Area-Speed-Efficient Transpose-Memory Architecture for Signal-Processing Systems

Mohamed El-Hadedy[*]
hadedy@Illinois.edu

Divya Patel[†]
dnp6sm@virginia.edu

Martin Margala[‡]
martin_margala@uml.edu

Kevin Skadron[§]
skadron@virginia.edu

ABSTRACT

This paper presents the design and analysis of a high-speed implementation of a new transpose memory architecture. The proposed memory structure achieves almost 4X improvement in speed while consuming 46% less area, compared to prior work. For example, an 8X8 transpose memory with 12-bit input/output resolution has been implemented in 140 slices on a Virtex-7 Xilinx FPGA platform, achieving 107.83 Gpbs, clocked at 647MHz. The new transpose memory architecture allows 3.5X speed up in performance for the 2D-DCT algorithm, compared to previous work, while consuming 28% less area, and 2D-IDCT achieves 3X speed up compared to prior work, while consuming 20% less area.

Keywords

FPGA, Signal Processing, Adaptive Systems

1. INTRODUCTION

A wide-range of applications such as computer graphics, medical imaging and telecommunications all rely on signal-processing technology. Signal processing requires fast math, often on complex numbers, but many applications require computations in real-time: i.e., the signal is a continuous function of time, which must be sampled and converted to digital form and analyzed for real-time monitoring or control purposes. The processor must thus execute algorithms performing discrete computations on the samples as they arrive. Many media processors and digital signal processors (DSPs) use special memory architectures that are able to fetch multiple data and instructions for supplying multiple computational functional units at the same time.

Many signal-processing algorithms, such as discrete cosine transform (DCT) and inverse DCT (IDCT) [2] must repeatedly transpose matrices. Transposing a matrix using conventional operations—reading out rows and writing columns from/to the cache, or vice-versa—is expensive, requiring many clock cycles (and power). These read/write operations are a form of overhead, and a prime target for optimization to improve the performance and energy efficiency of algorithms involving transpose operations. Also DWT [8], FFT[13], and encryption[21] require transposition operations as they dominated by matrix techniques.

To support efficient transposition, memory architectures have been proposed (see Related Work, below) that allow direct access to both the rows and columns (in contrast to conventional memory structures that only allow row access).

Transpose memory can be implemented either with shift registers or SRAM. RAM needs to transpose the column or row elements. Solutions based on RAM lead to high latency, which have a high cost in area [14, 16, 12]. Implementations of the transpose memory with size configurable[3], and low latency [6, 5], are based on shift-register structures because of their flexibility and low over head control.

This paper describes the changes necessary to previously-proposed transpose-memory organizations to support operations on both edges of the clock, doubling throughput. This module can be used in a wide variety of DSPs and other organizations to support DCT, IDCT, and other algorithms requiring efficient transpose operations such as 2D-FFT [19, 21]. This paper describes the implementation of the double-edge transpose memory unit, as well as its use within both DCT and IDCT units.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 presents the transpose memory architecture. Section 4 presents the transpose memory performance along with comparisons with previously schemes. Sections 5 and 6 present the potential ap-

[*]Dr.Mohamed El-Hadedy is a research scientist with the Coordinated Science Laboratory, University of Illinois.edu at Urbana-Champaign, 1308 West Main street, Urbana, Illinois 61801-2307. This work was done while Dr. Mohamed EL-Hadedy was a Research Associate with the Department of Computer Science at the University of Virginia, 85 Engineer's Way, P.O.Box 400740, Charlottesville, Virginia

[†]Divya Patel is an undergraduate student with the Department of Computer Science at the Univeristy of Virginia, 85 Engineer's Way, P.O.Box 400740, Charlottesville, Virginia

[‡]Martin Margala is the chair of the Department of Electrical and Computer Engineering at the university of Massachusetts Lowell, Ball Hall Room 301, One University Avenue, zip-code: 01854, Lowell, Massachusetts

[§]Professor.Kevin Skadron is the chair of the Department of Computer Science at the University of Virginia, 85 Engineer's Way, P.O.Box 400740, Charlottesville, Virginia, USA

.

plications, which could benefit from the proposed TRM. Finally, some concluding remarks about the applications of this memory architecture are presented.

2. RELATED WORK

In the past, several architectures for transpose memory have been proposed. In 1995, Kovac et al. [10] introduced an 8x8 transpose memory as an array of register pairs. The data are input to the transpose memory in row-wise fashion until all the 64 registers are loaded. Then the transpose memory outputs the transposed version serially. The transpose memory has a latency of 64 clock cycles. On the other hand, for transposing 8x8 matrix by using our new architecture, the latency in this case is just four cycles.

In 2001, Agostini et al. [1] describe a transpose memory architecture for a row-column DCT architecture on FPGAs that relies on two RAM structures. While the first RAM is receiving the data from the first stage of the 1D-DCT, the second stage of 1D DCT reads the input values column by column from the other RAM. Those two RAMs are controlled by a control block, which decides whether a RAM should be in Read or Write mode at each memory-access step. The authors' reason for using RAMs for the transpose memory implementation is the availability of RAM blocks on the FPGA. In addition, the use of registers in FPGAs is costly in terms of logic cells. However, block RAM is inefficient when both row and column access are frequent. Although we rely on using the FPGA flip-flops and look-up tables LUTs to build the new transpose memory, careful design for efficiency achieves better performance in both area and performance compared to all prior works.

In this paper, our new memory transpose architecture is an improvement upon two prior implementations of similar FPGA-based, flip-flop-based transpose memory organizations. In the method proposed in 2007 by El-Hadedy et al. [6], a single-edged memory subsystem for data transposition is detailed. This subsystem, in its NxN implementation, takes N clock cycles to saturate all of its cells with values, and then N clock cycles to output the values before the next set of data can be input into the transpose memory. The transpose memory can only receive values in the horizontal direction and can only output values in the vertical direction. Thus, 2N clock cycles are consumed to obtain each transposed output set.

The method proposed in 2010 by El-Hadedy et al. [5] remedied this shortcoming by creating a memory subsystem that allows values to be input and output in both the horizontal and vertical directions. This allows for decreased latency because, while data is being output in a particular direction, input data can be fed into the transpose memory subsystem in the same direction, i.e. in a pipe-lined fashion. This means that, for every N cycles (in an NxN implementation), a new set of inputs will be loaded and a new set of outputs will be produced. Over many inputs, the number of clock cycles consumed to obtain each output converges to N cycles, which is half that of the 2007's memory transpose implementation.

Both architectures rely on using a register file consisting of connected cells to shift the data in horizontal and vertical dimensions based on the inputs' direction. The difference comes from using a different cell architecture. For instance, in [5], the cell consists of an input 2x1 multiplexer for choosing which input should process first. This is followed by a set of flip-flops (register) acting on the positive edge of the clock to store the data, which are shifting every-cycle. The output from the register is connected to a 2x1 de-multiplexer to choose to which direction (X-direction or Y-direction) the output should be assigned.

On the other hand, the cell architecture of the new TRM relies on using a 2x1 multiplexer to choose which input dimension should be processed—the row or column inputs. This is followed by two sets of flip-flops, one of them acting on the positive edge and the other on the negative edge. The outputs of these registers can then be used without a 2x1 de-multiplexer. Although the actual hardware changes are small, these insights yield a significant improvement in terms of speed. With the new architecture, by using the double-edge register, we decrease the latency by half, plus we increased the maximum frequency by removing the combinational circuit of the 2x1 de-multiplexer in the prior work [5]. This speeds up the new memory by almost 4x compared to [5]. In other words, this new transpose memory module can produce a transposed output matrix every N/2 clock cycles, assuming an NxN memory transpose system.

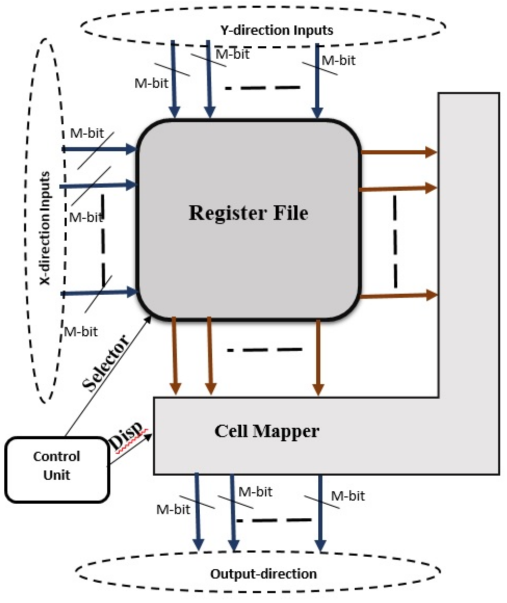


Figure 1: NxN M-bit Transpose architecture

3. TRANSPOSE MEMORY ARCHITECTURE

As shown in Fig. 1, the architecture of the transpose memory consists of three primary components: the register file, the cell mapper, and the control unit. The register file, shown in Fig. 2, operates on M-bit-long inputs. Each cell in the register file has a clock and an asynchronous reset signal, which synchronizes operation and reset. In addition, a selector signal dictates the direction of data flow within the memory transpose matrix: X or Y. At the end, the TRM has a display signal, which controls the direction of the outputs.

3.1 Cell

Each cell receives data from both X and Y directions, as shown in Fig. 3, and directly streams one direction's input to the appropriate output, shifting the data on each half-

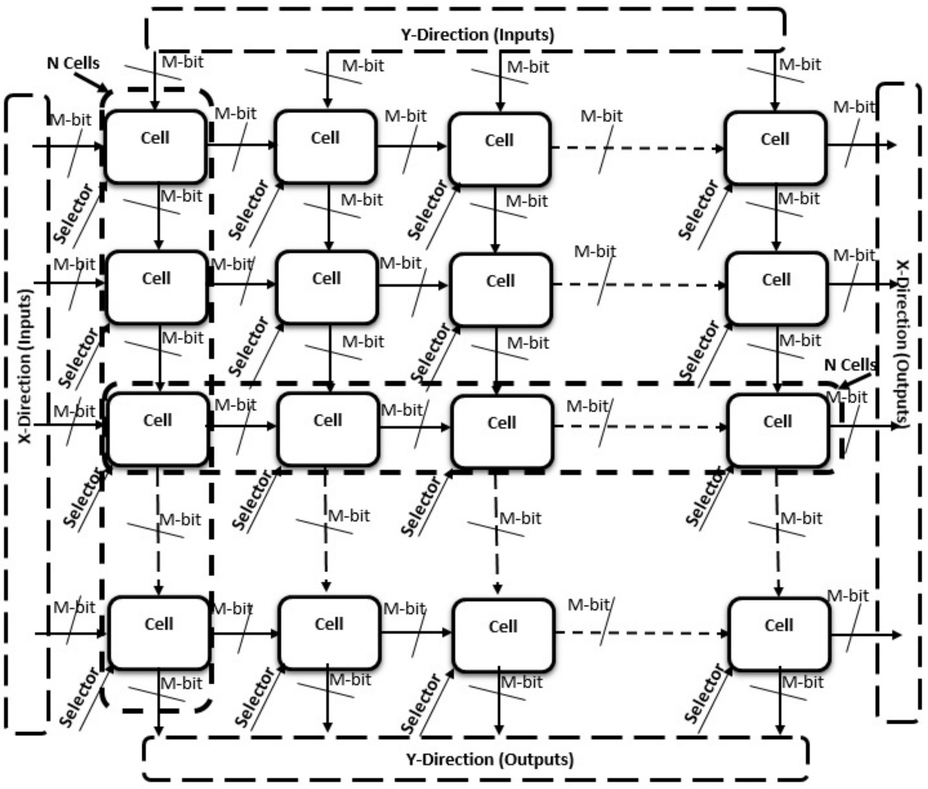
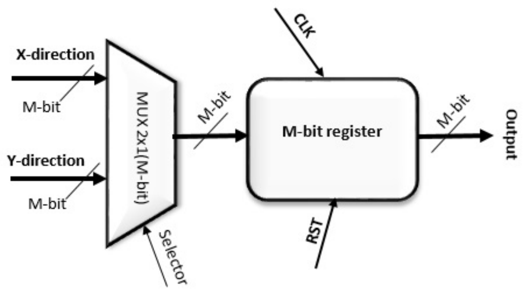
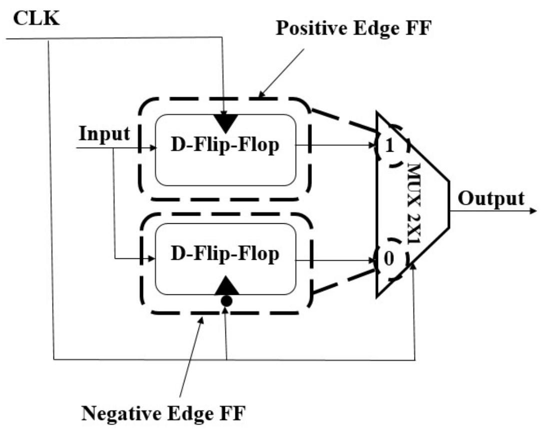


Figure 2: NxN M-bit register file architecture



(a) M-bit Cell Architecture



(b) Register structure

Figure 3: Register file internal architecture

clock edge. The cell consists of a 2x1 (M-bit) multiplexer, as shown in Fig. 3a, which is controlled by a 1-bit selector signal, and an M-bit register. The 2X1 multiplexer is used to select from from which direction the M-bit register receives data. The M-bit register is used to transfer the multiplexers' output on either the positive or negative edge of the clock CLK. The double-edge register architecture, as shown in Fig. 3b, is based on two sets of flip-flops running in parallel, one for the positive edge and the other for the negative. Both sets are connected to a 2x1 multiplexer, and the control bit of the multiplexer is connected to the clock.

By this way, the TRM process the data in both edges of the clock. Removing the de-multiplexer from the TRM in [5] decreases the propagation delay, which helps raise the maximum frequency compared to [5]. The register also has an asynchronous reset signal RST, which has a priority over the clock CLK and is used to clear the cell's output

3.2 Cell Mapper

As shown in Fig. 4, the cell mapper works as a multiplexer, which takes the outputs (X and Y directions) of the register file as inputs. The "Disp" signal determines which output values (X or Y) will be the output from the cell mapper on

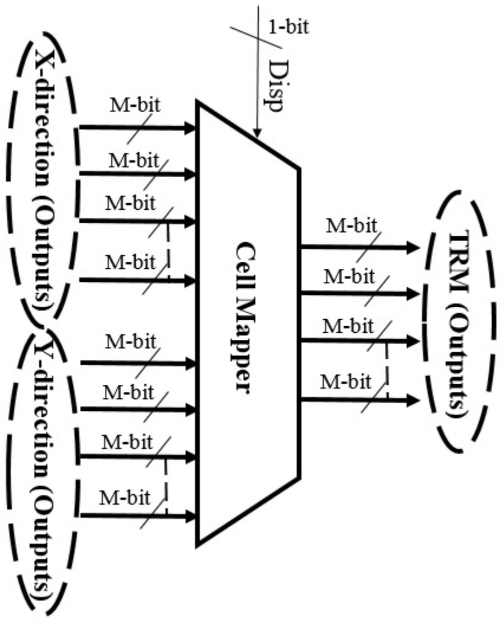


Figure 4: Cell mapper architecture

every clock edge.

3.3 Control Unit

The size of the control unit depends on the(TRM) dimensions. For instance, if the TRM is NxN, the control unit will act as a 2N-bit counter, counting on both edges of the clock. The most significant bit of the control unit (2N-1) determines the direction of the inputs and the outputs. The selector signal is connected to the most significant bit of the control unit. On the other hand, the disp's signal is connected to the inverse of the same bit. This gives the ability while the input comes from the x-direction, for the output to be taken from the Y-direction.

4. TRANSPOSE MEMORY PERFORMANCE

In this paper, the functionality of the proposed TRM was verified on the Xilinx Virtex-7 XC7VX485T-2FFG1761 device. The prior works were implemented on the Xilinx Virtex XCV800 [6, 5], so for fair comparison, the prior works are re-implemented on the new Virtex-7 (VC707). In this paper, VHDL is used for describing the prior and proposed works on the FPGA platform and was synthesized using ISE design suite 14.7.

Fig. 5 and Fig. 6 show the resource utilization of the 8X8 dual-edge TRM with different input/outputs resolutions on Virtex-7 platform. As shown in Fig. 7, the total area of the area of the proposed memory is a function of the input/output resolution. The area steadily increases with word size, while maximum frequency is almost unchanged, it varies from 626 to 656 MHz.

In FPGA platforms, the area is typically reported based on the total number of slices. A slice contains some number of Look-Up-Tables (LUTs), flip-flops (FFs) and multiplexers (MUX). For example, a Virtex-7 contains four LUTs and eight flip-flops [7]. As shown in Fig. 6, the proposed dual-edge TRM relies on using these LUTs and FFs, and the utilization increases with resolution.

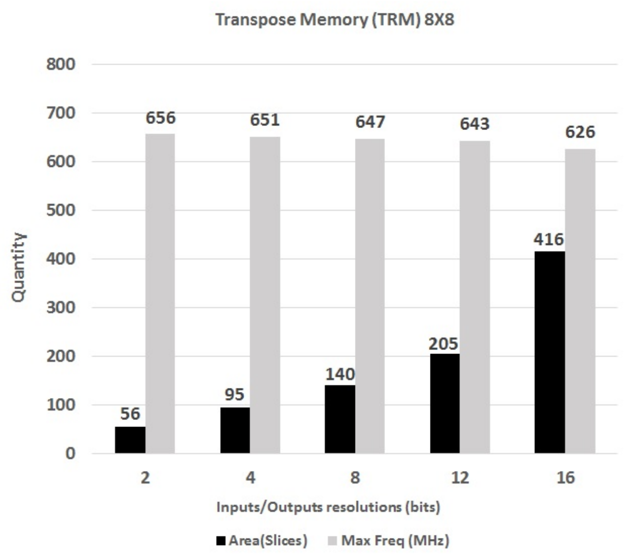


Figure 5: TRM 8X8 Performance, Area/Throughput

$$Throughput = \frac{Number\ of\ input\ bits \times Max frequency}{Number\ of\ clock\ cycles\ per\ block} \quad (1)$$

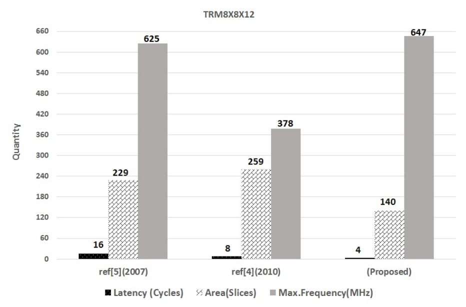


Figure 7: TRM 8X8X12 comparisons on VIRTEX-7, Area/max.freq comparison

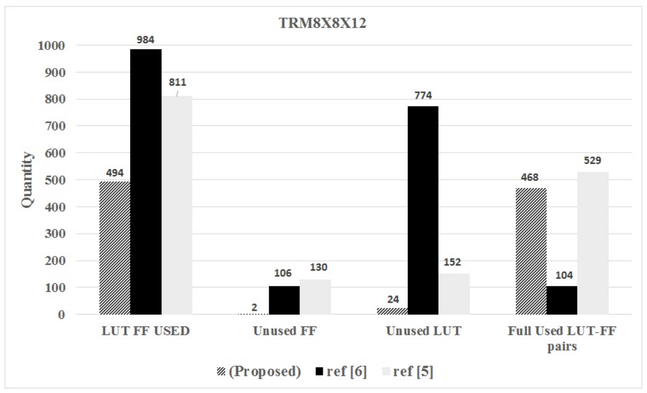


Figure 8: TRM 8X8X12 on ViRTEX-7, Resource utilization comparison with 12-bit word-size

As shown in Fig. 7, in terms of area, the proposed TRM is 39% smaller than [6] and 46% smaller than [5]. In terms

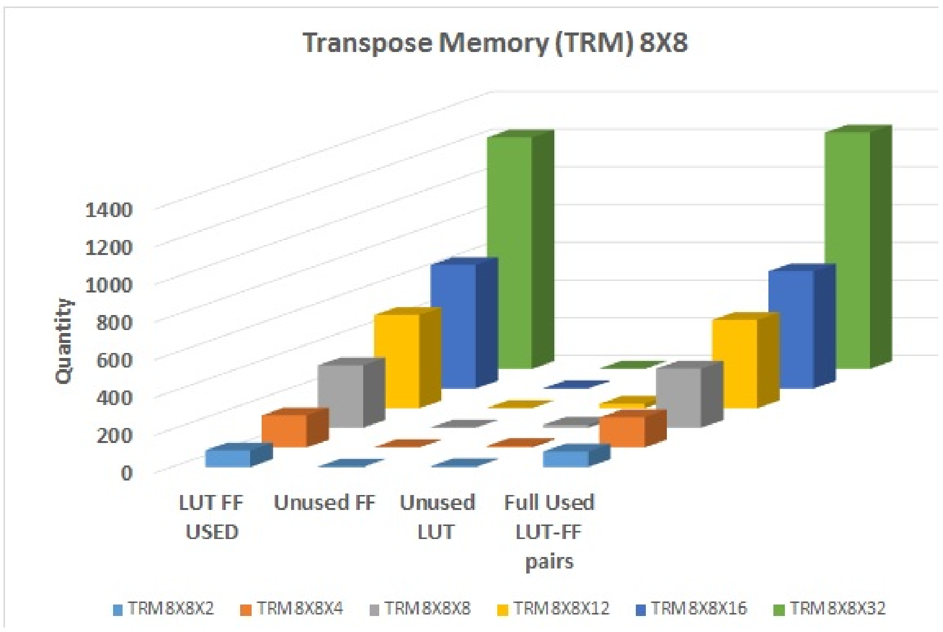


Figure 6: TRM 8X8 Performance, Resource utilization comparison with different word-sizes

of max. frequency, the proposed TRM works at 647 MHz, which is 3.5% better compared to [6] and 71% better compared to [5]. In terms of latency, for the TRM8X8, the proposed TRM has fewer cycles of latency compared to prior works; for example, for the 8X8 TRM, the proposed TRM has just 4 cycles, compared to 8 cycles in [5] and 16 in [6].

The results of Fig. 8 show that the proposed TRM does consumes 350% more full LUT-FF pairs than [6] and 11% less than [5]. However, for *total* LUT-FFs used (unused FF + unused LUT + Full used LUT-FF pairs), the proposed TRM consumes almost 50% less than [6], and 39% less than [5]. The proposed TRM, in terms of LUT-FF pairs compared to prior work, is expensive due to using these LUTs for building the double-edge sets of registers per each cell in the register file, but is still able to yield area savings and performance improvement.

These results in Fig. 7 and Fig. 8 show that the proposed TRM, using just LUTs and FFs, achieves better performance in terms of area and frequency compared to prior work. By applying equation. 1 on the posted data in Fig. 7

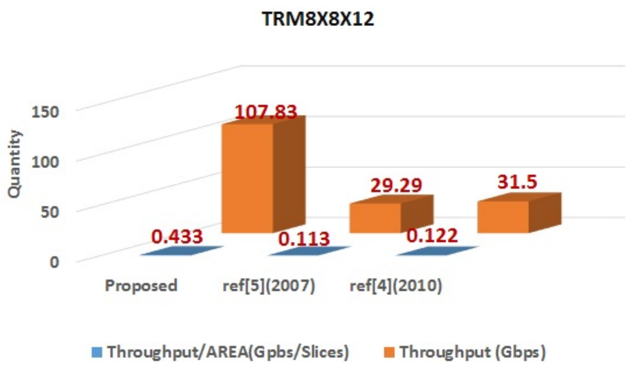


Figure 9: Speed/Area Performance

and Fig. 8, we can find in Fig. 9 that the proposed TRM is 3.7X faster than [6] and around 3.4X times faster than [5].

5. APPLICATIONS

For better comparison, the proposed transpose memory has been integrated in both 2D-DCT and 2D-IDCT blocks [4]. Specifically, the 2D-DCT and 2D-IDCT implementations and transpose-memory structures in [5] have been re-implemented on the Virtex-7.

5.1 2D-DCT component

Numerous applications such as lossy compression of images (e.g. JPEG [15] and watermarking [6]) rely on using the DCTs, to spectral methods for numerical solutions of partial differential equations[20].

Although 2D-DCT can be performed on blocks of various size, experiments have shown that compression is always a trade-off. One can always get sharper images by keeping more information. Experience shows that 8x8 blocks provide a good balance between fidelity and compression [17]. Equation 2 describes the formula of the 2D-DCT (omitting normalization and other scale factor), where N and M represents each dimension size, f(i,j) is the intensity of the pixel in row i and column j, and F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix. For instance, in 8X8 2D-DCT N = M = 8.

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i,j) \cos\left[\frac{\pi u}{2N}(2i+1)\right]$$
$$\cos\left[\frac{\pi v}{2M}(2j+1)\right] f(i,j)$$

where,

$$\Lambda(i) = \begin{cases} \frac{1}{\sqrt{2}} & , \text{for } \varepsilon = 0 \\ 1 & , \text{otherwise} \end{cases} \tag{2}$$

Computing a 2-dimensional DCT is typically achieved by two 1D-DCT computations, one in the X and one in the Y dimension, with a transpose unit between them. This is known as a row-column algorithm. El-Hadedy et al [5] relied on a combinational architecture to build the 1D-DCT followed by a register processing the data every cycle to decrease the effect of the critical path. In this paper, we used the same structure of the 1D-DCT in the prior work while modifying the end-stage register to perform every half cycle by applying the approach in Fig. 3b.

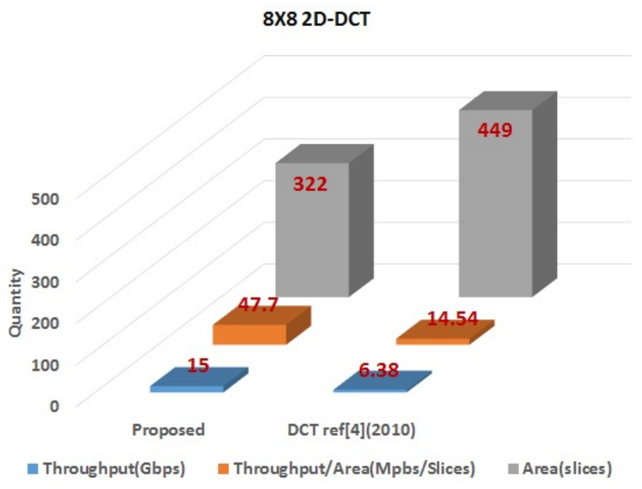


Figure 10: Performance result of the 8X8 2D-DCT using the proposed TRM

The performance comparison between [5] and the proposed double-edge TRM in Fig. 10 shows that the double-edge TRM improves the performance of 2D-DCT, with 3.5X speedup and a 28% reduction in area.

5.2 2D-IDCT component

The IDCT decodes an image back into the spatial domain from a frequency-domain representation of the data better suited to compression. It is the inverse operation of the DCT in section 5.1.

The 2D-IDCT consists of three blocks. The first and the last blocks are 1D-IDCT and the middle block is the transpose memory. The 1D-IDCT in this paper relies on using the modified Loeffler's technique [4] with modifications in [5], so that one 1D-ICT operation requires 11 multiplications and 29 additions, using the pipelined approach as shown in [5]. Fig. 11. Each $\sqrt{2}C_n$ block consists of three multiplication and three adders/subtractor [6].

A shown in Fig. 12, by integrating the proposed TRM in the 2D-IDCT, a speedup of 3X is achieved compared to the prior work [5], while the total area decreased by 20%.

6. COMPACT 2D-DCT PROCESSOR

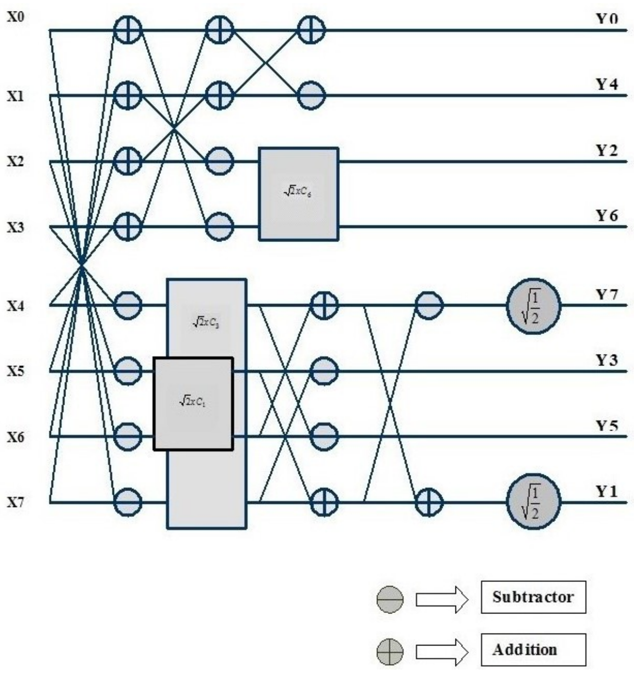


Figure 11: Modified 1D-IDCT [6]

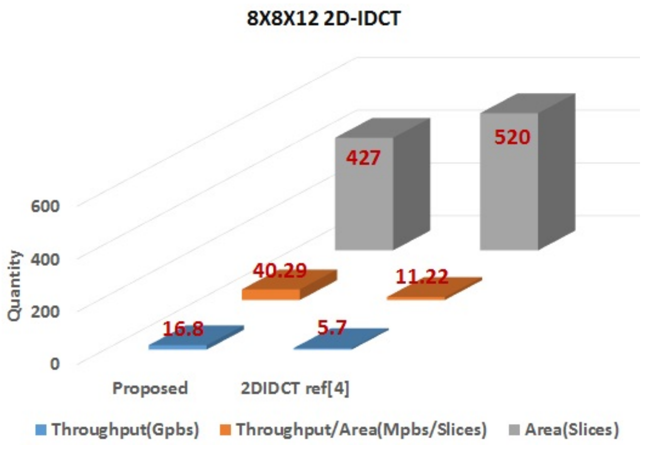


Figure 12: Performance result of the 8X8X12 2D-IDCT using the proposed TRM

Area is critically important for many applications, such as medical, military, and space applications. Instead of capturing the data from a local sensor and transmitting the raw data, it is preferable to compress it, or perhaps perform local analysis and send only results. These operations rely on algorithms such as DCT and IDCT.

The speed improvements of the TRM in section 4 can be invested to build a compact version of the 2D-DCT that relies on using one 1D-DCT connected to the TRM and looping back. This provides almost the same speed as the fastest version of the prior work [5], with a much smaller area.

As shown in Fig. 13, the new processor processes 8X8 blocks with 8-bit input resolution per element. It consists of a "padder," parallel data-bus, 1D-DCT, TRM, and the control unit. The total latency of this processor is 10.5 cycles.

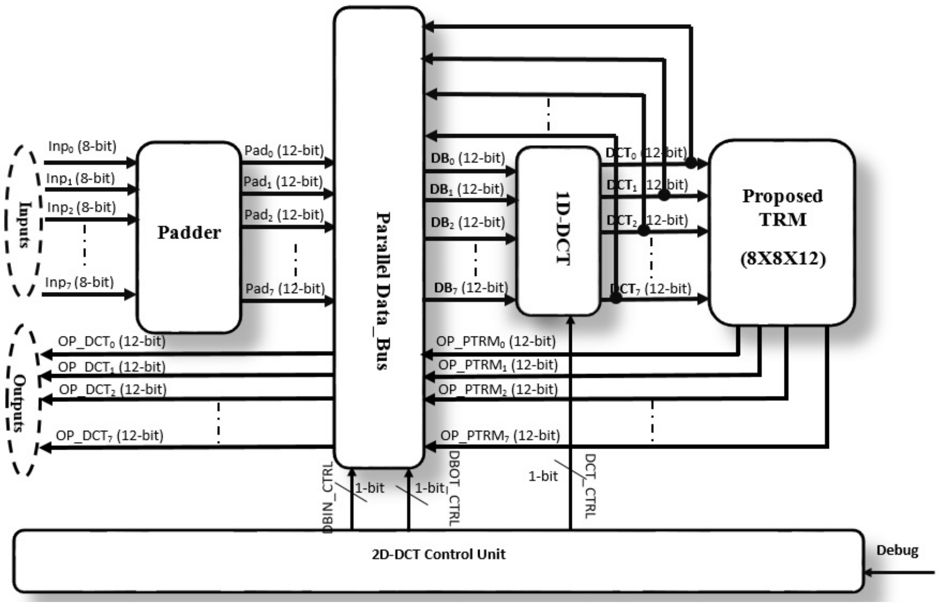


Figure 13: 2D-DCT compact processor

6.1 Padder

It is a combinational circuit, which converts the input stream resolution from 8-bit to 12-bit width by adding four zeros on the most significant bits (The extra bits are needed by the 1D-DCT).

6.2 Parallel Data_Bus

This works as two parallel multiplexers. The first multiplexer takes the output streams of the padder and the TRM and sends them to the 1D-DCT unit according to the control unit's DBIN_CTRL(1-bit) signal. The second multiplexer takes the output streams from the TRM (for debugging purposes) and the 1D-DCT and sends them as 2D-DCT outputs according to the control unit's DBOT_CTRL signal. The same architecture of the 1D-DCT and TRM in section 5.1 and section 3, respectively are used.

6.3 2D-DCT Control Unit

The control unit consists of a 5-bit counter, which is reporting the output every a half cycle. It controls the parallel data-bus unit and DBIN_CTRL(1-bit) through DBOT_CTRL(1-bit), DBIN_CTRL(1-bit), and DCT_CTRL respectively.

6.4 The Performance of the 2D-DCT Compact Processor

The processor has been implemented in 256 slices, requiring 13.5 cycles to process 96 byte blocks of data, achieving throughput of 5.6 Gps at 100 MHz. The throughput/area (Mbps/Slices) ratio of the compact processor is 22.4, which is higher than the ratio of the implementation of the 2D-DCT in [5] by 54%. On an FPGA, this frees up area, which allows a smaller FPGA to be used, or allows the FPGA to support a greater amount of functionality. It is also suggestive of the potential savings in an ASIC implementation.

Table 1 shows a comparison between the implementation of the 2D-DCT by using the proposed TRM and others. The throughput/Area (Mbps/slices) ratio of both fast and compact processors are higher than the ratio of the prior work. For instance, the ratio of the fast processor is higher than the ratio of the implementation in ref[18, 11] by 15 times. Even though, the maximum frequency in ref[9] is higher than the proposed fast version by 12.8%, the throughput/area ratio of the fast version is 34 times higher than it. The throughput of the proposed fast version is almost 2X higher than [18, 11], almost 5X higher than the implementation in [11], and almost 8X higher than in [9].

7. CONCLUSIONS

In this paper, we presented an FPGA implementation of a new transpose memory architecture that leverages both edges of the clock to improve throughput. In fact, with careful organization, the transpose memory itself can achieve a speedup of almost 4X over prior work, while consuming 46% less area. In transpose-heavy algorithms that rely heavily on transpose operations, such as 2D-DCT and 2D-IDCT, we also implement the computation logic to benefit from new data every half cycle. The resulting architecture achieves 3.5X speedup on 2D-DCT and 3X speedup on 2D-IDCT. This new TRM architecture allows a more compact DCT architecture that needs only a single stage of 1D-DCT, by looping data back through the TRM to reuse the computation hardware, maintaining high performance while further reducing area. Both normal and compact implementations of the 2D-DCT by using the proposed transpose memory show a significant improvements compare to the prior works in terms of speed and area.

The future work in this area will be implementing more algorithms, which are benefiting from the low latency the proposed memory showed. Also integrating this memory structure in DSPs for signal processing applications.

Acknowledgements

This work was supported in part by NSF grant no. CDI-1124931 and by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semicon-

Table 1: Performance Comparison

	Proposed Fast	Proposed Compact	ref[18]	ref[11]	Ref[9] arch(1)	Ref[9] arch (2)
Device	Virtex-7 XC7VX485t	Virtex-7 XC7VX485t	XC2VP3	Spartan XC3S500E	Virtex-7 XC7VX330T	Virtex-7 XC7VX330T
Throughput (Gpbs)	15	5.6	8.36	3.44	1.84	1.97
Area	322	256	2823	1145	1354	1110
Throughput/Area (Mbps/Slices)	47.7	22.4	3.03	3.07	1.39	1.82
Maximum Frequency (MHz)	300	100	107	84.81	338.5	256

ductor Research Corporation program sponsored by MARCO and DARPA.

8. REFERENCES

[1] L. Agostini, I. Silva, and S. Bampi, “Pipelined fast 2d dct architecture for jpeg image compression,” in *Integrated Circuits and Systems Design, 2001, 14th Symposium on.*, 2001, pp. 226–231.

[2] N. Ahmed, T. Natarajan, and K. Rao, “Discrete cosine transform,” *Computers, IEEE Transactions on*, vol. C-23, no. 1, pp. 90–93, Jan 1974.

[3] J. Bruguera and R. Osorio, “A unified architecture for h.264 multiple block-size dct with fast and low cost quantization,” in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 407–414.

[4] K. Bukhari, G. Kuzmanov, and S. Vassiliadis, “Dct and idct implementations on different fpga technologies,” in *Proceedings of ProRISC 2002, pp-232-235*, 2002.

[5] M. El-Hadedy, S. Purohit, M. Margala, and S. Knapskog, “Performance and area efficient transpose memory architecture for high throughput adaptive signal processing systems,” in *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, June 2010, pp. 113–120.

[6] M. Elhadedy, A. Madian, H. Saleh, M. Ashour, and M. Aboelsaud, “Hardware implementation of the encoder modified mid-band exchange coefficient technique (mmbec) based on fpga,” in *Microelectronics, 2007. ICM 2007. Internatonal Conference on*, Dec 2007, pp. 43–46.

[7] U. Guide, *7 Series FPGAs Configurable Logic Block*, 1st ed., Xilinx, San Jose, CA.

[8] R. Jain and P. Panda, “Memory architecture exploration for power-efficient 2d-discrete wavelet transform,” in *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, Jan 2007, pp. 813–818.

[9] P. Kitsos, N. Voros, T. Dagiuklas, and A. Skodras, “A high speed fpga implementation of the 2d dct for ultra high definition video coding,” in *Digital Signal Processing (DSP), 2013 18th International Conference on*, July 2013, pp. 1–5.

[10] M. Kovac and N. Ranganathan, “Jaguar: a fully pipelined vlsi architecture for jpeg image compression standard,” *Proceedings of the IEEE*, vol. 83, no. 2, pp. 247–258, Feb 1995.

[11] E. Kusuma and T. Widodo, “Fpga implementation of pipelined 2d-dct and quantization architecture for jpeg image compression,” in *Information Technology (ITSim), 2010 International Symposium in*, vol. 1, June 2010, pp. 1–6.

[12] Y. Li, Y. He, and S. Mei, “A highly parallel joint VLSI architecture for transforms in H.264/AVC,” *Signal Processing Systems*, vol. 50, no. 1, pp. 19–32, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11265-007-0111-4

[13] Y. Ma, “An effective memory addressing scheme for fft processors,” *Signal Processing, IEEE Transactions on*, vol. 47, no. 3, pp. 907–911, Mar 1999.

[14] A. Madisetti and J. Willson, A.N., “A 100 mhz 2-d 8 times;8 dct/idct processor for hdtv applications,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 5, no. 2, pp. 158–165, Apr 1995.

[15] N. Ponomarenko, K. Egiazarian, V. Lukin, and J. Astola, “Additional lossless compression of jpeg images,” in *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, Sept 2005, pp. 117–120.

[16] G. Ruiz and J. Michell, “Memory efficient programmable processor chip for inverse haar transform,” *Signal Processing, IEEE Transactions on*, vol. 46, no. 1, pp. 263–268, Jan 1998.

[17] G. Sullivan and R. Baker, “Efficient quadtree coding of images and video,” *Image Processing, IEEE Transactions on*, vol. 3, no. 3, pp. 327–331, May 1994.

[18] A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, “A pipelined fast 2d-dct accelerator for fpga-based socs,” in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, ser. ISVLSI ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 331–336. [Online]. Available: http://dx.doi.org/10.1109/ISVLSI.2007.13

[19] W. Wang, B. Duan, C. Zhang, P. Zhang, and N. Sun, “Accelerating 2d fft with non-power-of-two problem size on fpga,” in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, Dec 2010, pp. 208–213.

[20] Wikipedia, “Discrete cosine transform,” 2015, [Online; accessed 20 December 2015]. [Online]. Available: http://en.wikipedia.org/wiki/Discrete_cosine_transform

[21] X. Zhang and K. Parhi, “Implementation approaches for the advanced encryption standard algorithm,” *Circuits and Systems Magazine, IEEE*, vol. 2, no. 4, pp. 24–46, Fourth 2002.