Design and Implementation of An RNS-Based 2-D DWT Processor

Yong Liu and Edmund M-K. Lai, *Senior Member, IEEE*

Abstract — *Discrete wavelet transform has been incorporated as part of the JPEG2000 image compression standard and is used in many consumer imaging products. This paper presents a 2-dimensional biorthogonal DWT processor design based on the residue number system. The symmetric extension scheme is employed to reduce distortion at image boundaries. Hardware complexity reduction and utilization improvement are achieved by hardware sharing. Our implementation results show that the design is able to fit into a 1,000,000-gate FPGA device and is able to complete a first level 2-D DWT decomposition of a 32×32-pixel image in 205 μs* [1].

Index Terms — Discrete Wavelet Transform, Residue Number System, Field Programmable Gate Array.

I. Introduction

Wavelets are mathematical functions that enable one to analyze a signal with resolutions matched to its scale. Compared with the traditional Fourier analysis methods, wavelet transforms have the advantage that signals which are transient, or with discontinuities and sharp spikes can be properly analyzed. The developments on wavelets and wavelet transforms in recent years have led to numerous applications in areas such as computer vision, image compression, denoising noisy data, and sound synthesis. Recently the image compression standard JPEG2000 has incorporated discrete wavelet transform (DWT) into its core specifications [1]. As a result of standardization, DWT is now being used in various consumer imaging products.

As DWT requires intensive computation, for some real-time applications it is necessary to implement the wavelet transform algorithms in hardware so that timing constraints are met. DWT can be realized in hardware using FIR filters [4] which basically only involves additions and multiplications. How these arithmetic operations are implemented in hardware depends on the number system used to represent the data.

Residue number systems (RNS) are suitable for implementations of high-speed digital signal processing devices because of their inherited parallelism, modularity, fault tolerance and localized carry propagation [10]. Arithmetic operations, such as addition and multiplication, can be carried out more efficiently in RNS than in conventional two's complement systems. Hence that makes RNS a good candidate for implementing DWT. Some RNS-based DWT implementations have been reported in the literature [5][6][7]. However, these implementations concentrate only on 1-D DWT analysis or synthesis using orthogonal filter banks. A complete design of RNS-based 2-D DWT has, to the best of our knowledge, not been previously reported.

In this paper, we present our design and implementation of an RNS-based biorthogonal 2-D DWT processor. The requirements for filter coefficient word length are studied. The RNS-based FIR filter bank data path, controllers and memory interface are designed and simulated using a Field Programmable Gate Array (FPGA) development board.

The rest of the paper is organized as follows. First, a brief review of wavelet transforms and residue number systems is provided in Section II. Then, some design considerations are discussed in Section III. In Section IV, the design of the RNS-based filter bank data path, various controllers and memory interface are described. The performance results are then presented in Section V. Lastly, conclusions are given in Section VI.

II. Preliminaries

This section provides brief technical backgrounds on residue number systems, discrete wavelet transform and symmetric extension that are necessary to understand the rest of the paper.

A. Residue Number Systems

A residue number system is defined by a moduli set, which consists of n pairwise relatively prime integers $\{m_0, m_1, \ldots, m_{n-1}\}$. The useful computational range M of such a number system is the product of all moduli in the moduli set:

$$M = \prod_{i=0}^{n-1} m_i \qquad (1)$$

Such a residue number system is able to uniquely represent unsigned numbers in the range $[0, M-1]$ and signed numbers in the range $\left[-\frac{M-1}{2}, \frac{M+1}{2}-1\right]$ for odd values of M, or

[1] The authors are with the Centre for Multimedia and Network Technology, School of Computer Engineering, Nanyang Technological University, Singapore 639798. (e-mail: asmklai@ntu.edu.sg)

$\left[-\frac{M}{2},\frac{M}{2}-1\right]$ for even values of *M*. These are called the dynamic ranges of the system.

A number *X* within the dynamic range can be represented by the list of its residues x_i with respect to the moduli m_i defined in the moduli set. The RNS representation of *X* is denoted

$$X=\{x_0,x_1,\ldots,x_{n-1}\} \quad (2)$$

where for $i=0,1,\ldots n-1$,

$$x_i=\begin{cases} X \bmod m_i, & X\ge 0 \\ (M-X)\bmod m_i, & X<0 \end{cases} \quad (3)$$

In RNS, addition, subtraction and multiplication can be performed entirely on the residue representation of the operands. Let the RNS representations of *X* and *Y* be given by $\{x_0,x_1,\ldots,x_{n-1}\}$ and $\{y_0,y_1,\ldots,y_{n-1}\}$ respectively. Then

$$|X\circ Y|_M=\{|x_0\circ y_0|_{m_0},|x_1\circ y_1|_{m_1},\ldots.,|x_{n-1}\circ y_{n-1}|_{m_{n-1}}\} \quad (4)$$

where the operation $\circ$ can be either addition, subtraction or multiplication.

Conversion from the 2's complement representation to RNS representation is referred to as the *forward* conversion. The residue of a number *X* with respect to modulus m_i is given by

$$|X|_{m_i}=\left|\sum_{i=0}^{n-1} b_i|2^i|_{m_i}\right|_{m_i} \quad (5)$$

where b_i is either 0 or 1.

For reverse (RNS to binary) conversions, the Chinese Remainder Theorem (CRT) can be applied. The CRT states that binary/decimal representation of a number can be obtained from its RNS representation through (6), provided all elements of the moduli set are pairwise relatively prime.

$$\begin{aligned}|X|_M&=(x_{n-1}\,|\,x_{n-2}\,|\ldots|\,x_1\,|\,x_0)_{RNS}\\ &=\left|\sum_{i=0}^{n-1}\hat{M}_i|\alpha_i x_i|_{m_i}\right|_M\end{aligned} \quad (6)$$

where $\hat{M}_i=M/m_i$ and $\alpha_i=\left|\hat{M}_i^{-1}\right|_{m_i}$. α_i is called the multiplicative inverse of $\hat{M}_i$ with respect to m_i.

B. Discrete Wavelet Transform

In DWT, an arbitrary square integrable function is represented as a superposition of a family of *Wavelet* basis functions that are generated by the translation and dilation of the mother wavelet of the family. As in the pyramid algorithm proposed by Mallat [11], the DWT coefficients at an arbitrary level *j* can be computed from the DWT coefficients of its previous level *j+1*, which is expressed as follows:

$$c_j(k)=\sum_m h_0(m-2k)c_{j+1}(m) \quad (7)$$

$$d_j(k)=\sum_m h_1(m-2k)c_{j+1}(m) \quad (8)$$

where c_j and d_j are the scaling coefficient and the wavelet coefficient of level *j* respectively, whereas $h_0(n)$ and $h_1(n)$ are the dilation coefficients corresponding to the scaling and wavelet functions respectively.

The dilation coefficients $h_0(n)$ and $h_1(n)$ are also the coefficients of a low pass and a high pass filter respectively. As a result, the scaling and wavelet coefficients at level *j* are obtained by filtering the scaling coefficients at level *j*+1 using an analysis quadrature mirror filter bank (QMF).

Scaling coefficients at level *j+1* is obtained by combining the scaling and wavelet coefficients at level *j*:

$$c_{j+1}(k)=\sum_m c_j(m)h_0'(k-2m)+\sum_m d_j(m)h_1'(k-2m) \quad (9)$$

Each level of the 2-D DWT operation requires two stages of 1-D DWT operations as shown in Fig. 1. First, 1-D DWT is performed on the row data, producing high pass (H) and low pass (L) outputs. A second stage 1-D DWT is executed on the columns of the H and L outputs of the first stage to obtain four sub-images (HH, HL, LH, LL). Further decomposition can be made on the LL sub-image in a similar way. In this way, an image is decomposed into a set of sub-images with various resolutions corresponding to the different scales.

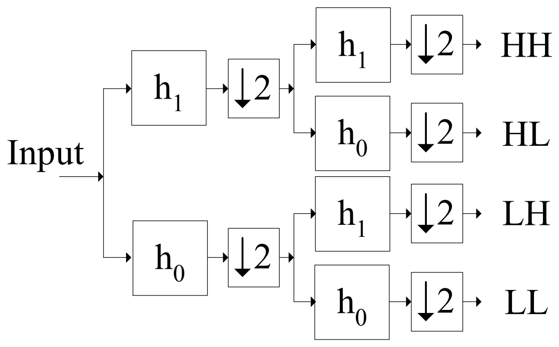

Fig. 1. Filter Bank Structure of 2-D DWT

C. Symmetric Extension

Since filter outputs at the boundary are not well defined, digital filtering of finite length data sequences introduces boundary distortions. Therefore, data sequences have to be extended beyond the boundary to eliminate boundary distortion. However, zero padding and periodic extension, which are traditionally used to extend data sequences, introduce undesired effects like expansiveness and artificial discontinuity. Thus, symmetric extension is proposed to solve these problems. When symmetric extension is applied to a 1-

dimensional data sequence of length N_0, both ends of the sequence are symmetrically extended by reflecting the first or last few samples of the sequence. Depending on the type of filter used, one of the four types of symmetric extension is applied to the input data sequence. In our design, because the filters used are of odd-length symmetric, (1,1)-symmetric extension is applied [2].

After filtering, only a portion of output sequence, of length N_0, will be kept to avoid expansiveness. For (1,1)-symmetric extension, data samples between I_L and I_R are kept. I_L and I_R are determined by the equations below.

$$\begin{aligned} I_L &= \left\lfloor \frac{L_e + L_f}{2} \right\rfloor \\ I_R &= I_L + \frac{S_{r/c}}{2} - 1 \end{aligned} \qquad (10)$$

where L_e is the length of symmetric extension, L_f is the length of the filter, and $S_{r/c}$ is the size of row or column of the image matrix before the filtering is taken place

III. Design Considerations

A. Wavelet Filter Coefficients

In wavelet based image processing, the choice of wavelet filters affects both the reconstructed image quality and system design. Among all available wavelet filters, the Daubechies 9/7-tap filter has been reported to give good overall performance on regularity, impulse response, step response as well as the PSNR values of the reconstructed images [13]. Also, it is one of the supported filter banks of the JPEG2000 standard. Thus, the Daubechies 9/7-tap filter is chosen to implement the filtering unit of the DWT processor. The filter coefficients are listed in Table 1.

TABLE 1
Filter Coefficients of the Daubechies 9/7 Filter Bank

Low Pass Filter		High Pass Filter	
$h(0)$	0.852699	$g(0)$	0.788486
$h(\pm 1)$	0.377402	$g(\pm 1)$	0.418092
$h(\pm 2)$	-0.110624	$g(\pm 2)$	-0.040689
$h(\pm 3)$	-0.023849	$g(\pm 3)$	-0.064539
$h(\pm 4)$	0.37828		

B. Dynamic Range Determination and Moduli Set Selection

The wordlength required for RNS filtering computations is determined by experimentation. With the assumption that input image data are represented in 8 bits, experimental results show that the use of 24-bit dynamic range is able to obtain reconstructed images with PSNR values higher than 54dB. Therefore, we consider that a dynamic range of 24 bits as adequate.

Many moduli sets with 24-bit dynamic range are available. Our estimation on system delay shows that a system implemented using moduli set {255, 256, 257} will likely results in shortest delay [15]. This moduli set is of the form $\{2^n-1, 2^n, 2^n+1\}$, which has the advantage of low cost forward conversion and modulo reduction. The reverse conversion architecture is also relatively simple. Thus the use of this moduli set can significantly reduce hardware complexity and delay [12].

C. Development Environment

An FPGA development board is used to implement the design. A block diagram of the board is shown in Fig. 2. The development board has a 1,000,000-gate FPGA chip, 32-MB of DDR SDRAM and other supporting modules like input/output ports, clock generator and LEDs. The 2-D DWT processor design is mapped onto the FPGA chip and utilizes the DDR SDRAM as its external storage.

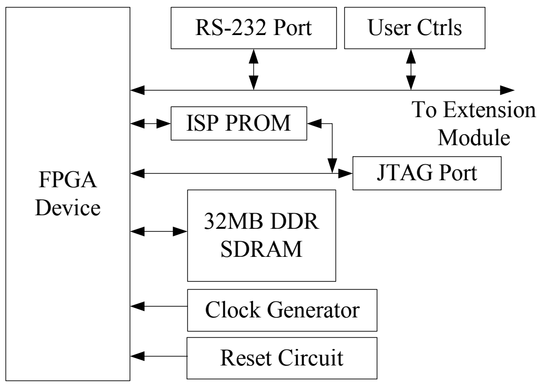

Fig. 2. FPGA Development Board Block Diagram [16]

IV. The 2-D DWT Processor

The 2-D DWT processor consists of the following key components: RNS-based FIR filter bank data path, buffering and DDR interfacing unit, SDRAM controller, main controller, clock management circuitry and the host interface. A block diagram of its structure is shown in Fig. 3.

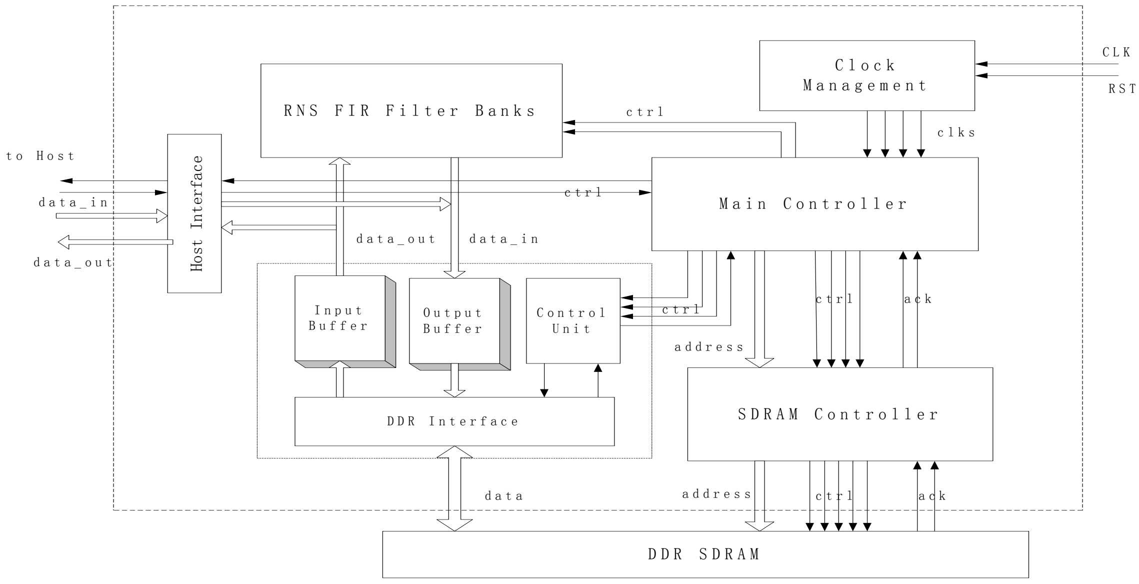

Fig. 3. Organization of the 2-D DWT Processor

A. *The RNS-based FIR Filter Bank Unit*

This unit consists of two sets of RNS-based FIR filters. Each of them is made up of three forward converters and three sub-filter banks corresponding to the three moduli in our chosen moduli set, as well as a reverse converter.

1) *Forward and Reverse Converters*

Forward and reverse conversions are necessary for any RNS-based system to convert its inputs from 2's complement format to RNS representation and its outputs from RNS back to 2's complement respectively. Data conversions result in an increase of hardware cost and delay of the system. Hence, they are considered as overheads to an RNS system and need to be minimized. Relatively simple converters are available for the moduli set $\{255, 256, 257\}$.

Forward converter architecture, which can handle signed numbers, for the modulo-255 and modulo-257 channels is a modified version of the ones discussed in [14] and [3] respectively. For the modulo-255 channel, input bits except the signed bit are first divided into 8-bit segments. A 2-entry look-up table is used to handle the sign bit of the input. If the sign bit is zero, which means that input is positive, the output of the look-up table (LUT) will be zero. Otherwise, the output will be the residue of the most negative value of the input with respect to 255. The LUT output and other segments are added together using adders with carry-end-around to yield the residue of X with respect to modulus 255. Forward conversion for the modulo-257 channel is handled in a similar way.

Note that forward conversion for the modulo-256 channel is achieved simply by keeping the least significant 8 bits of the 2's complement data.

The reverse converter structure that is based on a new formulation of the Chinese Remainder Theorem and uses only adders was proposed in [12]. However, it is limited to converting numbers in RNS back to unsigned numbers. In order to cope with signed numbers, we added a comparison and subtraction circuit to the original design.

2) *RNS-based FIR Sub-filters*

The FIR sub-filters in this design are realized in transposed form. It involves modulo multipliers and adders. Since the filter coefficients are known a priori in our design, it is possible to implement the modular multiplier as look-up tables (LUT), which can reduce both the delay and hardware cost.

In our design, 27 look-up tables (LUT), each with 8-bit width and 256 entries, are required to store all the results of modulo multiplications. The modulo-257 operation generally produces a 9-bit result. However, very few results have a non-zero 9th bit (having the output value of 256). Hence storage for the 9th bit is saved by substituting in some combination logic to generate the 9th bit of the output for the modulo-257 channel.

3) *Pipelining of the RNS-based Filter Banks and LUT Sharing*

The delay of each RNS sub-filter is less than that of a conventional 2's complement filter. However, data conversions introduce significant added delay to the RNS filter. A four-stage pipeline, depicted in Fig.4, is therefore introduced to improve the throughput. The first stage consists of the forward converters, the second stage the LUT-based multipliers, the third stage the modular adders and the last stage the reverse converter.

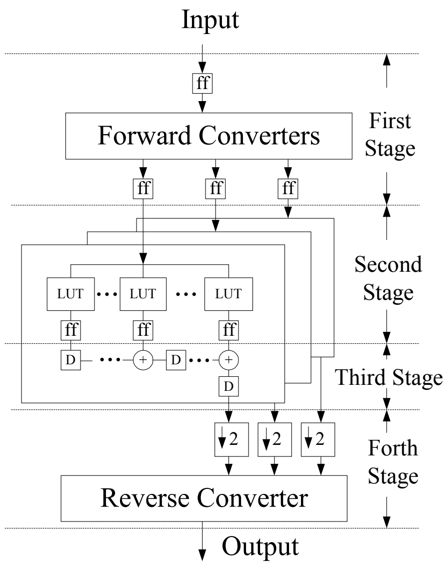

Fig. 4. Pipelining of the RNS-based FIR Filter Bank

The delay for each stage along the pipeline is different. For example, the delay for accessing a look-up table is much shorter than that for reverse conversion. Since the clocking frequency of the pipeline is limited by the slowest stage, hardware complexity can be reduced by sharing the same set of look-up tables between the two filter banks, as illustrated in Fig. 5. In each cycle, the look-up tables are first accessed by inputs to filter bank 0 and the outputs of look-up tables are kept in a set of registers. Then, the look-up tables are accessed by inputs to filter bank 1. In this way, significant amount of hardware is reduced without incurring any extra delay.

4) Reverse Converter Sharing Between HPF and LPF

The output of each filter is down-sampled by two to reduce the data rate after filtering. This down-sampling operation is performed before the reverse conversion as shown in Fig. 5. At the cycle where the output needs to be kept, RNS outputs from the low pass filter are fed into the reverse converter, while those from the high pass filter are stored in registers. At the cycle where the output needs to be discarded, reverse conversion is performed on the output from the high pass sub-filter of the previous cycle, which is stored in a register. Hence, the reverse converters are utilized fully and only one reverse converter is required for each filter bank.

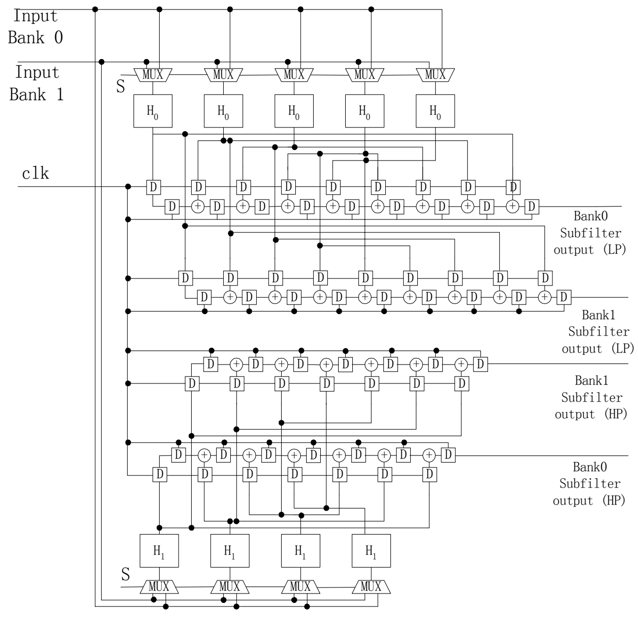
Input Bank 0
Input Bank 1
S
MUX
H_0
clk
D
Bank0 Subfilter output (LP)
Bank1 Subfilter output (LP)
Bank1 Subfilter output (HP)
Bank0 Subfilter output (HP)
H_1
S
MUX

Fig. 5. Two RNS-based FIR Sub-filter Banks with LUT Sharing

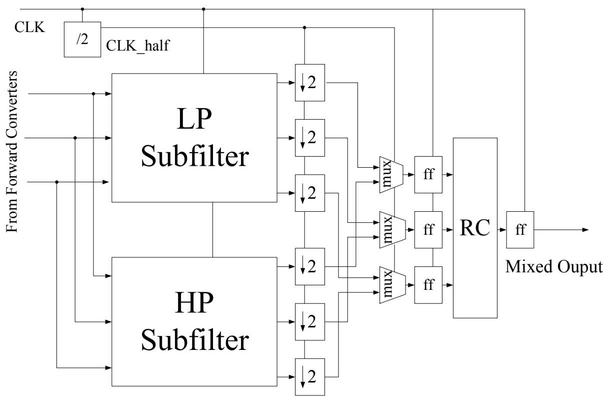

Fig. 6. RNS-based FIR Filter Bank with Reverse Converter Sharing Between LPF and HPF

B. *The Main Controller*

Fig. 7 shows the operation flow of the main controller. The main controller enters initialization mode after reset. During initialization, the main controller sends out reset signals to all other units, initializes its internal counters and waits for the external SDRAM to be ready. After the processor is initialized, the main controller requests image data from the host and stores them into the predefined area of the external SDRAM. When all image data are stored, the main controller instructs the filter banks to perform 2-D DWT on all the image data. Because the capacity of the internal RAM is not large enough to store a complete image in most cases, only part of an image will be read into the internal RAM for processing. After it is processed and stored back to the external SDRAM, other part of the same image will then be read into the internal RAM for processing. This process repeats until all parts of an image are processed and the processor will continues to process next image. If further decomposition is required, the low pass sub-images will be fed into the filter banks again. When all required processes are finished, the processed data are sent back to the host.

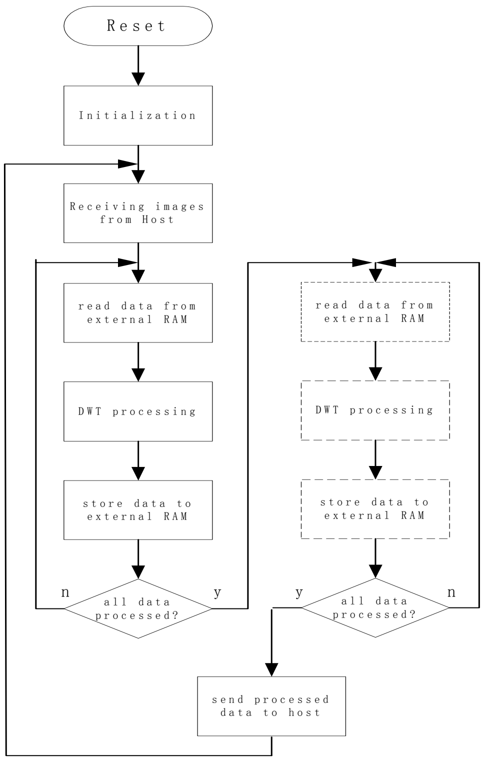
Reset
Initialization
Receiving images from Host
read data from external RAM
DWT processing
store data to external RAM
all data processed?
n
y
read data from external RAM
DWT processing
store data to external RAM
all data processed?
y
n
send processed data to host

Fig. 7. Main Controller Operation Flow

1) *Address Generation*

The first stage of a 2-D DWT is performed on the row data of the image, while the second stage is performed on the column data. Thus, output of first stage needs to be transposed before being fed into the second stage. This transposition is accomplished by using different address generation schemes for row and column processing.

In our design, the external RAM stores image data row by row. For example, consider a 4-by-4 image, memory locations 0 to 3 store pixels in first row, memory locations 4 to 7 store pixels in second row and so on. For this reason, the address for accessing external RAM is divided into three portions as illustrated in Fig. 8. Column and Row portions control which column and row of the image are being accessed respectively. The Predefined Area portion specifies a block of memory locations for a specific image.

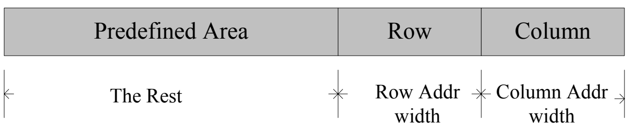

Fig. 8. Address to External SDRAM

Pseudo codes of the address generation schemes for row and column processing are listed in Table. 2. At each time, *N* rows and *N* columns of data are read into the row and column input buffers respectively since the external DDR RAM works in burst mode, which means that a number of data words stored in consecutive locations can be read or written efficiently. The number of data words read in is determined by a parameter called burst length, which is set during the initialization. Thus, the column counter is increased by the burst length each time.

TABLE 2
PSEUDO-CODE FOR ADDRESS GENERATION SCHEMES

Row	Column
If row_count < img_size then	*If col_count < img_size then*
for loop_count =0; loop_count <	*for loop_count =0; loop_count <*
N;loop_count ++	*N;loop_count ++*
for col_count=0;col_count <	*for row_count=0;rowl_count <*
img_size; col_count+burst_length	*img_size;row_count++*
ADDR(0 to col_width –1) <=	*ADDR(0 to col_width –1) <=*
col_count;	*col_count;*
ADDR(col_width to	*ADDR(col_width to*
row_width-1)<=row_count;	*row_width-1)<=row_count;*
ADDR(the rest)<= predefined	*ADDR(the rest)<= predefined*
area;	*area;*
end for;	*end for*
row_count+1;	*col_count+burst_length*
end for	*end for;*
end if;	*end if;*

2) *Data Scheduling During DWT Processing*

If the number of data processed by stage 1 of a 2-D DWT filter banks is N^2, then the number of data processed by each of the stage 2 filter banks will be only $N^2/2$. Therefore, only one filter bank is needed in the second stage that processes both the high pass and low pass outputs from the first stage without delaying the whole process.

Fig.9 illustrates how data are scheduled into both the filter bank0 and filter bank1. Basically, filter bank0 is configured for row processing while filter bank1 is configured for column processing. In cycle 0, filter bank0 performs row processing on image 0 and results in two sub-images (Img 0 L and Img 0 H). In cycle 1, filter bank 0 continues to row process image 1. At the same time, filter bank 1 performs column processing on the two sub-images generated in Cycle 0 by filter bank 0. Generally, in Cycle *X*, filter bank 0 performs row processing on Image *X* and filter bank 1 performs column processing on sub-images of Image *X-1*. In this manner, we can fully utilize the two filter banks and obtain the decomposed sub-images at each DWT processing cycle (except the starting and the ending cycles).

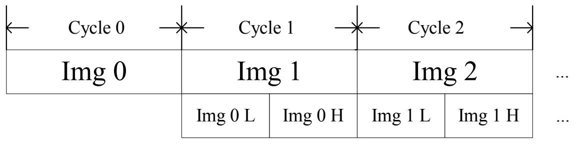

Fig. 9. Data Scheduling

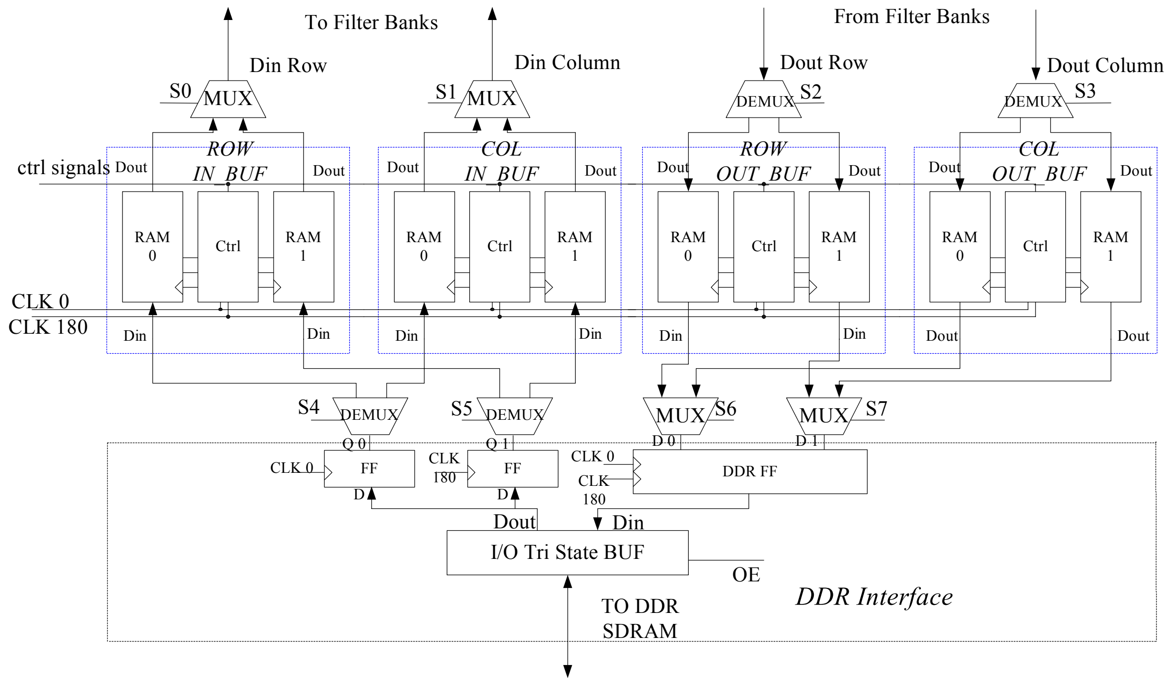

Fig. 10. Structure of the Buffering and DDR Interfacing Unit

C. *The DDR SDRAM Controller*

The DDR SDRAM Controller provides control signals to the external DDR SDRAM. It initializes the memory devices, manages SDRAM banks, and refreshes the memory device at appropriate intervals. The controller translates read/write requests and addresses from the main controller into all the necessary SDRAM command signals.

Special care is needed to handle refresh requests. A refresh request is sent when the refresh counter hits a certain preset value. That may result in conflict if the main controller sends out the read/write command to the RAM in the same interval. The following simple mechanism is used to resolve the conflict. If the read/write request comes before or at the same time as the refresh request, higher priority will be given to the read/write request. That means the SDRAM controller allows the read/write to take place first. After read/write has completed, it will acknowledge the refresh request and send out the refresh command. If the refresh request comes first, the main controller will be notified so that it won't send out any read/write request until refreshing is complete. Since refreshing is delayed by at most one read/write cycle, the preset value of the refresh counter is set to a value that is slightly smaller than the required value to keep the data in the external RAM safe.

D. *Buffering and DDR Interfacing Unit*

The buffering and DDR interfacing unit implements two functions. Firstly, it provides a dual clock rate (DDR) data interface between the DWT processor and the external DDR SDRAM. Secondly, it buffers the data read from the external SDRAM and the processed output data from the filter bank unit.

The structure of the buffering and DDR interfacing unit is shown in Fig. 10. It consists of the DDR interface, row and column input buffers, row and column output buffers. Data transfers to and from the external RAM are double buffered by using two separate 180 degrees out of phase clock signals (CLK 0 and CLK 180).

The buffering module has to work with the DDR interface as well as the filter bank unit. Therefore, there are two factors we need to consider when designing the buffering module. The first factor is that in order to work with the external DDR SDRAM, inputs/outputs between the DDR interface and input/output buffers are separated into two, with one of them responding to the original clock (CLK0), and the other one responding to the inverted clock (CLK180). The second factor is that the RNS-based filter banks are designed to work independently, so they may read in or output data at the same time.

The buffering module is organized as two input buffers (ROW_IN_BUF, COL_IN_BUF) and two output buffers (ROW_OUT_BUF, COL_OUT_BUF). ROW_IN_BUF and ROW_OUT_BUF temporally store input data and output data of the row filter bank, respectively. In the same manner, COL_IN_BUF and COL_OUT_BUF are to work with the column filter bank.

Each input or output buffer consists of two blocks of RAM (RAM0 and RAM1) and one controller. Each block of RAM is organized into 16-bit width and its size depends on the size of image to be processed. RAM0 is to work with Q0 (input buffer) or D0 (output buffer) of the DDR interface. Likewise, RAM1 is to work with Q1 or D1 of the DDR interface. The controller is responsible for generating addresses and all the necessary control signals for both blocks of RAM. Since each buffer has its individual controller, they can work independently and simultaneously.

The controller has two different operation modes: external mode and internal mode as depicted in Fig. 11. External mode is used when the DWT processor reads or writes data to the external DDR SDRAM. In external mode, RAM0 is clocked by CLK0, while RAM1 is clocked by CLK180. Upon receiving the READ command from the main controller, new data from the external SDRAM are read in and stored into both of the input buffers. Even numbered columns are stored into RAM0 and odd numbered columns are stored in RAM1. Likewise, if the WRITE command is received, data in both output buffers are written to the external RAM.

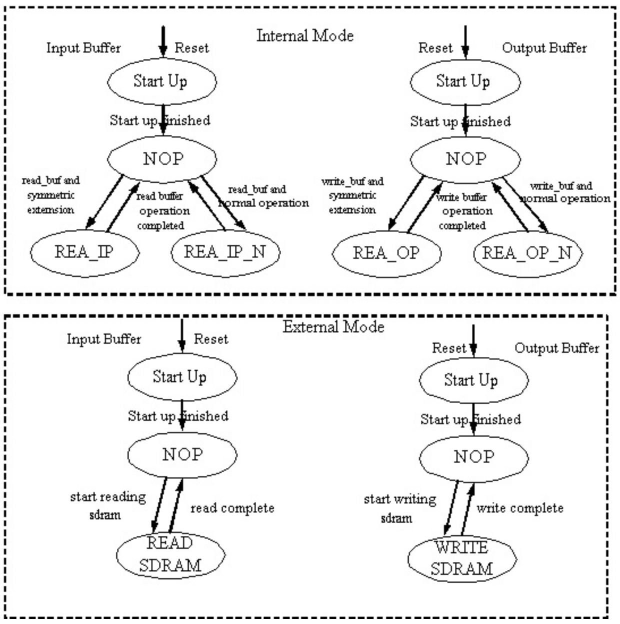

Fig. 11. State Diagrams of the Buffering and DDR Interfacing Unit

In the internal mode, both RAM0 and RAM1 are clocked by a single clock signal (CLK0). And RAM0 and RAM1 operate in an alternative manner. The operation of input buffer is as followed. Data requests are sent by the main controller to request data from any or both of the input buffers. The main controller also needs to indicate whether the data read out from the input buffers are organized in a symmetric extension fashion or a normal fashion. Similarly, if the main controller wants the any or both of output buffers to store output data from the filter banks, it will send out data receive command. The output buffers will then store the output data in to their RAMs. In order to facilitate the latter writing to external SDRAM, even numbered columns and odd numbered column data will be stored in RAM0 and RAM1 respectively.

1) Symmetric Extension of Input Data

Symmetric extension of the input data is realized by arranging and supplying the input data to the filter banks in a special pattern. Consequently, no extra memory space and extra hardware are required for this operation. Since data are stored in different ways for the row and column input buffers, data are arranged and supplied to the filter banks differently.

For row input buffer, let us consider a simplified example of an 8-by-8 image and (1,1) symmetric extension with extension length of 4. The row input buffer stores several rows of the images each time. RAM0 and RAM1 supply data to filter bank0 at alternate cycles. Fig. 12 illustrates the symmetric extension operation on the first row of the image. Addresses for both RAM0 and RAM1 and data sequence are also shown in Fig. 12. Likewise, Fig. 13 shows the symmetric extension operation on the first column of the image, assuming that the first four columns of data are stored in the COL_IN_BUF. Unlike the row input buffer, because all even numbered columns are stored in RAM0 and all odd numbered columns are stored in RAM1, only one block of internal RAM is used to supply data to the column filter bank at any one time.

RAM0	
Addr	Data
0	00
1	02
2	04
3	06
4	10
…	…

RAM1	
Addr	Data
0	01
1	03
2	05
3	07
4	11
…	…

(a) Data in ROW_IN_BUF

RAM	0	1	0	1	0	1	0	1
Addr	2	1	1	0	0	0	1	1
Data	04	03	02	01	00	01	02	03
Mode	Initial Extension				Normal			
RAM	0	1	0	1	0	1	0	1
Addr	2	2	3	3	3	2	2	1
Data	04	05	06	07	06	05	04	03
Mode	Normal				Final Extension			

(b) Address and Data Sequence generated

Fig. 12. (1,1) Symmetric Extension on Row Input Buffer

RAM0		
Addr	Data	
0	00	02
2	10	12
4	20	22
6	30	32
8	40	42
…	…	…

RAM1		
Addr	Data	
0	01	03
2	11	13
4	21	23
6	31	33
8	41	43
…	…	…

(a) Data in COL_IN_BUF

RAM	0	0	0	0	0	0	0	0
Addr	8	6	4	2	0	2	4	6
Data	40	30	20	10	00	10	20	30
Mode	Initial Extension				Normal			
RAM	0	0	0	0	0	0	0	0
Addr	8	10	12	14	12	10	8	6
Data	40	50	60	70	60	50	40	30
Mode	Normal				Final Extension			

(b) Address and Data Sequence generated

Fig. 13. (1,1) Symmetric Extension on Column Input Buffer

E. Other Units

The clock management unit generates clock signals with the same frequency but different phases for the processor. It is implemented by making use of the dedicated digital clock management block provided by the FPGA device.

The host interface allows the 2-D DWT processor to communicate with the host computer. Through the host interface, original image data are downloaded from the host for decomposition and the processed data are uploaded to the host for further processing.

V. Implementation Results

The DWT processor, which is able to process 32×32-pixel images, is coded in VHDL and synthesized into the target FPGA device and simulated. These VHDL models are parameterized so that the synthesized processor can process larger images. The complexity of various controllers is independent of the size of the image, but the size of input and output buffers need to increase to cope with the larger images. Table 3 shows the detailed timing for different parts of the RNS-based filter bank. Table 4 shows the FPGA resources consumed by various modules of the DWT processor. Altogether it occupies 93% of the slices and 20% of the block RAM available in the target FPGA device.

Trade-offs between slices and block RAM can be made in the synthesis of look-up table based modular multipliers. These look-up table based modular multipliers can be synthesized into slices (as shown in Table 4) or into block RAM. If they are synthesized into block RAM, the utilization will be reduced to 37% of total number of slices available. At the same time, block RAM usage will be increased to 88%.

All control units and the DDR interfacing and buffering unit are working under a main clock signal of frequency 100 MHz, while the RNS-base filter banks are clocked by a 25 MHz clock signal derived from the main clock. From the simulation results, under these clock frequencies and a burst length of four for the external SDRAM, it takes 205 μs to finish a first level 2-D DWT decomposition of a 32×32 image, of which 72 μs is for accessing the external SDRAM and 133 μs is for FIR filtering.

VI. Conclusion

This paper reports on the design and implementation of a 2-D biorthogonal DWT processor based on RNS arithmetic. The design of the RNS-based filter banks, various control units and the scheduling of data are discussed in detail. The synthesis results show that the entire design is able to fit into a 1,000,000-gate FPGA device. And this design has been successfully simulated.

Table 3
Timing Details for the RNS-base filter bank (ns)

Channels	Forward Converter	Sub-filters		Reverse Converter
		LUT	Mod Adder and Delay	
255	16.889	6.916	6.776	
256	0	6.148	4.261	32.577
257	18.837	6.933	7.571	

Table 4
FPGA Resource Consumption

	Slices	Flip-flop	4-input LUT	Block RAM	Max frequency (MHz)
RNS Filter Banks	3335	1634	6070	0	28.043
Main Controller	453	172	860	0	100
DDR Interfacing and Buffering	865	505	1542	4K Bytes (8 blocks)	100
SDRAM Controller	140	73	263	0	122

References

[1] ISO/IEC FCD 15444-1 2000v1.0 JPEG 2000 Image Coding System.

[2] Christopher M. Brislawn. "Symmetric extension transforms". *Wavelet Image and Video Compression*, P. N. Topiwala (Eds.), pp. 83-91, London: Kluwer Academic, 1998.

[3] F. Pourbigharaz and H. M. Yassine. "Simple binary to residue transformation with respect to 2^m+1 moduli". *IEE Proceedings on Circuits Devices and Systems*, Vol. 141, No.6 pp. 522-526, Dec. 1994.

[4] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley: Wellesley-Cambridge Press, 1996.

[5] J. Ramirez, A. Garcia, L. Parrilla, A. Lloris, and P. G. Fernandez. "Implementation of RNS analysis and synthesis filter banks for the orthogonal discrete wavelet transform over FPL devices". *Proceedings of 43rd IEEE Midwest Symposium on Circuits and Systems*, Aug. 2000.

[6] J. Ramirez, A. Garcia, P. G. Fernandez, L. Parrilla, and A. Lloris. "Implementation of canonical and retimed RNS architectures for the orthogonal 1-D DWT over FPL Devices". *Conference Record of the 34th Asilomar Conference on Signals, Systems and Computers*, Vol. 1, pp. 384-388, 2000.
[7] J. Ramirez, U. Meyer-Base, F. Taylor, A. Garcia, and A. Lloris. "Design and implementation of high-performance RNS wavelet processors for custom IC technologies". *Journal of VLSI Signal Processing*. vol. 34, pp. 227-237, Kluwer Academic, 2003.
[8] Kavish Seth and S. Srinivasan. "VLSI implementation of 2-D DWT/IDWT cores using 9/7-tap filter banks based on the non-expansive Symmetric extension scheme". *Proceedings of the 15th International Conference on VLSI Design*, pp. 435-440, Jan, 2002.
[9] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, eds. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.
[10] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw-Hill, 1967.
[11] S.G. Mallat, "A Theory of multiresolution signal decomposition: The wavelet representation", *IEEE Trans. on Pattern Recognition and Machine Intelligence*. Vol. 11, Jul. 1989.
[12] Yuke Wang, Xiaoyu song, Mostapha Aboulhamid and Hong Shen. "Adder Based Residue to Binary Number Converters for (2^n-1, 2^n, 2^n+1)". *IEEE Trans. on Signal Processing*, vol. 50, no. 7, pp. 1772-1779, Jul. 2002.
[13] J. D. Villasenor, B. Belzer, and J. Liao. "Wavelet Filter Evaluation for Image Compression". *IEEE Trans. on Image Processing*, Vol. 4, No. 8, pp. 1053-1060, Aug, 1995.
[14] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.
[15] Yong Liu and Edmund M-K Lai. "Moduli Set Selection and Cost Estimation for RNS-based FIR Filter and Filter Bank Design". Submitted to *Design Automation for Embedded Systems*, Kluwer Academic Publishers.
[16] *Virtex-II V2MB1000 Development Board User's Guide, Version 3.0*, Memec Design, Dec. 2002.

Yong Liu received the B.A.Sc. (Hons) degree in computer engineering from Nanyang Technological University, Singapore in 2002. He is currently pursuing his M. Eng. degree in the School of Computer Engineering, Nanyang Technological University, Singapore.

Edmund M-K. Lai (M'82-SM'95) received the B.E.(Hons) and PhD degrees in 1982 and 1991 respectively from the University of Western Australia, both in electrical engineering. Starting as an engineer in the semiconductor manufacturing industry in Hong Kong in 1982, he became a faculty member of the Department of Electrical and Electronic Engineering, The University of Western Australia in 1985 while pursuing his PhD part-time. In 1990, he joined the Department of Information Engineering, the Chinese University of Hong Kong as Lecturer. After a two-year stint as an independent consultant in signal processing education, software and hardware system development in Perth, Western Australia between 1998 and 1999, he joined the School of Computer Engineering, Nanyang Technological University in Singapore as Associate Professor in December 1999. His current research interests include digital signal processing theory and applications, information theory and ultra-wideband communications.
.