2-25-2005

Discrete Wavelet Transform Core for Image Processing Applications

Andreas E. Savakis
Rochester Institute of Technology

Richard Carbone
Rochester Institute of Technology

Recommended Citation
Andreas E. Savakis, Richard Carbone, "Discrete wavelet transform core for image processing applications", Proc. SPIE 5671, Real-Time Imaging IX, (25 February 2005); doi: 10.1117/12.596109; https://doi.org/10.1117/12.596109

Discrete Wavelet Transform Core for Image Processing Applications

Andreas Savakis and Richard Carbone

Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623

ABSTRACT

This paper presents a flexible hardware architecture for performing the Discrete Wavelet Transform (DWT) on a digital image. The proposed architecture uses a variation of the lifting scheme technique and provides advantages that include small memory requirements, fixed-point arithmetic implementation, and a small number of arithmetic computations. The DWT core may be used for image processing operations, such as denoising and image compression. For example, the JPEG2000 still image compression standard uses the Cohen-Daubechies-Favreau (CDF) 5/3 and CDF 9/7 DWT for lossless and lossy image compression respectively. Simple wavelet image denoising techniques resulted in improved images up to 27 dB PSNR. The DWT core is modeled using MATLAB and VHDL. The VHDL model is synthesized to a Xilinx FPGA to demonstrate hardware functionality. The CDF 5/3 and CDF 9/7 versions of the DWT are both modeled and used as comparisons. The execution time for performing both DWTs is nearly identical at approximately 14 clock cycles per image pixel for one level of DWT decomposition. The hardware area generated for the CDF 5/3 is around 15,000 gates using only 5% of the Xilinx FPGA hardware area, at 2.185 MHz max clock speed and 24 mW power consumption.

1. INTRODUCTION

The two-dimensional Discrete Wavelet Transform (DWT) has shown considerable promise in image processing applications, such as the JPEG2000 still image compression standard [1] and image denoising [2]. A hardware DWT core could be integrated into digital camera or scanner to perform image processing inside the device. Research on DWT architectures and hardware implementations includes efficient filter/subsample-based architectures as well as implementations based on the lifting scheme [3]. In [4], a VLSI model is simulated to estimate hardware performance. A scalable architecture for performing many different DWTs using the lifting scheme is proposed in [5]. A comparison of both a convolution-based and a high-throughput pipelined lifting scheme-based DWT architecture is presented in [6]. A VLSI architecture for performing the Cohen-Daubechies-Feauveau (CDF) 9/7 Integer Wavelet Transform is proposed in [7]. The work of [8] presents a power efficient architecture for performing the DWT using the lifting scheme and techniques for improving performance at the cost of increased power consumption and vice-versa. Architectures used to improve 2-D DWT performance using the lifting scheme are described in [9]. A "folded" architecture of the pipelined lifting-based DWT implementation, reusing multipliers to reduce hardware area and improve hardware utilization is proposed in [10]. The work of [11] provides analysis on hardware characteristics of convolution-based and lifting scheme-based DWT each using both regular and shift-add multipliers for the CDF 9/7, while [12] and [13] provide unique scalable architectures using the line-based lifting scheme method to reduce memory requirements. More recently, [14-16] present efficient architectures for lifting-based DWTs.

A flexible hardware architecture for performing the DWT on a digital image is presented in this paper. This architecture is based on a variation of the lifting scheme technique and provides advantages that include smaller memory requirements, fixed-point arithmetic, instead of more costly floating-point, and fewer arithmetic computations. In addition, the architecture is flexible in that it can be configured to perform different variations of the DWT. For example, the JPEG2000 still image compression standard

Real-Time Imaging IX, edited by Nasser Kehtarnavaz, Phillip A. Laplante,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 5671

uses the CDF 5/3 and CDF 9/7 DWT for lossless and lossy image compression respectively [17]. The proposed architecture allows the hardware for either of these two DWTs to be configured automatically.

Following this introduction, Section 2 provides a brief overview of the DWT using the Lifting Scheme. Section 3 explains the design and implementation of the DWT core. The DWT core is modeled using MATLAB and VHDL, and the VHDL model is synthesized to a Xilinx FPGA to prove hardware functionality. The CDF 5/3 and CDF 9/7 versions of the DWT are both modeled and used as comparisons throughout this paper. Section 4 provides performance results, and is followed by conclusions in Section 5.

2. THE LIFTING SCHEME

Lifting was proposed by Sweldens as an alternative approach to computing the DWT [3]. The lifting scheme calculates the DWT using spatial domain analysis, and consists of a series of *Split*, *Predict* and *Update* steps that modify, or lift, one set of samples to be used in the next step, as shown in Figure 1. The split step separates odd and even samples, and the predict step predicts values in the odd set, as follows:

$$\text{odd}_{\text{new}} = \text{odd}_{\text{old}} + \alpha\left(\text{even}_{\text{left}} + \text{even}_{\text{right}}\right)$$

where α as the predict step coefficient. The Update step uses the new wavelet coefficients in the odd set to update the even set producing "smooth" or "scaling" coefficients:

$$\text{even}_{\text{new}} = \text{even}_{\text{old}} + \beta\left(\text{odd}_{\text{left}} + \text{odd}_{\text{right}}\right)$$

where β as the update step coefficient. A more complex DWT involving more neighbors can be performed by merely adding more predict and update steps to the method described above, with the final predict and update steps producing the wavelet and scaling coefficients respectively. If needed, scaling steps can be added to adjust the DC gain of the wavelet and scaling coefficients. To perform the inverse DWT the lifting steps are simply performed in reverse order using the inverse operations (Figure 2). The main advantages of computing the DWT via the lifting scheme are that it is fast, it is calculated fully in-place, has symmetric forward and inverse transforms, and has potential for integer wavelet transform. The CDF 9/7 coefficients are widely used because they provide near optimal performance in image compression applications.

3. DESIGN AND IMPLEMENTATION

The objective of the proposed DWT design is to allow maximum flexibility, while maximizing memory efficiency. Requirements for a flexible design include the ability to utilize any user-defined wavelet. To accomplish this, the DWT core is preconfigured with a set of lifting and scaling steps. In order to configure the DWT core for a particular wavelet, the lifting step extraction must be performed prior to configuration to obtain the necessary lifting and scaling steps. An arbitrary number of lifting steps can be defined, and the number of scaling steps needed is two. The lifting/scaling steps are defined by real-number signed coefficients. Each lifting step can use a rounding function, as required by the particular wavelet.

The inputs and outputs of the DWT core are preconfigured to define various parameters, such as the maximum image width and height allowed, the maximum number of levels of decomposition possible, and the size of memory, data and address bus. Once the DWT core is configured and implemented in hardware, many options are available at run-time. The DWT core can perform both the forward and inverse DWT on an image of any height or width. The DWT core can also perform the DWT on local smaller portions of the image. This is particularly useful for cases where the DWT is applied to one or more tiles of an image, as is the case with JPEG2000 [1]. Finally, the DWT core is portable. Since it is written in standard hardware description language, it can be ported to various types of hardware, such as FPGAs and ASICs.

Memory efficiency is one of the goals of the present design. This was accomplished by performing computations in place, minimizing the amount of image buffering, and reducing the amount of swap memory space. The proposed wavelet core implements the "line-based" DWT in a manner that requires only one pass per row/column. As a result, however, each lifting step requires separate multiplication units. Since the lifting step coefficients do not change at run time, the DWT core uses fixed-point shift-add multiplier hardware that is specifically tuned for each lifting step coefficient instead of using regular floating-point multipliers for real number coefficients. These units automatically generate the necessary hardware to perform multiplication of an integer number by any precision signed floating point number. These units are faster and use less area/power than general purpose floating-point multipliers.

The inverse DWT using the lifting scheme is symmetrical and therefore performed by reversing the lifting steps in the forward DWT. The DWT core takes advantage of this by reusing the ALU components from the forward DWT by rerouting the data path in the reverse direction and changing the additions to subtractions and vice-versa. As a result, the additional hardware overhead in terms of control and routing for the reverse DWT is minimal.

3.1 Design

The components of the DWT core include the Wavelet-2D, Memory Controller, Multi-Lift-Reorder-2D, Lift-Reorder-2D, Multi-Lift-Reorder-1D, Lift-Reorder-1D, Lift-1D, Sliding-Window and Lift-ALU-group (Figure 3), where state machine controller units are associated with most modules. The functionality of the modules is described next in a top down fashion.

The Wavelet-2D module is the actual DWT core, and interconnects the Multi-Lift-Reorder-2D unit to the Memory Controller unit. The Memory Controller module provides a synchronous read/write interface to asynchronous SRAM memory; it translates the 2-D column and row address format to the 1-D memory address bus format using adders and multipliers, and coordinates read and write access that share a single data bus.The Multi-Lift-Reorder-2D module performs the 2-D forward/inverse DWT for N levels of decomposition of an image, using multiple single-level decompositions recursively applied to the LL subband of each level. It employs the related control logic and the Lift-Reorder-2D unit. The Lift-Reorder-2D module performs the 2-D forward/inverse DWT for one level of decomposition of an image using control logic and the Multi-Lift-Reorder-1D unit. The Multi-Lift-Reorder-1D module performs the 1-D forward/inverse DWT on N lines (rows or columns) of an image using control logic and the Lift-Reorder-1D unit. For the forward DWT, the Multi-Lift-Reorder-1D unit first operates horizontally to the rows of an image and then vertically to the columns of the image. For the inverse DWT the reverse operations are performed. The Lift-Reorder-1D module performs the 1-D forward/inverse DWT with reordering on one line of an image. The related machine controller unit is used to control the Lift-1D unit to perform the lifting scheme and reordering on one line. For the forward DWT the lifting scheme is performed first and is followed by reordering. For the inverse DWT it is the opposite. The Lift-1D module performs either the lifting scheme or reordering on one image line of an image. The Sliding Window and Lift-ALU modules are described next.

3.1.1 Sliding Window Method

The standard lifting scheme calculates the scaling and wavelet coefficients by incrementally overwriting the original pixel values with the intermediate lifting step results until the final scaling and wavelet coefficient values are obtained. Multiple passes of each row, one for each lifting and scaling step, are necessary to eliminate the need for intermediate full-line storage buffers. Therefore, the CDF 5/3 DWT requires two passes of each row and the CDF 9/7 requires six passes of each row.

To reduce the memory I/O bandwidth, the "Sliding Window Method" algorithm is presented, so that regardless of the number of lifting steps there is only a single pass of each row of pixels. The concept behind this method is to buffer only the necessary pixels required to calculate the final scaling and

wavelet coefficients. Based on the analysis of the lifting pixel dependency graph, it was determined that the lifting scheme can be implemented with the use of minimal pixel buffers. As a result, the lifting scheme requiring only a single pass of each row can be implemented. Rather than storing the results from the intermediate lifting steps into separate buffers, only the intermediate values are buffered and used to calculate the final scaling and wavelet coefficients. These coefficients are then used to overwrite each of the original pixels, so that in place computation is accomplished.

It was determined that for a wavelet with N lifting steps an N+2 size pixel buffers are required regardless of whether or not scaling steps are present. Therefore, the CDF 5/3 with two lifting steps requires a four-pixel buffer and the CDF 9/7 with four lifting steps requires a six-pixel buffer.

Figure 4 illustrates the CDF 5/3 forward DWT on a row of eight pixels using the sliding window method. First the sliding window buffer reads in two pixels from memory into a four-pixel storage buffer as shown in Figures 4 (a, b, and c). After the third pixel is read into the buffer, the first set of scaling and wavelet coefficients is calculated in Figure 4d, and is stored into the buffer shown in Figure 4e. Another two pixels are read into the buffer filling the buffer in Figure 4f and g. As the pixels are read from memory and shifted into the head of the filled buffer, the scaling and wavelet coefficients are shifted out from the tail of the buffer are written back to memory. This process is repeated until the entire row has been processed (Figures 4h through q). The buffer has written all of its pixels back into memory and is now ready to process another row.

The Sliding-Window module includes a pixel storage buffer and arithmetic logic that is primarily used to perform the lifting scheme and reordering. The CDF 5/3 Sliding-Window unit consists of a four-pixel buffer along with two Lift-ALU units. The CDF 9/7 Sliding-Window unit consists of a six-pixel buffer along with four Lift-ALU units and two Scale-ALU units. The Lift-ALU-Group unit contains all the individual lift and scale units required to perform the lifting scheme. The lift and scale units are connected together to perform either the forward or inverse DWT. The Lift-ALU-Group generates the necessary hardware based on an array of Lift-ALU and Scale-ALU units defined in a configuration file.

The Lift-ALU unit is an ALU that performs a lifting step operation on a pixel (C) given its neighboring left (A) and neighboring right (B) pixels, lifting step coefficient (K) and, if needed, an extra constant (L). The lifting step coefficient (K) can be any positive/negative real number. A floor, ceiling, round, or truncate function is applied to the result of the multiplication to keep an integer value for the integer wavelet transform. The format for the lifting step is shown below:

$$C_{new} = C + \lceil K(A + B + L) \rceil$$

The lifting step equations for the CDF 5/3 and CDF 9/7 biorthogonal wavelets have L values of 0 and use the floor function as shown below:

$$C_{new} = C + \lfloor K(A + B) \rfloor$$

The lift unit has to ability to perform symmetric extension near the image border. For example if pixel C is on the leftmost border of the image there is only the neighboring right pixel (B) and no neighboring left pixel (A). Therefore using the symmetric extension method pixel B is used in place of pixel A and the lifting equation becomes the following:

$$C_{new} = C + \lfloor K(B + B) \rfloor = C + \lfloor K(2B) \rfloor$$

The Scale-ALU unit is an ALU that performs a scaling step operation on a pixel (A) given a scaling step coefficient (K). The scaling step coefficient (K) can be any positive/negative real number. A floor, ceiling, round, or truncate function is applied to the result of the multiplication to keep an integer value for the integer wavelet transform. The format for the scaling step is shown below.

$$A_{new} = \lceil K(A) \rceil$$

The FXP-Multiply unit is a multiplier that performs fixed-point multiplication of an N-bit signed/unsigned integer number by a constant positive/negative real number. For synthesis purposes the generic value representing the real number coefficient is an array of integers indicating the location of each '1' in the binary number. The position of the least significant '1' is indicated as 1, the position of

the bit to its left is 2, etc. Another integer generic is used to indicate the location of the decimal point, 1 meaning there is one decimal place. For example the number 1.6875 or binary 1.1011 is formatted as Coefficient Array = {1,2,4,5} and Decimal Location = 4.

3.2 Implementation

The DWT core was modeled in various stages before the final implementation on an FPGA. A high-level MATLAB description, that has analogous inputs and output to the hardware DWT core, was used to verify the core functionality. A VHDL description using VHDL'93 was used to provide a low-level hardware description of the DWT core. The VHDL DWT core model serves as functional and timing verification of the DWT core. This model exactly mimics the operation of the actual hardware DWT core providing not only functional verification, but also timing information/verification. The second purpose of this model is to serve as synthesizable code for the target hardware platform providing all the necessary inputs and outputs.

The VHDL code is written at a low RTL level using data flow and structural descriptions with the exception of simple behavioral logic used for state machines. This allows the DWT to be synthesized with little effort and, if desired, aids in full-custom ASIC design using schematic capture. Arithmetic units are all custom-designed and do not have to be inferred at synthesis time. The DWT core makes extensive use of VHDL's built-in hardware generation capabilities such as generate and generic statements to tailor the hardware to the user-defined wavelet.

The XSV-300 FPGA prototyping board from Xess Corporation was used for testing the hardware DWT core. The XSV-300 contains a 240-pin Xilinx XCV300 Virtex FPGA that is a 300,000 gate-equivalent FPGA using lookup tables. The XSV-300 contains two independent banks of 512k x 16 SRAM for a total of 2 MB to be interfaced directly with the FPGA. The VHDL DWT core is synthesized to a Xilinx FPGA using the Xilinx ISE 4.2 software.

4. RESULTS

The MATLAB implementation of the DWT core was verified against the standard convolution-based DWT to demonstrate correct functionality of the DWT core. The traditional convolution-based CDF 9/7 DWT was applied to the Lena 512x512 pixel grayscale image for 1 to 7 levels of decomposition. Simple wavelet image denoising techniques resulted in improved images up to 27 dB PSNR. Next the MATLAB model of the DWT core using both the actual full precision lifting coefficients and the approximated lifting coefficients (within 0.0001) for the CDF 9/7 wavelet are applied to the Lena image for 1 to 9 levels of decomposition. The design target for maximum error was to keep the difference between the actual and approximated coefficients all below 0.0001. This tolerance was chosen based on desired hardware vs. precision constraints and provided satisfactory results. The PSNR difference measurements, which even after 9 levels of decomposition are well above 58dB.

A 75x84 pixel image called "rapper" is used to verify the VHDL model. Both the CDF 5/3 and CDF 9/7 forward and inverse DWTs were applied for 1 to 6 levels of decomposition using a clock period of 10 ns. The results illustrate that an average of approximately 14.5 clock cycles per pixel are required to perform the DWT. Eight clock cycles are required for the necessary memory read and writes while the remaining six clock cycles are used for control and arithmetic functions.

Reordering of the interleaved wavelet and scaling coefficients is the same for each implementation. Each row of pixels would be read and then written back to with the reordered configuration. This is repeated for all rows of the image and then all the columns of the image. An NxM pixel image requires 2NM reads and 2NM writes for a total of 4NM additional memory reads and writes for reordering. All convolution-based DWTs have the same memory bandwidth regardless of size of the wavelet. The two high pass and low pass filters/convolution kernels each read all the pixels in a row and write the results to a buffer. This is repeated for all rows of the image and then all the columns of the

image. Therefore, an NxM pixel image requires 4NM reads and 4NM writes for a total of 8NM memory reads and writes for either the convolution-based CDF 5/3 or CDF 9/7 DWT.

The memory bandwidth requirement using the lifting scheme varies depending on the number of lifting and scaling steps. Every row pixel is read once for each lifting stage, however, only half of the pixels are written back to memory each lifting stage. For each scaling stage, only half of the pixels in a row are read from and written to memory. This is repeated for all image rows and then all image columns. The lifting scheme based CDF 5/3 DWT involves only two lifting steps and no scaling steps. Therefore, an NxM pixel image requires 4NM reads and 2NM writes for a total of 6NM memory reads and writes. The lifting scheme based CDF 9/7 DWT has four lifting steps and two scaling steps requiring a much higher memory bandwidth. An NxM pixel image requires 12NM reads and 4NM writes for a total of 16NM memory reads and writes.

The memory bandwidth requirements for the DWT core presented in this paper is lower than both the convolution-based and lifting scheme implementations. Figure 5 illustrates the memory bandwidth requirements in terms of memory reads and writes for performing the forward DWT on an image for 1 to 9 levels of decomposition. The DWT core reads and writes each pixel only once, regardless of the wavelet used. The lifting scheme based CDF 5/3 there only has two lifting steps and no scaling steps, therefore, 2NM reads and 2NM writes are required for a total of 4NM memory reads and writes. The DWT core was synthesized using CDF 5/3 and CDF 9/7 wavelets, for maximum speed and minimum hardware area, using Ripple-Carry and Carry-Look-Ahead (CLA) architectures.

When synthesized for maximum speed the amount of combinational logic generated for the CDF 9/7 is more than double that for the CDF 5/3 DWT. The synthesized sequential logic for the CDF 9/7 is slightly more than that of the CDF 5/3 due to the similar control logic that is generated.

Figure 6 shows the maximum hardware speed determined during synthesis. In all cases, the speed of the CDF 5/3 DWT is more than double that of the CDF 9/7. This is most likely due to the long delay paths through the FXP-Multiply units generated for the real-number CDF 9/7 coefficients. Using the timing results, an image of size 512x512 requires 512x512x14.5 = 3,801,088 clock cycles for a 1-level DWT. Running at the maximum clock speed of 7.326 MHz this would yield one transformed 512x512 image every 0.519 seconds, or approximately two images per second. Improving a hardware performance to 100 MHz via more aggressive synthesis constraints and/or custom circuit design would improve performance to approximately 26 transformed images per second.

The information for the synthesized CDF 5/3 DWT core is shown in Table 1. The CLA version is the fastest running up to 2.185 MHz. The ripple-carry version is the smallest with only a 15,918 equivalent gate count using only about 5% of the available hardware resources of the Xilinx FPGA.

5. CONCLUSIONS

A flexible hardware architecture for performing the DWT on a digital image was presented in this paper. This architecture uses a variation of the lifting scheme technique that provides advantages, such as smaller memory requirements, fixed-point arithmetic instead of more costly floating-point, and less arithmetic computations. In addition the architecture is flexible in that it can be configured to perform many different variations of the DWT.

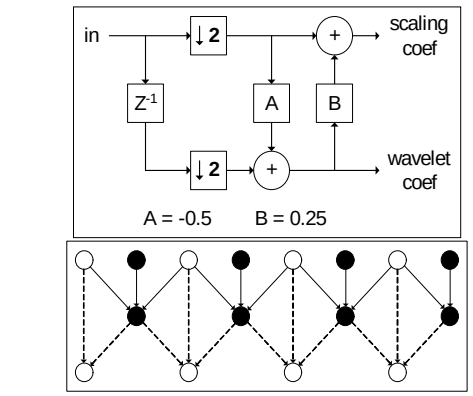
The CDF 5/3 hardware produces identical results to its theoretical MATLAB model. The fixed-point CDF 9/7 deviates very slightly from its floating-point MATLAB model with a ~59dB PSNR deviation for nine levels of DWT decomposition. The execution time for performing both DWTs is nearly identical at ~14 clock cycles per image pixel for one level of DWT decomposition. The hardware area generated for the CDF 5/3 is ~16,000 gates using only 5% of the Xilinx FPGA hardware area, and 2.185 MHz maximum clock speed.

ACKNOWLEDGMENTS

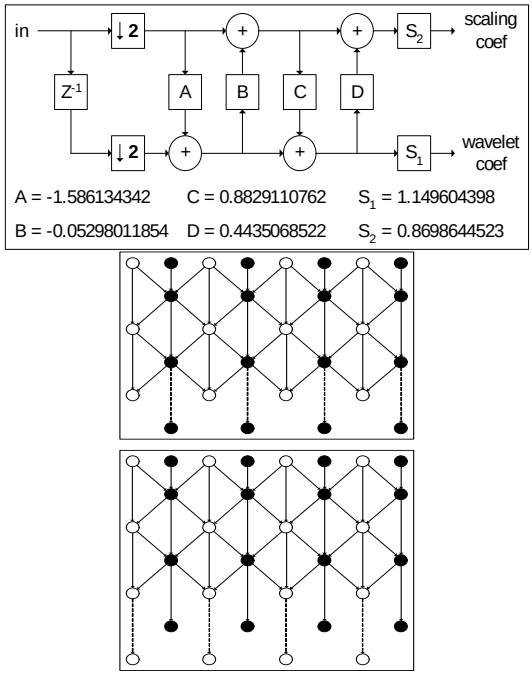
The authors would like to thank Dr. Marcin Lukowiak for his constructive feedback and assistance during the course of this research.

REFERENCES

[1] D. Taubman and M. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and*
[2] S. G. Chang, Bin Yu, M. Vetterli, 'Spatially adaptive wavelet thresholding with context modeling for image denoising," *IEEE Trans. Image Processing*, Vol. 9, pp.1522-1531, Sept. 2000 .
[3] W. Sweldens. "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions." *Proc. SPIE*, vol. 2569, pp. 68-79, 1995.
[4] A. Grzeszczak, M. K. Mandal, S. Panchanathan, and T. Yeap. "VLSI Implementation of Discrete Wavelet Transform," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 421-433, Dec. 1996.
[5] K. Andra, C. Chakrabarti, and T. Acharya. "A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform," *IEEE Transactions on Signal Processing*, vol. 50, no. 4. April 2002.
[6] J. M. Jou, Y.-H. Shiau, and C.-C. Liu, "Efficient VLSI Architectures for the Biorthogonal Wavelet Transform by Filter Bank and Lifting Scheme," *IEEE Int. Symp. Circuits and Systems*, pp. 529-533, Sydney, Australia, May 2001.
[7] M. Martina, G. Masera, G. Piccinini, and M. Zamboni. "A VLSI Architectures for IWT (Integer Wavelet Transform)," *Proc. 43rd IEEE Midwest Symp. Circuits Systems*, Lansing, MI, Aug. 2000.
[8] M. Grangetto, M. Martina, G. Masera, G. Piccinini, F. Vacca, and M. Zamboni. "FPGA Power Efficient Inverse Lifting Wavelet IP," *Proc. 35th IEEE Int. Asilomar Conf. Signals, Systems Computers*, Pacif Groove, CA, Nov., 2001.
[9] P. McCanny, S. Masud, and J. McCanny. "An Efficient Architecture for the 2-D Biorthogonal Discrete Wavelet Transform," *Proc. ICIP'01*, Thessaloniki. Greece, 2001.
[10] C. Lian, K. Chen, H. Chen, and L. Chen, "Lifting Based Discrete Wavelet Transform Architecture for JPEG2000," *IEEE Int. Symp. Circuits and Systems*, vol. 2, pp. 445-448, May 2001.
[11] V. Spliliotopoulos, N. Zervas, Y. Andreopoulos, G. Anagnostopoulos, and C. Goutis. "Quantization Effect on VLSI Implementations for the 9/7 DWT Filters," *Proc. ICASSP'01*, Salt Lake City, UT, vol. 2, pp. 1197-1200, May 2001.
[12] W. Chang, Y. Lee, W. Peng, and C. Lee. "A Line-Based, Memory Efficient and Programmable Architecture for 2D DWT using Lifting Scheme," *IEEE Int. Symp. Circuits and Systems*, 2001.
[13] C. Huang, P. Tseng, and L. Chen. "Efficient VLSI Architectures of Lifting-Based Discrete Wavelet Transform by Systematic Design Method," *IEEE Int. Symp. Circuits and Systems* 2002.
[14] H. Liao, M. K. Mandal, and B.F. Cockburn, "Efficient architectures for 1-D and 2-D lifting-based wavelet transforms" IEEE Trans. Signal Processing, Vol. 52, pp. 1315-1326, May 2004.
[15] Pei-Yin Chen, "VLSI implementation for one-dimensional multilevel lifting-based wavelet transform," IEEE Trans. Computers, Vol. 53, pp.386-398, April 2004.
[16] C.T Huang, P.C. Tseng; L.G. Chen, "Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform, IEEE Trans. Signal Proc., Vol. 52, pp. 1080-1089, April 2004.
[17] A. Skodras, C. Christopoulos, and T. Ebrahimi. "The JPEG 2000 Still Image Compression Standard," *IEEE Signal Processing Mag.*, vol. 18, pp. 36-58, Sept. 2001.



(a)



(b)

Figure 1. Split, Predict, and Update Steps of Forward DWT using the Lifting Scheme (a) using CDF 5/3 wavelet and (b) using CDF 9/7 wavelet.

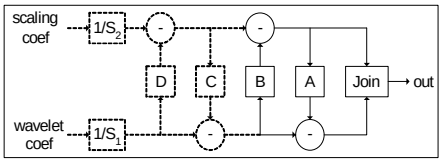


Figure 2. Inverse DWT using the Lifting Scheme with Multiple Lifting and Scaling Steps.

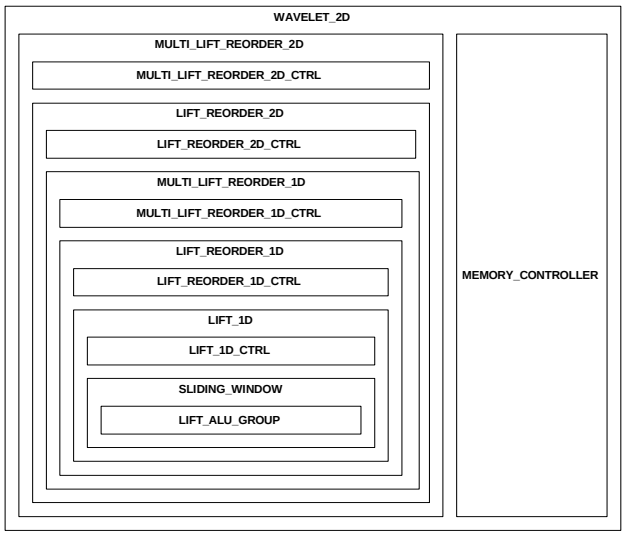


Figure 3: Discrete Wavelet Transform Core Component Diagram

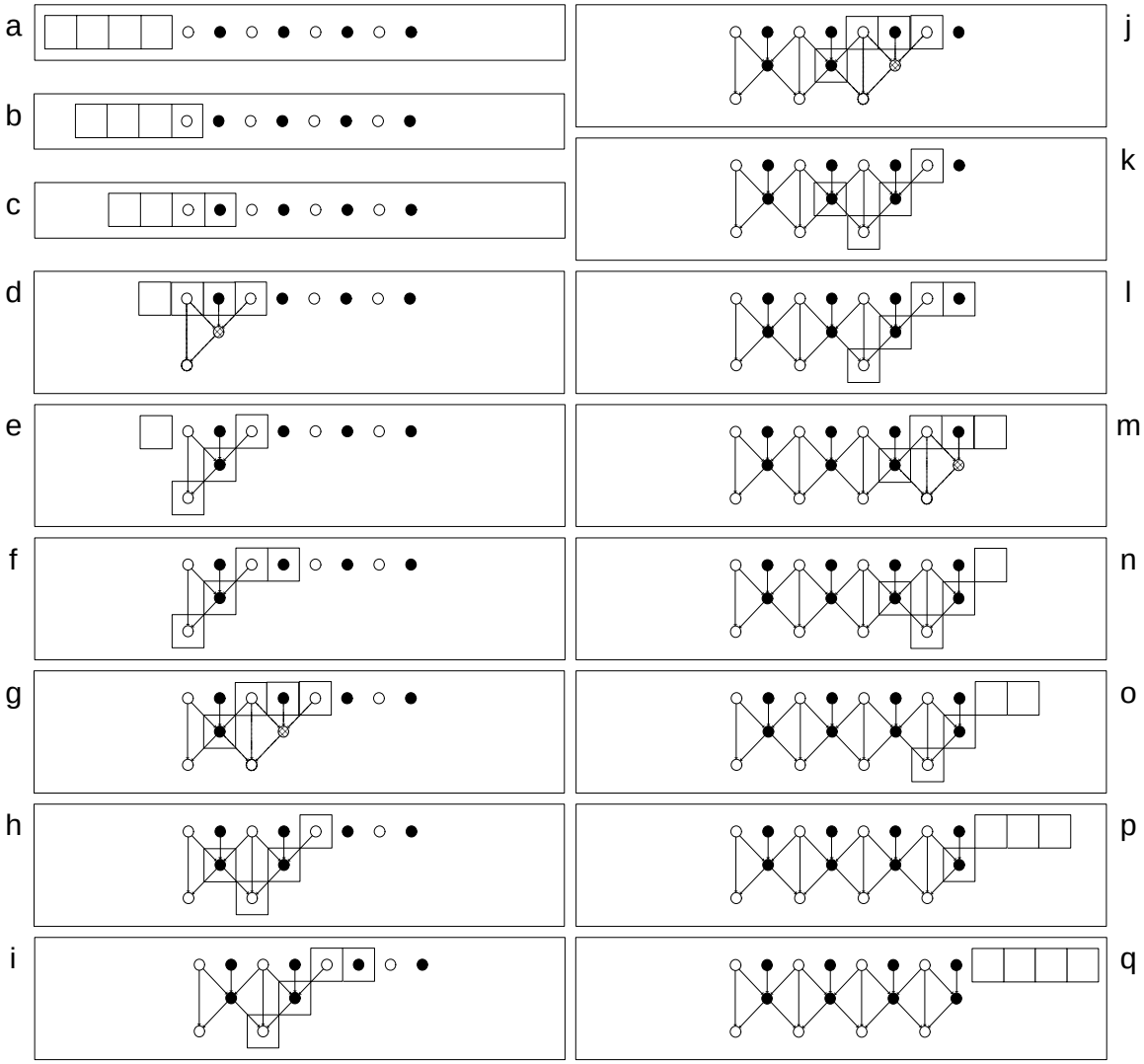


Figure 4: Forward CDF 5/3 DWT of Row of Pixels using Sliding Window Method

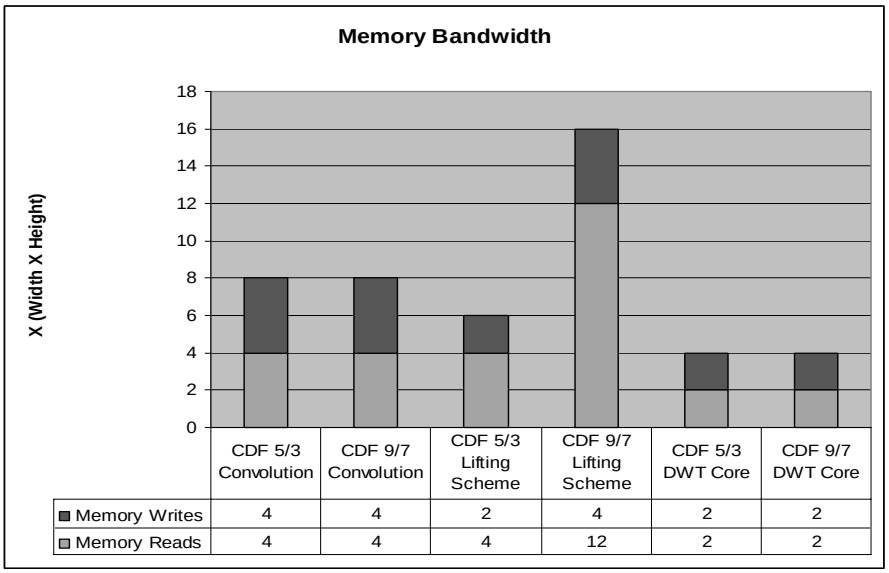


Figure 5: Normalized Memory Bandwidth (Read/Write) for CDF 5/3 and CDF 9/7 DWT using Convolution, Lifting Scheme, and DWT Core

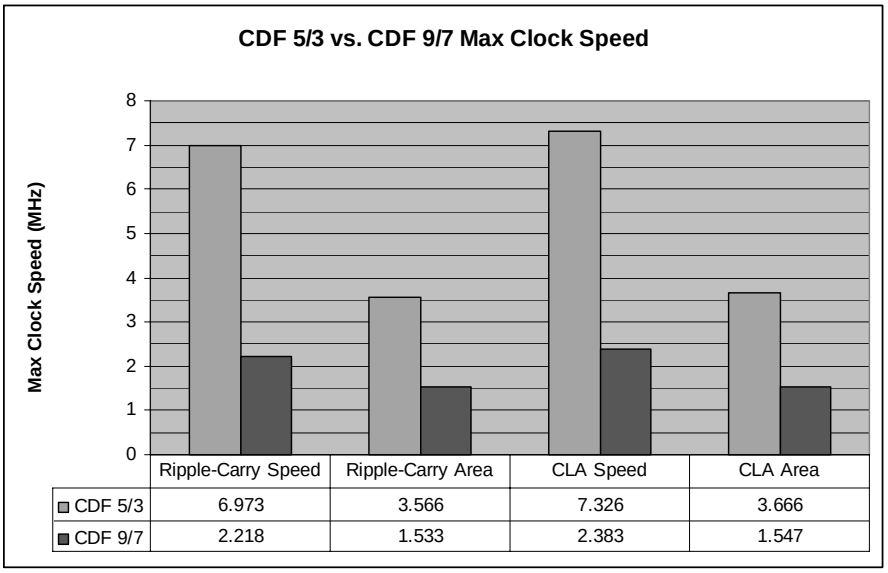


Figure 6: Maximum Clock Speed for CDF 5/3 and CDF 9/7 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders

Architecture	Optimization	Max Clock Speed (MHz)	Min Clock Period (ns)	Equivalent Gate Count
Ripple-Carry	Speed	2.185	457.711	16,410
	Area	1.561	640.595	15,918
CLA	Speed	2.185	457.704	17,628
	Area	1.676	596.674	16,656

Table 1: Maximum Clock Speed and Equivalent Gate Count for Hardware CDF 5/3 DWT Synthesized for Maximum Speed/Minimum Area using Ripple-Carry/CLA adders