Design of Vector Register Architecture in DSP Processor for Efficient Multimedia Processing

Chou-Pin Wu and Jen-Ming Wu

Abstract—**In this paper, we present an efficient instruction set architecture using vector register file hardware to accelerate operation of general matrix-vector operations in DSP microprocessor. The technique enables in-situ row-access as well as column access to the register files. It can reduce the number of memory access significantly. The technique is especially useful for block-based video signal processing kernels such as FFT/IFFT, DCT/IDCT, and two-dimensional filtering. We have applied the new instruction set architecture to in-loop deblocking filter processing in H.264 decoder. Performance comparisons show that the required load/store operations for the in-loop deblocking filter can be reduced about 42%. The architecture would improve the processing speed, and code density in DSP microprocessor especially for video signal processing substantially.**

Index Terms—**Register File, DSP microprocessor, instruction set architecture, video signal processing, H.264**

I. INTRODUCTION

Much of the video signal processing standards such as H.264 and MPEG4 assume a CODEC model that uses block-based signal processing, for example, motion compensation, transform, quantization, entropy coding and deblocking filtering,...etc[1, 2]. These standards divide the frame into many macroblocks (e.g. 16 x 16 pixels per macroblock and 8 bits per pixel in H.264). These frames are compressed and decompressed block by block.

In the recent years and the coming future, the demand of video signal processing become more and more popular in portable devices, e.g. cell phones, PDA, digital media player, ...etc. These portable devices may need to support different video signal processing standards of MPEG-2/4 or H.264 for satisfying various applications. Therefore, embedded DSP microprocessors capable of supporting various needs by loading different programs are attractive for best flexibility. Many embedded DSP microprocessors have developed multimedia instructions in order to satisfy the characteristics of multimedia applications. Due to the fact that the data width of a processor is usually several times wider than the data type of images or videos of which the data type is usually 8-bit wide. The most popular approach is the SIMD-like instructions that operate on the image/video data concurrently. For example, a 32-bit DSP processor could support four 8-bit SIMD processing in parallel.

Data optimization and rearrangement is an important approach to achieve smaller code density and efficient processing of the DSP processor for video-streaming. The performance can be improved by changing the way data is organized and placed. The popular multimedia processing kernels, such as DCT/IDCT, horizontal/ vertical pixel padding, matrix/vector, matrix/matrix processing, require matrix transposition. However, transposition is expensive and cumbersome due the hardware architecture of the processor. It has been challenging in algorithm to do matrix transposition within the conventional register file [7].

In nature of images, the block-based video signal

Manuscript received Nov. 4, 2007; revised Nov. 30, 2007 .
The authors are with the Inst. of Communications Engineering , Dept. of Electrical Engineering, National Tsing Hua University, Taiwan, ROC.
Email: jmwu@ee.nthu.edu.tw

processing algorithms process data along the rows as well as along the columns. However, the architecture of register file in current DSP microprocessors is generally in the sequential fashion, or one dimensionally scalar. When these block-based video signal processing algorithms are executed, they usually need to process data along the rows and as well as the columns. This implies that the matrix, or the block, needs to be transposed frequently. As a result, after one dimensional processing (or row processing), they need to write the processed data back to memory, transpose, and then reload data to the register file. The transposition, therefore, consumes significant amount of time and instructions for memory access. In order to alleviate this problem, we present a new register file architecture design that is more suitable for block-based signal processing, called 2-D vector register architecture.

A register file architecture that provides row-access and column-access has been proposed in [8]. We build on the previous work and extend on it. Specifically, our main contributions are:

- We realize the hardware implementation of vector register file architecture and minimize the cost of read ports and the write port.
- We have addressed the potential data hazard problem that would rise from 2-D vector register file architecture.
- We have applied the 2-D vector register file architecture to practical de-blocking filter in the context of H.264 standard.

The remainder of the paper is organized as in the following. In Section II, we introduce the concept of the conventional 1-D scalar and the proposed 2-D vector data register architecture. Section III describes the proposed architecture and implementation. Section IV explains the performance comparisons. Finally, conclusions are drawn in Section V.

II. MOTIVATION OF VECTOR REGISTER FILE

In most of the DSP microprocessors, the architecture of data registers are one- dimensionally scalar. Fig.1. shows a 8x32 bits data register architecture using one 1:8 demultiplexer and two 8:1 multiplexers for two read ports and one write port. There are eight 32-bit scalar registers, R0, R1,..., R7. To address these eight registers, each of the read or write port requires 3-bit addressing.

When we use the register file to execute matrix–vector computation or block-based operation, we must execute the operation in two dimensions. In H.264, we partition the macroblock of 16 x 16 pixels into 4x4 subblocks. Each subblock consists of 4x4 pixels represented by *a, b,..., p* as shown in the Fig.2(a). Each pixel is represented by 8 bits/pixel.

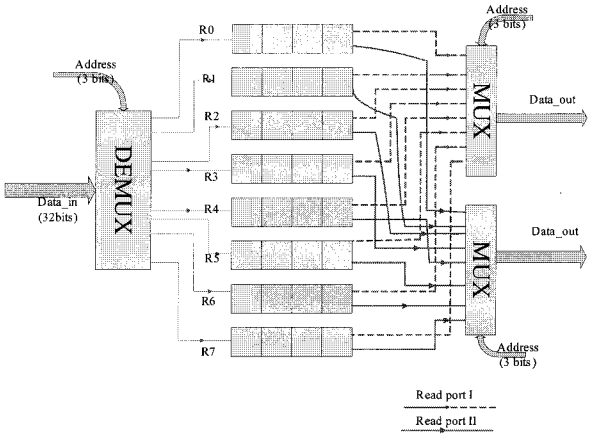


Fig. 1. The implementation of 1-D scalar registers.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

(a) 4x4 sub-blocks

R1	a	b	c	d
R2	e	f	g	h
R3	i	j	k	l
R4	m	n	o	p

(b) Four 32 bits registers

	C1	C2	C3	C4
R1	a	b	c	d
R2	e	f	g	h
R3	i	j	k	l
R4	m	n	o	p

(c) Sixteen 8 bits registers

Fig. 2. (a) 4x4 subblocks with each subblock of 4x4 pixels represented by *a, b,..., p.* (b) Permutation of the elements in the 1-D scalar register. R1 to R4 represents the different 32-bit registers. (c) The permutation of the sub-blocks in the proposed 2-D vector registers. R and C represent the row and column operation mode. These registers are addressed by R1,..,R4 and C1,..,C4.

In a load-store architecture microprocessor, when the processor executes the arithmetic operations, the 1-D scalar architecture has to load data into the register from memory first. Fig.2(b) shows the location of these subblocks in the 1-D scalar register with each register of 32 bits that it loads each row of the subblock into the same register. For example, pixels *a, b, c,* and *d* in the first row of the subblock are loaded into register R1, and so on. There are usually only two read ports (i.e. the two MUXes as in Fig. 1) in the register architecture. So, we can only address and process a 32-bit data operation from two of the registers. However, in the video signal processing, we usually need to process four pixels in the same row or column at the same time. This creates a limitation of the 1-D scalar architecture in Fig. 2(b). The 1-D scalar architecture could not perform the 32-bit column processing in the sub-block which would require to read one pixel from each of the four registers (R1,...R4) simultaneously. It has to store the data back the data memory temporarily after the row processing, update the address and reload the same data block after transposition, i.e. loading pixels *a, e, i,* and *m*, in the first column of the subblock into register R1 and so on. For the same pixel in subblock, we must load twice and store twice in order to finish row and column processing.

The limitation of 1-D scalar register architecture for block-based video processing motivates us of finding a better solution. If we can find a way to process the data in both row and column directions in the register directly, we can save requirement of instructions for load/store operations and reduce the program code size. The idea is to simply break the 32-bit scalar register into four 8-bits registers. Each 8-bit register can be addressed independently as shown in Fig.2(c). The proposed architecture is capable of two-dimensional operation by the ability of column and row register addressing, and hence called 2-D vector register architecture. We can use it to execute matrix operation or block-based algorithms in many video processing standards.

One potential problem that may be introduced from the 2-D vector register file is that data hazard may occur when the processor switches between row-access mode and column-access mode. There are several methods that can be considered to handle the data hazard problem. One way to mitigate the problem is to unpipeline the processor or insert delays every time the processor switch from row-access mode to column access mode, and vice versa. However, this will waste cycle times and slow down the processing. Another popular and powerful approach is to adopt register bypassing to improve processor performance and eliminate certain data hazards [9]-[10]. However, the extensive register bypassing process pipeline would result to significant impact on critical path, cycle time, area, and power consumption. Recent researche advocates the partial bypassing architecture to alleviate the impact [11].

III. THE ARCHITECTURE DESIGN AND IMPLEMENTATION

We design a set of *vector registers files* which actually divides the 32-bit scalar register into four 8-bit registers. Each 8-bit register element can be independently addressed. The eight 32-bit register file become thirty-two 8-bit register file array as shown in Fig .3.

The original scalar register architecture in Fig. 1 allows only 1-dimensional addressing (R1-R4). The "vector register" can access the 32-bit values of the register in either row or column fashion. As a result, the vector registers operate in two modes: row addressing mode and column addressing mode. In the row addressing mode, the value of register is processed in horizontal direction (R1-R4) to Arithmetic Logic Unit (ALU). While in the row addressing mode, it is processed in vertical direction (C1-C4). We can use one bit to decide the mode of vector registers.

As the result of two dimensions, we can operate the data in the "vector register" in any direction. When we execute matrix operation or block-based function, we can calculate these data using R1-R4 in row mode first. After the row dimension operation we do not need to store and reload the data to and from the memory to finish the transposition. We can operate the data using C1-C4 in another dimension temporarily. The elimination of write-back and reloading would significantly reduce the number of memory access and hence power consumption.

To realize the vector register architecture, we do not need to duplicate the read ports in Fig. 1 from two read ports to four read ports. We could just add one bit that can decide the mode of the vector registers. Consequently, the

implement cost of the proposed architecture is to expand the two read ports MUX from three bits to four bits to decode the mode bit as shown in Fig. 3.

In general, the vector registers architecture design is useful in all matrix-vector operations. It can decrease the computation complexity. The proposed architecture reduces the program latency due to much less memory access.

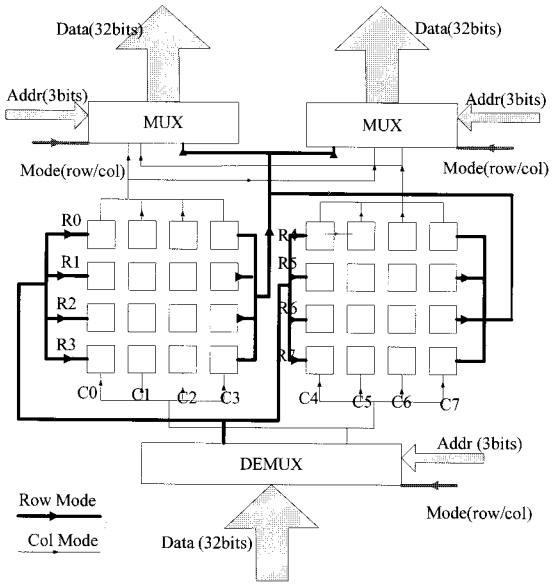


Fig. 3. Implementation of the proposed 2-D vector register file architecture. Because of two dimensions, we use one bit to address the operation mode (row or column).

IV. Experiment Result Of Deblocking Filter In H.264

In various video coding standards, the process of encoding and decoding are based on blocks (e.g. block-based motion compensation, or DCT/IDCT). As a result, artificial blocking effect is introduced on the edges of the subblocks. To remove this artificial blocking effect and reduce the image distortion, H.264/AVC enforces the deblocking filters inside its coding and decoding loop[1].

The deblocking filter process is invoked for the luma and chroma components. Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges for the same macroblock.

Inputs to the deblocking filter include pixels, boundary strength (BS), and threshold values as shown in Fig. 4. The operation of deblocking filter is decided according to boundary strength and threshold values[4,5,6].

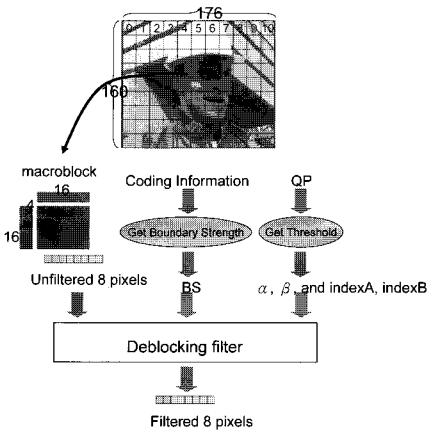


Fig. 4. The inputs to and outputs from the deblocking filter.

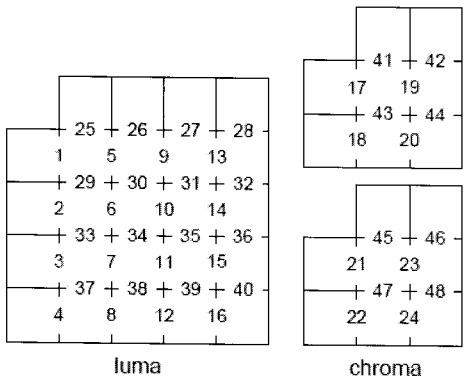
In the process of deblocking filter, we load the value of pixels to registers from the memory and then filtered the vertical edges. After the vertical filtering, we can immediately filter the horizontal edges using the column register addressing of vector registers. As a result, we can save the need of writing back to memory for transpose and reloading to register as in the scalar register architecture. The elimination of write-back and reloading would significantly reduce the number of memory access and hence power consumption for memory access.

We have conducted several experiments to compare the performance for different cases. In order to simplify the issue, we define some assumptions in our experiment:

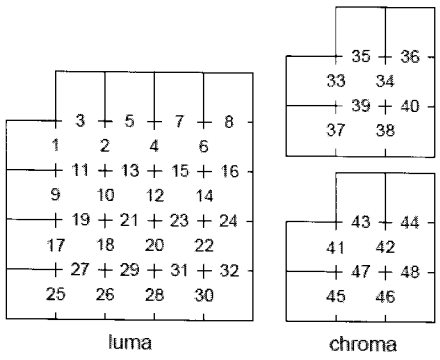
1. There are two read ports and one write port in the register as well as the SRAM memory.
2. There are total of eight 32-bit scalar registers or sixteen 8-bit vector register elements as data registers for fair comparison.
3. The instruction set architecture is 32 bit and the system data bus width is also 32 bits.
4. The system allows two parallel operations in one cycle, except the store operation.

For the filtering process, the instructions needed are load, store, vertical filtering, and horizontal filtering. Two store operations can not execute in parallel within one cycle due to the single write port limitation in SRAM.

Here we exam three different cases in the experiment. The Case A in the experiment uses 8 x 32 bits 1-D scalar registers, capable of accommodating two subblocks. The Case B uses two 8 x 32 bits 1-D



(a)



(b)

Fig. 5. (a) The order of the A case which use the original register. (b) The order of the C case which use the vector register .

scalar registers, capable of accommodating four subblocks. Case C uses the proposed 2-D vector architecture which has four 16 x 8 bits vector registers, capable of accommodating four subblocks.

In the original register, the filtering order is horizontal and then vertical as shown in Fig 5(a). In the proposed architecture we can operate in two dimensions. We can filter its vertical edge and horizontal edge without storing the block back to memory. Take the first subblock in the upper left corner in the macroblock as shown in Fig. 5(b) as example. It is the left vertical edge (edge 1) being filtered first, followed by its upper horizontal edge (edge 2) and then its right vertical edge (edge 3). We can finish these three edges with the upper-left subblock stay in the register and only its neighbor subblocks (left, up, and right) swapping between the register and the memory. Filtering the block's lower horizontal edge(edge 10) requires the subblock beneath it. So we would filter this edge when it comes to the next row. For each subblock, we can filter its vertical edge and horizontal edge without storing the block back to memory.

The result of the required number of load/store instructions for deblocking filtering experiment on one macroblock is summarized in Table I. Filter_V represents deblocking the vertical edges and Filter_H represents the deblocking the horizontal edges.

In our experiment, because the reduced number of load/store, the number of required instructions drops to 640. The required load instructions drops from 384 (in original register architecture) to 224 (in vector register architecture), 42% reduction of memory access. Without surprise, the same amount of 42% reduction occurs to store operation. The number of instructions reduces 1/3 and the number of cycles decreases 32.3%. All the reduction comes from the reduced load/store operations.

Table 1. The result of the deblocking filtering experiment on one macroblock.

	A case (original register)	B case (original register)	C case (vector register)
Load/Store	384/384	256/256	224/224
Fliter_V/ Filter_H	96/96	96/96	96/96
# of Insts Used	960	704	640
% (inst.)	100%	73.3%	66.67%
Cycles Used	486	358	329
% (cycle)	100%	73.6%	67.7%

V. CONCLUSIONS

In this paper, we have presented the use of 2-D vector register file in place of 1-D scalar register. Experiment results show that the processing capability of the proposed architecture is very good. It can reduce the number of memory access to achieve high performance. The power consumption is also reduced due to less memory access. The architecture would also improve the code size due to less load/store operations. The code size is also an important issue for embedded systems. Because it only add one multiplexer and two de-multiplexers to decode the operation mode, the proposed is easy to implement.

ACKNOWLEDGMENTS

The authors would like to thank the funding support of this work by Ministry of Economics, MOE (Taiwan), under grant No. 96-EC-17-A-01-S1-038.

REFERENCES

[1] "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," JVT G050, 2003.

[2] Iain E. G. Richardson,"H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia", *John Wiley and Sons*, 2003

[3] Philip P. Dang, "An Efficient Implementation of In-loop Deblocking Filters for H.264 Using VLIW Architecture and Predication," STMicroelectronics Inc., *IEEE Intl Conf on Consumer Electronics*, 8-12 Jan. 2005 pp.291-292

[4] Peter List, Anthony Joch, Jani Lainema, Gisle Bjntegarrd, and Marta Karczewicz, "Adaptive Deblocking Filter," *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 13, No. 7, July 2003.

[5] Shen-Yu Shih ,Cheng-Ru Chang and Yung-Long Lin "A Near Optimal Deblocking Filter for H.264 Advanced Video Coding" *IEEE Asia and South Pacific Conference on Design Automation*, Jan. 2006

[6] Yu-Wen Hung, To-Wei Chen, Bing-Yu Wang, and Liang-Gee Chen, "Architecture Design For Deblockong Filter In H.264/JVT/AVC," *IEEE Int'l Conf. On Multimedia Expo(ICME)*, Vol. 1, 6-9 July 2003, pp. 693-6

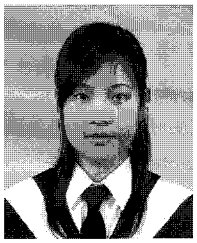
[7] B. Hanounik X. Hu, "Linear-time Matrix Transpose Algorithms Using Vector Register File With Diagonal Registers" *Proc. IEEE 15th Int'l Conf. on Parallel and Distributed Processing*, April, 2001

[8] Yoochang Jung, Stefan G. Berg, Donglok Kim, and Yongmin Kim, "A Register File with Transposed Access Mode," *IEEE Int'l Conf. On Computer Design*, Sept. 2000, pp. 559-560

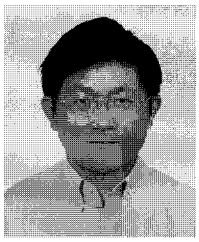
[9] Fan, K.; Clark, N.; Chu, M.; Manjunath, K.V.; Rajiv Ravindran; Smelyanskiy, M.; Mahlke, S," Systematic register bypass customization for application-specific processors," *IEEE Int' Conf.on Application-Specific Systems, Architectures, and Processors,* 24-26 June 2003

[10] A Shrivastava, E. Earlie, N.D Dutt, A. Nicolau, "Retargetable pipeline hazard detection for partially bypassed processors," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 14, Issue 8, Aug. 2006 pp.791-801

[11] A. Shrivastava, S. Park, E. Earlie, N.D. Dutt,A. Nicolau, Y. Paek, Y, "Automatic Design Space Exploration of Register Bypasses in Embedded Processors," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 26, Issue 12, Dec. 2007 pp. 2102 - 2115



Chou-Pin Wu received the B.S. degree in Electrical Engineering from National Dong-Hwa Univ., Taiwan in 2005 and M.S. degree in Communications Engineering from National Tsing Hua University Taiwan in 2007. She is currently with MediaTek Inc., Taiwan as member of technical staff. Her research interests include multimedia signal processing, embedded SoC design, and MIMO wireless communication technologies for mobile communications.



Jen-Ming Wu received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1988, and Ph.D. degree from University of Southern California, CA, in 1998, all in electrical engineering. From 1998 –2003, he has been with Sun Microsystems Inc. in Silicon Valley as senior design engineer. Since 2003, he has been with Inst. of Communications Engineering, Dept. of Electrical Engineering, National Tsing Hua University as Assistant Professor. Prof. Wu has worked on various fields of Electrical Engineering including digital signal processing, microprocessor architecture, wireless communication transceiver design, and high speed IO interface with optical transceivers, digital signal processing, and microprocessor architecture. Currently, his research works focus on embedded SoC design for H.264 and wireless baseband communication technologies for Mobile WiMAX and MB-OFDM UWB applications.