

An Efficient Self-Transposing Memory Structure for 32-bit Video Processors

Mahdi Nazm Bojnordi, Naser Sedaghati-Mokhtari, Omid Fatemi, Mahmoud Reza Hashemi
Nano-Electronics Center of Excellence, School of Electrical and Computer Engineering,
University of Tehran, Tehran, Iran
{m.bojnordi, n.sedaghati}@ece.ut.ac.ir, omid@fatemi.net, hashemi@comnete.com

Abstract— Many 2-D data processing applications can be simplified and represented by use of 1-D operations. Such tools, however, require applying both vertical and horizontal operations to the data blocks. The data transposing units is preferred to be used by the designers rather than applying individual operations for horizontal and vertical directions. Hence, designing a cost efficient and extendible transposing memory is a key issue for these applications. This paper proposes an efficient management strategy for using the SRAM modules in order to make a self-transposing memory architecture (STMA). In addition to its lower cost compared to flip-flop based buffers, the proposed architecture is more than 29% faster than usual SRAM based memory units. Simulations indicate that using the STMA in the H.264/AVC deblocking filter results in 60% speed improvement.

Keywords—Self-Transposing memory

I. INTRODUCTION

Many signal processing algorithms need to apply very complex 2-D operations to blocks of data. Representing 2-D operations by simpler 1-D ones have been widely considered in the literature to decrease the algorithm complexities [1]-[3]. These algorithmic modifications significantly reduce the implementation complexity, as well. In many memory bound applications, like video and image processing, 1-D tools are still the processing bottlenecks for real-time implementations. Hence, these tools have been targeted by the designers for cost and speed optimization.

In these applications, the newly defined 1-D operations are sequentially applied to horizontal and vertical lines of the desired data block. In order to decrease cost of the hardware implementation, horizontal and vertical operations are considered to be similar. Therefore, a processing engine is required for processing the data block in both horizontal and vertical directions. Also, a memory management is necessary to select the proper data line for the processing engine. Therefore, these architectures are composed of two main parts:

- *Transposing Unit (TU)*: transposes the data block and prepares appropriate data for both horizontal/vertical processing modes.
- *1-D Processing Engine (1-DPE)*: applies the required vertical and horizontal operations to the data lines of the input matrix.

The TU is responsible for transposing the data block in order to switch between the vertical and horizontal processing modes of the 1-DPE. In the TU, data accessing strategy is very important. Using simple memories results in decreasing the hardware area and power consumption. In turn, the processing speed decreases. In order to increase the processing speed, multiple access memory modules, e.g. register files, can be employed. Using these architectures increases hardware area and power consumption and hence, they are not suitable for low power applications.

In this paper, an efficient 32-bit self-transposing memory architecture (STMA) for using in video and image applications is proposed. STMA uses the SRAM modules for maintaining the block data; hence, it is cost-efficient. It can be adopted in various tools of image and video processing architectures, like DCT, IDCT, and deblocking filter. Based on the work in [4], a 1-D 32-bit processing engine for H.264/AVC deblocking filter is implemented to evaluate the STMA. Simulation results indicate that 60% speed improvement is achieved by using the STMA in the H.264/AVC deblocking filter.

The paper is organized as follows. The proposed structure for self-transposing memory is presented in Section II. The architecture is analyzed in Section III. Section IV presents experimental results and comparison. Finally, the paper finishes by the conclusion in Section V.

II. THE PROPOSED MEMORY STRUCTURE FOR 8×8 SELF-TRANSPOSING MEMORY

In many block-based video and image coding standards, images are divided to 8×8 data blocks to perform an efficient compression [5]-[7]. 2-D processing tools are applied to 8×8 data blocks, as well. An 8×8 data block, called M , is composed of sixty four 8-bit data entries, known as pixels. The pixels are the smallest accessible data elements in video and image applications. High-performance processors and accelerators, however, access to the data of sizes greater than a pixel. For example, 32-bit data bus is very frequently used in existing system-on-chips [8].

Typically, in a 32-bit video/image processing system, pixels are maintained in the memory in the form of quadruples of neighboring pixels (Q_i). Therefore, the memory map of the block M is as illustrated in Figure 1. The STMA is proposed based on this memory mapping. Its detailed structure is shown in Figure 2. The STMA has two ports for simultaneous reading

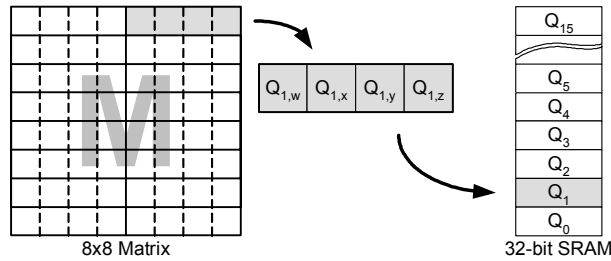


Figure 1 Mapping the 8×8 M data block to the 32-bit memory.

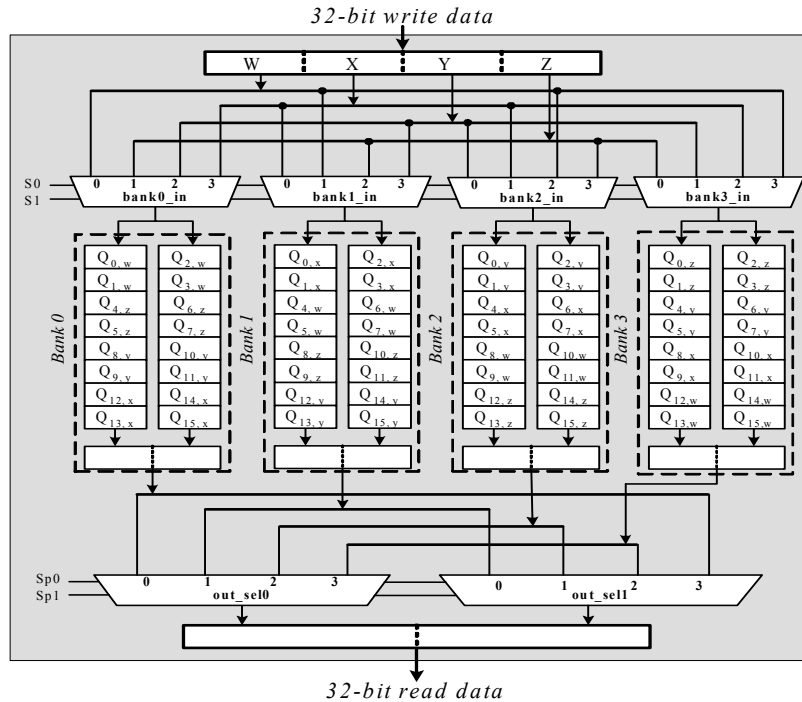


Figure 2 Structure of dual-port 32-bit STMA.

and writing. Each row of the block M can be stored in the STMA in the form of two 32-bit data words. Therefore, for reading or writing each row of the block M two clock cycles are required. The STMA employs four memory banks. Each bank is composed of two odd and even 8-bit two-port SRAM modules.

Input data, 32-bit words, are written to the memory in the form of four individual 8-bit parts called w , x , y , and z . A simple controller is considered that controls writing and reading to/from the memory banks. Controlling signals for the memory banks are generated easily by reordering the input write and read address lines and using a few logic gates. The controller structure for the proposed self-transposing memory

architecture is shown in Figure 3. Writing and reading data to/from the STMA are described as follows.

A. Writing to the STMA

In the STMA write process, each 32-bit input data is divided to four 8-bit data, i.e. pixels. According to the STMA write address, the pixels are instructed to specific memory banks. Data selection for the memory banks is performed using four 4-to-1 multiplexers for the input of the memory banks. TABLE I shows the addressing mechanism used for writing the data to the memory banks. We note that, there are two SRAM modules inside each memory bank, namely *odd* and *even*. Write address for these modules are produced according to the STMA write address. The same addresses are used for all

modules. However, write enable signals of the SRAM modules are controlled in order that only one module be enabled while writing the data. This way, each pixel is written in a specific module of a specific memory bank. As an example, suppose the write address is 1011. According to TABLE I, all the odd RAM modules are selected for this address. The pixels w , x , y , and z are instructed to the banks numbered 2, 3, 0, and 1 respectively. On the other hand, generated write address for the SRAM modules is 101. Therefore, the pixels are written to the address 101 of all odd SRAM modules.

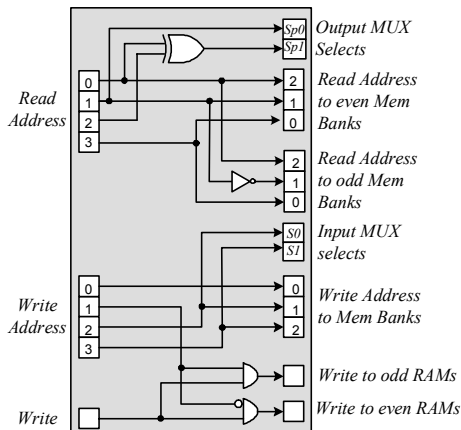


Figure 3 The proposed controller architecture.

TABLE I MEMORY BANK ADDRESSING STRATEGY FOR WRITE OPERATION.

Write Address	Bank Address	Odd/even Memory Part	W	X	Y	Z
0000	000	Even	0	1	2	3
0001	001	Even	0	1	2	3
0010	000	Odd	0	1	2	3
0011	001	Odd	0	1	2	3
0100	010	Even	1	2	3	0
0101	011	Even	1	2	3	0
0110	010	Odd	1	2	3	0
0111	011	Odd	1	2	3	0
1000	100	Even	2	3	0	1
1001	101	Even	2	3	0	1
1010	100	Odd	2	3	0	1
1011	101	Odd	2	3	0	1
1100	110	Even	3	0	1	2
1101	111	Even	3	0	1	2
1110	110	Odd	3	0	1	2
1111	111	Odd	3	0	1	2

B. Reading from STMA

Data arrangement of the pixels occurs during writing the data to the SRAM modules. In other words, by the described data distribution in the write process, the data block is prepared for transposing. And it will be completed by the read process. For this goal, corresponding data are selected and instructed to the memory output port using an especial addressing strategy. The memory bank addressing strategy for read operation is shown in TABLE II. For example, if the input read address is 0110, two separated parts are read from the banks 3 and 0. Data are read from addresses 000 and 010 of banks 3 and 0, respectively.

TABLE II MEMORY BANK ADDRESSING STRATEGY FOR READ OPERATION.

Read Address	First Bank		Second Bank	
	Address	Number	Address	Number
0000	000	0	010	1
0001	100	2	110	3
0010	000	1	010	2
0011	100	3	110	0
0100	000	2	010	3
0101	100	0	110	1
0110	000	3	010	0
0111	100	1	110	2
1000	001	0	011	1
1001	101	2	111	3
1010	001	1	011	2
1011	101	3	111	0
1100	001	2	011	3
1101	101	0	111	1
1110	001	3	011	0
1111	101	1	111	2

TABLE III COMPARING THE STMA WITH OTHER MEMORIES.

STMA	8-bit SRAM		32-bit SRAM*	
	one-port	two-port	one-port	two-port
Clock cycles	31	128	48	44

* Intermediate registers are required

III. DESIGN ANALYSIS

The STMA is valuable for using in both dedicated hardware and platform based system implementations. It offers a fast transposing mechanism by the lowest cost. For transposing every 8×8 data block, 16 clock cycles are needed to write all rows of the data block. Respectively, 16 clock cycles should be spent to read the transposed block. After writing 15 words of the data block, the transposed block can be read. Therefore, there is 31 clock cycles required for transposing every data block using STMA. Since in many applications, horizontal and vertical processes are sequentially applied to the data block, the proposed STMA satisfies the processing speed requirements.

Comparing with the cases using a single 8-bit two-port or one-port SRAM module, the STMA transposes the data block 72% and 75% faster, respectively. Using 32-bit memory instead 8-bit reduces number of reading and writing operations. In turn, extra circuitry is needed for data block transposing. TABLE III compares the STMA with the mentioned memories in terms of the required clock cycles for transposing. According to this table, STMA is at least 29% faster than 32-bit SRAM for transposing an 8×8 data block.

IV. DEBLOCKING FILTERING USING THE STMA

According to the H.264/AVC Recommendation [7], deblocking filtering algorithm is defined as a conditional process for updating the pixels of a macroblock to eliminate the horizontal and vertical blocking artifacts. The filter is applied to the horizontal and vertical line of pixels (LOP). Hence, this algorithm is suitable for evaluating the proposed STMA. For this goal, 32-bit H.264/AVC deblocking accelerator is designed. The architecture is composed of two parts, as follows:

A. Deblocking engine

Based on [4], a deblocking engine for processing an LOP is designed. This engine is able to process and update the pixels of each input LOP at every clock cycles.

B. Memory unit

Two memory managements are considered for the proposed accelerator. The first is using a single two-port SRAM module and several shift registers. And the second employs the proposed STMA for transposing. The total processing is affected by each of these memory managements that will be discussed in the next section.

V. SIMULATION RESULTS

For simulations, the deblocking engine is implemented by the Verilog HDL and synthesized by a standard 0.18 μm CMOS library. Synthesis results show that maximum clock frequency of 100 MHz is achieved for the deblocking engine.

Using the designed deblocking engine, two basic and STMA-based architectures are presented. The processing power of these architectures are compared in TABLE IV. According to the results of this table, the STMA-based architecture is 60% faster than the basic architecture.

TABLE IV COMPARING TWO DIFFERENT ARCHITECTURES OF THE DEBLOCKING FILTER.

	CPMB*	MBPS**	Supported resolution (working at 100MHz)
STMA-based	192	520833	(2320x1904)@30
Basic	814	208333	(1472x1200)@30

* CPMB: Clock cycles per macroblock

** MBPS: Macroblock per second

VI. CONCLUSION

A cost efficient and extendible transposing memory management strategy for SRAM modules has been presented. According to the proposed mechanism, 32-bit self-transposing memory architecture (STMA) for video and image processors has been implemented. The proposed STMA is 29% faster than single two-port memories in transposing the data block. It also can improve the total processing speed more than this in various applications. As an example, the STMA is used in a H.264/AVC deblocking filter. Simulation results show that 60% processing speed improvement is achieved because of using the proposed STMA.

REFERENCES

- [1] N.L. Cho, and S. U. Lee, "A Fast 4x4 DCT Algorithm for the Recursive 2-D DCT", *IEEE Trans. on Signal Processing*, Vol. 40, No.9, Sep. 1992
- [2] S. C. Chan, and K. L. Ho, "A New Two-Dimensional Fast Cosine Transform Algorithm", *IEEE Trans. on Signal Processing*, Vol. 39, No.2, Feb. 1991
- [3] Y. Huang, and J. Wu, "A Refined Fast 2-D Discrete Cosine Transform Algorithm", *IEEE Trans. on Signal Processing*, Vol. 47, No.3, Mar. 1999
- [4] M.N. Bojnordi, O. Fatemi, M.R. Hashemi, "An efficient deblocking filter with self-transposing memory architecture for H.264/AVC," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, pp. 925-928, France, May 2006.
- [5] Video Coding for Low Bit Rate Communication. ITU-T Recommend H.263, Feb. 1998.
- [6] *Information Technology - Coding of Audio-Visual Objects - Pan 2; Visual*, ISONEC 144496-2, 1999.
- [7] "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264/ISO/IEC 14496-10*, Mar. 2005.
- [8] W. Wolf, "The Future of Multiprocessor Systems-on-Chips," *Design Automation Conference*, pp. 681-685, 2004.