A High Throughput Parallel Pipeline Architecture Hardware Design of a One Dimensional Multicore SPIHT Algorithm

N P Raju Mandapaka[1], Venkata Satish Dhulipudi[2]
Design Engineer in Space Systems[1], Assistant Professor[3]
Department of ECE[2]
Ananth Technologies Limited East Godavari Dist, INDIA[1] Prasiddha College of Engineering and Technology East Godavari Dist, INDIA[2]
Corresponding authors' email id: *mnprawinraj@gmail.com*

Abstract

A desirable characteristic for display memory compression is support for the raster-scan processing order and the fixed target compression ratio. Set partitioning in hierarchical trees (SPIHT) is an efficient two-dimensional compression algorithm that guarantees a fixed target compression ratio, but its one-dimensional (1D) variation has received little attention, even though its 1D nature supports the raster-scan processing order. This paper proposes a novel hardware design for 1D SPIHT. The algorithm is modified to exploit parallelism for effective hardware implementation. For the encoder, dependences that prohibit parallel execution are resolved and a pipelined schedule is proposed. For the parallel execution of the decoder, the algorithm is modified to enable estimation of the bitstream length of each pass prior to decoding. This modification allows parallel and pipelined decoding operations, leading to a high-throughput design for both encoder and decoder. However, in an actual system, several modules, including the video codec and the external memory, are connected via a system bus. In this environment, the gain in compression and the gain in transfer are very different. For a practical FMC design, not only the compression rate but also the transfer efficiency should be carefully considered. In this paper, a bus-aware interface

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

design for one dimensional (1D) lossless FMC is proposed to reduce the amount of bus transfers. The proposed bit stream merging and tailed burst length schemes increase the transfer gain such that it is close to the gain in compression. The proposed 1D lossless FMC schemes are implemented in Verilog language and verified in an environment containing a high efficiency video coding (HEVC) encoder and a system bus.

Keywords: *Set partitioning in hierarchical trees (SPIHT), one dimensional (1D, high efficiency video coding (HEVC)*

I. INTRODUCTION

Due to the strong demand for high-resolution videos in many applications, compression is inevitable for the storage and transmission of videos. As popular video resolutions trend toward higher resolutions, the resource overhead is also increasing, even for the temporary storage of input frames. Most standard video codecs, such as H.264 and high efficiency video coding (HEVC), use temporal redundancy between successive frames. Through inter-predictions using current and reference frames, only residual data are transferred. In hardware- based encoders, input frames from a camera and reference frames are stored in external memory and the data are loaded for compression operations whenever necessary. To store high-resolution frames with resolutions which exceed 1920x1080, a large amount of memory space is required. In addition, a significant amount of time is required to read and write a large amount of data. Data waiting time often becomes a bottleneck and determines the overall video encoding time. To relieve this bottleneck, frame memory compression (FMC) is widely adopted. Input or reference frames are encoded before they are written to external memory. The encoded data are then read and decoded when requested. An FMC module is located between the external memory and a video codec, and compression algorithms for an FMC module are independent of codec algorithms. Various algorithms and organizations have been proposed for FMC [1-20]. Previous works can be classified into lossy and lossless compression schemes. The lossy compression scheme reduces both

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

the memory bandwidth and memory space at some sacrifice of quality [1-7]. Lee et al. use the best intra-mode of the H.264 encoder to determine the scan order of differential pulse code modulation (DPCM) in FMC. The DPCM results are compressed further by Golomb-Rice coding. If the compression rate is less than 50%, a quantization parameter (QP) is increased and the same process is repeated until a 50% compression rate is met [1]. In multi-view coding (MVC), the overhead for memory access is especially large. For FMC with MVC, simplified intra-prediction is performed from the best 4x4 intra-mode of MVC. Non-linear quantization and Huffman-based entropy coding are then conducted in sequence. Here, the quantization strengths and Huffman tables are selected depending on the image content characteristics. The external memory bandwidth in MVC is reduced by 69.5% with a marginal degradation of the coding efficiency of MVC [4]. The compression rate of lossless compression schemes [8-20] is generally lower than that of lossy schemes, and the compression rates are not fixed. At times, the compressed data size is larger than the original data size. Thus, FMC with lossless compression is of no use to reduce the memory space. However, it is still useful for reducing the memory bandwidth, and it does not affect the coding efficiency of the video encoder at all. Silveira et al. propose a reference frame context-adaptive variable-length compressor (RFCAVLC) using four types of coding tables, resulting in a compression rate of 24% [8]. Zhou et al. focus on the optimum organization to enable the video encoder to access the memory randomly, with several recompression algorithms [10-13] integrated into an HEVC test model (HM). They show that each recompression algorithm achieves bandwidth reductions ranging from 52.36% to 57.60% [9]. In the other work [14], a pixel granularity adaptive directional prediction scheme is adopted so as not to depend on the results from the video encoder. In addition, dynamic kth- order Unary/Exp-Golomb Rice coding is used to reduce the data bandwidth even more. Previous works have proposed good solutions focusing on the data compression rate. However, in many actual systems, several modules including a video codec are connected via a system bus. Accesses to external memory are also done through a system bus. Thus, for a practical FMC design, not only the compression performance but also the transfer efficiency should be carefully considered. Video

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

compression is the important tool in transmission. It is done to reduce memory and to process large amount of data. Video codecs used are MPEG-2, MPEG-4, H.264 [1] for 3D processing, motion detection and estimation, feature extraction and analysis, multispectral processing, interpolation and super resolution, video segmentation and tracking process. As raw data without compression has high memory traffic and takes longer time for memory access cycles. For progressive transmission [2] and easy bit rate control, frame compression algorithm is proposed. In this paper, Set partitioning in hierarchical trees (SPIHT) algorithm [3] is applied for video applications. This algorithm has hardware utility architecture and less arithmetic coding is required. The distributed arithmetic provides real time manipulation of videos. Discrete Wavelet transform [4] analyze the frames at multiresolution and reduces computational redundancy. It decomposes into subbands and gives higher flexibility and performs well in low bit rates. As it is a pipelined architecture, 2D DWT [5] does not require frame memories in every step. The 2D DWT divides the input image in two stages as row wise followed by column wise. The frame compression is to compress the video frames which are independent in size before being stored into external memory. The objective of SPIHT algorithm is to compress video frames and provides clear picture quality and increased throughput. SPIHT process multiple bit-planes in a single cycle. The following features of SPIHT are high encoding/decoding speed, optimized embedded coding, high compression ratio, fine image quality, exact bit rate specification. The memory access time for both encoder and decoder is less. The redundant pixels are removed and most significant pixels are transmitted first. Thus space and bandwidth is reduced such that hardware cost is also reduced. It achieves high speed and compression ratio. Therefore small memory size and latency is obtained. Video codecs store reconstructed or reference frames into external frame memory to execute an inter prediction which is based on temporal correlation. Typically, video codecs access the frame data through a bus. The stored data are often recompressed, and thereby reducing the frame memory size or bus bandwidth. This method is called frame memory compression (FMC) [1- 10]. FMC reduces frame memory access costs but requires an additional hardware. Furthermore, low read/write latency is demanded not to affect

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

the processing speed of video codecs. Therefore, unlike video codecs such as high efficiency video codec (HEVC), low computation is important rather than high coding efficiency in FMC. Therefore, temporal correlation is not exploited and spatial correlation is used restrictedly. Recently, the resolution size is exceeding 4K UHD (3840×2160) and video codecs require very highthroughput. The throughput of FMC is increasing as well. Until now, various FMC algorithms have been proposed. Lee et al. propose a differential pulse code modulation (DPCM) - variable length coding (VLC)-based FMC algorithm [1]. However, this algorithm generates the bitstream with a variable length. In addition, the bitstream has dependence inside, and therefore the highthroughput is not supported. To solve this problem, FMC algorithms without VLC have been studied. Kim et al. propose prediction and coding methods which are effective in hardware implementation [3]. The prediction method called hierarchical average and copy prediction (HACP) offers high prediction accuracy using diverse average values within a block. Moreover, the coding method called significant bit truncation (SBT) enables the number of the bits generated within a block to be calculated in advance, leading to the high-throughput. Guo et al. propose a prediction method called multi- mode DPCM and averaging (MDA) which effectively compresses various images. In addition, a coding method called semi-fixed length (SFL) is proposed to predict the bitstream length in advance, thereby achieving very highthroughput [6]. Despite the high-throughput, HACP-SBT and MDA-SFL only support lossless coding, and therefore have limited applications. To support lossy coding, the quantization method in [1] can be applied to those algorithms. However, HACP-SBT and MDA-SFL insert dummy bits to make the bitstream length foreknowable. As the target compression ratio (CR) increases, the ratio of those dummy bits in the bitstream increases, resulting in decrease in coding efficiency. In case that the CR and image quality are the most important factors to consider when choosing an FMC algorithm, lossless coding is most appropriate. It is difficult to assure that the same FMC is used for the video encoder and decoder. Therefore, the FMC, which is lossy coding, may make a reference frame mismatch between encoding and decoding, which is continuously accumulated. For this reason, HACP-SBT and MDA- SFL, which are

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

used as FMC in video codecs, only consider lossless coding. However, lossless coding algorithms are not suitable for hardware systems. Since the length of the bitstream generated by the lossless coding is variable, the frame memory size and bandwidth vary as well. Furthermore, random access on the reference frame required by video codecs is difficult. For example, since the bitstream having a variable length is sequentially stored in frame memory, it is difficult to calculate an initial memory address for each search range. If the bitstream is discontinuously stored in frame memory to support easy random access, the bus utilization to access the frame memory is significantly reduced. For example, dummy bits are inserted into the data transfer and the burst mode of the bus protocol is not fully utilized. This reduces the compression gain in the bus bandwidth utilization. Discrete wavelet transform (DWT) - set partitioning in hierarchical trees (SPIHT) is one of lossy coding algorithms [11]. The DWT-SPIHT achieves an accurate target CR and supports lossless coding as well. In addition, this algorithm is based on bit-plane coding and effective in coding efficiency due to errors occurring in lower bit-planes. However, the processing speed is slow due to the dynamic processing order. In addition, existing dependence inside bitstream and each pass interferes with the speed. To solve this problem, various methods including hardware architectures have been studied [5, 7, 12-15]. Kim et al. propose an SPIHT hardware design of which throughput is about 1 bit-plane per cycle [7]. In this design, the processing cycle is fixed regardless of the block size, and therefore the throughput can be continuously improved by increasing the block size. However, this scalable throughput has some problems with hardware implementation. First, a 2D DWT is composed of two 1D DWTs whose transform directions are horizontal and vertical. When the horizontal transform ends, the vertical transform starts. Therefore, as the block size increases, latency and temporal memory cost increase. Second, there is a stage of transposing coefficients to bit-planes between DWT and SPIHT, which is implemented in memory elements. As the block size increases, the size of the memory element proportionally increases. Third, when the height and width of a square block are doubled, the processing speed increases four times. Therefore, achieving the target system speed is difficult and the hardware utilization decreases. Fourth, as the block size

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

increases, the number of bits to be coded in each pass of SPIHT increases with each cycle. Consequently, the burden on the packer module to shift and merge the generated bits increases. Likewise, the complexity of the parser module increases. Satellites deployed by NASA currently only make use of lossless image compression techniques during transmission. There have been a few driving reasons behind NASA‟s decision to transmit unaltered data. First, the downlink channels have provided enough bandwidth to handle all of the data a satellite‟s sensors collected in real time. Second, there has been a lack of viable platforms with which a satellite could perform significant image processing techniques. Lastly, transmitting unaltered data reduces the risk of corrupting the data-stream.

As NASA deploys satellites with more sensors, capturing an ever-larger number of spectral bands, the volume of data being collected is beginning to outstrip a satellite‟s ability to transmit it back to Earth. NASA‟s most recent satellite Terra contains five separate sensors each collecting up to 36 individual spectral bands. The Tracking and Data Relay Satellite System (TDRSS) ground terminal in White Sands, New Mexico, captures data from all of these sensors at a rate of 150Mbps [16]. As the number of sensors on a satellite grows, and thus the transmission rates increase, they are providing a driving force for NASA to study lossy methods of compressing images prior to down linking. Current technologies have been unable to provide NASA with a viable platform to process data in space. Software solutions suffer from performance limitations and power requirements. At the same time traditional hardware platforms lack the required flexibility needed for post-launch modifications. After launch they cannot be modified to use newer compression schemes or even implement bug fixes. In the past, a modification to fixed systems in satellites has proven to be very expensive. The correction to the Hubble telescope‟s flawed 94-inch-wide primary mirror approached $50 million [4].

By implementing an image compression kernel in a reconfigurable system, it is possible to avoid these shortcomings. Since such a system may be reprogrammed after launch, it does not suffer from conventional hardware‟s inherit inflexibility. At the same time the algorithm is computing in custom hardware and can perform at the required

Page 1-26 © MANTECH PUBLICATIONS 2018. All Rights Reserved

rates, while consuming less power than a traditional software implementation.

A VIDEO display system, such as that in a television, often includes a memory system that stores the video data for display in frame memory and performs a variety of quality enhancement procedures. These procedures include color enhancement and data overdrive for LCD [1-2], aging compensation for OLED, and load compensation for PDP [3-4]. Inclusion of a memory system increases the cost of a video display system, and such costs change with the resolution of the display panel [5]. When video data are stored in external DRAM, the associated extensive memory traffic requires a huge bandwidth for that external memory [6-8]. To reduce memory cost and bandwidth, display systems often compress frame data and reduce the amount of data to be stored in memory. Desirable characteristics for display memory compression include support of the raster-scan processing order and maintenance of the fixed target compression ratio. Set partitioning in hierarchical trees (SPIHT) [9] is an efficient compression algorithm that guarantees maintenance of the target compression ratio, but SPIHT has only been studied extensively for two-dimensional (2D) compression, while its one-dimensional (1D) variation [10-11] has received less attention, even though the 1D nature supports the raster-scan processing order. This paper proposes a novel algorithm and hardware design that efficiently processes a 1D SPIHT.

Compression algorithm requirements for video display systems are different from those in standard video compression formats such as H.264 or HEVC. The first requirement is that the algorithm needs to process the data in raster-scan order, which is the order in which the display system receives frame data from the video source. Second, the algorithm must meet the target compression ratio because the size of the memory is fixed in a display system. If the size of the compressed data is larger than the memory size, some data cannot be stored in memory. Third, the complexity of the compression algorithm needs to be much lower than that of the standard compression because a complex compression algorithm may require expensive hardware, which increases the overall hardware cost even though the frame memory size is reduced by compression. This third requirement prohibits an aggressive compression tool, thus the target compression ratio for display

Page 1-26 © MANTECH PUBLICATIONS 2018. All Rights Reserved

is relatively small, ranging from 10% to 50%.

Several compression algorithms have been proposed for video compression for display. Simple compression algorithms such as JPEG-LS use only spatial correlation [12]. JPEG-LS is not very effective for display because it adopts variable length coding (VLC), thus it cannot guarantee a constant compression ratio. This is a common problem in other compression algorithms that adopt VLC [5, 13-14]. Block truncation coding (BTC) [2] and significant bit truncation (SBT) [15] code a block in a foreknowable length. However, BTC represents pixels in a block by using a small number of representative values, typically two values, thus the reconstructed image often has significant distortion. The coding efficiency of SBT is inferior to that in other VLC-based algorithms or transform-based algorithms because SBT codes all the pixels in a block using the same bit length, and a meaningless number of bits are coded.

SPIHT is a wavelet transform-based algorithm [9] that codes bit-plane by bit-plane in descending order from the upper bit-plane to the lower bit-plane with the more significant bits being coded first. Thus, it achieves high compression efficiency. In addition, it can be terminated when the coded bitstream length reaches the target bit length (TBL). As a result, SPIHT guarantees a compression ratio that meets the TBL.

The conventional SPIHT algorithm and SPIHT variations use a two-dimensional (2D) tree structure to partition an image into 2D blocks of coefficients. This conventional 2D compression scheme is efficient because it takes advantage of data correlations in both horizontal and vertical directions. However, 2D SPIHT cannot be used effectively for video display systems because the input data are generally received in raster-scan order. The 2D SPIHT algorithm requires a large amount of memory because of the mismatch between the input data order and the compression operation order when the compression operation is performed with a 2D block as the processing unit.

As a result, a 2D SPIHT algorithm requires memory with a size equal to block height × image width, where block height represents the vertical size of the 2D processing block and image width represents the width of the input image to be compressed. The width of

Page 1-26 © MANTECH PUBLICATIONS 2018. All Rights Reserved

an image increases in many applications and the memory requirement for 2D compression increases in proportion.

To reduce the large memory requirement of a 2D SPIHT algorithm, a one-dimensional (1D) SPIHT algorithm has been proposed [10-11]. The 1D SPIHT algorithm uses a 1D tree structure with a 1D block as the compression unit. As a result, 1D SPIHT does not need a large amount of memory, a key advantage over 2D SPIHT. Therefore, 1D SPIHT is effective in terms of hardware implementation, reducing the cost associated with the memory buffer. Generally, 1D SPIHT achieves a lower coding efficiency than 2D SPIHT because 1D SPIHT cannot utilize data redundancy in the vertical direction. Nevertheless, the low memory requirement of 1D SPIHT is an important advantage that allows it to be used frequently, which is beneficial as real applications often demand memory requirement reductions.

Although an algorithm for 1D SPIHT compression has been previously proposed, a hardware implementation scheme for that algorithm has not been investigated. This paper proposes an efficient hardware design for implementation of the 1D SPIHT algorithm. An important difference in the proposed design from that in previous hardware designs is that multiple frequency bands need to be processed in parallel. As the multiple bands have interdependences, the processing order of these multiple bands must be scheduled effectively. In addition, the bitstream format needs to be carefully defined to enhance the degree of parallelism. To this end, the original 1D SPIHT algorithm has been modified to allow efficient parallel executions with a slight degradation of compression efficiency. The hardware design when implemented with Verilog HDL achieves throughputs of 7.04 Gbps and 7.63 Gbps at the operating frequencies of 143 MHz and 167 MHz for the encoder and the decoder, respectively. Hardware logic gate counts of 37.2K and 54.1K are used by the encoder and the decoder, respectively.

II. EXISTING SYSTEM

High-throughput of about one bit-plane per cycle. Fig shows the pipelining schedule of the encoder design in [7] of which the decomposition level is two. In this figure, vertical and horizontal axes represent passes in each band and execution time, respectively. In addition, one rectangle and the number inside represent one cycle time

Page 1-26 © MANTECH PUBLICATIONS 2018. All Rights Reserved

and the bit-plane number, respectively. During the first two cycles, represented in white color, all passes do not work in parallel to avoid the dependences. However, all passes work in parallel from the third cycle, which is represented in light gray color. The delayed bit-planes are represented in dark gray color. As this design processes about one bit-plane per cycle, regardless of the block size, throughput can be improved by increasing the block size. However, the required hardware resources, such as buffers, significantly increase as the block size increases. Moreover, hardware utilization may decrease because a fine control of throughput is difficult. Therefore, it is required to divide one large block into several small blocks and process them in parallel to achieve an area-efficient and high-throughput design.

Encoder design:

This section describes the proposed hardware design for a SPIHT encoder. In the proposed design, an earlier 1D SPIHT algorithm is modified to exploit parallelism effectively so as to implement the algorithm in hardware. To this end, dependences in SPIHT are analyzed and a parallel and pipelined scheduling of operations is proposed This subsection describes the data dependences in the SPIHT algorithm that make it difficult to utilize hardware resources fully. The subsection also describes a modified processing order for the SPIHT algorithm. The modification improves SPIHT hardware utilization by avoiding dependences.

The proposed algorithm processes three passes, RP, SP, and FRP as shown in Fig.. There are two dependences that prohibit individual passes from being processed simultaneously (see Fig. 1 (a)). The first dependence exists between the SP and the FRP. This dependence can be avoided by delaying the execution of FRP by one cycle after the SP is executed.

The second dependence exists between the SP of the higher pass band and that of the lower pass band. To avoid this dependence, the processing order needs to be changed such that the SP phase of the lower pass band is delayed by one cycle after the SP phase of the higher pass band. The next paragraph describes a processing schedule that avoids the data dependences shown in Fig. 1 (a).

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

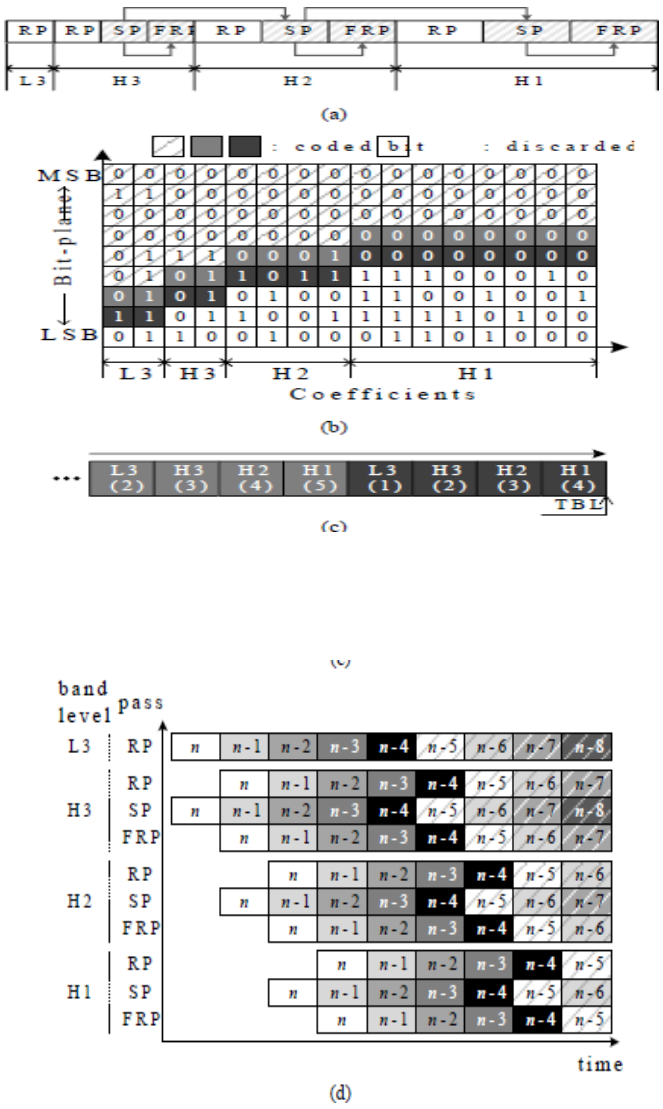


Fig. 1. Dependence analysis and pipeline schedule. (a) Dependences among passes (b) an example of which modified bit-planes are processed in parallel (c) generated bitstream of (b) (d) Pipelined execution avoiding dependence violation

Fig. 1 (b) shows a proposed processing order that delays processing at the lower levels. In that figure, the horizontal axis presents the 16 DWT coefficients at three decomposition levels. The vertical axis presents the bit-planes from the 0th bit-plane to the 8th bit-plane. The darkness in each region indicates the bit-planes that are processed simultaneously. For example, the 3rd bit-plane of H3 is processed

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

simultaneously with the 4th bit-plane of H2 and the 5th bit-plane of H1. In the next cycle, one less bit-plane is processed such that the 2nd bit-plane of H3, the 3rd bit-plane of H2, and the 4th bit-plane of H1 are processed in parallel. In this proposed order, a higher level is processed earlier than a lower level. As a result, dependence violation among SPs in different bands is avoided, thus only the dependence between the SP and the FRP in the same band needs to be handled by delaying the execution of the FRP by one cycle after the SP (see the pipelined execution presented in Fig. 4 (d)).

The change of processing order, as shown in Fig. 4 (b), affects the bitstream format because the bitstream is stored in the order in which it is generated. The next figure, Fig. 4 (c), shows example bitstreams in which the 2nd bit-plane of L3, the 3rd bit-plane of H3, the 4th bit-plane of H2, and the 5th bit-plane of H1 are stored in sequence. In the next cycle, the 1st bit-plane of L3, the 2nd bit-plane of H3, the 3rd bit-plane of H2, and the 4th bit-plane of H1 are stored in sequence. Fig. 4 (c) shows the bitstream format in this case. Suppose that the length of the generated bitstream reaches the TBL when the 4th bit-plane of H1 is generated and the SPIHT operation then stops. In this case, H3 stores the bitstream down to the 2nd bit-plane, whereas H2 and H1 store the bitstreams down to the 3rd bit-plane and the 4th bit-plane, respectively. The white region in Fig. 4 (b) indicates the discarded bit-planes.

The processing order shown in Fig. 4 (b) leads to the pipeline schedule shown in Fig. 4 (d). At the first cycle (cycle = 0), the RP of L3 and the SP of H3 are processed in the first bit-plane (indicated by the nth bit-plane). At cycle 1, the RP and FRP of H3 in the nth bit-plane are processed. In addition, the SP of H2 in the nth bit-plane as well as the RP of L3 and the SP of H3 in the second bit-plane (indicated by the (n-1)th bit-plane) are processed. From cycle 3 onward, all passes in all bands are processed for different bit-planes. Consequently, a coding result for a single bit-plane is generated for every cycle.

Although the proposed schedule is designed to avoid data dependences, it can also help improve coding efficiency. In general, the information stored at a higher level may be more important than that at a lower level. For example, the importance of H3 may be greater than that of H2 which, in turn, may be more important than H1. The pipelined

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

schedule results in a processing order that allocates more bits for a higher level than the number for a lower level. Consequently, coding efficiency can be improved by allocating more space for a higher level than for a lower level. The next subsection presents an additional coding scheme to improve coding efficiency.

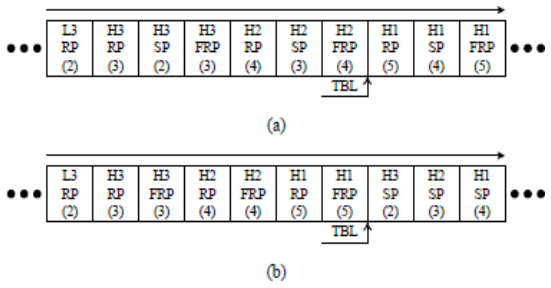


Fig. 2 Relocation of sorting bits (a) generated order (b) relocated order

Relocation of sorting bits:

The proposed improvement is accomplished by changing the storage location in the bitstream for the three passes, RP, FRP, and SP. If the bitstream is generated consistent with the processing order, it is stored as shown in Fig. 5 (a). The number inside the parentheses of each box in the figure indicates the bit-plane number. For example, L3 RP (2) in the leftmost box represents the RP of the L3 band of the 2nd bit-plane. Suppose that the bitstream length reaches the TBL after the FRP of the 4th bit-plane of H2 (indicated by the arrow). In such a case, all of the bitstream for the RP and FRP of the 5th bit-plane and that for the SP of the 4th bit-plane of H1 cannot be included in the final bitstream. Note that the sorting bits generated by SP are useful only when the related RP and FRP bits are available. In this example, the sorting bits of the SP of the 3rd bit-plane of H2 are useful for decoding the RP and FRP of the 3rd bit-plane of H2 (also for the SP of the 3rd bit-plane of H1). However, these related bits are not available due to the TBL limitation (note that the RP and FRP of the 3rd bit-plane of H2 are encoded during the next cycle and that the bitstream of those passes is excluded). Therefore, the sorting bits are useless in this case; moreover, they are wasted even when they are included in the bitstream. This waste of bitstream content takes place whenever the SP bitstream is included in the bitstream while its related bitstream is excluded. To avoid bitstream waste, the pass locations are reordered such that the sorting bits obtained by SP are moved to right after the related magnitude bits obtained by RP and FRP. As the sorting bits are stored later than the magnitude bits, there is a greater chance for the sorting bits than the magnitude bits to be excluded from the generation bitstream. In this example,

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

the new order of the pass locations is shown in Fig. 5 (b). The SP of the 3rd bit-plane of H2 is moved to right after the RP and FRP of the 5th bit-plane of H1. The bitstream reaches the TBL at the end of the FRP of the 5th bit-plane. As a result, the RP and FRP of the 5th bit-plane are included in the bitstream, whereas the SP of the 3rd bit-plane of H2 is excluded. Note that the inclusion of the RP and FRP of the 5th bit-plane of H1 has a positive impact on image quality, whereas the exclusion of the SP of the 3rd bit-plane of H2 does not have a negative impact

Decoder architecture:

An important requirement in decoder design is to determine the starting position of each pass in the bitstream prior to the start of the pass. This is necessary because each pass needs to access the corresponding bitstream for decoding. The description that follows shows that, because it is impossible to obtain the starting position of each pass, it is necessary to modify the bitstream format to allow pre-calculation of the bitstream length of every pass.

To explain why it is impossible to obtain the starting position of each pass before it is decoded, we first discuss the length of the bitstream generated in each pass. Table I presents all of the possible cases of bitstreams that are generated by the three passes, SP, FRP, and RP. The columns of Table I show the passes, the initial encoding state, the significance test results that lead to state transitions, the final state after the significance test, and the generated bitstream. In the next three paragraphs, the problem associated with determining length of the generated bitstream is explained in detail for each of the three pass types.

For the SP, the initial state can be either LISa (Type A) or LISb (Type B), as shown in Fig. 2. When the initial state is LISa, only a single sorting bit is generated, regardless of the significance test result (see right-most column in Table I). When the initial state is LISb, there are two significance tests (lines 11 and 25 in Fig. 2) and the length of the bitstream depends on the results of the significance test in line 25 in Fig. 2. When the significance test returns „1“, the state moves from LISb to LISa and two additional sorting bits are generated (line 10 in Fig. 2). In addition, a single sorting bit for the set in LISb (line 24 in Fig. 2) is generated. As a result, three sorting bits are generated. When the significance test returns „0“, the coefficient remains in LISb,

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

and only a single sorting bit is generated (line 24 in Fig. 2). The number of bits generated by the SP varies (i.e., either 1 or 3) depending on the significance test result. Thus, the length of the bitstream for the SP cannot be predicted in advance by a decoder, and an additional scheme is necessary for prior calculation of the SP bitstream length.

For the FRP, the initial state is always LISa and the two coefficients in LISa are tested. Depending on the magnitude of the two coefficients, four possible transitions (LIP, LIP), (LSP, LIP), (LIP, LSP), and (LSP, LSP) are possible. In all cases, two magnitude bits are generated. In contrast, a sign bit is generated only for a transition to LSP. Therefore, the length of the generated bitstream is not always the same, and another scheme is necessary for prior calculation of the FRP bitstream length.

For the RP, the initial state can be either LIP or LSP. When the initial state is LIP, the final state remains LIP when the significance test returns „0". In this case, only a single magnitude bit is generated (see right-most column in Table I). If the significance test returns „1", indicating a transition from LIP to LSP, two bits are generated: one magnitude bit and one sign bit. Thus, the length of the bitstream is 2. When the initial state is LSP, the next state is always LSP and a single magnitude bit is generated. Therefore, the length of the generated bitstream is either one or two depending on the present state and the significance test result; consequently, prior calculation of the RP bitstream length is impossible. In practice, the RP problem is identical to that in the FRP, indicating that the same scheme can be used for both the FRP and the RP.

The above discussion shows that the length of the generated bitstream varies and is dependent on the significance test result. This dependence makes it difficult for a decoder to obtain the bitstream length of each pass before the pass is decoded. In the next part of this subsection, two schemes that permit prior calculation of bitstream length for the SP and the RP (and FRP) are discussed.

For prior calculation of SP bitstream length, the operation of the SP is changed so that three sorting bits are always generated. In other words, two dummy bits are generated in addition to the sorting bit when the initial state is LISb and the significance test

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

returns „0". The generation of two dummy bits in the SP degrades the coding efficiency because these two bits do not carry any information. To avoid wasting these dummy bits, they are used to store the bitstream that is generated later. This dummy bit reuse scheme is illustrated in Fig. 6.

An additional buffer is used to store the address of the dummy bits. In the example shown in Fig. 6, the dummy bits are generated at addresses 501 and 508. Note that each location stores two dummy bits (and the bitstream generated later). In this example, the TBL is assumed to be 512. Therefore, the 512nd and 513rd bits in the generated bitstream are supposed to be discarded; instead, they are stored in the spaces addressed at 501 and 502

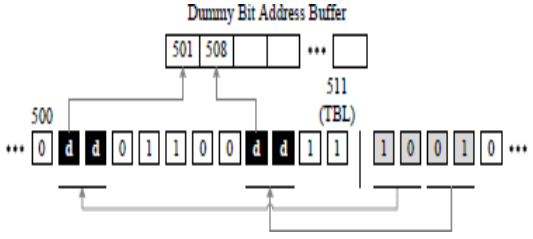


Fig. 3 Dummy bit generation and reuse

Respectively. Similarly, the 514th and 515th bits are stored in the spaces addressed at 508 and 509. In this manner, no bitstream space is wasted; thus, there is no efficiency degradation resulting from the generation of the dummy bits for the SP. A decoder can identify the existence of the dummy bits when the state remains LISb in the SP.

Therefore, the bits in the dummy bit space are stored in a separate buffer space and are used later for decoding. The proposed dummy bit reuse scheme requires additional memory space and hardware logics to manage the dummy bits. The sizes of those memory spaces and hardware logics are presented.

For prior calculation of bitstream length, another scheme that is applicable for both the RP and FRP is adopted from BPS. Recall that the number of magnitude bits is fixed (1 for RP and 2 for FRP), although the length of the bitstream varies depending on the significance test (Table I).

By separating the magnitude bits from the sign bits and storing them in two different positions in the bitstream, prior calculation of the length of the magnitude bits is possible because this value is fixed. After the magnitude bits are decoded, the length of the sign bits can also be calculated. In order to separate the magnitude and sign bits, the magnitude bits are stored from left

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

to right in the bitstream, whereas the sign bits are stored from right to left. In other words, the bitstreams for the RP and FRP are stored in two different positions: one on the left storing the magnitude bits and the other on the right storing the sign bits. Note that there is dependence of the decoding of the magnitude bits on the decoding of the sign bits because the length of the sign bits is obtained after the magnitude bit is known. Therefore, sign bit decoding for the RP (and FRP) is delayed by a single cycle after magnitude bit decoding of the corresponding RP (and FRP).

Half-block processing scheme:

The magnitude bits and sign bits are separated for the parallel execution of the decoder. However, separate storage of the magnitude and sign bits degrades coding efficiency, thereby producing an additional challenge. Such efficiency degradation may be substantial in cases where the proposed algorithm processes a large block. Fig. 7 (a) shows an example that indicates the possibility of coding efficiency degradation. In this example, a DWT decomposition level of three and a coding block size of 64 coefficients are assumed. Each box represents a pass, and the number inside the box indicates the maximum number of bits that can be generated in the pass. As shown in Fig. 7, the magnitude bits are stored from the left to the right, whereas the sign bits shown in figure

Stored in the opposite direction. Therefore, the magnitude and sign bits eventually overlap as the length of the generated bitstream reaches the TBL. In Fig. 7 (a), the magnitude bits for the RP of H1 overlap the sign bits for the corresponding pass. In this case, only the sign bits are stored in the bitstream, whereas the magnitude bits are excluded from the generated bitstream, as the sign bits carry more important information than that in the magnitude bits.

However, these sign bits are useless because the corresponding magnitude bits are not stored in the bitstream. The number of the discarded bits can be as large as 32 when the overlap is that for the RP of H1. The discard of this large number of bits may cause a substantial coding efficiency degradation. Therefore, a scheme to reduce the number of discarded bits is needed.

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

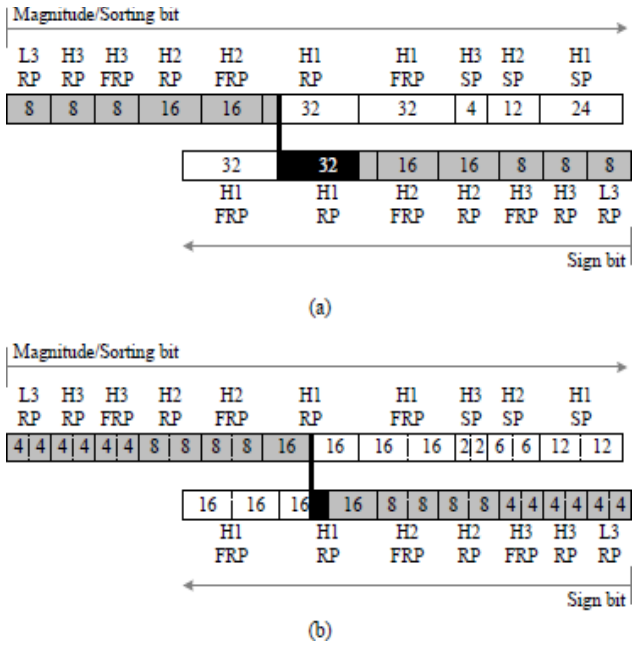


Fig. 4 Examples of the half-block processing scheme. (a) One part for one pass (b) two parts for one pass

To reduce the number of discarded bits, the size of the blocks to be encoded (and decoded) by each pass is halved and the half-blocks are processed in parallel. Parallel execution is possible because each half-block can be encoded independently. Fig. 7 (b) shows the half-block processing scheme. In this example, the RP operation for H1 is decomposed into two RP operations, each of which processes half of the coefficients in H1. In the proposed parallel execution, H1 is halved into two sub-blocks of 16 coefficients, each of which is encoded in parallel. Note that the maximum number of bits generated for each sub-block is 16; hence, the waste of generated bits caused by the overlapping of the magnitude and sign bits is reduced by half. Such a parallel execution scheme may increase the hardware cost, e.g., the start address calculation of each pass for the bitstream. Further partitioning into smaller sub-blocks is not attempted in this paper in order to avoid an additional increase in hardware cost.

Hardware architectures:

This block presents the hardware implementation of the proposed encoder and decoder. The organization of the proposed

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

hardware implementation is described in detail, and its performance and cost are compared with previous hardware implementations.

Verilog hardware description language is used to implement the proposed algorithm in hardware. The target compression ratio is two, the block size is 1×64, and the DWT decomposition level is three. Note that the implementation can be easily modified for other compression ratios, block sizes, and decomposition levels. To allow fair comparisons, all experimental results presented in this section are obtained from monochrome image compression.

Fig. 5 (a) shows the hardware organization of the proposed 1D SPIHT encoder. The input data for the encoder are DWT coefficients. These are transferred to the Transpose Unit, which reorganizes the data structure of the DWT coefficients for bit-plane by bit-plane processing in the Encoder Core Unit. The proposed architecture processes a single bit-plane per cycle.

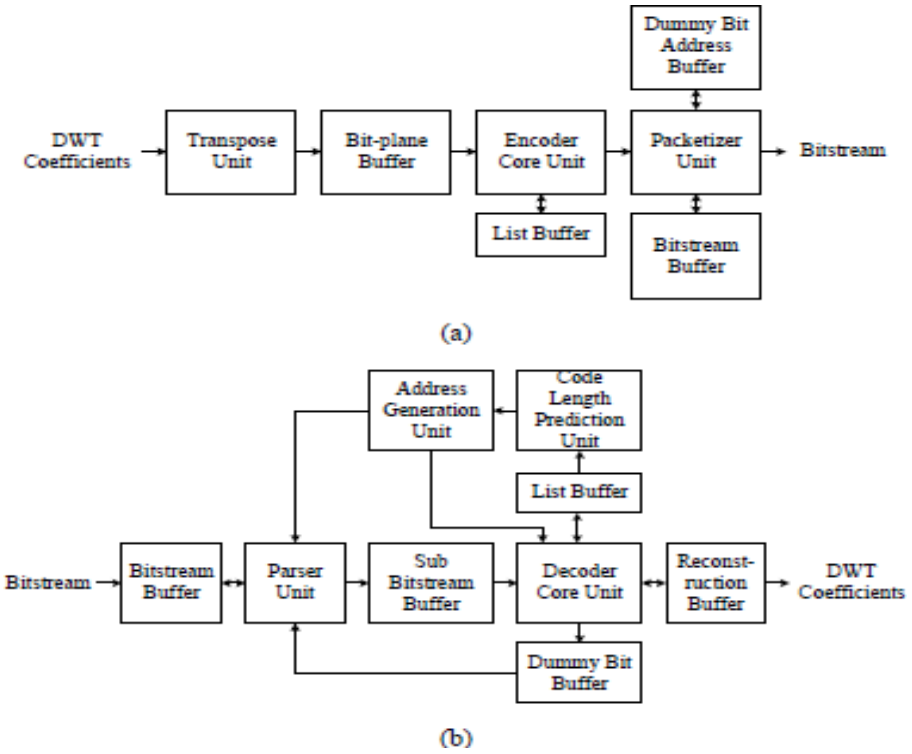


Fig. 5 Block diagram of the proposed 1D SPIHT hardware. (a) encoder (b) decoder

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

To this end, the Encoder Core Unit includes 20 processing units: 8 for RP, 6 for FRP, and 6 for SP. The Dummy Bit Address Buffer stores the address of the dummy bits generated by the Encoder Core Unit and the Packetizer Unit. The Dummy Bit Address Buffer size is 160 bits. The List Buffer stores the encoding status among Null, LIP, LSP, LISa, and LISb. The Packetizer Unit combines the bitstreams generated by multiple passes in the predefined format. In addition, it counts the number of dummy bits and uses them to store the bitstream that is coded after reaching the TBL.

The 1D SPIHT decoder organization is shown in Fig. 5 (b). The input bitstream is stored in the Bitstream Buffer, which stores the bitstream of the block to be decoded. The Parser Unit identifies the parts of the bitstream to be processed in individual passes and stores them in the Sub-Bitstream Buffer. The Code Length Prediction Unit and the Address Generation Unit calculate the address of each pass for the bitstream to decode and send this information to the Parser Unit. The Decoder Core Unit processes 20 units. However, as the sign bits that are decoded in the RP and FRP are delayed by a single cycle, 34 units are processed simultaneously. The List Buffer size in the decoder is 384 bits, which is twice as large as that in the encoder because sign bit decoding is delayed by a single cycle. The bits in the dummy bit position are temporarily stored in the Dummy Bit Buffer and are used later when the decoded bitstream length reaches the TBL.

III. PROPOSED SYSTEM

Multi-core spiht:

The parallel and pipelined method in [7] enables the number of processing cycles to be fixed regardless of the block size. Therefore, as the block size increases, the number of generated bits per cycle increases, consequently resulting in high-throughput. However, the hardware utilization may greatly decrease depending on the target speed or the incoming input data speed. Fig. 6 shows an example of execution in [7] in which FMC is connected to a bus with 64 bit-width and the DWT decomposition level is set by two. In this figure, input data come into the FMC with the target processing speed of 8 pixels/cycle in the burst mode. Fig. 6(a) shows the execution of an 8×8 block while Fig. 6(b) shows that on a 16×16 block. Gray and white squares in Fig. 9 represent sign and magnitude bit-planes, respectively, and dotted squares represent the delayed bit-

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

planes In Fig. 6(a), input pixels with 8 bit-width are transformed into wavelet coefficients in the 2D DWT module and then transposed to 10 bit-planes in the transpose module. The transposed bit-planes are transferred to the 2D SPIHT modules.

As shown in Fig. 7, the number of cycles to encode a single 8×8 block is 12 because delayed bit-planes, corresponded to the initial delayed two cycles, are added to avoid dependences. Therefore, the SPIHT hardware design shows throughput of 5.33 (=64/12) pixels/cycle when the block size used in FMC is 8×8, which is represented by a triangle mark in Fig. 3(c). In this case, the processing speed of SPIHT is slower than the target speed of 8 pixels/cycle which is represented by a square mark, and therefore the SPIHT design cannot work on the fly.

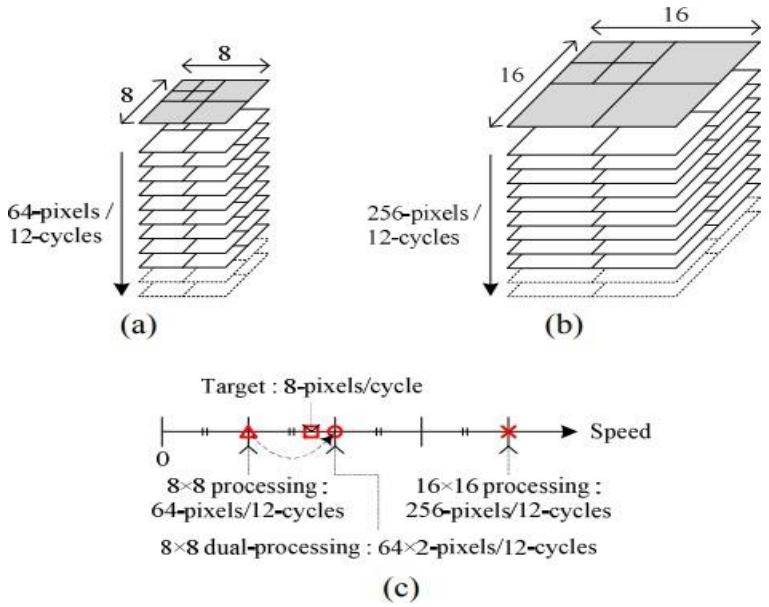


Fig. 6 Relationship between block size and throughput when the DWT decomposition level is two (a) processing an 8×8 block, (b) processing a 16×16 block, (c) target speed and processing speeds of (a) and (b).

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

Typically, the coding unit is a square-shaped block and its height and width values are determined in power of 2. When the height and width of an 8×8 block are doubled to increase the speed of SPIHT, the block size becomes 16×16. As shown in Fig. 3(b), the speed of SPIHT for a 16×16 block is 21.33 (=256/12) pixels/cycle, which is represented by a cross mark in Fig. 3(c). This speed is 2.67 times faster than the target speed. It leads to low hardware utilization because numerous hardware resources are exploited for the unnecessary speedup. If two 8×8 blocks are processed in parallel and pipelined manner, 128 (=64×2) pixels are processed in 12 cycles. In this case, the SPIHT design has the throughput of 10.67 pixels/cycle, which is represented by a circle mark and becomes closer to the target speed of 8-pixels/cycle. As a result, this design is more efficient in hardware utilization, achieving the required processing speed.

IV. RESULTS

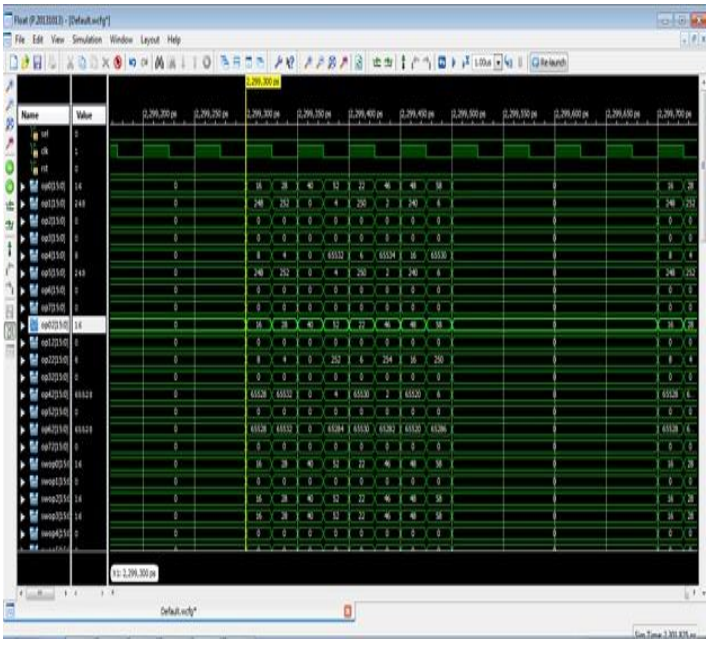


Fig. 7 Simulated Results

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

CONCLUSION AND FUTURESCOPE

CONCLUSION

This paper extends the high-throughput SPIHT hardware design in [7] to be fine-scalable and exploits it as an FMC integrated with the enhanced encoder. The proposed hardware design processes small-sized blocks in parallel and pipelined manner, resulting in similar coding efficiency and low hardware cost compared to the previous design with the same throughput. In addition, the proposed pass reduction scheme reduces hardware costs, which are particularly critical in packer and parser modules. The proposed design for a lossy FMC in video codecs shows higher coding efficiency than the previous FMC design, and it is more effective in whole codec systems requiring limited frame memory size, burst transferring protocol, and data random access.

FUTURE SCOPE

Further, this project is enhanced by using clock skewing technique to further reduce the combinational path delay. Input arriving time, combinational path delay, maximum output delay shows significant ruins in circuit designing. Latency improvements plays critical role in now a days. With Fixed Order SPIHT the throughput of the system increases by roughly an order of magnitude while still matching the original algorithm‟s PSNR curve. There is a sharp increment in the verification quality, for the signals recreated from Modified SPIHT than that of SPIHT method. This method can be utilized for other medical sign to decrease encoding and translating time as a result of having great PSNR at low bit rate than SPIHT. In future upgrade of this work will be carried out by different compression methods for the change of feature related work.

REFERENCES

I. Yiqi Tew, Kokshiek Wong, “An Overview of Information Hiding in H.264/AVC Compressed Video”, IEEE Transactions on Circuits and Systems for Video Technology, Vol.24, No.2,pp. 305-319, Feb. 2014.

II. Said Amir and Pearlman William A., “A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees” IEEE Transactions On Circuits and Systems For Video Technology, Vol.6,pp. 243-250, June 1996.

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

III. Sunwoong Kim, Donghyeon Lee, Jin-Sung Kim, and Hyuk-Jae Lee,"A High-Throughput Hardware Design of a One-Dimensional SPIHT Algorithm", IEEE Transactions on multimedia,Vol.18,No.3,pp.392-404,Jan. 2016.

IV. Huang C., Seng P., and Chen L., "Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform," IEEE Transactions on Signal Processing, vol. 53, no. 4, pp. 1575-1586,2005.

V. Y. Hu and C. C. Jong, "A memory-efficient high-throughput architecture for lifting-based multi-level 2-D DWT," IEEE Trans. Signal Process., vol. 61, no. 20, pp. 4975–4987, Oct. 2013.

VI. T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 6, pp. 529–534, Jun. 2003.

VII. Sonja Grgic, Mislav Grgic, Yuen Cheng, "Performance Analysis of Image Compression using Wavelets", IEEE transactions on Industrial Electronics, Vol.48, No.3,pp. 682-695, June 2001.

VIII. I. Venkata Anjaneyulu, P. Rama Krishna, "FPGA Implementation of DWT-SPIHT Algorithm for Image Compression", International Journal of Technology Enhancements and Emerging Engineering Research, 2015.

IX. Athar Ali Moinuddin, Ekram Khan, and Mohammed Ghanbari, "Low Complexity, Efficient and Embedded Color Image Coding Technique", IEEE Transactions on Consumer electronics, Vol.54, No.2, pp. 787-794, July 2008. [

X. H.-C. Kuo and Y.-L. Lin, "A hybrid algorithm for effective lossless compression of video display frames," IEEE Trans.on Multimedia, vol.14, no.3, pp.500– 509, Jun. 2012.

XI. A. Said and W. A. Pearlman, "A New, Fast, and Efficient Image Codec Based on Set Partitioning in

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights

Hierarchical Trees," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 3, pp. 243-250, Jun. 1996.

XII. F. W. Wheeler and W. A. Pearlman, "SPIHT Image Compression Without Lists," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 6, pp. 2047-2050, Jun. 2000.

XIII. T. W. Fry and S. A. Hauck, "SPIHT Image Compression on FPGAs," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, No. 9, pp. 1138-1147, Sep. 2005.

XIV. P. Corsonello, S. Perri, G. Staino, M. Lanuzza, and G. Cocorullo, "Low Bit Rate Image Compression Core for Onboard Space Applications," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 16, No. 1, pp. 114-128, Jan. 2006.

XV. C.-C. Cheng, P.-C. Tseng, C.-T. Huang, and L.-G. Chen, "Multi-mode Embedded Compression Codec Engine for Power-Aware Video Coding System," IEEE Transactions on Circuits and Syst

Page 16-41 © MANTECH PUBLICATIONS 2018. All Rights