Available online at www.sciencedirect.com

SCIENCE DIRECT®

Real-Time Imaging 10 (2004) 277–284

www.elsevier.com/locate/rti

Real-time DSP implementation of motion-JPEG2000 using overlapped block transferring and parallel-pass methods

Byeong-Doo Choi, Kang-Sun Choi, Min-Cheol Hwang, Jun-ki Cho, Sung-Jea Ko*

Department of Electronics and Computer Engineering, Korea University, Sungbuk-Ku, Seoul 136-701, South Korea

Available online 13 October 2004

Abstract

This paper presents a real-time implementation of Motion-JPEG2000 encoder using a fixed-point DSP chip. Among several modules in JPEG2000 encoder, the lifting algorithm for discrete wavelet transform (DWT) and the embedded block coding with optimized truncation (EBCOT) comprise more than 85% of the encoding complexity. Thus, it is very important to design and optimize these two modules in order to increase the performance of the hardware implementation. First we propose an overlapped block transferring (OBT) method that can significantly improve the performance of the lifting algorithm for DWT by increasing the cache hit rate. We show that the execution time of the lifting scheme can be further reduced by programming the DSP software using the single instruction multiple data (SIMD) instructions and the super scalar pipeline structure. Moreover, we introduce a parallel-pass method for fast implementation of EBCOT. This method reduces the processing time of EBCOT by processing the three coding passes of the same bit-plane in parallel. Experimental results show that our developed Motion-JPEG2000 DSP system meets the common requirement of the real-time video coding [30 frames/s (fps)] and is proven to be a practical and efficient DSP solution.
© 2004 Elsevier Ltd. All rights reserved.

1. Introduction

JPEG2000 compression standard has been created to provide high compression efficiency compared to JPEG [1]. It includes a rich set of features such as improved compression efficiency, lossy to lossless compression, multiple resolution representation, embedded bit-stream, region-of-interest (ROI) coding, and error resilience [2–4].

Motion-JPEG2000 (MJP2) is intended to create a new coding system required by video communication market and applications based on JPEG2000. The core technology of MJP2 targets an intra-based coding system, which differs from the current moving pictures standards, MPEG (MPEG-1, 2 and 4). It is well known that MPEG outperforms Motion-JPEG in compression efficiency because MPEG takes advantage of motion prediction between pictures. However, it is notable that MJP2 outperforms MPEG-2 and MPEG-4 in both the compression rate and error resiliency as presented in the recent study [5]. In particular, the advantage of MJP2 is outstanding in error prone environments. This is very important for consumer application as well as professional broadcasting systems.

DSP technology applied to various multimedia applications is also evolved fast. Recently, in the DSP technology, the single instruction multiple data (SIMD) instructions become usable [6]. As packing several small data types into a larger register, the SIMD instructions manipulate and process multiple data in an instruction, and thus reduce the execution time drastically.

In this paper, we present an embedded MJP2 system structure to encode video in real-time. The architecture primarily consists of three modules: the video acquisition module which obtains image data from two analog cameras, the MJP2 encoder module, and the local area network (LAN) module to transmit encoded code-streams via the Internet. For the MJP2 encoder, we propose the overlapped block transferring (OBT)

*Corresponding author. Tel.: +82 2 3290 3228; fax: +82 2 925 5883.
E-mail address: sjko@dali.korea.ac.kr (S.-J. Ko).

1077-2014/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2004.08.003

method, based on the cache performance to improve DWT. Instead of the line-based lifting scheme. An image is divided into overlapped subblocks and then each overlapped subblock is processed by a 2-D lifting algorithm to increase the cache hit rate. We show that the OBT-based lifting scheme with the SIMD instructions and the super scalar pipeline structure of DSP can increase the performance of the DWT drastically. Moreover, we propose a parallel-pass method for fast implementation of EBCOT. This method reduces the processing time of EBCOT by processing the three coding passes of the same bit-plane in parallel.

The paper is organized as follows: The proposed system level architecture is presented in Section 2. The OBT-based lifting scheme with the SIMD instructions and parallel-Pass processing for EBCOT are proposed in Section 3. In Section 4, the performance of the proposed system is discussed and conclusions are given in Section 5.

2. The implemented MJP2 system architecture

Fig. 1 shows the proposed block diagram for a hardware implementation of MJP2 encoder. This system is under development with ALTERA MAX7256 (256 LEs) and TMS320C6416 (600 MHz, 4800 MIPS, 128 kb cache). The video acquisition module captures NTSC and RS-170 analog video. The analog video is digitized into YUV 4:2:2 formatted video with two separate fields. These two fields are merged into a frame by an FPGA in Fig. 1. The frame generated is fed to the MJP2 encoder module for compression. Since the input to the MJP2 encoder has YUV format, the intercomponent color transform in the preprocessing of JPEG2000 is not required. The transmission module packetizes and delivers generated bit-stream via the Internet.

As shown in Fig. 1, the FPGA used in our hardware implementation plays an important role in transferring various data and controlling each module in the system. Fig. 2 depicts four main functions implemented in the

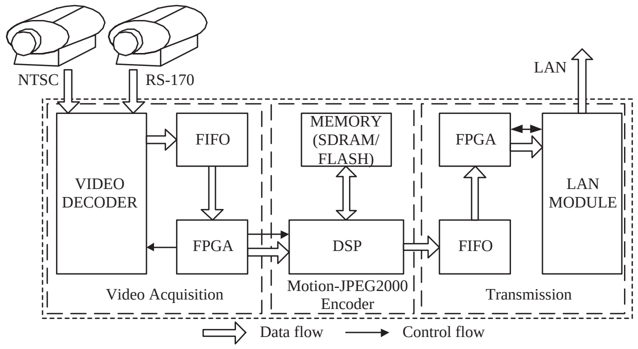

Fig. 1. Proposed block diagram for a hardware implementation of Motion-JPEG2000 encoder.

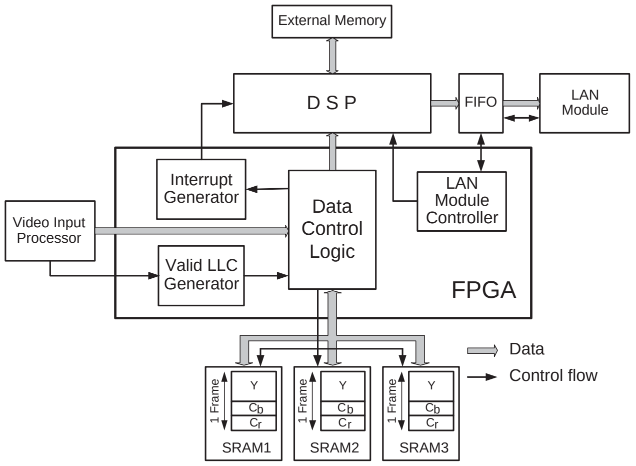

Fig. 2. Main functions implemented in the FPGA and interaction with external devices in the proposed hardware system.

FPGA in the proposed system. The FPGA is consumed up to 80%.

Video input processor chip represented as the video decoder in Fig. 2 digitizes analog video into YUV format video. The video input processor generates video of size 720 × 480 pixels. The input video can be cropped into the one of a desired size. For example, a concentric image of size 640 × 480 pixels can be extracted from the video. The video input processor generates a line-locked system clock (LLC) every 27 μs for all signals including blank signal as well as real pixel signal. The valid LLC generator in Fig. 2 produces a clock (valid LLC) only when the current signal outputted from the video input processor resides in the desired region. The data control logic (DCL) in the FPGA moves the current signal into the high speed SRAM.

The video input processor produces the valid pixel data for each field whose format is Cb0Y0Cr0Yl-Cb2Y2Cr2Y3 · · ·. To merge two parity fields into a frame whose pixel data are arranged in the order of the color component like Y0Y1Y2Y3 · · · Cb0Cb2- · · · Cr0Cr2 · · ·, the DCL should calculate the appropriate address where the current value is placed.

In order to cope with the case that the DSP may encode a frame slower than the output rate of the video input processor, the proposed system has three temporary high-speed SRAM's, each of which holds only one frame. Since the DCL contains the information about which SRAM is being used or is filled up with the latest video, the DCL always enables only one SRAM until the SRAM becomes full. Then the DCL changes the destination SRAM to next SRAM by rotating the destination circularly. After an SRAM has been filled up, the DCL signals the interrupt generator to fetch the latest image into the external memory associated with DSP chip via direct memory access (DMA) channels.

The LAN module controller indicates the buffer status of FIFO connected with the LAN module. The FIFO is used to suppress the difference of the throughputs of the DSP and the LAN module. Each time a buffer of the LAN module has a space for new data, the LAN module requests the FIFO to send data. The DSP output bitstream into the FIFO whenever the LAN module controller indicates that the FIFO has a space to be filled in.

3. The proposed software architecture of MJP2

Among several modules in JPEG2000 encoder, the lifting algorithm for discrete wavelet transform (DWT) and the embedded block coding with optimized truncation (EBCOT) comprise more than 85% of the encoding complexity. Thus, it is very important to design and optimize these two modules in order to increase the performance. The latest DSP chip can enable the real-time implementation of the DWT and adaptive binary arithmetic coding [6]. Utilizing the hardware features of the DSP chip, we optimize wavelet filtering and the EBCOT algorithm.

3.1. OBT-based lifting scheme for efficient cache utilization

The lifting algorithm, an alternative technique of computing the wavelet coefficients efficiently, performs the line-based DWT. Thus, lifting requires less computation as well as less memory. However, in a point of view of memory management, it still has severe cache-miss problems during the execution of the vertical wavelet filtering. Fig. 3 shows the reason the cache-misses occur in the vertical filtering with 128 set cache. When filtering the first 128 pixels of the first column, all data have to be fetched. When the 129th pixel is needed, the cache stores the oldest data and fetches the pixel

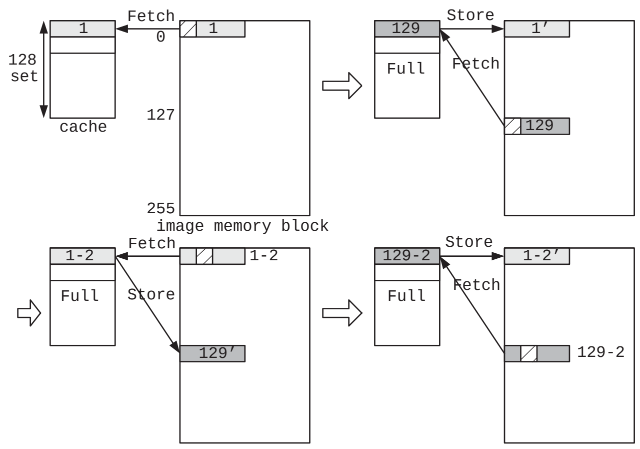

Fig. 3. Cache-misses in vertical wavelet filtering.

data. The same process executes in the 130th pixel, the 131st pixel, and so on. In the second column, the needed data that have been fetched does not exist in the cache any more. They have to be fetched again. Hence, all pixel data have to be fetched before they are filtered. Each of fetching takes 8 cycles of CPU processing. This is a lot of burden on CPU performance.

To reduce the cache miss drastically by increasing both the temporal locality and the spatial locality, we propose an OBT-based lifting scheme. This scheme uses hierarchical memory structure with two layers on DSP. Fig. 4 shows the memory structure of the DSP. The CPU interfaces directly to dedicated level-one data caches (Ll) of 16 Kbytes. This cache operates at the full speed of CPU access. It is a two-way set associative cache with a 64-bytes line size and 128 sets. A second level (L2) data memory is entirely mapped SRAM.

It services cache misses from the LI cache. The original image data are in the external memory mapped DRAM. Data transfer between the L2 and the external memory operates by a direct memory access (DMA) controller. An OBT-based lifting scheme partitions an entire image memory into blocks to fit into the cache size and reorder the processing sequence for the efficient lifting algorithm. It reduces the cache miss rate in the horizontal filtering.

Generally, in order to perform a lifting scheme, the image rows are filtered in the horizontal direction, and the image columns are filtered in the vertical direction, but the modified lifting scheme in the proposed method is performed block by block. The block must have the same size as one of the way in the LI cache. This is the key feature of the proposed approach.

Before the wavelet lifting transform starts, the data in the block 1 of the input image block are transferred from the input image memory to the L2 by the DMA. When the filtering in the horizontal direction starts, the CPU needs the first line of the block 1 and finds it in the L2 and fetches it in the L1 cache. After the filtering of the first line finishes, it moves to the second line of the block 1. Next, it moves to the third line and the fourth line, and so on. After the filtering of the last line of the block 1 finishes, the whole block 1 is located in the L1 cache, and then the filtering in the vertical direction starts in the block 1 in the L1 cache. No data fetching is needed in this vertical filtering. Therefore, during the filtering both of the horizontal direction and the vertical direction in the block 1, cache-miss occurs only when the CPU needs the next horizontal line. This is the reason the proposed method has prominent performance.

After the whole filtering of the block 1 finishes, the block 2 fetches in the way 1 in the L1. After finishing the filtering in the block 2, the block 3, starts to fetch from the L2 to the way 0 of the L1. At the same time, the filtered data of the block 1 in the way 0 move to the L2 line by line. The filtered data of the block 1 are already

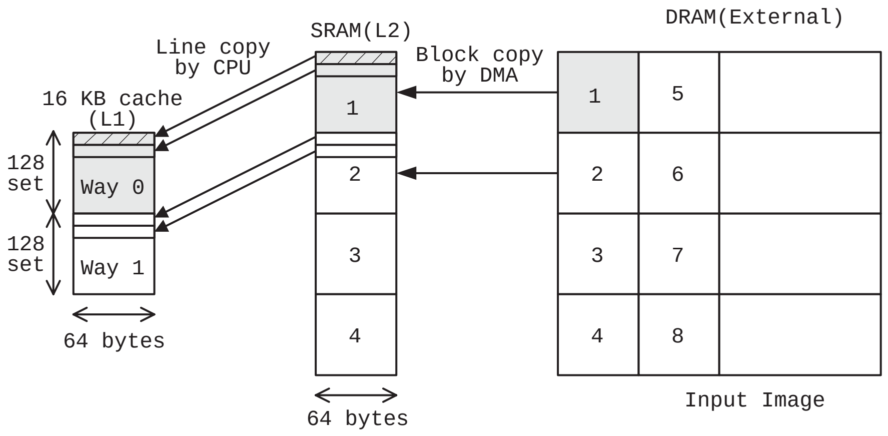

Fig. 4. Cache memory manipulation for efficient lifting algorithm.

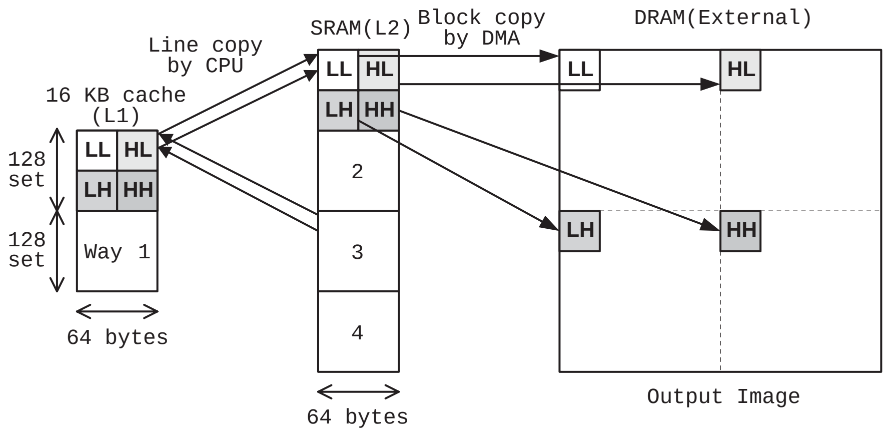

Fig. 5. Rearrangement of filtered data.

divided by each band such as LL, LH, HL, HH before they move to the L2. After finishing the block 1 movement to the L2, the data in each band transfer from the L2 to their correct final position in the output image memory by DMA. Fig. 5 shows the movement of the filtered data. Rearrangement of filtered data is needed to prepare for block coding.

Data transfer between the L2 and the external memory operates independently with data transfer between the L1 and the L2. As block data transferring is operated by DMA without CPU execution, the overhead of block transferring does not affect total processing time. Therefore, the proposed method increases cache-hit rate without any load to the CPU. We achieve this by reordering the sequence of the wavelet lifting, and by controlling DMA.

This method can perfectly remove a cache-miss problem. However, one problem exists in the proposed method because the wavelet filtering cannot be performed independently without coefficients of adjacent blocks. The filter needs next block coefficients at the edges to calculate exactly correct results. In the proposed method, the vertical lifting process is not related to this problem. Vertically two adjacent blocks are always loaded in the L1 cache in the same time. But in the horizontal lifting process, it is not easy to escape this problem.

In order to solve this problem, when the next right block is loaded, a part of the right side of the left block should be overlapped with this block. This method is referred to as overlapped block transferring (OBT) method. In Fig. 6, memory blocks used for double buffering from external memory to the L2 cache are laid overlapping each other along the vertical direction. Area 1 in dark gray is completely wavelet processed, whereas Area 2 in light gray contains data lifted partially. Thus, the next block for the 2-D lifting is placed to include Area 2 as well as Area 3. The remaining horizontal lifting steps for values in Area 2 are completed and the 2-D lifting scheme is processed for Area 3.

3.2. *Optimization of the lifting algorithm using SIMD instructions and the super scalar pipeline structure*

Although the lifting algorithm reduces computational complexity as well as memory used, the lifting algorithm still has much of computational burdens. In an attempt to improve the performance of the lifting algorithm

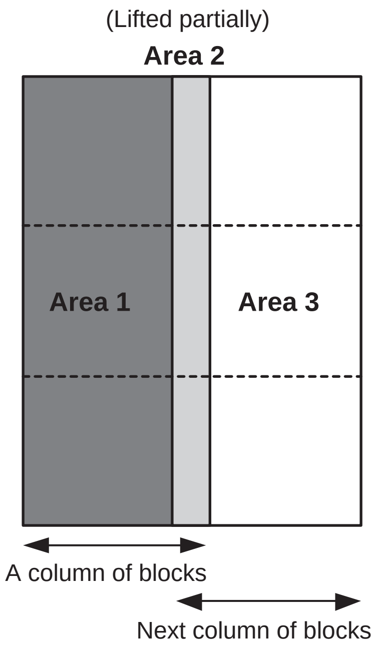

Fig. 6. Overlapped block configuration.

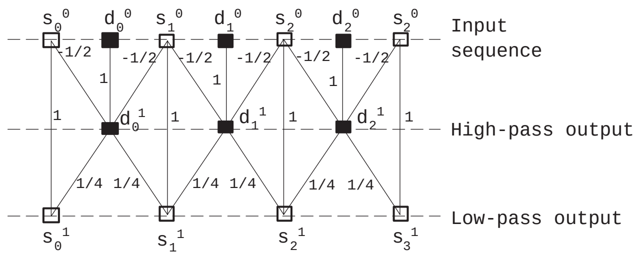

Fig. 7. Lifting prediction/update steps for the (5,3) filter-bank.

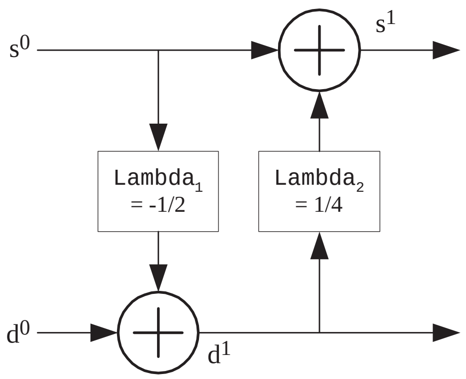

Fig. 8. Block diagram for the lifting steps for the (5,3) filter-bank.

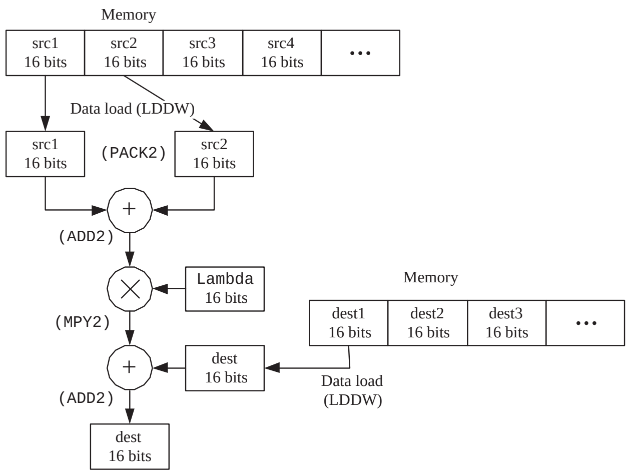

Fig. 9. Basic operation used in the lifting scheme for the integer (5,3) filter-bank.

further, we implement the lifting algorithm using the SIMD instructions of DSP and the super scalar pipeline structure.

The lifting operation consists of several steps. The basic idea is to first compute a trivial wavelet transform, called the lazy wavelet transform, by splitting the original 1-D signal into odd and even indexed sub-sequences, and then modifying these values using the alternating prediction and updating steps. Fig. 7 depicts an example of the lifting steps corresponding to the integer (5,3) filter-bank which is used in the reversible path for the lossless compression in JPEG2000. The sequences $\{s_i^0\}$ and $\{d_i^0\}$ denote the even and odd sequences, respectively, that are results of the lazy wavelet transform of the input sequence. The even and odd samples are regarded as initial values of low-pass and high-pass components, respectively. Then, values of high-pass component are updated by adding values of low-pass component with a weight. Next, values of low-pass component are updated by adding values of high-pass component with another weight. By iterating such update procedures, 1-D DWT is realized.

The prediction and updating steps are given as

$$d_i^1 = d_i^0 - \tfrac{1}{2}(s_i^0 + s_{i+1}^0), \tag{1}$$

$$s_i^1 = s_i^0 + \tfrac{1}{4}(d_{i-1}^1 + d_i^1). \tag{2}$$

Eqs. (1) and (2) are implemented by sequential update using a weighted sum of the adjacent values. Fig. 8 represents this lifting steps as a block diagram, where the lambdas, called lifting factors, in Fig. 10 are –1/2 and 1/4 for (1) and (2), respectively.

Fig. 9 shows the flowchart of the basic operation in the lifting scheme for the integer (5,3) filter-bank. The SIMD instructions, PACK2, ADD2, MPY2 and LDDW, are used. The PACK2 instruction takes the lower half-words from two source word data and packs them both into one destination word data. ADD2 and MPY2 instructions make two half-word data add and multiply on upper and lower register halves, respectively. LDDW instruction loads double word from memory with an unsigned constant offset or register offset. In this lifting scheme using SIMD, different pipeline hazards do not exist.

Thus, for an $N \times M$ image, a generic C code for the basic operation of the lifting scheme requires $2 \cdot N \cdot M$ 16-bit additions, $N \cdot M$ 16-bit multiplications, and $3 \cdot N \cdot M$ 16-bit data load. In general, a 16-bit multiplication, a 16-bit addition, and a 16-bit data load take 4 cycles, 1 cycle, and 5 cycles for execution, respectively. Thus, the generic C code requires total $21 \cdot N \cdot M$ cycles for processing an $N \times M$ image.

For the SIMD instruction can read 64 bits, that is, four 16-bit values are loaded simultaneously. In addition, DSP provides multiplication and addition instructions for two 16-bit values. Since two instructions can be used simultaneously by using the super scalar pipeline structure, four 16-bit values can be added or multiplied at the same time. Thus, using both the SIMD instructions and the super scalar pipeline structure reduces execution time drastically. For the case described above, the identical operation optimized requires only $5 \cdot N \cdot M$ cycles for processing an $N \times M$ image.

3.3. Parallel-pass architecture for EBCOT

Embedded block coding with optimal truncation (EBCOT) is the most complicated part in JPEG2000. The context of a sample coefficient is formed according to the significant state of the sample and its eight neighbors within a 3×3 context window, and the context data goes into the arithmetic coder. Each bit-plane is encoded through three coding passes, called significant propagation pass (Pass 1), magnitude refinement pass (Pass 2) and clean up pass (Pass 3). During each pass, all the samples of the bit-plane are scanned to determine whether or not each sample is encoded in the current pass. Therefore, all the samples need to be scanned three times, requiring a lot of processing time.

To solve this problem, we propose a fast context modeling method based on the parallel-pass scheme. The strategy is to process the three coding passes of the same bit-plane in parallel.

In EBCOT, a proper coding pass for the sample must first be determined, and then the sample is encoded during the coding pass. In this manner, each sample in the bit plane is encoded in one of the three passes. In order to reduce the processing time, three passes could be processed in parallel. However, the parallel processing causes a problem. If the three coding passes are concurrently executed, a sample in Pass 3 can become significant prior to its neighboring samples in Passes 1 and 2, resulting in a wrong implementation of EBCOT. Moreover, in EBCOT, the processing results of samples in Pass 2 or 3 depend on those of Pass 1. However, in parallel pass mode, samples in Pass 2 or 3 can not use the results of Pass 1.

In order to solve this problem, the coding operations for Passes 2 and 3 are delayed by one column to use the result of Pass 1, and Passes 2 and 3 are simultaneously processed. Figs. 10 and 11 show the proposed scheme. The results of four samples (numbered as 1) are stored after they are encoded in Pass 1. Then, the samples (numbered as 2,3) are encoded in Pass 2 or 3. In this case, the results of four samples (numbered as 1) are used as neighbors for Passes 2 and 3. After Passes 2 and 3 are completed, the two columns in box move to the right by one stripe. As a consequence, all three passes are encoded in one scan.

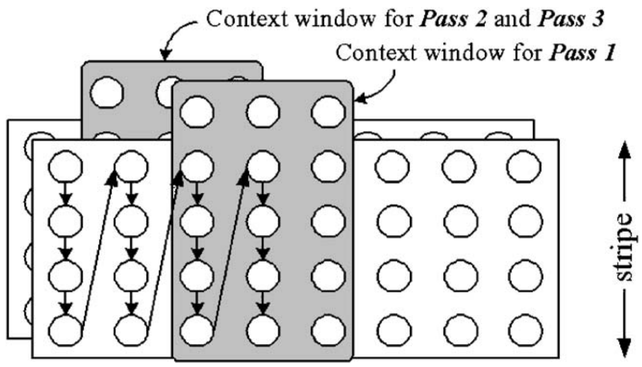
Context window for *Pass 2* and *Pass 3*
Context window for *Pass 1*
stripe

Fig. 10. Context windows in parallel processing.

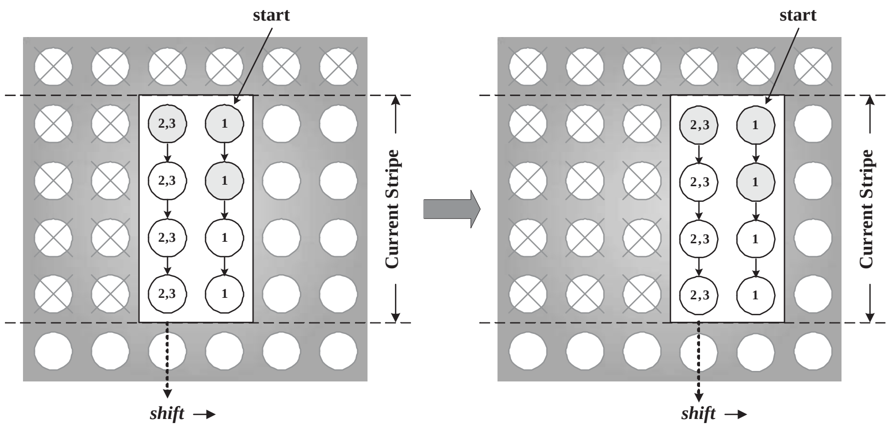

Fig. 11. Context windows in parallel processing.

4. Experimental results

The proposed OBT-based lifting scheme with the SIMD instructions and the parallel-pass processing are demonstrated in this section.

Table 1 shows a comparison of execution time of 2-D DWT for several image sizes. As shown in Table 1, the lifting method using the proposed OBT memory management scheme reduces the execution time of the lifting algorithm significantly. Note that the execution time is more reduced with the increase of the image size. The lifting scheme using the proposed OBT in our embedded MJP2 system performs over five times faster than the generic line-based lifting method.

Table 1
Comparison of execution time for different lifting schemes for Lena image

Image size	Generic (ms)	OBT (ms)
352×288	12.24	4.37
512×512	58.07	11.31
640×480	68.90	13.25

Table 2
Comparison of execution time of the (9,7) filter-bank lifting scheme between the generic C code and the SIMD implementation using the super scalar pipeline structure for 640×480 Lena image

Lifting direction	Generic C code (ms)	SIMD implementation (ms)
Horizontal	7.75	2.26
Vertical	7.75	2.39

Table 3
Comparison of the consuming time of the EBCOT between the non-parallel and parallel-pass processing for 640×480 Lena image

Pass type	Non-parallel (ms)	Parallel (ms)
Pass 1	14.49	14.76
Pass 2	7.02	4.43
Pass 3	26.75	17.37
Total	48.26	36.56

Table 2 shows the performance improvement by using both the SIMD instructions and super scalar pipeline structure of DSP. Since both the horizontal and the vertical lifting are applied to every pixel, the DSP SIMD implementation executes the lifting scheme about 3.4 times faster than the generic lifting method. Thus, our embedded MJP2 system using the two proposed schemes simultaneously performs encoding video over 18 times faster than the generic MJP2 encoder.

Table 3 shows the performance improvement by using the proposed parallel-pass algorithm for EBCOT. As shown in Table 3, for Pass 1, the proposed method does not affect the execution time because there is no difference between the proposed method and the generic method. However, for Passes 2 and 3, the proposed method reduces the calculation time up to 41% (Pass 2) and 32% (Pass 3). This result indicates that the proposed method significantly reduces the processing time for scanning and masking in case of Pass 2 and 3 by reusing the parameter and data used in Pass 1. In general, the computation complexity of the whole EBCOT can be reduced by 24% as compared with the generic architecture.

To measure performance of the proposed system, Table 4 shows time consumed and other dimensions of the design space, which are cycles, usage memory size and power consumption. Most computation time is consumed for the calculation in the EBCOT Tier-1 coder. The resources of the FPGAs in video acquisition module and transmission module are consumed up to 80% and 60% respectively.

Table 5 presents measured calculation time of different processing unit of JPEG2000, when various solutions for the realization of JPEG2000 are utilized. *Fossel's* implementation uses FPGAs (Xilinx XCV300E) with an additional processor support (ARM7) and *Meerwald's* implementation is parallel coding solution

Table 4
Performance and consumed resources for the proposed system

Modules	Algorithm	Time (ms)	Cycles (Kcycles)	Memory (Mbytes)	Power consumption
Video acquisition module (FPGA)	Pre-processing	3.40	170	922	3.3 V 80 mA
MJP2 encoding module (DSP)	DWT	4.65	2,790	922	
	Quantization	1.02	612	461	3.3 V
	EBCOT Tier-1	36.56	21,936	677	850 mA
	EBCOT Tier-2	14.18	8,508	28	
Transmission module (FPGA)	Packetization	0.41	21	40	3.3 V 60 mA
Total		60.32	34,037	2000	3.3 V/1A

Table 5
Comparison of total calculation time of JPEG2000 encoding for 256×256 Lena image

Implementation	*Fossel's* (ms)	*Meerwald's* (ms)	Proposed (ms)
Wavelet	4.0	421	1.5
EBCOT Tier-1	76.8	426	14.5
EBCOT Tier-2	13.7	200	7.9
Total	84.5	1047	23.9

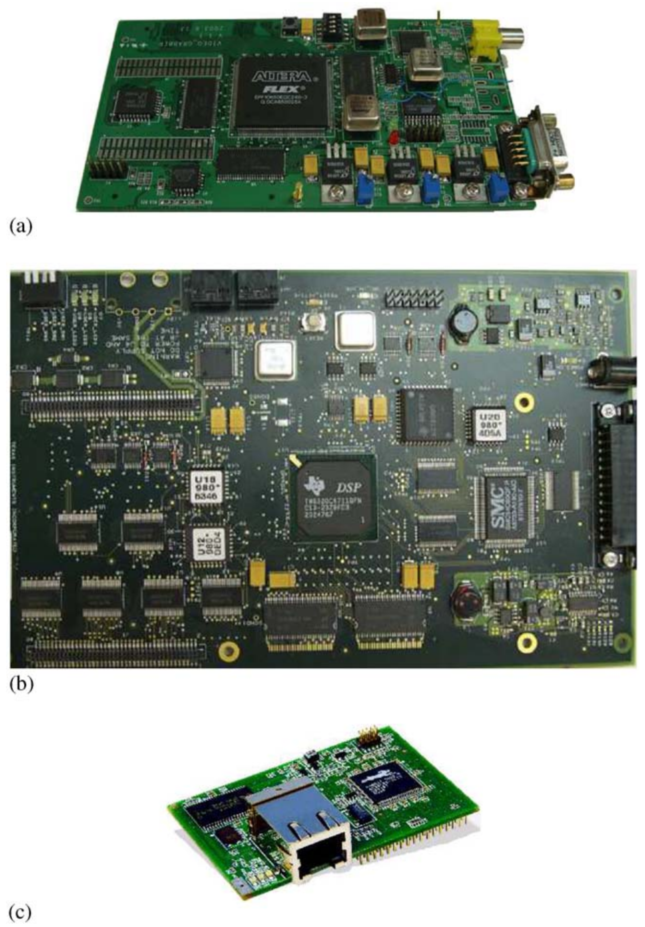
Fig. 12. The developed embedded Motion-JPEG2000 encoding system. (a) The video acquisition module. (b) The MJP2 encoder module. (c) The LAN Module.

on multiprocessors (Pentium II 500 MHz). The proposed system is about 3 times faster than *Fossel*'s implementation [9] and 70 times faster than *Meerwald*'s that [10]. Note that the processing time of DWT and EBCOT Tier-1 is reduced remarkably.

Fig. 12 shows the developed Motion-JPEG2000 encoding system consisting of the video acquisition module, the MJP2 encoder module, and the LAN module described in Section 2. By connecting these modules, the whole system can encode videos obtained from two cameras as well as transmit bitstream via the Internet in real-time.

5. Conclusions

In this paper, we have presented a real-time embedded Motion-JPEG2000 encoding system using a fixed-point DSP chip. To improve the performance of the system, we have proposed OBT-based lifting scheme to increase the cache hit rate. The OBT-based lifting scheme is over five times faster than the line-based lifting scheme. Moreover, the usage of the SIMD instructions and the super scalar pipeline architecture of DSP has reduced the wavelet execution time by over three times. In addition, we showed that the proposed parallel-pass algorithm can significantly reduce the execution time of EBCOT. Consequently, the MJP2 implementation on a fixed-point DSP meets common requirement of real-time video coding [30 frames/s (fps)] and is proven to be a practical and efficient DSP solution.

References

[1] Rabbani M, Joshi R. An overview of the JPEG2000 still image compression standard. Signal Processing: Image Communication 2002;17:3–48.
[2] Taubman DS, Marellin MW. JPEG2000: Image compression fundamentals, standards and practice. Dordrecht: Kluwer Academic Publishers; 2002.
[3] Information Technology—JPEG2000 Image coding system: Part 1. ISO/IEC International Standard 15444-1 2000.
[4] Information Technology—JPEG2000 Image coding system: Part 5—Reference Software. ISO/IEC International Standard 15444-5 2001.
[5] Yu W, Qiu R, Fritts J. Advantages of motion-JPEG2000 in video processing. In: Proceedings of the SPIE, Visual Communications and Image Processing 2002;4671:635–45.
[6] TMS320C64x Technical Overview. Texas Instruments 2001.
[9] Fossel S, Fottinger G, Mohr J. Motion JPEG2000 for high quality video systems. IEEE Transactions on Consumer Electronics 2003;49:787–91.
[10] Meerwald P, Norcen R, Uhl A. Parallel JPEG2000 image coding on multiprocessors. In: Proceedings of the International Parallel & Distributed Processing Symposium, 2002.