFP). Original pixels will be filtered first, and then using these filtered pixels to predict subsequent 8×8 blocks. For an *intra_4x4* and *intra_8x8* block, 13 neighbors and 25 filtered neighbors are needed, respectively. According to the Luma *intra_8x8* modes, the maximum number of filtered neighbors is 17, as illustrated in Table 2. Hence, only 17 (i.e. 136 bits) filtered pixels need to be stored. In the *intra_4x4* process, the prediction formula of each mode except DC mode has the same form:

$$prediction\_out = (A+2B+C+2) >> 2.$$

Compared with the share-based intra prediction generator [8]-[10], the proposed base-mode predictor not only reduces area cost (due to elimination of four adders) but also guarantees that four predicted pixels will be generated in one cycle of each *intra_4x4* modes, including DC mode. In particular, we use this base mode predictor to generate the four predicted pixels in one cycle. In the RSFP, the form of formula is identical to that in *intra_4x4*, and also can be rewritten to the same form*:*

$$filtered\_out = (A+2B+C+2) >> 2.$$

Hence, we can share the hardware resource to generate filtered pixels, as shown in Figure 16(a). Notice that an additional process, neighbor distribution, is needed to add in *intra_8x8* process because we only store 17 filtered pixels instead of 25.

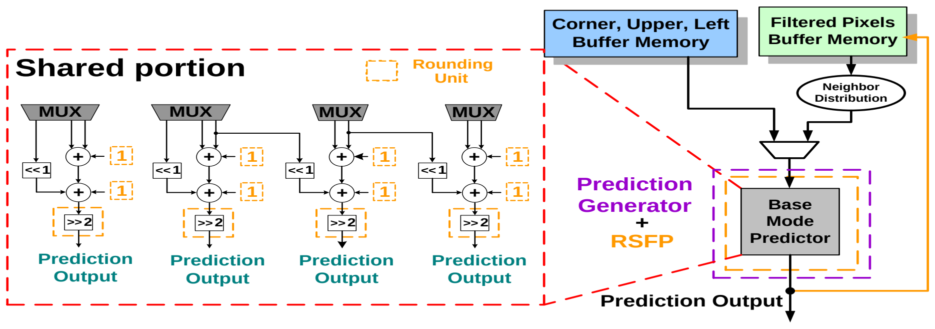
Table 2: Numbers of filtered pixels in intra 8x8 modes

Prediction Modes of Intra_8x8	**# of filtered neighbors**
0 (Vertical)	8
1 (Horizontal)	8
2 (DC)	0,8,16
3 (Diagonal down left)	16
4 (Diagonal down right)	17

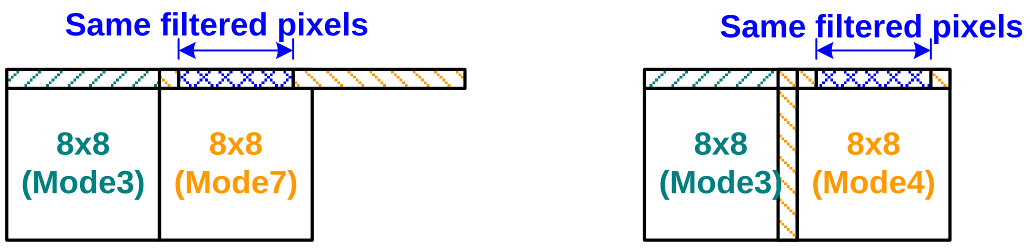
5 (Vertical right)	17
6 (Horizontal down)	17
7 (Vertical left)	16
8 (Horizontal up)	8

$$\textit{Extra latency} = \underline{\underline{\left\lceil \frac{M}{P} \right\rceil}} + \underline{\underline{\left\lceil \frac{M-N}{P} \right\rceil}} + \underline{\underline{\left\lceil \frac{M}{P} \right\rceil}} + \underline{\underline{\left\lceil \frac{M-N}{P} \right\rceil}} \text{, where } \begin{cases} M = 0, 8, 16, \text{ or } 17 \\ N = 0 \text{ or } 6 \\ P = 4 \end{cases} \qquad (1)$$

In a four-parallel intra prediction module, the latency of an 8x8 block will be increased to 0~5 cycles according to the different modes of 8×8 blocks. In order to reduce the latency penalty, we reserve filtered pixels when the mode of the first/third is equal to 3 or 7 and second/fourth 8x8 block is equal to 0, 2 (if upper is available), or 3~7 (i.e. the value N = 6. Otherwise, N = 0). Then these filtered pixels are directly used to predict second/fourth 8×8 block. To clarify the extra latency, Eq. (1) lists the decoding extra latency in an *intra_8x8* MB, and Table 2 summarizes the # of filtered pixels in each 8×8 intra coding mode (i.e. the value of M). In particular, we list some examples to clarify the processing behavior of an intra 8×8 block in Figure 16(b). If the modes of first and second 8×8 blocks are 3 (diagonal bottom left) and 7 (vertical left) or 3 (diagonal bottom left) and 4 (diagonal bottom right), only 10 and 11 pixels are needed to be filtered while decoding the second 8×8 block, as described in Figure 16(b).



(a)



(b)

Figure 16: (a) Architecture of an intra 8×8 decoding module, and (b) behavior of shared filter.

(4) *Simulation Results*

To enhance system performance, our proposal is designed to optimize area, buffer size, and latency. We use two 0.64kb Line SRAMs which are connected to a 32-bit system bus to make decoding pipeline simple, and 0.688kb SRAM to store reused neighboring pixels. Table 3 shows the average cycles for decoding an I-MB in different video sequences of our proposed design for 30fps HD1080 video format at working frequency of 100MHz with MBAFF and Luma *intra_8x8*. The overhead of latency is less than 5% compared to preliminary architecture [11]. The overall area and buffer memory size for supporting H.264 BP/MP/HP are 14063 gates in UMC 0.18um technology and 688 bits, as shown in

Table 4. The overheads for supporting Luma *intra_8x8* are 10% and 7.5% compared to [10].

Table 3: The average cycles for different video sequences.

Test Video Sequence	**Intra Prediction @ BL [11]**	**Proposed Intra Prediction @ HP**	**Cycle Overhead**
Foreman	342.68	355.81	3.8%
Grandma	275.63	285.28	3.5%
Suzie	294.90	307.28	4.2%

Table 4: Comparison results

	Chen et al. [10]	**Proposal**	**Overhead**
Profile	MP	HP	
Process	0.18um	0.18um	
Working Frequency	87M	100M	
Gate Count	12785	14063	10%
Memory (bit)	640	688	7.5%

C. The Reduced Patterns Comparison Embedded Compressor/Decompressor

(1) *The Proposed RPCC Embedded Compression Algorithm*

(i) Algotrithm

The proposed compression scheme adopts pattern-based and 4x2 block-grid. The CR is fixed as two and each 4x2 unit (64 bits) will be compressed into 32-bit data package. Because fixed CR results in regular amount of coded data, the EC assures the ability of random access without extra memory to register the segment address of coded data. In addition, the 4x4 block unit is the basic coding unit in H.264 standard, we partition each 4x4 block into two 4x2 blocks. Thus, 4x2-based block-grid lessens the coding latency and makes the data access more efficient.

The proposed algorithm is shown in Figure 17. There are three parts in the overall EC method: 1) MBPTC, 2) RPCC and 3) average coding. In MBPTC algorithm, we partition a 4x2 matrix into eight bit planes and search the Start Plane (SP) in four continuous layers which are close to MSB with 2 bits as the first compression step

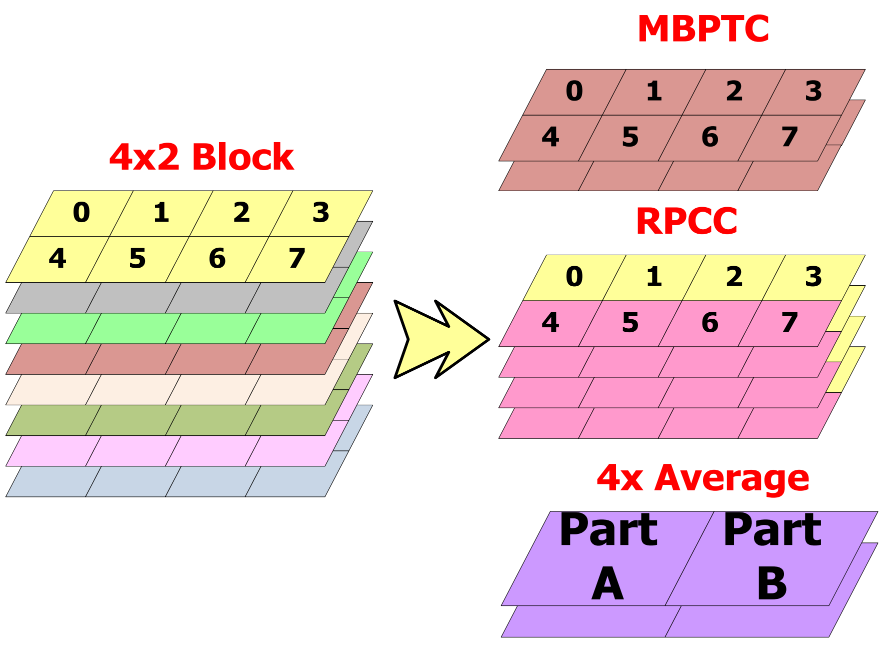


Figure 17 Compression methods of our proposed algorithm

The second step is to deal with remaining bit planes after MBPTC by RPCC. As shown in Figure 18, we partition a 4x1 section with 4 bits into four 4x1 layers. According to the coding threshold adopted, the RPCC will select the left or right strategy. While we set the coding threshold to level 2, RPCC compares layer 1 and layer 2 with eight 4x1-based patterns at the same time. If there is no error in layer 1 and layer 2, RPCC adopts the left strategy to compress the 4x1 section. Otherwise, RPCC adopts right strategy. According to the simulation result with different thresholds, while the right strategy is adopted, the right strategy is often in worse case. We exploit this feature to improve the drawback in 4x1-based PCC algorithm.

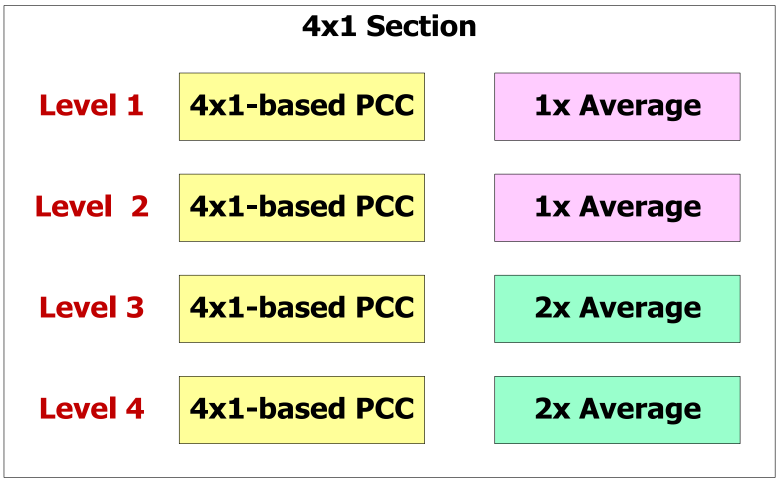


Figure 18 Reduced patterns comparison coding concept

The third part is the average coding scheme which deals with the two residual continuous bit planes after RPCC. We partition these bit planes into two 2x2-based parts and calculate the average value in each 2x2-based part. The coded data format is shown in Figure 19.

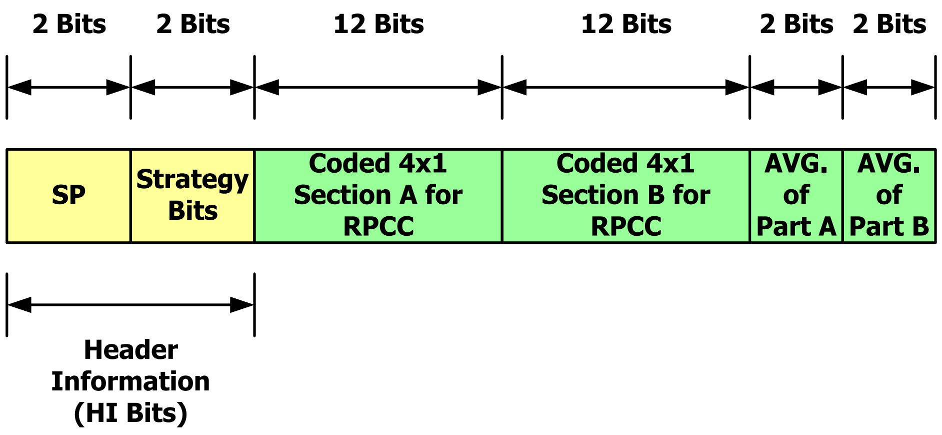

2 Bits
2 Bits
12 Bits
12 Bits
2 Bits
2 Bits
SP
Strategy Bits
Coded 4x1 Section A for RPCC
Coded 4x1 Section B for RPCC
AVG. of Part A
AVG. of Part B
Header Information (HI Bits)

Figure 19 The compressed 32-bit data format

(ii) Design of Patterns

For a 4x2 block, the bit plane consists of 8 bits, leading to 28 (= 256) possible number of bit planes. However, most of bit planes do not often appear in an image and contribute the less visual quality of decoded image. In addition, some different bit planes can provide the proximate visual quality. Thus, we focus on the design of a small set of visually sensitive predefined bit planes as shown in Figure 20. By inverting the polarization (0s and 1s) of predefined bit planes, eight patterns representing edges and lines are generated. Each pattern is represented by a 3-bit index as the number of patterns is eight.

0	0	0	0		1	1	1	0
1	0	0	0		1	1	0	0

Figure 20 Four predefined bit planes

(iii) Formula

We derive the formula (1) from the simulation result. It is about the PSNR loss of 4x1-based PCC algorithm. *i* is the number of 4x1 error bit plane. P_m is the error rate and P_n is the ratio of error rate per position in each 4x1 bit plane as described in Table 5. As described in the previous section, we can setup the different coding thresholds (Level 0~4) in 4x1-based PCC algorithm to obtain corresponding weight (W_i) as described in Table 6. We can exploit the formula to estimate for the PSNR loss in 4x1-based PCC algorithm while the previous parameters are modified.

$$\text{PSNR Loss (4x1 - based)} = \sum_{\mathrm{i}=0}^{4} \left[\mathrm{C}_{\mathrm{i}}^{4} \left(P_m \cdot P_n \right) \cdot \left(1 - P_m \cdot P_n \right)^{4-i} \right] \cdot W_i \qquad (1)$$

Table 5 Ratio of error rate per position in each 4x1 bit plane

W_i	Level 0	Level 1	Level 2	Level 3	Level 4
W_0	0	0	0	0	0
W_1	240	112	48	16	0
W_2	720	224	48	0	0
W_3	720	112	0	0	0
W_4	240	0	0	0	0

Table 6 Weights under different coding thresholds

P_n	Error Rate (%)	Total Ratio (%)
P_0	8.68	32.54
P_1	4.65	17.43
P_2	4.65	17.43
P_3	8.70	32.60

Figure 21 shows the distribution of PSNR loss in all thresholds. It helps improving the coding performance of the algorithm.

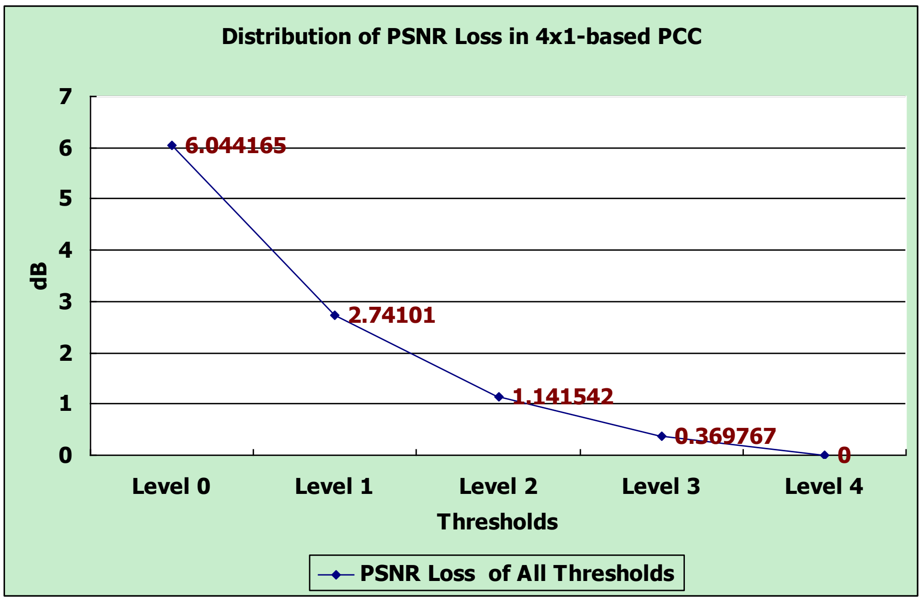


Figure 21 Distribution of PSNR loss in 4x1-based PCC algorithm

(2) *Proposed Architecture*

(i) Modified Bit Plane Truncation Coding

The hardware design of MBPTC is improved from original BPTC. It is a combinational block to deal with 4x2 pixels to obtain Start Plane (SP) and 4x2-plane component for each 4x2 array. In Figure 22, we employ three 8-input OR gates as thresholds to control the value of SP. The bits of layer 1, 2 and 3 are used to be input of 8-input OR gate individually.

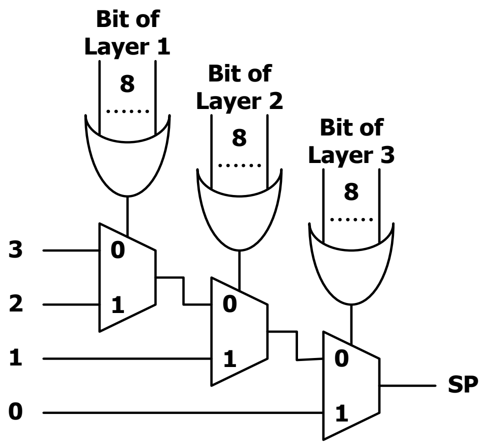


Figure 22 Hardware design for the MBPTC

(ii) Reduced Patterns Comparison Coding

RCPP is a combinational block to deal with coded data by MBPTC. As shown in Figure 23, SP selects four layers to be compressed and threshold is exploited to choose the strategy to be adopted. The SP is produced by MBPTC and the threshold is defined by users with different levels as described in Table 5. (Here we adopt Level 2)

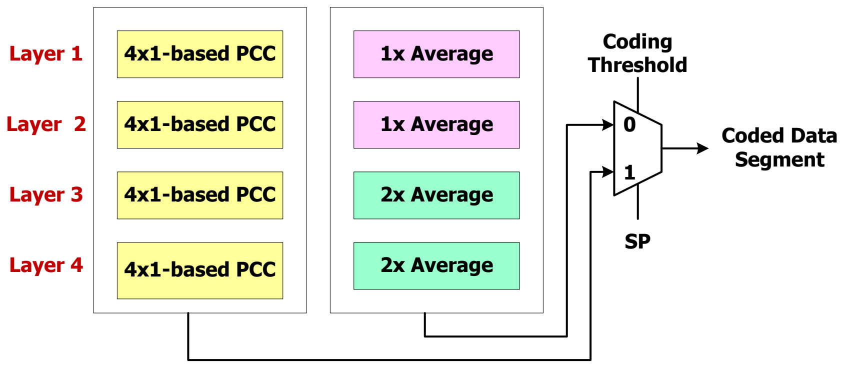


Figure 23 The hardware architecture of RPCC

(iii) Data Rearrange

Data unpacking is a simple reverse process of encoding. The decoder focuses on putting the data on proper positions. According to the coded data format, the SP selects the initial bit plane of decoding. The continuous four layers are then placed on corresponding positions depending on strategy bits. Afterward the average of part A and B is placed on the continuous two bit planes after the four layers.

(iv) Overall Design of Encoder and Decoder

The overall compressor design is shown in Figure 24. It takes one cycle to deal with 4x2 block. Here each MB takes 16 cycles to be encoded.

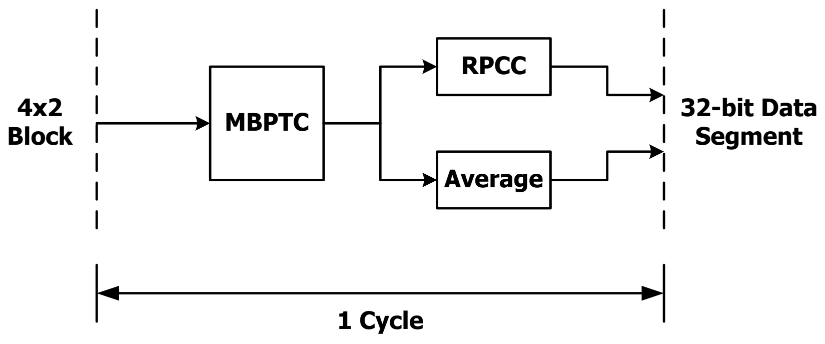


Figure 24 Data flow of the encoder

For providing data to MC, the decompressor needs to support higher throughput. The actual architecture of decompressor design is shown in Figure 25. A 4x2 block takes one cycle to be decoded. Under the design, each MB takes 16 cycles to be decoded.

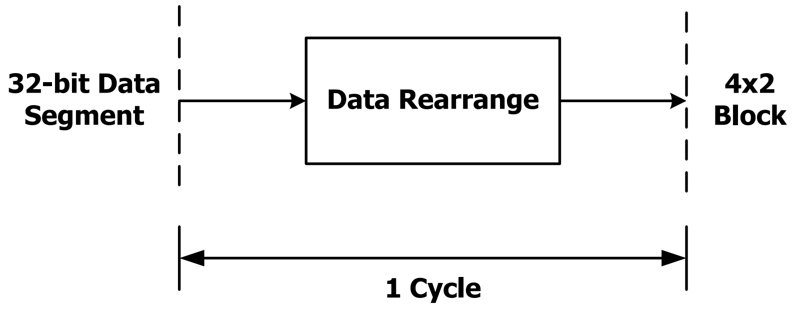


Figure 25 Data flow of the decoder

(3) *System Integration And Verification*

Figure 26 shows the overall block diagram of this system. The adopted H.264 decoder works at 5, 100 and 150 MHz respectively to perform CIF, HD 1080 AVC, HD 1080/720 SVC at 30 frames/per second (FPS). The embedded compressor compresses the data from deblocking filter into 64-bit data segment which is stored in external memory. The embedded decompressor decompresses the coded data segment from off-chip memory into 4x2-sized block which is sent to Motion Compensation. The bandwidth of system bus is 32 bits and the external memory is 32 bits per entry.

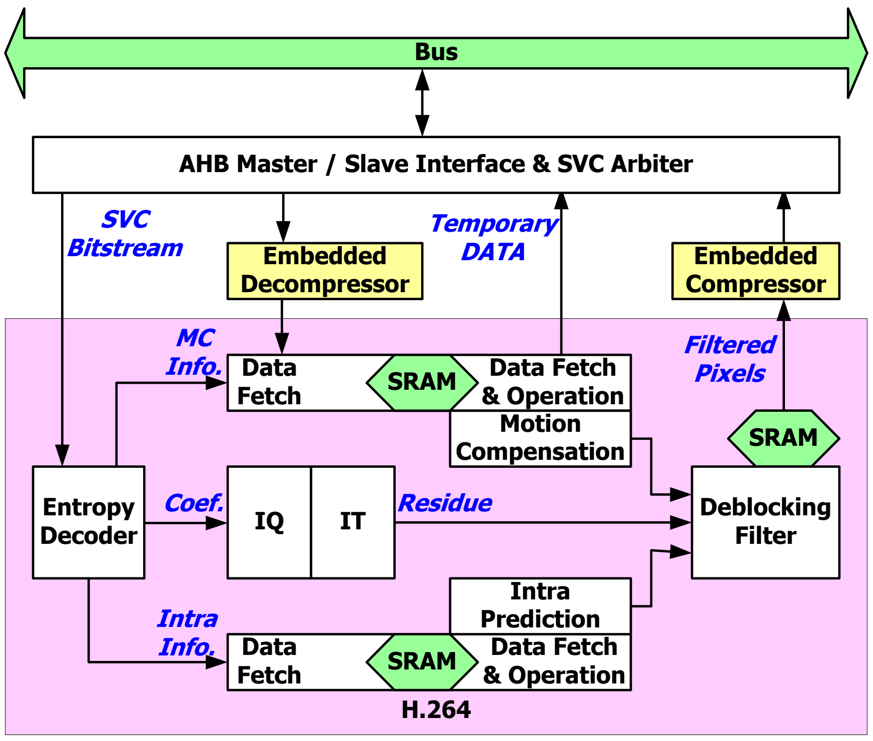


Figure 26 The architecture of our proposed H.264 decoder with EC capability

The related accesses of EC are partitioned into write accesses and read accesses. Write accesses from deblocking filter write the data to external memory and read accesses read the data from external memory to MC. Many methods have been proposed to improve embedded compression and all of them aim to improve the performance of embedded compression. However most of performance measured by these methods is fragmental, lacking verification from system level. In addition, we expect to precisely estimate the amount of read/write accesses on system view point. Thus, we employ "CoWare" to deal with the complicated problems. As shown in Figure 27, "CoWare" provides many functions to simulate a complete system and the user-defined means user's design. It makes more efficient that we can change the user-defined field relied on our demands. We add the proposed design and H.264 system into user-defined field. The AMBA interface between "CoWare" and user-defined is coded in System C and it provides a protocol to commutate each other. Furthermore, user-defined means all designs in this filed need to be coded in Verilog.

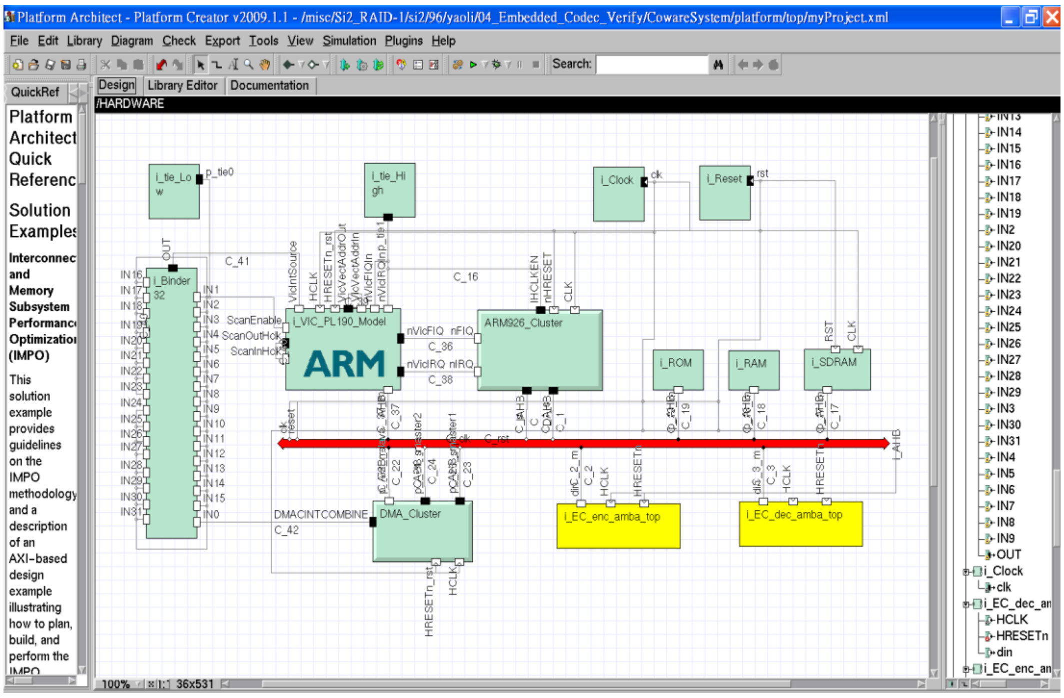

Platform Architect - Platform Creator v2009.1.1 - /misc/Si2_RAID-1/si2/96/yaoli/04_Embedded_Codec_Verify/CowareSystem/platform/top/myProject.xml
File Edit Library Diagram Check Export Tools View Simulation Plugins Help
Search:
QuickRef
Design
Library Editor
Documentation
/HARDWARE
Platform Architect Quick Reference
Solution Examples
Interconnect and Memory Subsystem Performance Optimization (IMPO)
This solution example provides guidelines on the IMPO methodology and a description of an AXI-based design example illustrating how to plan, build, and perform the IMPO
i_tie_Low
p_tie0
i_tie_High
i_Clock
clk
i_Reset
rst
i_Binder
C_41
C_16
i_VIC_PL190_Model
ARM
ARM926_Cluster
nVicFIQ
nFIQ
nVicIRQ
nIRQ
C_36
C_38
i_ROM
i_RAM
i_SDRAM
DMA_Cluster
DMACINTCOMBINE
C_42
i_EC_enc_amba_top
i_EC_dec_amba_top
i_AHB
100%
36x531
i_Clock
clk
i_EC_dec_amba_top
HCLK
HRESETn
din
i_EC_enc_amba_top

Figure 27 Block Diagram in CoWare simulation platform

Because the proposed algorithm provides fixed CR as two, the write access times after adding the EC are always half of original system. The reduction ratio of write access is 50%. The embedded decompressor decompresses the data from external memory to MC. Because the bandwidth of system bus is 32 bits and the external memory is 32 bits per entry, the original system takes 4x1 pixels as access unit. The read access behavior of MC with/without EC is analyzed as Table 6. The worst condition is the sub-pixel case. The 4x4 block needs a 9x9 block to complete the motion compensation. Thus, while original system needs 27 cycles to deal with this case, embedded compressor takes 15 cycles to do that. If the required 4x4 blocks of MC are aligned with the coded 4x4 blocks, original system with/without embedded compressor needs 2/4 cycles to deal with the best case. The special cases are included to (Align, Not Align), (Not Align, Not Align) and (Sub, Not Align). If required data of MC is not fit for 4x2 block-grids, it may increase extra access.

Table 7 All cases of read access required by MC with/without EC

Case of MV (x , y)	Access Cycles for System Without EC	Access Cycles for System With proposed EC	Reduction of Access Cycles Without EC (%)
(Align , Align)	4	2	50
(Align , Not Align)	4	2 / 3	50 / 25
(Align , Sub)	9	5	44.4
(Not Align , Align)	8	4	50
(Not Align , Not Align)	8	4 / 6	50 / 25
(Not Align , Sub)	18	10	44.4
(Sub , Align)	12	6	50
(Sub , Not Align)	12	6 / 9	50 / 25
(Sub , Sub)	27	15	44.4
AVG.	13.2	6.8 ~ 6.9	49.1 ~ 48.3

(4) *Simulation Result*

Software implementation of the proposed algorithm is integrated with JM 16.1. The reference frames are compressed by the proposed algorithm and then compared with those results from [15] and [16] respectively. The test sequences are akiyo, flower, football, foreman, mobile calendar, carphone, canoa, coastguard, waterfall and tempete in CIF format. For each sequence, computing the average PSNR value refers to the original sequence with 100 frames. Table 8 shows the comparison results. It can be found that our proposed solution based on non-transform compression scheme provides lower complexity and less gate count with acceptable PSNR loss and visual quality.

Table 8 Comparison of Simulation

		MHT + GRC [4]	DCT + MBPZC [5]	Proposed
Technology		UMC 90 nm	UMC 90 nm	UMC 90 nm
System		MPEG-2 Decoder	H.264 Decoder	H.264 / SVC
Processing Data Unit		8x1 Array (8 Pixels)	4x4 Array (16 Pixels)	4x2 Array (8 Pixels)
Working Frequency		100 MHz	100 MHz	100 MHz
Total Gate Count		20 K	30K	3.1 K
Cycle Count (Cycle)	***Encoder***	2 for first 1x8, 1 for Pipeline Stage	12 for first 4x4, 4 for Pipeline Stage	2 for each 4x4
	Decoder	2 for first 1x8, 1 for Pipeline Stage	4 for first 4x4, 2 for Pipeline Stage	2 for each 4x4
For a MB (Encoder/Decoder)		33 Cycles/33 Cycles	72 Cycles/34 Cycles	33 Cycles/32 Cycles
PSNR Loss		11.81 dB~14.57 dB	3.22 dB~8.39 dB	4.42 dB~7.10 dB (Average: 5.98)
Power Consumption		N/A	2.78 mW / 1.66 mW	228 uW / 130 uW

D. The Bitplane Truncation with Pattern Comparison Coding Embedded Compressor/Decompressor

(1) *Proposed BTPCC Embedded Compression Algorithm*

(i) Algotrithm

The proposed algorithm compresses a 4x2 block (64-bit) from the output of the deblocking filter. The CR is fixed at 2. After compressing, a 4x2 block will become a 32-bit segment. With fixed CR, the amount of the coded data is constant. Therefore, this compression can guarantee access times. Besides, in H.264 standard, a 4x4 block which is a basic coding unit can be partitioned into two 4x2 blocks.

Figure 28 shows the flowchart of the proposed compression algorithm. We divide the algorithm into four parts: 1) Pixel Truncation, 2) Selective Bitplane, 3) Rounding, and 4) Pattern Comparison. These parts will be described in the following paragraphs. The compressed 32-bit segment format is shown in Figure 29. The representation format consists of 2-bit Mode, 2-bit Start Plane (SP), 2-bit Decision L, 2-bit Decision R, 12-bit Coded Data L, and 12-bit Coded Data R.

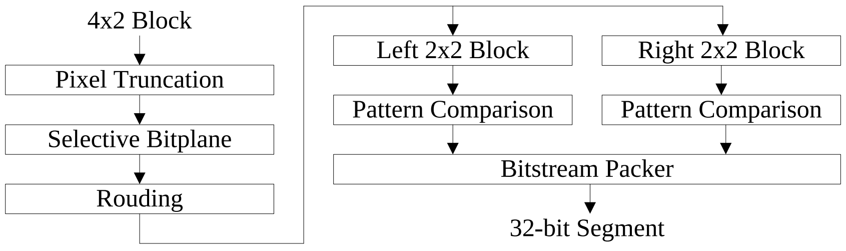


Figure 28 Compression flow of the proposed algorithm

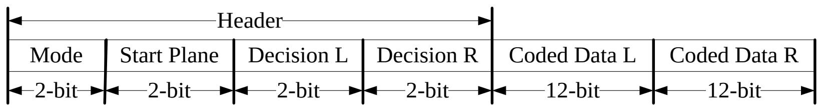


Figure 29 Compressed 32-bit segment format

A. Pixel Truncation

Figure 30 shows the flowchart of the pixel truncation. First, we calculate the average value (Avg.) of the 4x2 block and the difference value (Diff.) between maximum pixel and minimum pixel of the 4x2 block. Second, according to the average and the difference, we classify those 4x2 sub-blocks into five types as the following: 1) Avg. from 0 to 63 and Diff. less than 32, 2) Avg. from 64 to 127 and Diff. less than 64, 3) Avg. from 128 to 191 and Diff. less than 64, 4) Avg. from 192 to 255 and Diff. less than 32, and 5) no change. In type 1, if each pixel is larger than or equal to 64, we force the pixel to be 63. In type 2, if each pixel is less than 64, we force the pixel to be 64; if each pixel is larger than or equal to 128, we force the pixel to be 127. Types 3 and 4 are processed like types 2 and 1 respectively. In type 5, the original pixel value remains unchanged.

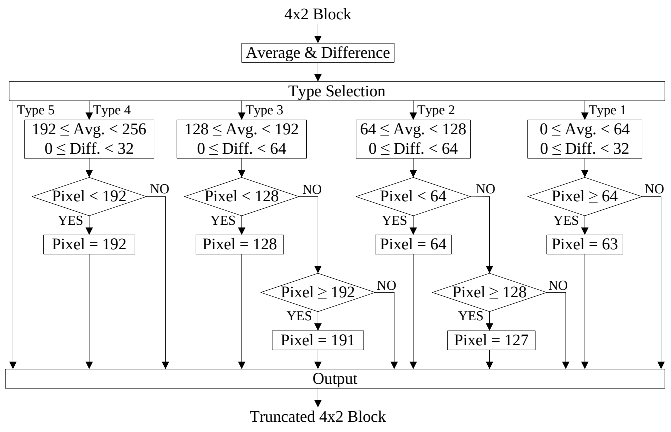


Figure 30 Flowchart of the pixel truncation

B. Selective Bitplane

Figure 31 shows the flowchart of the selective bitplane. Bitplane coding is a well-known method. We exploit bitplane as a basic unit to a group numbers, instead of pixel-wised basic unit. First, we consider a 4x2 block in which each pixel value is represented by 8-bit. A bitplane can be formed by selecting a single bit from the same position in the binary representation of each pixel.

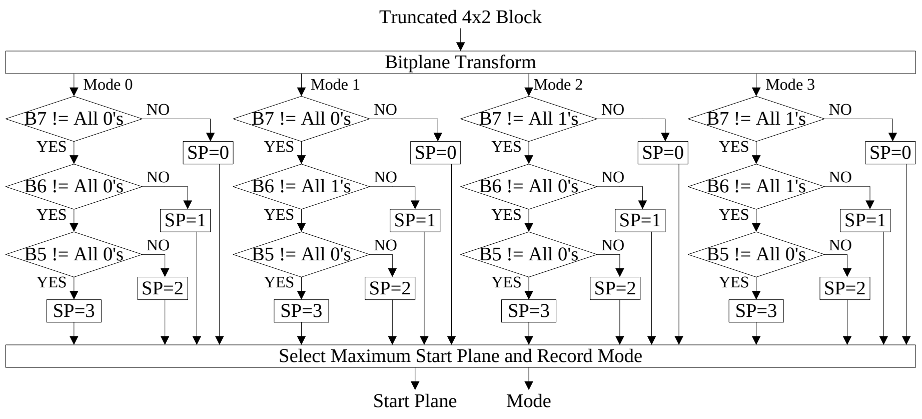
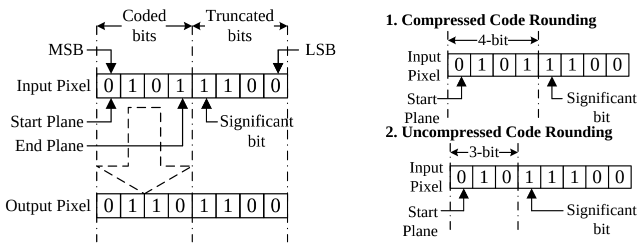


Figure 31 Flowchart of the selective bitplane

We define that B7 represents the MSB plane while B0 represents the LSB plane. Second, the start plane (SP) is searched for four successive bitplanes from the MSB bitplane with four modes as follows: 1) from B7 to B5 are all-0, 2) B6 is all-1; B7 and B5 are all-0, 3) B7 are all-1; B6 and B5 are all-0, and 4) B7 and B6 are all-1; B5 is all-0. In the first mode, if both B7 and B6 are all-0 and B5 is not all-0, then SP is equal to 1. Similarly, the other modes like as the first mode. Finally, the maximum start plane of four modes is selected to record the mode and start plane.

C. Rounding

Since lower bitplanes are truncated due to the limited budget, a simple rounding is applied here. The rounding is applied when the significant bit of the truncated bits is nonzero and the coded bits are not all 1's. In Figure 32(a), the simple idea is shown. This idea leads to a satisfied quality improvement. Two rounding modes are proposed because the pattern comparison has two data compressed formats. As shown in Figure 32(b), the first one is the compressed code rounding and the other is the uncompressed rounding. For pattern comparison, the first rounding method is applied to the first three types and the second rounding method is only for the final type.



(a) (b)

Figure 32 Flowchart of the rounding

D. Pattern Comparison

The final step encodes the preserving bitplanes. First, the truncated 4x2 block is partitioned into two 2x2 blocks that are called the left 2x2 block and the right 2x2 block as shown in Figure 33(a). In Figure 33(b), both the left 2x2 block and the right 2x2 block exploited the equal SP and compressed individually. Second, four types for a 2x2 block is classified as follows: 1) Group A, 2) Group B, 3) Group C, and 4) Uncompression. The first three types exploit a group of the eight patterns to compare with four successive bitplanes from SP and select one type which can hit three successive bitplanes. The three groups of the eight patterns are shown in Table I. If the first three types cannot hit larger than or equal to three bitplanes, the type 4 is chosen and three successive bitplanes from SP are stored.

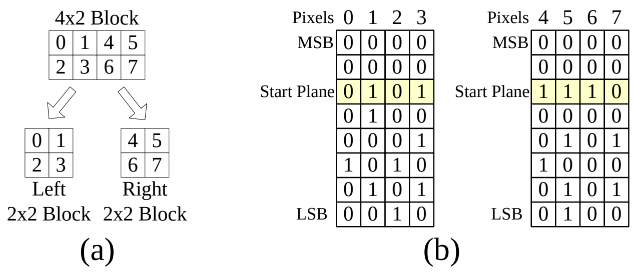


Figure 33 An example of partitioning 4x2 block

Table 9 Three Groups of Eight Patterns

Pattern No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Group A	0000	1111	1110	0111	0011	1100	0001	1000
Group B	0000	1111	1110	0111	1010	1001	0110	0101
Group C	0000	1111	1110	0111	1101	1011	0010	0100

(2) *Proposed BTPCC Embedded Compression Architecture*

(i) Compressor Design

Figure 34 shows the pipeline architecture of compressor design. We use two pipeline stages and each stage requires one cycle. The first stage is the pixel truncation. The second stage is composed of selective start plane, rounding, selective pattern comparison, and packer. This compressor encodes a 4x2 block in 2 cycles.

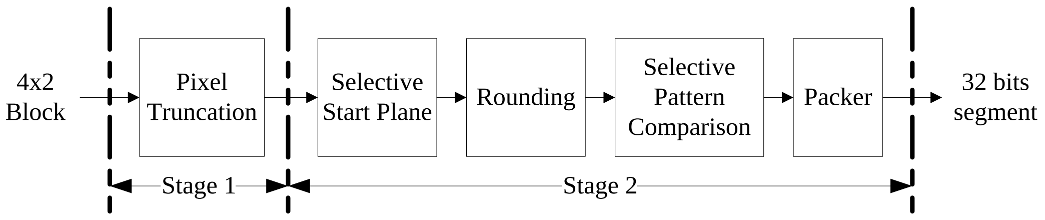


Figure 34 Compressor architecture

(ii) Decompressor Design

Figure 35 shows the pipeline architecture of decompressor. The decompressor only needs one stage with one cycle, including parser, start plane decoding, and pattern decoding. This decompressor reaches a higher throughput; therefore we can provide a higher random accessibility.

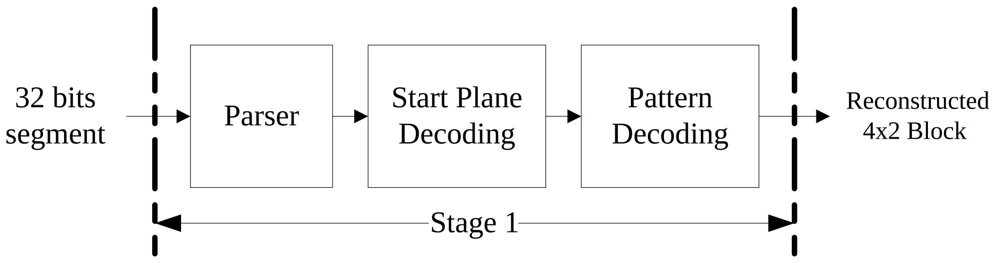


Figure 35 Decompressor architecture

(3) *System Integration*

The overall H.264 decoder [31] with the embedded compression codec is shown in Figure 36. The embedded compressor works between the deblocking filter and the external memory. The embedded decompressor works between the external memory and the motion compensation. To design address controller of EC is very simple since our compression ratio is fixed at two. Our system bus is 32 bits and the external memory is 32 bits per entry.

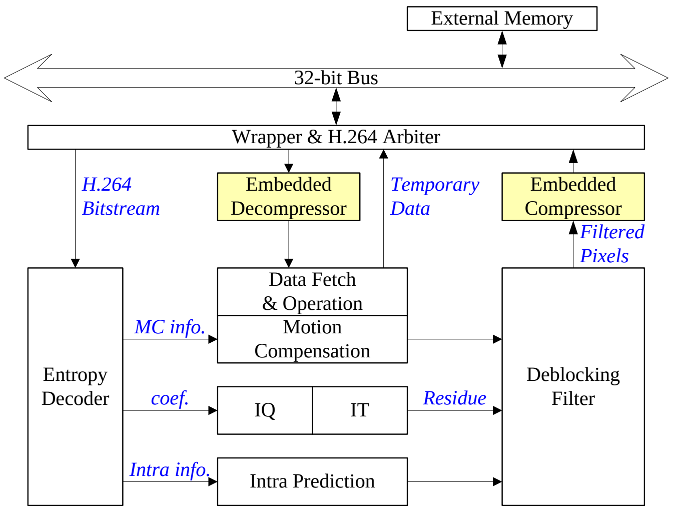


Figure 36 H.264 decoder with proposed embedded compression

The compatible H.264 decoder specification is HD1080+HD720@30fps and works at 150MHz. The compressor converts a 4x2 block from the deblocking filter into a 32 bits segment which is stored into the external memory. Comparing the data access times of the external memory for the system without EC, the data access times of our system is half. The decompressor converts a 32 bits segment into a 4x2 block which is sent to the motion compensation. Since our system bus is 32 bits and the external memory is 32 bits per entry, the system accesses once a data as four pixels. In Table II, we analyze the read times of the motion compensation with/without EC. The worst case is the (Sub, Sub) case. To finish the motion compensation, a 4x4 block needs a 9x9 block. Therefore, the system with/without proposed embedded compressor takes 15/27 cycles. The best case is the (Align, Align) case. Original system with/without embedded compressor needs 2/4 cycles to finish the best case. For the other cases when the required data of motion compensation are not fit for 4x2 block-grids, the access times become increased.

Table 10 All Cases of Read Access Requirement

Case of MV (x, y)	**Access Cycles for System without EC**	**Access Cycles for System with Proposed EC**	**Reduction of Access Cycles without EC**
(Align, Align)	4	2	50
(Align, Not Align)	4	2/3	50/25
(Align, Sub)	9	5	44.4
(Not Align, Align)	8	4	50
(Not Align, Not Align)	8	4/6	50/25
(Not Align, Sub)	18	10	44.4
(Sub, Align)	12	6	50
(Sub, Not Align)	12	6/9	50/25
(Sub, Sub)	27	15	44.4
Average	13.2	6.8~6.9	49.1~48.3

(4) *Experimental Results*

Table 11 PSNR Comparison shows the software result of the proposed algorithm which is integrated with JM16.2. The test sequences are Akiyo, Forman, Mobile, Stefan, and Station. Each test sequence executes 100 frames. And then the average PSNR value is calculated. Results show that the PSNR loss of the proposed algorithm is from 1.27 to 3.94dB.

Table 11 PSNR Comparison

Sequence	**Format**	**H.264 (dB)**	**Proposed (dB)**	**PSNR loss**
Akiyo	CIF	43.72	41.16	2.56
Forman	CIF	41.23	39.20	2.03
Mobile	CIF	37.61	34.14	3.47
Stefan	CIF	38.82	34.88	3.94
Station	HDTV	39.12	37.84	1.27

Table 12 shows the comparison among previous work. It can be found that our proposed hardware provides less hardware complexity and better visual quality. Especially, the proposed decoder just requires one cycle with higher random accessibility for embedded compression without degrading overall system performance. The power consumption of the proposed hardware is better than Lee's [16] and Wu's [31]. Figure 37 shows the Station sequence result of the original system with EC in HDTV format. The propagation of quality loss is unavoidable but video quality remains acceptable.

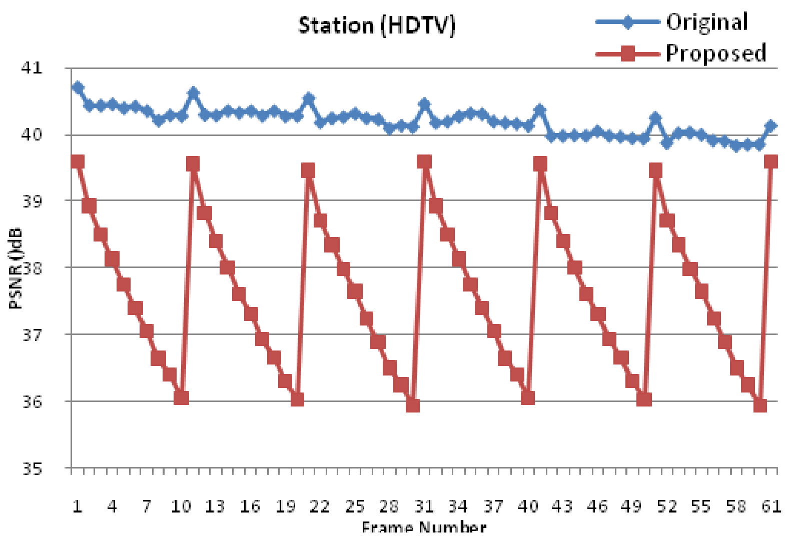


Figure 37 Simulation result of station sequence (HDTV)

Table 12 Comparison Among Previous Work

		Lee’s [3]	Wu’s [4]	This Work
Technology		CMOS 0.25um	UMC 90nm	UMC 90nm
System		MPEG-2 Decoder	H.264 Decoder	H.264/SVC Decoder
Working Frequency		100MHz	100M z	150MHz
Processing Data Unit		8x1 Block	4x4 Block	4x2 Block
T tal Gat C unt		20k	30k	4.9k
Cycle Count for 4x2 Block	**Encoder**	2 cycles	N/A	2 cycles
	Decoder	2 cycles	N/A	1 cycle
Cycle Count for a MB	**Encoder**	33 cycles	72 cycles	33 cycles
	Decoder	33 cycles	34 cycles	32 cycles
PSNR Loss		6.08dB~10.65dB	1.31dB ~ 4.48dB	1.27dB ~ 3.94dB
Power Consumption	**Encoder**	N/A	2.78mW	158uW
	Decoder	N/A	1.66mW	86uW

*The N/A is because of the processing data unit is 4x4 block in MBPZC

E. An Area-efficiently High-accuracy Prediction-based CABAC Decoder for H.264/AVC

(1) *Proposed prediction scheme:*

To replace SE Predictor [25], we used combinational logic and several buffer stored status of each stage (just like Figure 38). Because the bottleneck of throughput depended on the hit rate in the prediction-based CABAC decoder, we let the decoding flow more regular except the case we ready should know what the decoded bin is. In the previous work [25], they used MPS-based two-bit predictor to predict current bin, and it obtained about 70% hit rate. The 30% miss rate is the critical part decreased the performance. So, we enhance the average hit rate by our proposed prediction process.

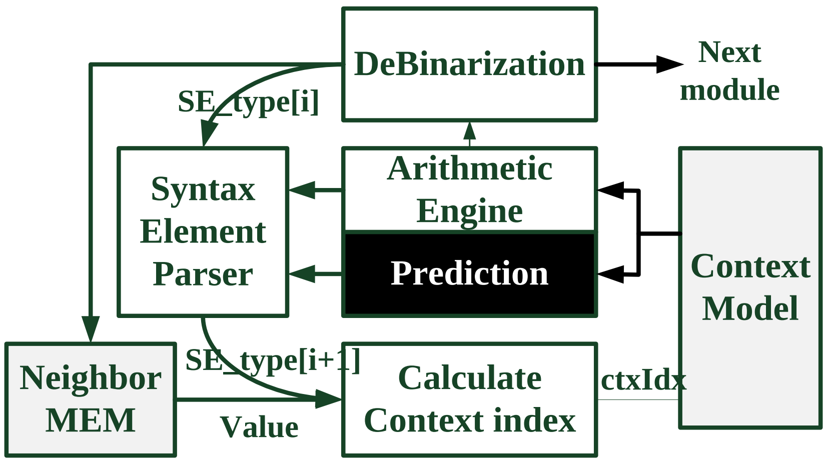


Figure 38: Block Diagram of Proposed CABAC Decoder

On the other hand, we often produce extra overhead for getting neighbour information and requesting data from external memory. So, we optimize our *memory system* to solve this problem. We prefetched and compressed the neighbour information to reduce the storage and the latency.

A. Prediction Process

This section we proposed three methods to raise throughput at limited resource by single-bin engine.

Raised Hit Rate – In the Figure 39(a) and Figure 39(b), they are traditional arithmetic engine decoding flow. In the beginning, we have current value of Range and Offset before we decoded the bin. After we read the value of LPS range, we can know offset is in the field of MPS or LPS. In fact, we can recognize the trend that the bin may be MPS before we decoded the bin. Before we get the LPS range, we already have current value of Range and Offset. So, if we observe the difference between Range and Offset, we can find out when difference is the larger and MPS rate is the higher. And then, following this principle we use two bits according as Table 13 to recognize which status is mostly happened. We can make sure next bin at status “00” and ‘11”, because the value are over the limit of standard. This result can significantly raise hit rate, but the other status still cause prediction miss. Actually, we also can use this method to pState and so on. The extra two bits can raise hit rate when we are at status “01” or “10” and can produce at the same time when processing difference. So, it couldn’t increase critical path in our prediction process. Finally, we can make sure at status “00” and “11” and have four-bit tips to predict what next bin is at status “01” and “10”. Fortunately, we overall got more than 90% hit rate in our simulation results by our prediction process.

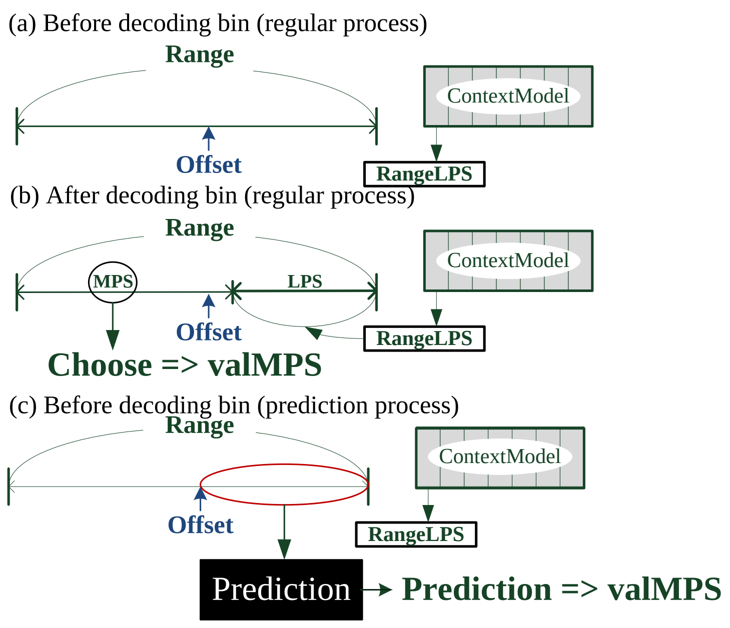


Figure 39: (a) Before decoding bin (regular process) (b) After decoding bin (regular process) (c) Before decoding bin (prediction process)

Table 13: Status of Difference

Status	***Value(pState)***	***Possible***
00	*Diff. < value(0)	LPS
01	value(0) ≦*Diff. < value(31)	LPS
10	value(31) ≦ *Diff. ≦ value(62)	MPS
11	*Diff. > value(62)	MPS

*Diff. = Range – Offse

(2) Reduced Stall Time – Furthermore, we still try how to reduce unnecessary stall times. Because of regular decoding flow, our prediction process depends on predicted bin to calculate ctxIdx, we may get miss penalty when predicting miss. Actually, not all of syntax element (SE) branch point need previous bin. So, we collect all of SE's finished bin and branch selection to know when we really should stall or not.

(3) Solved Data Hazard – On the other hand, pipeline architecture may cause unavoidable data hazard. When we calculate next ctxIdx, we may need neighbor block SE. Different with traditional decoding flow we avoid the worst case which we must stall to wait decoded bin. So, the other reasons we get data hazard are data still handled by other stage and data haven't updated to memory. The prefix one can be solved by forward path and suffix one can be solved by data reused (used buffer to hold on data). So, most of data hazard can be solved by our improvement.

B. Memory System

This section we proposed three methods to reduce cost, latency and memory bandwidth requirement.

(1) ***Solved Syntax Element Switching Overhead (SESO)*** –In the previous work [3], we can clearly understand the effect of SESO that seriously decreased performance. In the other issue, getting neighbor information may enhance the latency when we access data from external memory. When we request data from system bus, the latency may exceed over our estimate significantly. The reasons could be system clock are asynchronous with external memory, other module compete memory bandwidth requirement especially for Motion Compensation and so on. So, this may be a potential problem even if original storage aren’t huge than other module. The immediate solutions are included an internal memory to store all information we need, but this method will get large overhead when we upgrade to HD sequences. We should include almost 20 Kbits SRAM even double in MBAFF for CABAC. So, this is inefficient to implement. To overcome these unexpected problems, the best solutions are reduced the stored neighbor information. In our works, we prefetch the data to simplified buffer (described in following paragraph) when decoding first SE of MacroBlock (MB) and precalculate SE for neighbor MB. Therefore, we adjust the latency for getting neighbor data to average 1.333 cycles/MB.

(2) ***Reduced MEM. BW. Occupation*** – In our analysis, motion vector different (mvd) is the biggest part of all neighbor information. However, we shouldn’t use the total bits to calculate next ctxIdx. We show the control conditions at the Figure 40(a) from standard, and we find out most of cases can be determined by two bits. So, we store each mvd from 10 bits to 2 bits and use 5 bits to store extra mvd when it’s bigger than 3 just like Figure 40(b). When we need to refer the mvd, we should access 2 bits mvd and extra mvd which is bigger than 3from external memory. And, in our analysis, most of mvd are smaller than 2. Finally, we can efficiently reduce about 70% storage in our simulation by this method.

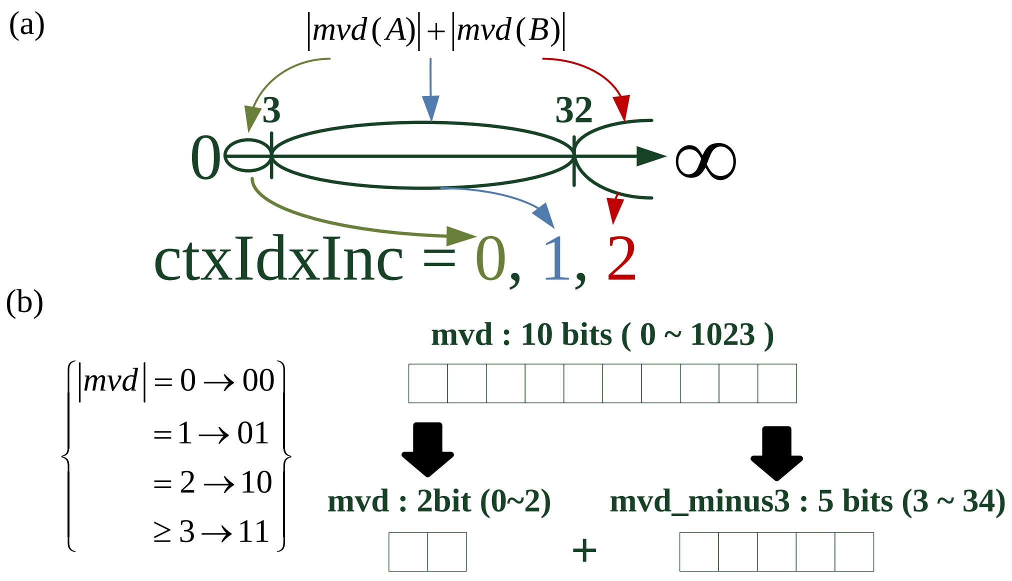


Figure 40: (a) ctxIdxInc control condition for mvd ; (b) proposed mvd reduction scheme

(3) Raised Buffer Efficiency – When we decode first bin of SE, we should access the same SE at the neighbor block which located in current MB or neighbor MB. Immediately, we need two kinds of buffer to handle data from neighbor MB or current MB and both of them occupy the largest percentage of buffer. Actually, we can combine two kinds of buffer to raise area efficiency. After we read the data from the buffer, some buffer can be written. Because of this way and accurately scheduling we can reduce cost certainly.

(2) *Proposed prediction-based Architecture*

This section implements our proposed prediction-based CABAC decoder architecture by our proposed scheme, as Figure 41.

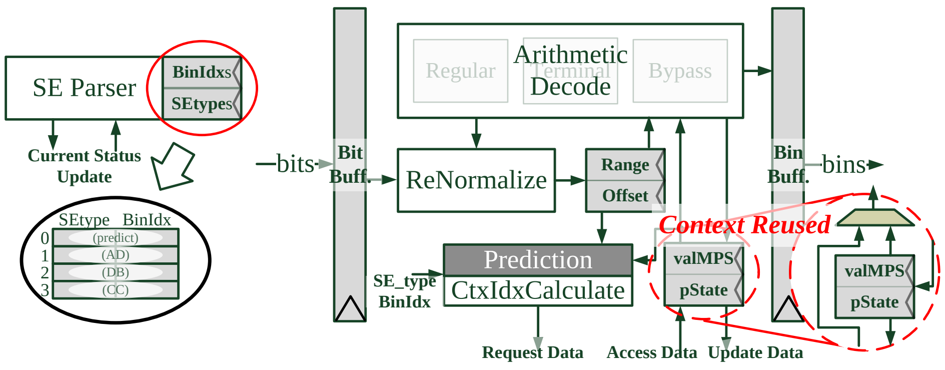


Figure 41: Architecture of Proposed Prediction Process

First, we parallel the bin-decoded process and ctxIdx- calculated process by two independent paths, and we record each status of pipeline stages in shift register from SE parser. So, we can break the data dependency by prediction process and controlled SE parser. By the way, we use multiplexer and buffers to choose context (valMPS and pState) or up which can be data reused or not.

Second, we may find a problem when we calculate the ctxIdx. That is we need neighbour information to calculate current ctxIdx. So, we increase one stage to deal with this problem and are shown in Figure 42. Actually, syntax elements are produced after second stage, and the third stage can't affect the pipeline. In the other problems, the mvd which is bigger than 3 is too dispersed to achieve high storage reduction. So, we implement a transfer unit to compress data. This unit compresses at most 6 5-bit mvd to adaptive 32-bit BUS, and the transmission times can be reduced significantly. Oppositely, we should increase an inverse transfer unit to decompress data when we need to reference data.

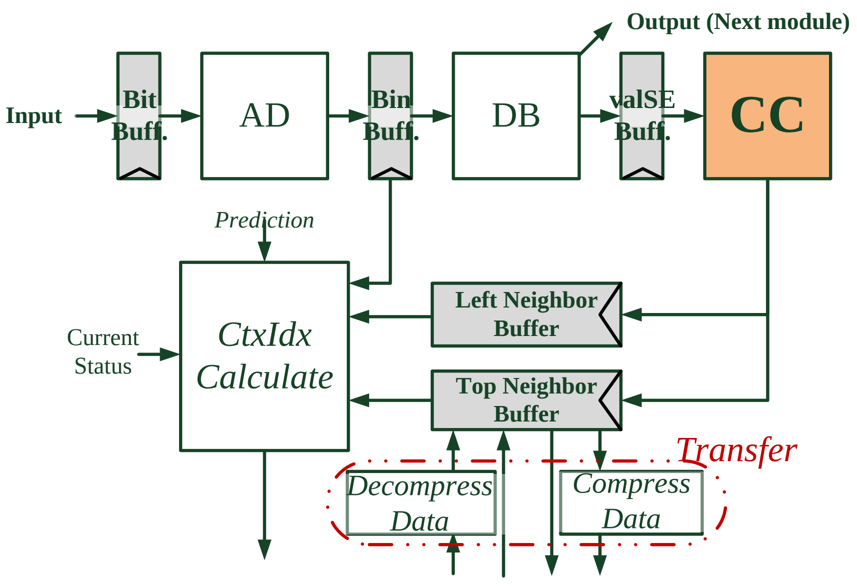


Figure 42: Architecture of Proposed Memory System

Third, we integrate the prediction process (Part A) and memory system (Part B) to our proposed CABAC decoder which is shown in Figure 43. In our design, this may cause data hazard problem in some special cases or SEs, for example, coded_block_pattern. Therefore, we should increase a forward process to deal with this problem and use multiplexer to select regular path or forward path. On the other hand, we should have a unit to deal with miss penalty. When we predict miss, the status registers can't be shifted. We should stall one cycle to calculate correct ctxIdx, and we use previous bin instead of predicted bin. Finally, we make a FSM to control initialization process and decoding process.

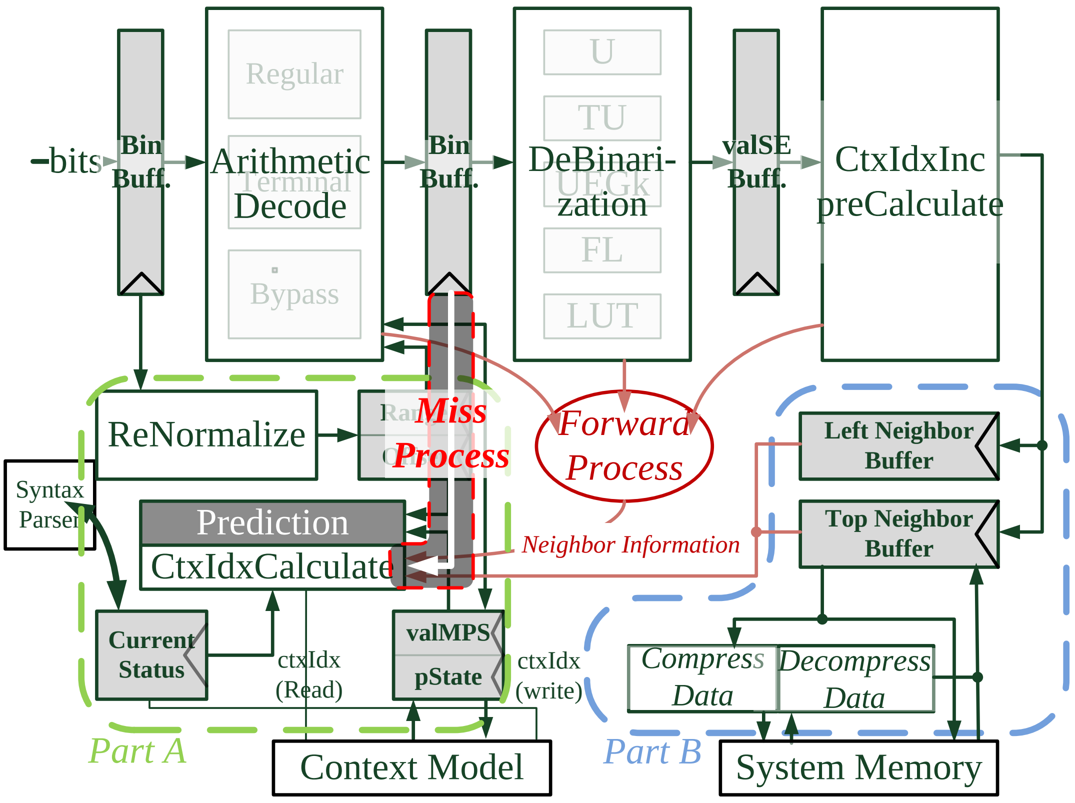


Figure 43: Architecture of Proposed CABAC Decoder

(3) *Experimental result*

To prove our prediction algorithm can adapt in different sequences, we make a simulation that we calculate hit rate at each bin of sequences in average. Table 14 shows average hit rate of the same resolution sequences, and we get hit rate from 90.75% to 94.27%. It means even in the worst case we still can keep more than 90% hit rate. Therefore, we raise higher performance according to formula of throughput than previous work [25] at the same conditions and mechanism. (All the sequences are encoded by JM 16.1[28] with 4:2:0 color format, QP: 28, GOP: IBP, Max. bit-rate and frame rate of 30 fps.)

Table 14: Hit Rate of Prediction Process

	*N	Original MPS Rate	Proposed Hit Rate	Increased Hit rate
QCIF	21	71.98%	91.70%	19.88%
CIF	22	73.36%	91.99%	18.63%
HD	6	78.42%	93.10%	14.68%
AVG.	49	73.27%	91.95%	18.68%

*N: number of test sequence

In other works, we reduce the storage and test in full-HD sequences. We count the range distribution of mvd and show our results which compared with our optimal and traditional solutions in Figure 44. And, we get reduction rate from 58.01% to 75.13%. It means we can reduce bandwidth requirement or system buffer utilities efficiently.

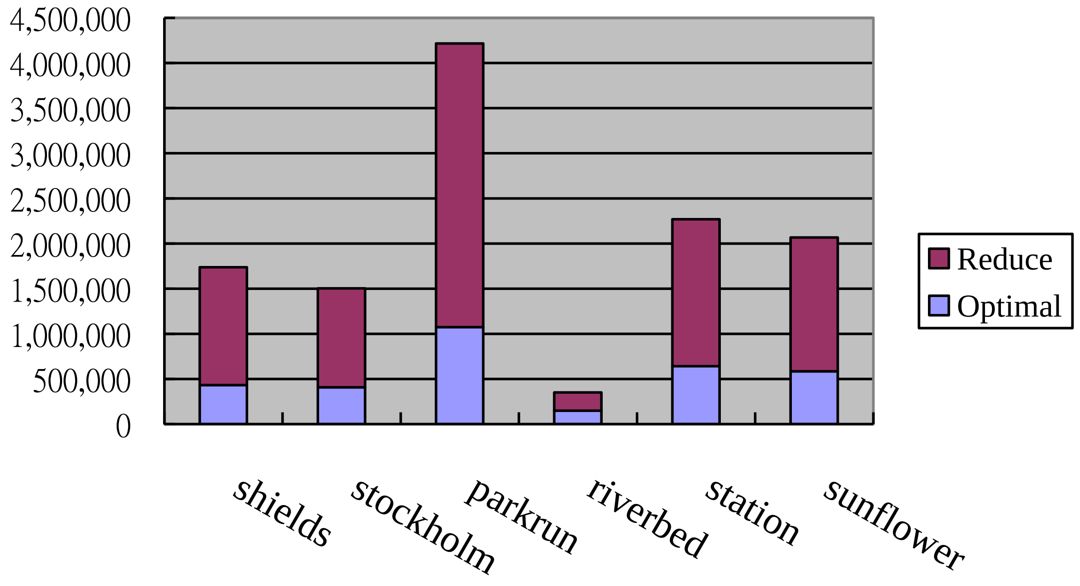


Figure 44: Reduce Rate of Memory System

The RTL simulation result shows that the proposed design can decode 0.95 bins per cycle in average. The synthesis results of proposed architecture and a performance comparison with previous works are shown in Table 15. By applying the mechanisms, the proposed architecture can efficiently reduce at least 45% hardware cost than previous works. However, we still can achieve Level 5.0 MP. The maximum throughput of the proposed design is 239.4 Mbins/s and the maximum working frequency is 249 MHz.

Table 15: Comparison of the Proposed Design and Previous Works

		CSVT 09'[27]	ISCAS 10'[28]	ISCAS 09'[26]	ISCAS 08'[25]	Proposed
Spec.		H.264/HP 1920x1088 @30fps	1920x1088 @30fps	1920x1088 @30fps	1920x1088@25fps	**H.264/AVC 1920x1088@30fps**
Technology		TSMC 0.18 um	UMC 90nm	UMC 90nm	N/A	**UMC 90nm**
Frequency		105 MHz (MAX:140 MHz)	MAX:264 MHz	MAX:222MHz	N/A	**150 MHz (MAX: 249MHz)**
Mechanism		Multi-Bin	Multi-Bin	Multi-Bin	Prediction-based	**Prediction-based**
Gate Count	**w/o Context Model**	34,955	N/A	N/A	N/A	**16,291 (17,838@249MHz)**
	with Context Model	76,333	42,372	82,400	N/A	**23,303 (25,828@249MHz)**
MEM. System	**Get neighbor Delay**	N/A	N/A	N/A	N/A	**1.333 idle/MB (*idle : 1 cycle)**
	Context Model	SRAM (3,528 bits)	Hybrid SRAM	Register File	N/A	**SRAM (3,360 bits)**
Average Bins/Cycle		0.71 (740x480@4Mb/s) 0.86(1920x1088@60Mb/s)	1.83	1.95 ~ 1.98	0.8333 (Hit Rate: 71.4%)	**0.9467 (Hit Rate: 91.95%)**
Maximum Throughput		120.4 (@ 140Mhz)	483.1 (@ 264MHz)	410.0 (@ 222MHz)	N/A	**239.4 (@249MHz)**

F. A Predefined Bit-Plane Comparison Coding for Mobile Video Applications

(1) *Proposed Embedded Compression Algorithm*

The proposed algorithm compresses a 4x2 block (64-bit) from the output of the deblocking filter. The CR is fixed at 2. After compressing, a 4x2 block will become a 32-bit segment. With fixed CR, the amount of the coded data is constant. Therefore, this compression can guarantee access times. Besides, in H.264 standard, a 4x4 block which is a basic coding unit can be partitioned into two 4x2 blocks.

Figure 45 shows the flowchart of the proposed compression algorithm. We divide the algorithm into four parts: 1) Pixel Truncation, 2) Selective Bitplane, 3) Rounding, and 4) Pattern Comparison. These parts are described in the following paragraphs. The compressed 32-bit segment format is shown in Figure 46. The representation format consists of 2-bit Mode, 2-bit Start Plane (SP), 2-bit Decision L, 2-bit Decision R, 12-bit Coded Data L, and 12-bit Coded Data R.

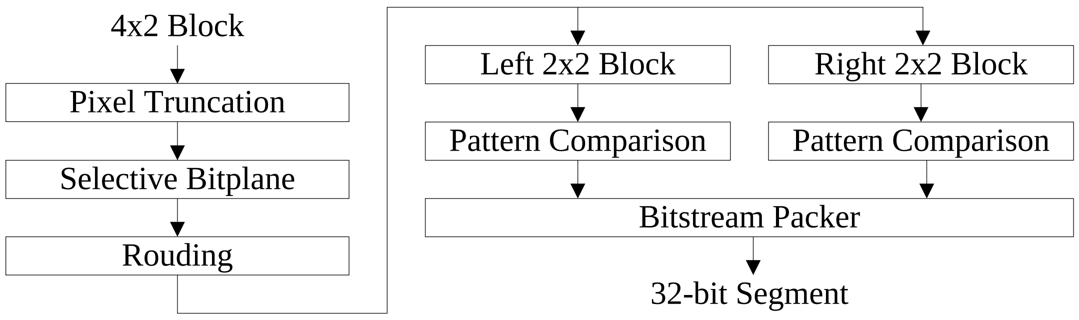


Figure 45: Compression flow of the proposed algorithm

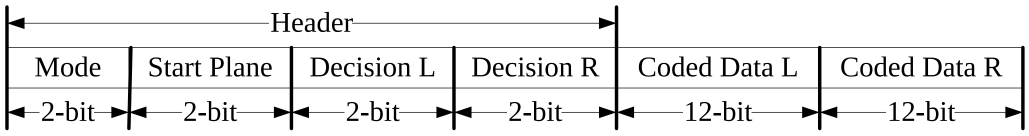


Figure 46: Compressed 32-bit segment format

A. *Pixel Truncation*

Figure 47 shows the flowchart of the pixel truncation. First, we calculate the average value (Avg.) of the 4x2 block and the difference value (Diff.) between maximum pixel and minimum pixel of the 4x2 block. Second, according to the average and the difference, we classify those 4x2 sub-blocks into five types as the following: 1) Avg. from 0 to 63 and Diff. less than 32, 2) Avg. from 64 to 127 and Diff. less than 64, 3) Avg. from 128 to 191 and Diff. less than 64, 4) Avg. from 192 to 255 and Diff. less than 32, and 5) no change. In type 1, if each pixel is larger than or equal to 64, we force the pixel to be 63. In type 2, if each pixel is less than 64, we force the pixel to be 64; if each pixel is larger than or equal to 128, we force the pixel to be 127. Types 3 and 4 are processed like types 2 and 1 respectively. In type 5, the original pixel value remains unchanged.

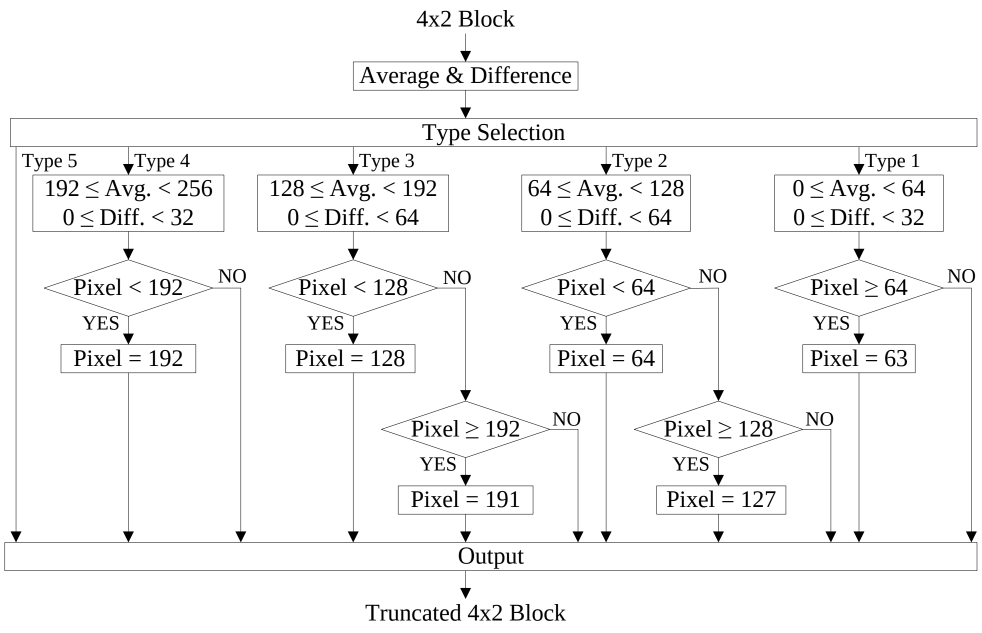


Figure 47: Flowchart of the pixel truncation

B. *Selective Bitplane*

Figure 48 shows the flowchart of the selective bitplane. Bitplane coding is a well-known method. We exploit bitplane as a basic unit to a group numbers, instead of pixel-wised basic unit. First, we consider a 4x2 block in which each pixel value is represented by 8-bit. A bitplane can be formed by selecting a single bit from the same position in the binary representation of each pixel.

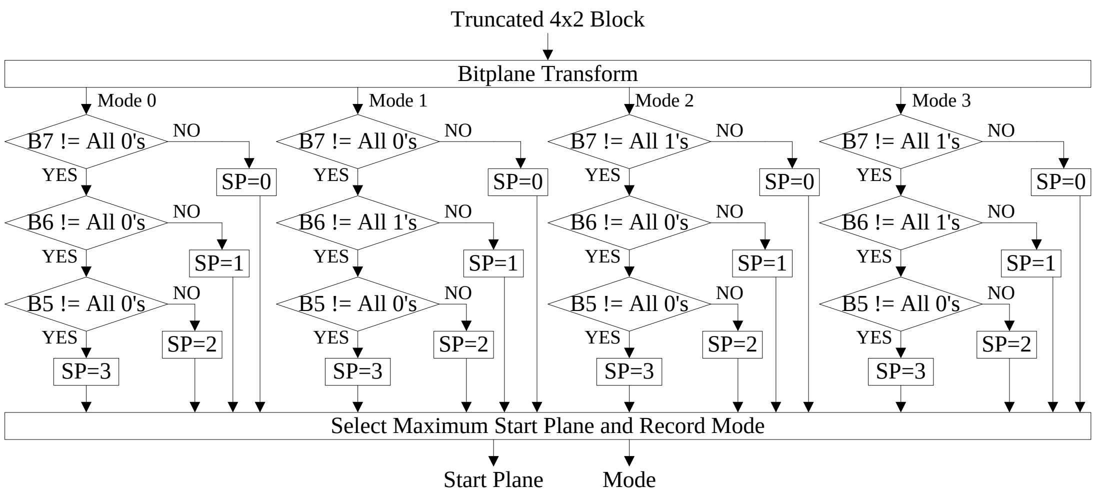
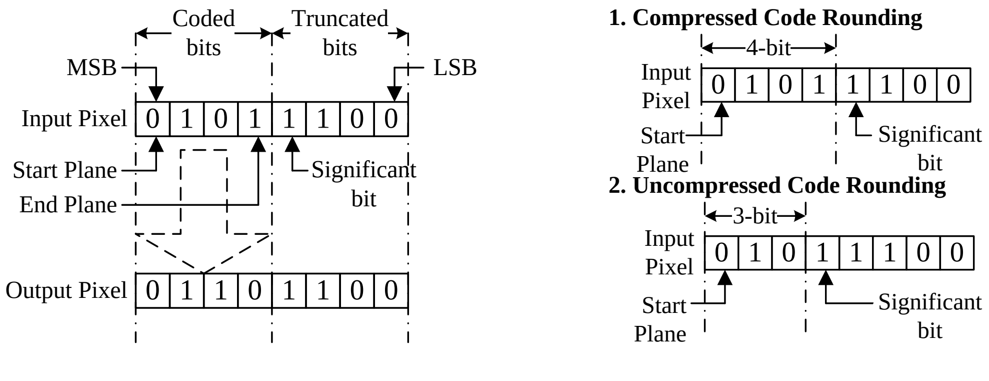


Figure 48: Flowchart of the selective bitplane

We define that B7 represents the MSB plane while B0 represents the LSB plane. Second, the start plane (SP) is searched for four successive bitplanes from the MSB bitplane with four modes as follows: 1) from B7 to B5 are all-0, 2) B6 is all-1; B7 and B5 are all-0, 3) B7 are all-1; B6 and B5 are all-0, and 4) B7 and B6 are all-1; B5 is all-0. In the first mode, if both B7 and B6 are all-0 and B5 is not all-0, then SP is equal to 1. Similarly, the other modes like as the first mode. Finally, the maximum start plane of four modes is selected to record the mode and start plane.

C. *Rounding*

Since lower bitplanes are truncated due to the limited budget, a simple rounding is applied here. The rounding is applied when the significant bit of the truncated bits is nonzero and the coded bits are not all 1's. In Figure 49(a), the simple idea is shown. This idea leads to a satisfied quality improvement. Two rounding modes are proposed because the pattern comparison has two data compressed formats. As shown in Figure 48(b), the first one is the compressed code rounding and the other is the uncompressed rounding. For pattern comparison, the first rounding method is applied to the first three types and the second rounding method is only for the final type.



(a) (b)

Figure 49: Flowchart of the rounding

D. *Pattern Comparison*

The final step encodes the preserving bitplanes. First, the truncated 4x2 block is partitioned into two 2x2 blocks that are called the left 2x2 block and the right 2x2 block as shown in Figure 50(a). In Figure 50(b), both the left 2x2 block and the right 2x2 block exploited the equal SP and compressed individually. Second, four types for a 2x2 block is classified as follows: 1) Group A, 2) Group B, 3) Group C, and 4)

Uncompression. The first three types exploit a group of the eight patterns to compare with four successive bitplanes from SP and select one type which can hit three successive bitplanes. The three groups of the eight patterns are shown in Table 16. If the first three types cannot hit larger than or equal to three bitplanes, the type 4 is chosen and three successive bitplanes from SP are stored.

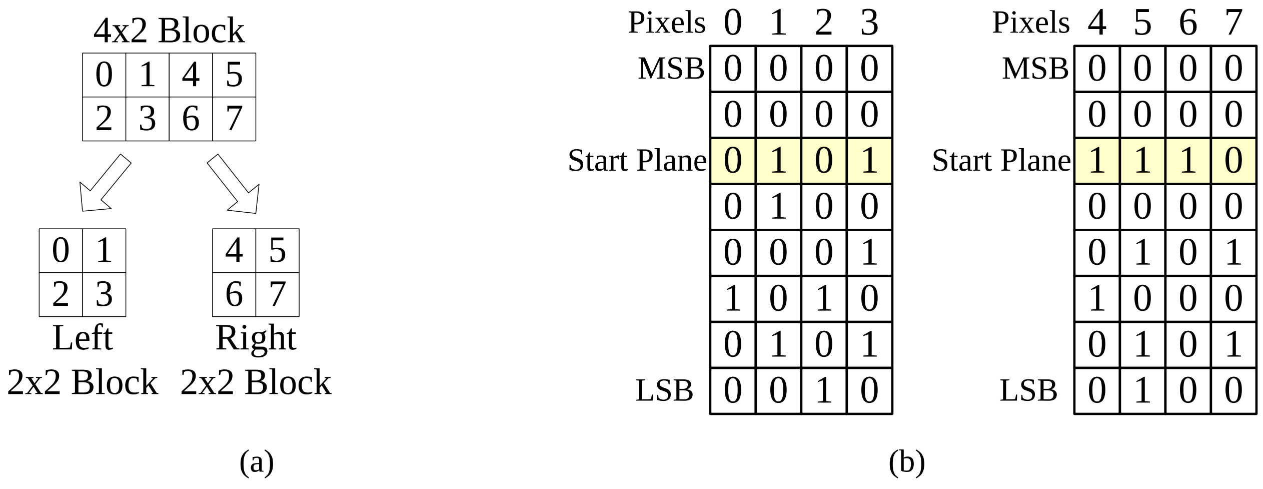


Figure 50: An example of partitioning 4x2 block

Table 16: Three Groups of Eight Patterns

Pattern No.	1	2	3	4	5	6	7	8
Group A	0000	1111	1110	0111	0011	1100	0001	1000
Group B	0000	1111	1110	0111	1010	1001	0110	0101
Group C	0000	1111	1110	0111	1101	1011	0010	0100

(2) *Proposed Architecture*

A. Compressor Design

Figure 51 shows the pipeline architecture of compressor design. We use two pipeline stages and each stage requires one cycle. The first stage is the pixel truncation. The second stage is composed of selective start plane, rounding, selective pattern comparison, and packer. This compressor encodes a 4x2 block in 2 cycles.

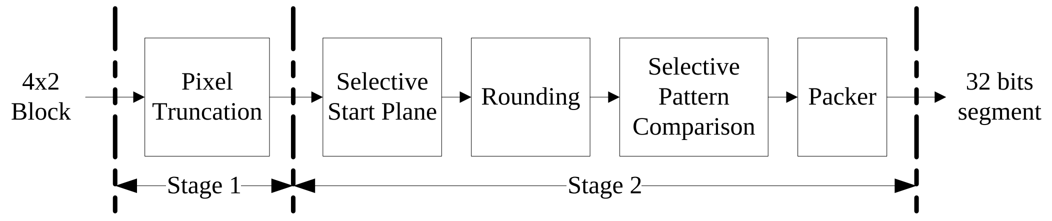


Figure 51: Compressor architecture

B. Decompressor Design

Figure 52 shows the pipeline architecture of decompressor. The decompressor only needs one stage with one cycle, including parser, start plane decoding, and pattern decoding. This decompressor reaches a higher throughput; therefore we can provide a higher random accessibility.

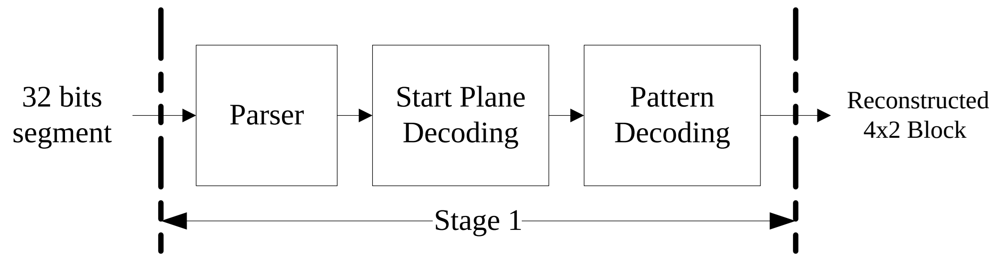


Figure 52: Decompressor architecture

(3) *System Integration*

The overall H.264 decoder with the embedded compression codec is shown in Figure 53. The embedded compressor works between the deblocking filter and the external memory. The embedded decompressor works between the external memory and the motion compensation. To design address controller of EC is very simple since our compression ratio is fixed at two. Our system bus is 32 bits and the external memory is 32 bits per entry.

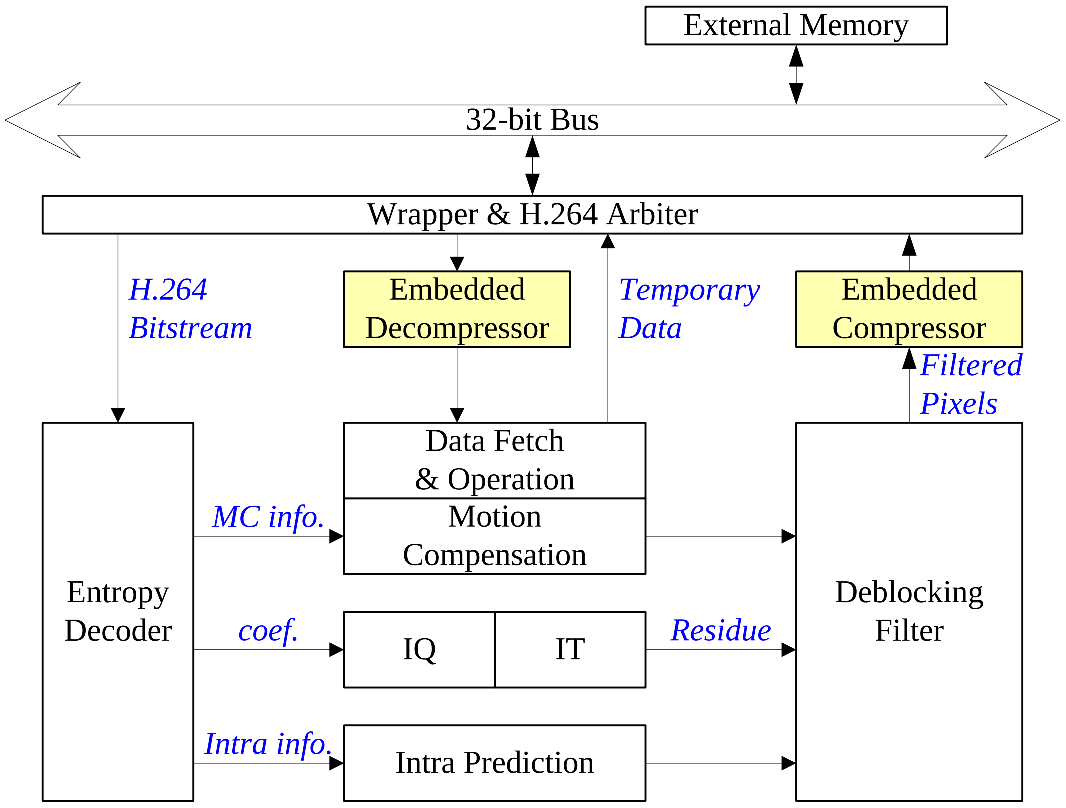


Figure 53: H.264 decoder with proposed embedded compression

The compatible H.264 decoder specification is HD1080+HD720@30fps and works at 150MHz. The compressor converts a 4x2 block from the deblocking filter into a 32 bits segment which is stored into the external memory. Comparing the data access times of the external memory for the system without EC, the data access times of our system is half. The decompressor converts a 32 bits segment into a 4x2 block which is sent to the motion compensation. Since our system bus is 32 bits and the external memory is 32 bits per entry, the system accesses once a data as four pixels. In Table 17, we analyze the read times of the motion compensation with/without EC. The worst case is the (Sub, Sub) case. To finish the motion compensation, a 4x4 block needs a 9x9 block. Therefore, the system with/without proposed embedded compressor takes 15/27 cycles. The best case is the (Align, Align) case. Original system with/without embedded compressor needs 2/4 cycles to finish the best case. For the other cases when the required data of motion compensation are not fit for 4x2 block-grids, the access times become increased.

Table 17: All Cases of Read Access Requirement

Case of MV (x, y)	Access Cycles for System without EC	Access Cycles for System with Proposed EC	Reduction of Access Cycles without EC
(Align, Align)	4	2	50
(Align, Not Align)	4	2/3	50/25
(Align, Sub)	9	5	44.4
(Not Align, Align)	8	4	50
(Not Align, Not Align)	8	4/6	50/25
(Not Align, Sub)	18	10	44.4
(Sub, Align)	12	6	50
(Sub, Not Align)	12	6/9	50/25
(Sub, Sub)	27	15	44.4
Average	13.2	6.8~6.9	49.1~48.3

(4) *Experimental Results*

Table 18 shows the software result of the proposed algorithm which is integrated with JM16.2. The test sequences are Akiyo, Forman, Mobile, Stefan, and Station. Each test sequence executes 100 frames. And then the average PSNR value is calculated. Results show that the PSNR loss of the proposed algorithm is from 1.27 to 3.94dB.

Table 18: PSNR Comparison

Sequence	Format	H.264 (dB)	Proposed (dB)	PSNR loss
Akiyo	CIF	43.72	41.16	2.56
Forman	CIF	41.23	39.20	2.03
Mobile	CIF	37.61	34.14	3.47
Stefan	CIF	38.82	34.88	3.94
Station	HDTV	39.12	37.84	1.27

Table 19 shows the comparison among previous work. It can be found that our proposed hardware provides less hardware complexity and better visual quality. Especially, the proposed decoder just requires one cycle with higher random accessibility for embedded compression without degrading overall system performance. The power consumption of the proposed hardware is better than Lee's [31] and Wu's [32]. Figure 54 shows the Station sequence result of the original system with EC in HDTV format. The propagation of quality loss is unavoidable but video quality remains acceptable.

Table 19: Comparison Among Previous Work

		Lee's [31]	Wu's [32]	This Work
Technology		CMOS 0.25um	UMC 90nm	UMC 90nm
System		MPEG-2 Decoder	H.264 Decoder	H.264/SVC Decoder
Working Frequency		100MHz	100MHz	150MHz
Processing Data Unit		8x1 Block	4x4 Block	4x2 Block
Total Gate Count		20k	30k	4.9k
Cycle Count for 4x2 Block	Encoder	2 cycles	N/A	2 cycles
	Decoder	2 cycles	N/A	1 cycle
Cycle Count for a MB	Encoder	33 cycles	72 cycles	33 cycles
	Decoder	33 cycles	34 cycles	32 cycles
PSNR Loss		6.08dB~ 10.65dB	1.31dB ~ 4.48dB	1.27dB ~ 3.94dB
Power Consumption	Encoder	N/A	2.78mW	158uW
	Decoder	N/A	1.66mW	86uW

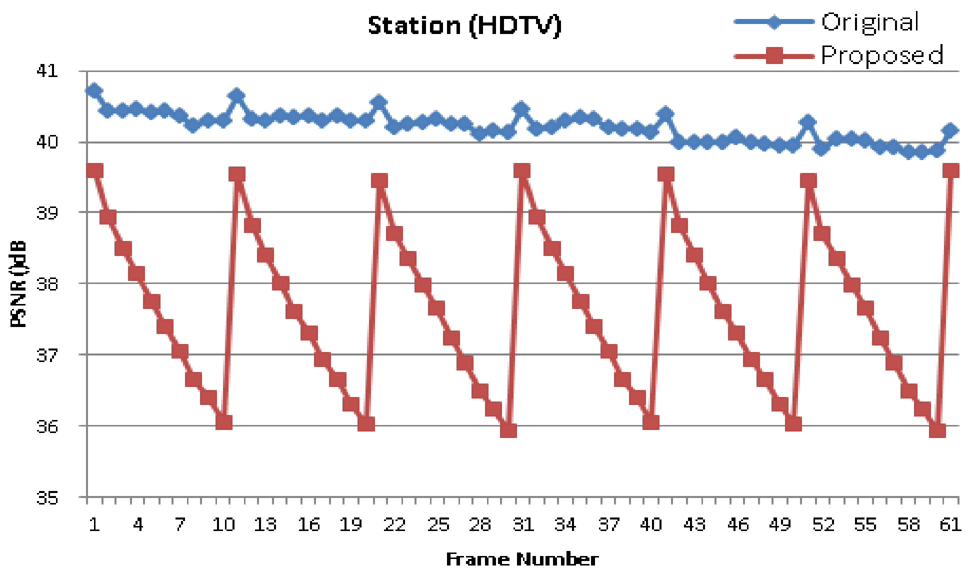


Figure 54: Simulation result of station sequence (HDTV)

三、 結論與討論

The objective of this project contains six topics:

(1) the embedded compressor/decompressor,

(2) the high profile intra predictor.

(3) The Reduced Patterns Comparison Embedded Compressor/Decompressor

(4)The Bitplane Truncation with Pattern Comparison Coding Embedded Compressor/Decompressor

(5) *An Area-efficiently High-accuracy Prediction-based CABAC Decoder for H.264/AVC*

(6) A Predefined Bit-Plane Comparison Coding for Mobile Video Applications

We described as below:

First, we proposed a flexible algorithm which achieves good coding efficiency and is suitable to be integrated with any video decoder. The proposed architecture is synthesized with 90-nm CMOS standard-cell library. The operation frequency is 108 MHz. The gate counts of proposed algorithm for compressor/decompressor are 15.8K/14.2K respectively. With the help of this embedded compression engine, we can reduce the bandwidth requirement and the external frame memory size. The proposed architecture costs 30K gate counts and deals with a 4x4 block unit while MHT costs 20K gate counts in dealing with a 1x8 pixels array. The proposed algorithm not only gains 5.29dB in picture quality but also achieves an area-efficient hardware implementation.

Second, we propose a high-profile intra predictor to support MBAFF and Luma *intra_8x8* decoding. The proposed memory hierarchy includes upper, left and corner memory buffer which reuses the neighboring pixels for follow-up prediction procedures. In Luma_8x8 decoding process, we propose base-mode predictors to minimize the additional hardware cost, latency penalty, and filtered pixel buffer memory size. Compared to the existing design [10] without supporting intra 8x8 coding, this design only introduce 10% and 7.5% of gate counts and SRAM overheads. The proposed design can achieve real-time processing requirement for HD1080 format video in 30fps under the working frequency of 100MHz.

Third, we have proposed a new embedded compression algorithm for mobile video applications. With these advantages of the proposed EC engine, we can lessen the size of external memory and bandwidth utilization to achieve the goal of power saving. Due to the fixed Compression Ratio, the proposed function is easy to be integrated with an H.264 system. The proposed architecture is synthesized with 90-nm CMOS standard-cell library and the gate counts of the proposed algorithm for embedded compressor/decompressor are 1.8K/3.1K respectively. The average PSNR loss of proposed algorithm is 5.98 dB. The working frequencies are 5 (CIF), 100 (HD 720) and 150 (HD 1080 + HD720) MHz depending on different operation modes.

Fourth, we have proposed a new embedded compression algorithm for mobile video applications. With these advantages of the proposed EC algorithm, we can lessen the size of external memory and bandwidth utilization to achieve power saving. The pipelined architecture of the proposed decompressor requires 1 cycle, thus the random accessibility becomes better. Due to the fixed CR, the proposed EC algorithm is easier to be integrated with H.264 decoder. From the experimental results, the PSNR loss of the proposed EC algorithm is from 1.27 to 3.94dB. The proposed architecture is synthesized with 90-nm CMOS standard-cell library and the gate counts of the proposed algorithm for compressor/decompressor are 4.0k/0.9k respectively. The working frequency is up to 150MHz@HD1080/720. For power consumption, the compressor is 158uW and the decompressor is 86uW.

Fifth, we proposed a prediction-based CABAC decoder architecture. In our design, prediction process combined parser and decoder to reduce most of SESO and raised the average hit rate to more than 90%. Memory system reduced about 70% of information to be stored and get low cost because of single-bin engine and buffer reused. Finally, we get low cost and memory bandwidth requirement and enough throughputs for real-time decoding full-HD sequences.

Sixth, we have proposed a new embedded compression algorithm for mobile video applications. With these advantages of the proposed EC algorithm, we can lessen the size of external memory and bandwidth utilization to achieve power saving. The pipelined architecture of the proposed decompressor requires 1 cycle, thus the random accessibility becomes better. Due to the fixed CR, the proposed EC algorithm is easier to be integrated with H.264 decoder. From the experimental results, the PSNR loss of the proposed EC algorithm is from 1.27 to 3.94dB. The proposed architecture is synthesized with 90-nm CMOS standard-cell library and the gate counts of the proposed algorithm for compressor/decompressor are 4.0k/0.9k respectively. The working frequency is up to 150MHz@HD1080/720. For power consumption, the compressor is 158uW and the decompressor is 86uW.

四、 參考文獻

[1] R. J van der Vleuten et al, "Low-complexity scalable DCT image compression", IEEE Proc. Image Processing, vol.3 pp. 837-840, Sep. 2000

[2] Yongje Lee; Chae-Eun Rhee; Hyuk-Jae Lee; "A New Frame Recompression Algorithm Integrated with H.264 Video Compression" Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium, pp. 1621-1624, May 2007

[3] A. Bourge and J. Jung, "Low-Power H.264 Video Decoder with Graceful Degradation," SPIE Proc. Visual Commun. and Image Process., vol. 5308, pp. 1234-1245, Jan. 2004

[4] T.-Y. Lee, "A New Frame-Recompression Algorithm and its Hardware Design for MPEG-2 Video Decoders," IEEE Trans. CSVT, vol. 13, no. 6, pp. 529-534, June 2003.

[5] Byeong Lee, "A new algorithm to compute the discrete cosine Transform" Acoustics, Speech, and Signal Processing, IEEE Transactions on Volume 32, Issue 6, pp:1243-1245, Dec 1984

[6] Joint Video Team, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.

[7] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560–576, July 2003.

[8] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen, "Hardware architecture design for H.264/AVC intra frame coder", ISCAS 2004.

[9] Esra Sahin and Ilker Hamzaoglu, "An Efficient Intra Prediction Hardware Architecture for H.264 Video Decoding", DSD 2007.

[10] Jia-Wei Chen, Chien-Chang Lin, Jiun-In Guo, and Jinn-Shyan Wang, "Low Complexity Architecture Design of H.264 Predictive Pixel Compensator for HDTV Application", ICASSP 2006.

[11] Ting-An Lin, Sheng-Zen Wang, Tsu-Ming Liu and Chen-Yi Lee, "An H.264/AVC Decoder with 4x4 Block-Level Pipeline," Proc. ISCAS, pp. 1810-1813, May, 2005.

[12] Tsu-Ming Liu and Chen-Yi Lee, "Design of an H.264/AVC Decoder with Memory Hierarchy and Line-Pixel-Lookahead," Journal of VLSI Signal Processing Systems, Aug. 2007.

[13] "ITU-T Recommendation H.264 and ISO/IEC 14496-10, Advanced Video Coding for Generic Audiovisual Services", May 2003.

[14] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560–576, July 2003.

[15] R. Manniesing, R. Kleihorst1, R. V. Vleuten1, and E. Hendriks, "Implementation of lossless coding for embedded compression," IEEE ProRISC, 1998.

[16] T. Y. Lee, "A New Frame-Recompression Algorithm and its Hardware Design for MPEG-2 Video Decoders," IEEE Trans. CSVT, vol. 13, no.6, pp. 529-534, June 2003.

[17] Y. D. Wu, Y. Li, and C. Y. Lee, "A Novel Embedded Bandwidth-Aware Frame Compressor for Mobile Video Applications," in Proc. IEEE Intelligent Signal Processing and Communication Syst. (ISPACS), pp. 1-4, Feb 2009.

[18] C. K. Yang and W. H. Tsai, "Improving block truncation coding by line and edge information and adaptive bit plane selection for gray-scale image compression," Pattern Recognition Letter, vol. 16, pp. 67-75, 1995.

[19] T. M. Amarunnishad, V. K. Govindan, and T. M. Abraham, "Block Truncation Coding Using a Set of Predefined Bit Planes," in Proc. IEEE Int. Conf. Computational Intelligence and Multimedia Applications (ICCIMA), vol. 3, pp. 73-78, Dec 2007.

[20] "Draft ITU-T recommendation and final draft international standard of Joint Video Specification (ITU-T Rec. H264-ISO/IEC 14496-10:2005 AVC)," JVT G050, 2005.

[21] J. Kim, and C. M. Kyung, "A Lossless Embedded Compression Using Significant Bit Truncation for HD Video Coding," IEEE Trans. CSVT, accepted, 2010.

[22] T. M. Liu, and et al., "A 125/spl mu/w, fully scalable MPEG-2 and H.264/AVC video decoder for mobile applications," in Proc. IEEE Int. Solid-State Circuits Conference (ISSCC), pp. 1576–1585, 2006.

[23] Joint Video Team (JVT) of ISO/IEC MPEG&ITU-T VCEG, "Joint Draft ITU-T Rec. H.264 | ISO/IEC 14496-10/Amd.3 Scalable video coding," *July 2007.*

[24] Detlev Marpe, Heiko Schwarz and Thomas Wiegand, “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,” in *Circuits and Systems for Video Technology, IEEE Transactions on, vol. 13, pp. 620-636, 2003.*

[25] Won-Hee Son and In-Cheol Park, "Prediction-based real-time CABAC decoder for high definition H.264/AVC," in *ISCAS 2008, pp. 33-36, 2008.*

[26] Yao-Chang Yang and Jiun-In Guo, "High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications," in *Circuits and Systems for Video Technology, IEEE Transactions on, vol. 19, pp. 1395-1399, 2009.*

[27] Pin-Chih Lin, Tzu-Der Chuang and Liang-Gee Chen, branch selection multi-symbol high throughput CABAC decoder architecture for H.264/AVC," in *ISCAS 2009, pp. 365-368, 2009.*

[28] Yuan-Hsin Liao, Gwo-Long Li and Tian-Sheuan Chang, “A high throughput VLSI design with hybrid memory architecture for H.264/AVC CABAC decoder,” in *ISCAS* 2010*, pp. 2007 – 2010, 2010*

[29] Joint Video Team (JVT) Reference Software JM 16.1.

[30] J. Kim, and C. M. Kyung, “A Lossless Embedded Compression Using Significant Bit Truncation for HD Video Coding,” *IEEE Trans. CSVT,* accepted, 2010.

[31] T. Y. Lee, “A New Frame-Recompression Algorithm and its Hardware Design for MPEG-2 Video Decoders,” *IEEE Trans. CSVT*, vol. 13, no.6, pp. 529-534, June 2003.

[32] Y. D. Wu, Y. Li, and C. Y. Lee, “A Novel Embedded Bandwidth-Aware Frame Compressor for Mobile Video Applications,” *in Proc. IEEE Intelligent Signal Processing and Communication Syst. (ISPACS),* pp. 1-4, Feb 2009.

[33] C. K. Yang and W. H. Tsai, “Improving block truncation coding by line and edge information and adaptive bit plane selection for gray-scale image compression,” *Pattern Recognition Letter,* vol. 16, pp. 67-75, 1995.

[34] T. M. Amarunnishad, V. K. Govindan, and T. M. Abraham, “Block Truncation Coding Using a Set of Predefined Bit Planes,” *in Proc. IEEE Int. Conf. Computational Intelligence and Multimedia Applications (ICCIMA),* vol. 3, pp. 73-78, Dec 2007.

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

   ■ 達成目標

   □ 未達成目標（請說明，以 100 字為限）

   　□ 實驗失敗

   　□ 因故實驗中斷

   　□ 其他原因

   說明：

2. 研究成果在學術期刊發表或申請專利等情形：

   論文：■已發表 □未發表之文稿 □撰寫中 □無

   專利：□已獲得 □申請中 ■無

   技轉：□已技轉 □洽談中 ■無

   其他：（以 100 字為限）

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）（以 500 字為限）

   在此計畫執行中，我們提供兩個可應用於行動通訊之下世代低功耗視訊解碼器元件，其中分別為：

   (A)An Area-efficient High-accuracy Prediction-based CABAC Decoder for H.264/AVC.

   (B)A Predefined Bit-Plane Comparison Coding for Mobile Video Applications.

   所提的低功耗視訊解碼方案,可提供行動視訊應用下,有效降低傳輸頻寬和節能的需求,有助於提升行動裝置的使用時限.

國科會補助專題研究計畫項下出席國際學術會議心得報告

日期：___年___月___日

計畫編號	NSC97－2221－E－009－167－MY3		
計畫名稱	應用於數位電視的雙模視訊解碼器		
出國人員姓名	郭明諭	服務機構及職稱	國立交通大學 電子工程學系
會議時間	2011年5月15日至2011年5月18日	會議地點	巴西 里約熱內盧
會議名稱	(中文)國際電路與系統研討會 (英文)IEEE International Symposium on Circuits and Systems		
發表論文題目	(中文)針對H.264/AVC之低硬體面積高精準度預測式內容適應性二元算術解碼器設計 (英文)An area-efficient high-accuracy prediction-based CABAC decoder architecture for H.264/AVC		

一、參加會議經過

今年的 2011 年 IEEE 電路與系統國際會議 (ISCAS) 在巴西 (Brazil) 里約熱內盧 (Rio de Janeiro) 城市舉行。此會議的目的在邀請世界各國研究人士以及科技業界人員共同探討最新進的電子電路與系統應用議題。以下是這次 2011 ISCAS 的重要議程，包含 (1) analog signal processing (2) biomedical circuits and systems (3) circuits and systems for communications (4) computer-aided network design (5) digital signal processing (including BSP) (6) life-science systems and applications (7) multimedia systems and applications (8) nanoelectronics and gigascale systems (9) neural networks and systems (10) nonlinear circuits and systems (11) power and energy circuits and systems (12) sensory systems (13) visual signal processing and communications (14) VLSI systems, architectures and applications (15) education in circuits and systems (16) special sessions (17) live demonstrations of circuits and systems。

會議的第一天是 tutorial，第二天到第四天是 keynote speech、technical lecture 與 poster。今年 tutorial 的議題主要包含有混和訊號、3D 圖學、低功率消耗的電路設計，在此即使對這方面領域涉略不多的聽眾來說，很快就能有效率了解概況，而且通常參加的人在幾十人內，有更多時間可以發表問題討論

解除困惑。

二、與會心得

學生本次是錄取會議的 poster 部分，不同於會議報告的方式，有更多的時間和機會與其他人交談並分享經驗。

除了聽一些 technical lecture，我也有去參加 keynote speech，印象最深也最有興趣的就是微軟工程師講解對於未來人體和電子晶片中間的 touch interface，這讓吾人想到最近當紅的 iPhone，就是強打良好的觸控式介面，使人面在使用電子產品的時候，更加人性化和也更增進不少娛樂性，而微軟現今也在電視遊樂器上 XBOX 引進 motion detection 方法，讓玩家不用任何的硬體設施介面就能模擬臨場的效果，讓人們在遊戲上除了考慮益智也能考驗姿體的協調度，讓玩遊戲機不再只是呆坐在位子上，而是讓玩遊戲也好像到戶外運動一樣，提高了電子產品在娛樂性上更高的層面，而微軟已經在實驗是否將一些 touching 的介面轉移到人體的思考和肌膚表面，透過腦波和身體的震動波偵測判別，將訊號輸入到電子晶片上，達到免手持硬體輸入介面以提高便利性。

除了演講上的新知吸收，這次會議中也有 coffee break、welcome party、banquet 能和來自各地的人士交流機會，除了更多了解不同領域還有不同研究環境的訓練方式，更提升自己不少英文聽力和口說能力，如何能不失禮貌且親和的與人在宴會或者餐桌上談話，也是一種學問。

三、考察參觀活動(無是項活動者略)

四、建議

如果是 poster 的話，如果可以有現場 demo 會更吸引人

五、攜回資料名稱及內容

1. 2011 ISCAS 電子檔案論文。

2. 2011 ISCAS 會議手冊。

3. 2011 ISCAS 紀念後背包。

4. 2011 數本電路設計相關紙本期刊。

5. 2012 各大電子電路 conference 的 call for paper。

六、其他