A TREE-BLOCK SCHEDULING ARCHITECTURE FOR SEPARABLE 2-D INVERSE DISCRETE WAVELET TRANSFORM

Yeu-Horng Shiau and Jer Min Jou

Department of Electrical Engineering,
National Cheng Kung University, Tainan, Taiwan, R. O. C.
Email:huh@j92a21.ee.ncku.edu.tw

ABSTRACT

In this paper, an efficient VLSI architecture for the 2-D inverse discrete wavelet transform is proposed. We adopt a tree-block pipeline-scheduling scheme in it for increasing computation performance and reducing temporary buffer. The scheme divides the input data into several wavelet blocks and processes these blocks one by one, so that the size of buffer for storing temporal data is greatly reduced to only the size of one block. The scheduling also makes the data flow tight and regular to meet high speed and low complexity. In addition, the architecture is pipelined efficiently to reach higher throughput rate. Each filter is designed regularly and modularly, so it is easily scalable for different filter lengths and different levels by adding some extra modules and some control signals. We also show that hardware utilization of the two filters is 100%. Due to its low hardware cost, small storage, regularity, and high performance, the architecture can be applied to real-time applications, such as MPEG-4 and JPEG-2000.

1. INTRODUCTION

Recently, there has been a great amount of interest in the field of Forward and Inverse Discrete Wavelet Transform (FDWT/IDWT) for image and video processing. The FDWT decomposes a nonstationary signal into a set of multiscaled wavelets, which are relatively more stationary and easier to be coded, and then these transformed components can be exactly recovered by IDWT. The advantage of DWT is favored over other transforms mainly from the fact that the DWT performs a mutiresolution analysis of a signal with localization in both time and frequency. This mutiresolution analysis is very similar to the way in which the human visual system interprets images, and such is a natural and efficient way to code and store the two-dimensional image data. However, software computation of the DWT requires a relatively long computation time. Therefore, designing a special integrated circuit for DWT is necessary and urgent to achieve higher computation speed, especially for real-time high-resolution video processing.

Up to now, a number of VLSI architectures for DWT have been designed and implemented to meet the requirement for real-time applications. Lewis and Knowles [1] first presented multipilerless 2-D forward and inverse architecture based on the four-tap Daubechies wavelet transform. As a result, their architecture doesn' t work efficiently for other wavelets. In [2]-[6], the architectures for FDWT have been developed and have worked efficiently in some applications. In [7]-[9], efficient VLSI architectures for both FDWT and IDWT was proposed and implemented. Most of the above architectures were aimed at the optimal design for the FDWT, which was modified to fit the manner in IDWT. Such design, on the one hand, is well done for combining the architecture of the FDWT and IDWT in a single chip, but on the other hand, these architectures were not the optimal solutions for the design of IDWT. Consequently, they might not work efficiently in the inverse manner. Kim and Lee [10] developed a scalable VLSI architecture employing a two-channel quadrature mirror filter (QMF) lattice for the one-dimensional (1-D) DWT. Acharya and Chen [11] presented a systolic architecture for 1-D DWT and it is suitable for both decomposition and reconstruction of signals. The above architectures could work efficiently for 1-D DWT, but it needs a large storage module for rearranging the data between the row and column computations when the architectures were extended to the separable 2-D DWT. It is impractical to implement in a single chip when the image size becomes too large. In [12], a low-cost VLSI architecture based on 2-D IDWT was presented but was not easily to scale up for different filter lengths. Chakrabarti and Mumford [14] proposed several architectures and scheduling algorithms for encoders and decoders based on 2-D DWT. Their motivation was to aid the designer in choosing one that is best suited for a specific application. While some applications require the latency or the computation time to be low others require the memory size or the area to be low. However, the filters used in their architectures were not active all the time. That means the computation time can be sped up.

It can be seen that there are a few architectures for 2-D IDWT and most of them are just modified from the architectures for FDWT as mention before. Actually, the behaviors of FDWT and IDWT are similar but not really all the same. The most difference is that in the FDWT the required computation for the next level is reduced while the required computation for the next level is increased in the IDWT. If we treated the behavior in the IDWT like the one in the FDWT, such design might not work efficiently in the inverse manner. In this paper, we consider the difference between FDWT and IDWT, so we concentrate on the manner in IDWT and propose a tree-block pipeline-scheduling scheme for the VLSI design. The rest of this paper is organized as follows. In Section II, we briefly described the subband structure of the IDWT. In Section III, we introduce the tree-block scheduling scheme. In Section

IV, we describe the proposed VLSI architecture. Then in Section V, we show the proposed architecture can be easily scalable for different filter lengths and different octave levels. In Section VI, we present some results. Finally, some conclusions are presented in Section VII.

2. INVERSE DISCRETE WAVELET TRANSFORM

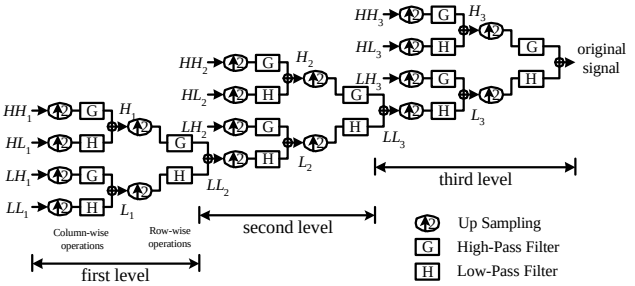


Fig. 1. The synthesis of 2-D separable IDWT for three-level.

The IDWT processes the input signal subbands from the coarsest resolution level to the finest one. Generally, the 2-D IDWT computation can be classified into separable and non-separable types according to its operating manner. Here we only consider the separable type. Fig. 1 shows the computation of a three-level 2-D separable IDWT synthesis process, in which each level is composed of two types of processing, the column-wise processing and the row-wise processing. Each processing consists of the high-pass filtering (G), the low-pass filtering (H), and up sampling. We denote the four inputs at *d*th-level computation along the column by HH_d, HL_d, LH_d, and LL_d, and denote the two inputs at *d*th-level computation along the row by H_d and L_d. Suppose that the length of filter (or filter tap) is of size *K*, the transfer functions of filters H and G are represented as follows:

$$\mathrm{H}(z)=\sum_{k=0}^{K-1} h_k z^{-k}\ ,\ \mathrm{G}(z)=\sum_{k=0}^{K-1} g_k z^{-k}$$

In the following section we first present our proposed architecture with four filter taps for both H(*z*) and G(*z*) and with three-level reconstruction for example, and then we will show it can be easily scalable for different filter lengths and different octave levels.

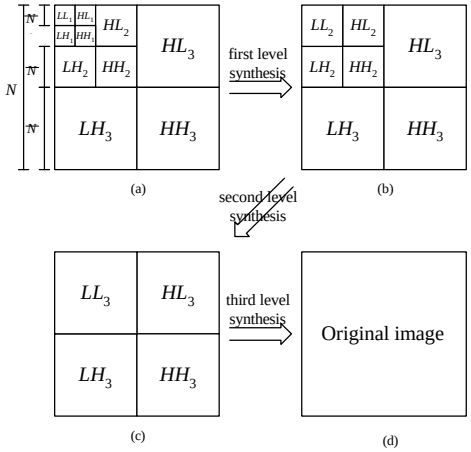


Fig. 2. An example for the processing of 2-D 3-level IDWT.

Now, we look close to the procedure of the reconstruction. Let us consider a three level 2-D IDWT for image synthesis. An *N*×*N* sequence of input data can be represented as Fig. 2(a). At the beginning of processing, the finer-coefficient subband labeled LL_2 in Fig. 2(b) is first reconstructed from the four coarsest subbands LL_1 , LH_1, HL_1 , and HH_1 in Fig. 2(a), and then the four subband LL_2, LH_2, HL_2, and HH_2 is used further to obtain the next finer scaled wavelet coefficients. This processing continues until the original image of size $N \times N$ is reconstructed (see Fig. 1(a)→(b)→(c)→(d)). Note that, because of the up sampling required for both column-wise and row-wise processing, the size of the synthesized subband is four times larger than the original after one level processing. Then, the required computation in the next level is relatively increased.

3. TREE-BLOCK SCHEDULE

We propose a new scheme for the 2-D IDWT called tree-block pipelining schedule. In this scheme, the schedule of the input subband coefficients is based on the depth-first (block-first) traverse, while the previous methods are based on the breadth-first (level-first) traverse [2]-[4]. Specifically, the breadth-first traverse just like Fig.1 first inputs the four coarsest subbands to reconstruct the finer subband that is further used with three other subbands in the next level; the processing is repeated level by level until the original image is reconstructed. In such method, at each level, the processing cannot be started until the four wavelet subbands are obtained, so this breadth-first traverse will need a great amount buffers to store temporal low-low subbands. For example, an *N*×*N* input sequence for the computation of three-level IDWT at least requires $N^2/4$ temporary buffers to store the finest subband coefficients; it is impractical for high-resolution image frame. Besides, waiting the produced coefficients makes the hardware idle some time, as a result, such design increases the overall computation time.

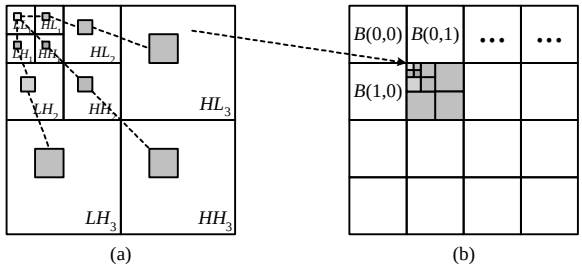


Fig. 3. Reorganizing of a wavelet tree into a wavelet block.

However, the tree-block scheduling, at the beginning, divides all the input subband coefficients into several wavelet trees [13]. Each tree contains a tree root, which is one coefficient in the coarsest subband, and its descendants, which are the coefficients relative to the tree root in all the finer scale subbands (see Fig. 3(a)). Then, the coefficients of each wavelet tree are reorganized to form a wavelet block as shown in Fig. 3(b) where *B*(*i*,*j*) represents the wavelet block at location (*i*,*j*). The concept of the wavelet block provides an association between wavelet coefficients and what they represent spatially in the image frame. Therefore, this scheduling scheme processes the frame

block by block for all levels not level by level until the whole frame is reconstructed. The advantages of the block-based processing are that (1): The next level computations within the same block can be started as early as possible and are easily pipelined to get a higher performance, and (2): the size of a wavelet block is invariant and is less than the image size, so the size of buffer for storing temporal data is greatly reduced from the whole frame size to the size of a wavelet block. For a D-level IDWT, we need only 4^{D-1} temporary buffers and the value of D is always small.

We assume that the synthesized level is D, and that the size of synthesized image is $N \times N$. Thus, the input sequence is divided into $N^2/4^D$ wavelet blocks each of which is of size $2^D \times 2^D$. Each wavelet block $B(i,j)$ contains 4 coefficients at the first level, which are $LL_1(i,j)$, $LH_1(i,j)$, $HL_1(i,j)$ and $HH_1(i,j)$, 12 coefficients at the second level, which are $LH_2(2i+r,2j+s)$, $HL_2(2i+r,2j+s)$, and $HH_2(2i+r,2j+s)$ for $0 \le r,s \le 1$, 48 coefficients at the third level, which are $LH_3(4i+r,4j+s)$, $HL_3(4i+r,4j+s)$, and $HH_3(4i+r,4j+s)$ for $0 \le r,s \le 3$, and $3 \cdot 2^{d+1}$ coefficients at the dth level, which are $LH_d(2^{l-1}i+r, 2^{l-1}j+s)$, $HL_d(2^{l-1}i+r, 2^{l-1}j+s)$, and $HH_d(2^{l-1}i+r, 2^{l-1}j+s)$ for $0 \le r,s \le 2^{l-1}-1$. Here $LL_d(x,y)$, $LH_d(x,y)$, $HL_d(x,y)$, and $HH_d(x,y)$ represent the wavelet coefficients at the coordinate (x,y) in the subband LL_d, LH_d, HL_d, and HH_d, and $L_d(x,y)$ and $H_d(x,y)$ represent the wavelet coefficients at the coordinate (x,y) in subband L_d and HH_d, respectively. The scheduling procedure for a wavelet block $B(i,j)$ is described below.

At Level d:

(1) For $0 \le r,s \le 2^{d-1}-1$, schedule $HH_d(2^{d-1}i+r, 2^{d-1}j+s)$ and $HL_d(2^{d-1}i+r, 2^{d-1}j+s)$ at time $T + (2/3)(4^{d-1}-1) + 2(2^{d-1}r+s)$. Compute the outputs $H_d(2^di+2r, 2^{d-1}j+s)$ and $H_d(2^di+2r+1, 2^{d-1}j+s)$. Schedule $LL_d(2^di+r,2^dj+s)$ and $LH_d(2^di+r,2^dj+s)$ at time $T + (2/3)(4^{d-1}-1) + 2(2^dr+s) + 1$. Compute the outputs $L_d(2^di+2r, 2^{d-1}j+s)$ and $L_d(2^di+2r+1, 2^{d-1}j+s)$.

(2) For $0 \le r,s \le 2^{d-1}-1$, schedule $H_d(2^di+2r, 2^{d-1}j+s)$ and $H_d(2^di+2r+1, 2^{d-1}j+s)$ at time $T + (2/3)(4^{d-1}-1) + 2(2^dr+s)+1$, and schedule $L_d(2^di+2r, 2^{d-1}j+s)$ and $L_d(2^di+2r+1, 2^{d-1}j+s)$ at time $T+(2/3)(4^{d-1}-1)+2(2^dr+s)+2$. Compute $LL_d(2^di+2r, 2^dj+2s)$, $LL_d(2^di+2r, 2^dj+2s+1)$, $LL_d(2^di+2r+1, 2^dj+2s)$, and $LL_d(2^di+2r+1, 2^dj+2s+1)$.

It can be seen that the procedure is executed block by block in the row major order. Thus, the beginning time T in each wavelet block $B(i,j)$ can be calculated as $\frac{2}{3}(4^D-1)\times(i+j\times\frac{N}{2^D})$.

4. PROPOSED ARCHITECTURE

Based on the tree-block scheduling described in Section III, the computations of a wavelet block $B(i,j)$ is shown in Fig. 4. According to the equations in Fig. 4, the data flow of the tree-block architecture is drawn in Fig. 5. To minimize the hardware cost, the data flow can be folded along x-direction and y-direction onto one processor. Then, an efficient architecture for 2-D IDWT computation is derived. Fig. 6 presents the block diagram of this high performance and low cost VLSI architecture, which is composed of one column filter, one row filter, two storage units, and a temporary buffer unit. From Fig. 4, the proposed inverse architecture first performs the column filtering and then the row filtering to process the decomposed subbands. Note that the two efficient filter structures are used to process the computation of each wavelet block in the row major order for all levels. The column filtering computes along the columns and produces the outputs in the row major order. The row filtering read the data from the column filtering and computes along the rows and produces the outputs in the row major order. The two storage units store the partial results for the column and row filtering, which will be used within current block or in other blocks. The temporary buffer unit stores the subband LL_d(for d >1) of a wavelet block. The details of each component are described in the following discussions. We first present our proposed architecture with four filter taps for both H(z) and G(z) and with three-level reconstruction for example, and then we will show it can be easily scalable for different filter lengths and different octave levels.

For d = 1 to D {
  For $q = (2^{d-1} \cdot i)$ to $(2^{d-1} \cdot i + 2^{d-1} - 1)$ {
    For $p = (2^{d-1} \cdot i)$ to $(2^{d-1} \cdot i + 2^{d-1} - 1)$ {
    Column-wise processing:
    $H_d(2p,q)=HL_d(p,q)h_0+HH_d(p,q)g_0+HL_d(p\text{-}1,q)h_2+HH_d(p\text{-}1,q)g_2$
    $H_d(2p+1,q)=HL_d(p,q)h_1+HH_d(p,q)g_1+HL_d(p\text{-}1,q)h_3+HH_d(p\text{-}1,q)g_3$
    $L_d(2p,q)=LL_d(p,q)h_0+LH_d(p,q)g_0+LL_d(p\text{-}1,q)h_2+LH_d(p\text{-}1,q)g_2$
    $L_d(2p+1,q)=LL_d(p,q)h_1+LH_d(p,q)g_1+LL_d(p\text{-}1,q)h_3+LH_d(p\text{-}1,q)g_3$
    Row-wise processing:
    $LL_{d+1}(2p,2q)=L_d(2p,q)h_0+H_d(2p,q)g_0+L_d(2p,q\text{-}1)h_2+H_d(2p,q\text{-}1)g_2$
    $LL_{d+1}(2p,2q+1)=L_d(2p,q)h_1+H_d(2p,q)g_1+L_d(2p,q\text{-}1)h_3+H_d(2p,q\text{-}1)g_3$
    $LL_{d+1}(2p+1,2q)=L_d(2p+1,q)h_0+H_d(2p+1,q)g_0+L_d(2p+1,q\text{-}1)h_2+H_d(2p+1,q\text{-}1)g_2$
    $LL_{d+1}(2p+1,2q+1)=L_d(2p+1,q)h_1+H_d(2p+1,q)g_1+L_d(2p+1,q\text{-}1)h_3+H_d(2p+1,q\text{-}1)g_3$
    }
  }
}

Fig. 4. The computations of a wave block $B(i,j)$.

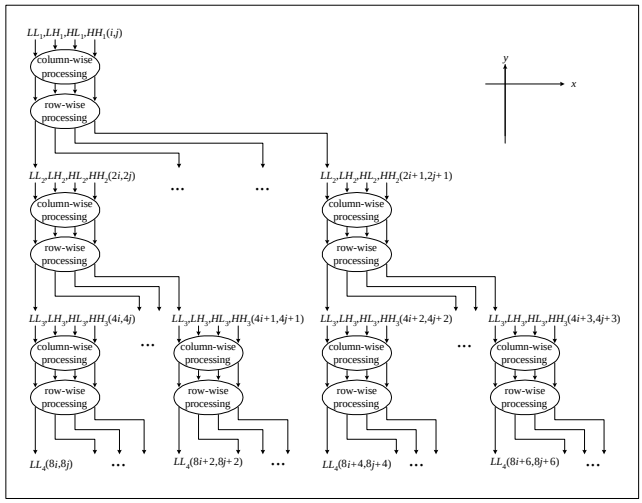


Fig. 5. The data flow for one wavelet block (D=3).

We now list the equations for column filtering in the dth level. Here we only show the equations for computing the outputs $H_d(i,j)$ because the equations for computing $L_d(i,j)$ is similar to the ones for $H_d(i,j)$. The equations are shown

as follows.

$$H_d(m,n)=HL_d(m,n)h_0+HH_d(m,n)g_0+HL_d(m\text{-}1,n)h_2+HH_d(m\text{-}1,n)g_2. \quad (1)$$

$$H_d(m+1,n)=HL_d(m,n)h_1+HH_d(m,n)g_1+HL_d(m\text{-}1,n)h_3+HH_d(m\text{-}1,n)g_3. \quad (2)$$

$$H_d(m+2,n)=HL_d(m+1,n)h_0+HH_d(m+1,n)g_0+HL_d(m,n)h_2+HH_d(m,n)g_2. \quad (3)$$

$$H_d(m+3,n)=HL_d(m+1,n)h_1+HH_d(m+1,n)g_1+HL_d(m,n)h_3+HH_d(m,n)g_3. \quad (4)$$

$$H_d(m+4,n)=HL_d(m+2,n)h_1+HH_d(m+2,n)g_1+HL_d(m+1,n)h_3+HH_d(m+1,n)g_3.$$

...

From the above computations, it can be seen that the coefficient $HH_d(m,n)$ appears in all the four equations (equation (1)-(4)) and so does the coefficient $HL_d(m,n)$. If we can process the four equations in parallel, the storage access times of $HH_d(m,n)$ and $HL_d(m,n)$ will be reduced and the computation time will speed up. According to this idea, the column filter is designed with four modules, M_u, where $1 \le u \le 4$. Each of them has the same structure and calculates one of the four equations. Fig. 7(a) and Fig. 7(b) show the structure of module M_u and the block diagram of the column filter, respectively. The module M_u consists of two multipliers and one adder. The $2n$-to-n reduced width multipliers we developed [15] are used here to reduce the area. When n is large enough, the $2n$-to-n multiplier needs only half area of a standard parallel multiplier. To reach higher performance, two registers are inserted into the output for pipelining with the row filter.

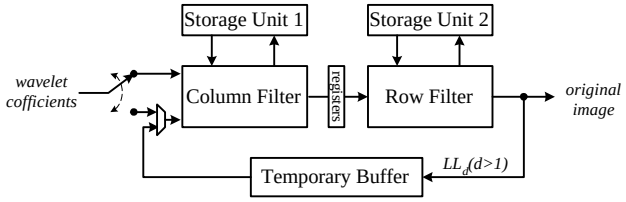


Fig. 6. The block diagram of the tree-block architecture.

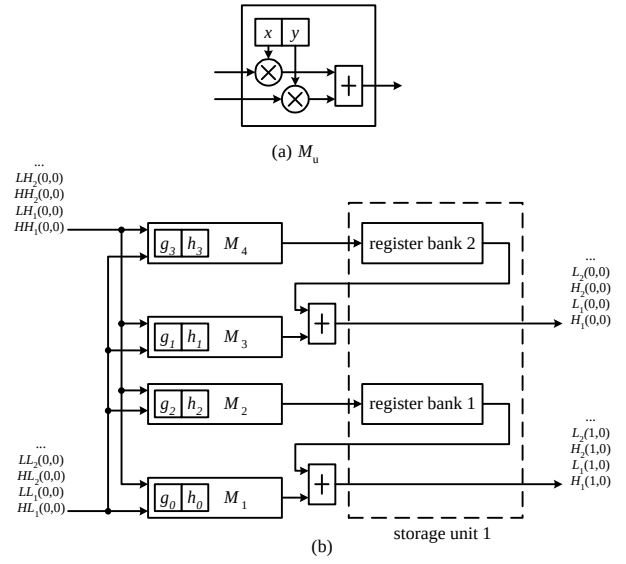


Fig. 7. (a) The structure of $M(u)$ and

(b) The block diagram of the column filter.

In Fig. 7(b), two coefficients, $HH_d(m,n)$ and $HL_d(m,n)$, are fed simultaneously into four modules to compute the four equations in parallel, but the results are only the partial sums of the product terms of four finer coefficients, $H_d(m,n)$, $H_d(m+1,n)$, $H_d(m+2,n)$, and $H_d(m+3,n)$. Thus, equation (1) to (4) should be changed as follows:

$$H_d(m,n)=HL_d(m,n)h_0+HH_d(m,n)g_0+P_d(m,n)$$

$$[P_d(m,n)=HL_d(m\text{-}1,n)h_2+HH_d(m\text{-}1,n)g_2].$$

$$H_d(m+1,n)=HL_d(m,n)h_1+HH_d(m,n)g_1+P_d(m+1,n)$$
$$[P_d(m+1,n)=HL_d(m\text{-}1,n)h_3+HH_d(m\text{-}1,n)g_3].$$

$$P_d(m+2,n)=HL_d(m,n)h_2+HH_d(m,n)g_2.$$

$$P_d(m+3,n)=HL_d(m,n)h_3+HH_d(m,n)g_3.$$

where $P_d(m,n)$, $P_d(m+1,n)$, $P_d(m+2,n)$, and $P_d(m+3,n)$ represent the partial sums of $H_d(m,n)$, $H_d(m+1,n)$, $H_d(m+2,n)$, and $H_d(m+3,n)$, respectively. To store these partial sums of the finer coefficients, storage unit 1 is used for column filtering as shown in Fig. 7(b). It can be seen that there are two register banks in storage unit 1. Based on the concept of wavelet block, each register bank consists of two types of register sub-banks, one for intra-block storing and another for inter-block storing, which are implemented with shift register routing networks shown in Fig. 8. Intra-block sub-bank with the size of 8 units stores the partial sums that will be used within the current wavelet block, while Inter-block sub-banks with the size of $2N$ units store the partial sums that will be used in the other wavelet block. Here we need an inter-block sub-bank in each level because the lifetimes of inter-block sub-banks in different levels are overlap. Since the column filtering computes the outputs in the row major order, we must store the partial sums of one-row coefficients. Thus, the size of the inter-block sub-bank for dth level is relative to the image size N and level d and is of size $\frac{N}{2^{3-d}}$. An example for the scheduling for the column filtering in a wavelet block is shown in Fig. 9, where Q_d represents the partial sum of $L_d(m,n)$. To understand the difference between the inter-block registers and intra-block registers, for instance, $P_1(2,0)$ and $P_1(3,0)$ in Fig. 9 belong to the other block, and they will be store in inter-block register bank. $P_2(2,0)$ and $P_2(3,0)$ are stored in intra-block register bank since they are within current wavelet block and will be used in the 7th clock cycle.

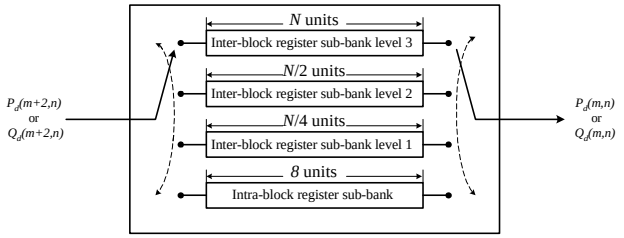


Fig. 8. The routing networks of the registers in storage unit 1.

Time unit	Input	Input / Read from storage unit 1	Output	Output/ Write to storage unit 1
1	$HH_1(0,0)$, $HL_1(0,0)$	$P_1(0,0)$, $P_1(1,0)$	$H_1(0,0)$, $H_1(1,0)$	$P_1(2,0)$, $P_1(3,0)$
2	$LH_1(0,0)$, $LL_1(0,0)$	$Q_1(0,0)$, $Q_1(1,0)$	$L_1(0,0)$, $L_1(1,0)$	$Q_1(2,0)$, $Q_1(3,0)$
3	$HH_2(0,0)$, $HL_2(0,0)$	$P_2(0,0)$, $P_2(1,0)$	$H_2(0,0)$, $H_2(1,0)$	$P_2(2,0)$, $P_2(3,0)$
4	$LH_2(0,0)$, $LL_2(0,0)$	$Q_2(0,0)$, $Q_2(1,0)$	$L_2(0,0)$, $L_2(1,0)$	$Q_2(2,0)$, $Q_2(3,0)$
5	$HH_2(0,1)$, $HL_2(0,1)$	$P_2(0,1)$, $P_2(1,1)$	$H_2(0,1)$, $H_2(1,1)$	$P_2(2,1)$, $P_2(3,1)$
6	$LH_2(0,1)$, $LL_2(0,1)$	$Q_2(0,1)$, $Q_2(1,1)$	$L_2(0,1)$, $L_2(1,1)$	$Q_2(2,1)$, $Q_2(3,1)$
7	$HH_2(1,0)$, $HL_2(1,0)$	$P_2(2,0)$, $P_2(3,0)$	$H_2(2,0)$, $H_2(3,0)$	$P_2(4,0)$, $P_2(5,0)$
8	...	...	...	...

Fig. 9. The schedule for the column filtering.

The equations for the row filtering, which are analogous to column filtering, are shown as follows.

$$LL_{d+1}(m,n)=L_d(m,n)h_0+H_d(m,n)g_0+L_d(m,n\text{-}1)h_2+H_d(m,n\text{-}1)g_2 \quad (5)$$

$$LL_{d+1}(m,n+1)=L_d(m,n)h_1+H_d(m,n)g_1+L_d(m,n\text{-}1)h_3+H_d(m,n\text{-}1)g_3 \quad (6)$$

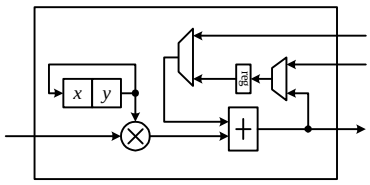
$$LL_{d+1}(m,n+2)=L_d(m,n+1)h_0+H_d(m,n+1)g_0+L_d(m,n)h_2+H_d(m,n)g_2 \quad (7)$$

$$LL_{d+1}(m,n+3)=L_d(m,n+1)h_0+H_d(m,n+1)g_0+L_d(m,n)h_2+H_d(m,n)g_2 \quad (8)$$

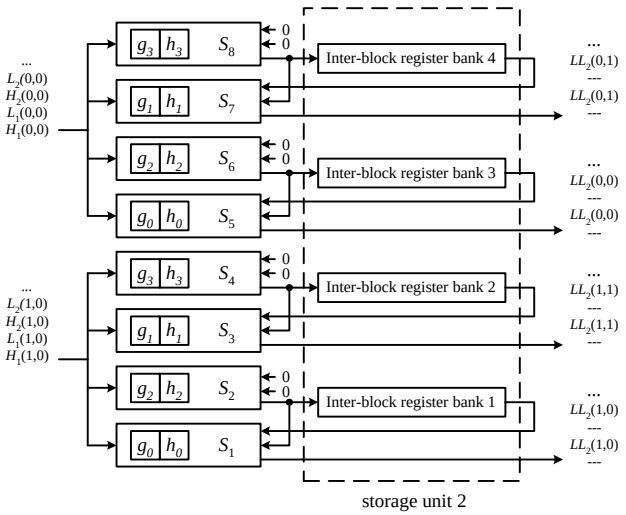
$$LL_{d+1}(m,n+4)=L_d(m,n+2)h_0+H_d(m,n+2)g_0+L_d(m,n+1)h_2+H_d(m,n+1)g_2$$

…

From above, $L_d(m,n)$ and $H_d(m,n)$ are in all of the four equations, so the idea of parallel processing is also considered. Besides, without storing the results, our design starts row filtering as soon as the results of the column filtering are produced. According to the scheduling in Fig. 9, two finer coefficients are produced every clock cycle in the column filtering, such as $L_d(m,n)$ and $L_d(m+1,n)$, so that the row filter must compute eight equations in parallel. Based on these concepts, the row filter is designed with eight modules, S_u, where $1 \le u \le 8$. All of them have the same architecture and deal with the partial sum of eight equations.



(a)



(b)

Fig. 10. (a) The structure of $S(u)$.

(b) The block diagram of the row filter.

Fig. 10(a) and Fig. 10(b) show the structure of module S_u and the block diagram of the row filter, respectively. In module S_u, we also use the 2*n*-to-*n* reduced width multipliers to reduce the area. Similar to the column filter, storage unit 2, which is shown in Fig. 10(b), is used here for storing the partial results of coefficients. It only consists of inter-block registers with each size of 7 units because the partial sum of intra-block coefficients will be soon used in the next clock cycle. The partial sum of intra-block coefficients are immediately written to and read from the registers in module S_u. Note that the row filtering processes the wavelet blocks in the row major order, so the lifetime of inter-block registers in a block is the time to process a wavelet block. The produced coefficients of subband LL_d (for d >1) are stored in the temporary buffer unit whose size is 4^{d-1} for dth level. These temporal coefficients in buffer will be soon used in the next level. An example for the scheduling for the row filtering in a wavelet block is shown in Fig. 11.

Time unit	Input	Output
2	$H_1(0,0),H_1(1,0)$	¾
3	$L_1(0,0),L_1(1,0)$	$LL_2(0,0),LL_2(0,1),LL_2(1,0),LL_2(1,1)$
4	$H_2(0,0),H_2(1,0)$	¾
5	$L_2(0,0),L_2(1,0)$	$LL_3(0,0),LL_3(0,1),LL_3(1,0),LL_3(1,1)$
6	$H_2(0,1),H_2(1,1)$	¾
7	$L_2(0,1),L_2(1,1)$	$LL_3(0,2),LL_3(1,3),LL_3(1,2),LL_3(1,3)$
8	$H_2(2,0),H_2(3,0)$	¾
9	$L_2(2,0),L_2(3,0)$	$LL_3(2,0),LL_3(2,1),LL_3(3,0),LL_3(3,1)$
10	$H_2(2,1),H_2(3,1)$	¾
11	$L_2(2,1),L_2(3,1)$	$LL_3(2,2),LL_3(2,3),LL_3(3,2),LL_3(3,3)$
…		…

Fig. 11. The schedule for the row filtering.

5. SCALABILITY FOR THE IDWT

In this section, we will show the extension for difference filter tap and different levels of the proposed architecture. When the high-pass filter tap extends to K_1 and the low-pass filter tap extends to K_2 (suppose K_1>K_2, and K_1 and K_2 are both even.), the equations for column filtering in the dth level are changed as follows.

$$H_d(m+2k,n)=\sum_{p=0}^{\frac{K_2}{2}-1} HL_d(m+k-p,n)h_{2p}+\sum_{q=0}^{\frac{K_1}{2}-1} HH_d(m+k-q,n)g_{2q}.$$

$$H_d(m+2k+1,n)=\sum_{p=0}^{\frac{K_2}{2}-1} HL_d(m+k-p,n)h_{2p+1}+\sum_{q=0}^{\frac{K_1}{2}-1} HH_d(m+k-q,n)g_{2q+1}.$$

for k = 0, 1, 2, …,(K_1-1).

It can be observed that $HL_d(m,n)$ appears in the first K_2 equations and $HH_d(m,n)$ appears in all the K_1 equations. Similarly, to reduce the storage access times, the architecture for column filtering, which is changed to Fig. 12(b), needs K_2 M_u-modules and (K_1-K_2) $M¢_u$-modules to calculate these K_1 equations at the same time. The structure of $M¢_u$-module is presented in Fig. 12(a). These K_2 M_u-modules compute the partial sum of the first K_2 equations every time while (K_1-K_2) $M¢_u$-modules compute the partial sum of the other (K_1-K_2) equations. To store these partial sums, the storage unit 1 needs (K_1-2) register banks and each register banks are the same as Fig. 13. The total size of storage unit 1 is calculated as follows.

Total size = (Intra-block registers + Inter-block registers) × number of register banks

$$= \left(N+\frac{N}{2}+\ldots+\frac{N}{2^{D-1}}+2^D\right)\times(K_1\text{-}2) \approx 2(K_1\text{-}2)\times N \text{ units}$$

The equations for the row filtering with K_1 high-pass filter tap and with K_2 low-pass filter tap are shown as follows.

$$LL_{d+1}(m,n+2k)=\sum_{p=0}^{\frac{K_2}{2}-1} L_d(m,n+k-p)h_{2p}+\sum_{q=0}^{\frac{K_1}{2}-1} L_d(m,n+k-q)g_{2q}.$$

$$LL_{d+1}(m,n+2k+1)=\sum_{p=0}^{\frac{K_2}{2}-1} L_d(m,n+k-p)h_{2p+1}+\sum_{q=0}^{\frac{K_1}{2}-1} L_d(m,n+k-q)g_{2q+1}.$$

for k = 0, 1, 2, …,(K_1-1).

From above, $L_d(m,n)$ appears in the first K_2 equations and

$H_d(m,n)$ appears in all the K_1 equations, so the row filtering consists of $2K_1$ S_u-modules as shown in Fig. 14. The K_1 S_u-modules compute the K_1 equations for $L_d(m,n)$, and the other K_1 S_u-modules compute the K_1 equations for $L_d(m+1,n)$. Similarly, those S_u-modules calculate only partial sums of the equations, which must be stored in the storage unit 2. The storage unit 2 includes $2(K_1\text{-}2)$ inter-block register banks, each of which is of size $2^D - 1$. The total size of storage unit 2 is calculated as follows.

Total size =Inter-block registers × number of register banks

$$=(2^D - 1)\times(2K_1 - 4) \text{ units}$$

6. RESULTS

It can be seen from the scheduling described in Section III that it requires two cycles to process the four coefficients in the first level and requires eight cycles to process the sixteen coefficients in the second level within a wavelet block. If the octave level extends to D, the number of cycles to process one wavelet block is as follows.

$$\text{Cycles} = 2 + 8 + 32 + \ldots + 2\cdot 4^{(D-1)} = \frac{2}{3}(4^D - 1).$$

Thus, the total number of cycles for processing an $N\times N$ image is shown below:

Total Cycles = the number of wavelet blocks

× Cycles for one block

$$= \frac{1}{4^D} N^2 \times \frac{2}{3}(4^D - 1) = \frac{2}{3}(1 - \frac{1}{4^D})N^2.$$

When D is large, it needs at most $(2/3)N^2$ cycles to process an image frame. In addition, the schedule also shows that both row filter and column filter achieve 100% hardware utilization and that the data flow is kept regularly. The total size of two storage units for an $N\times N$ image is as follows.

Total Size = storage unit 1 + storage unit 2

+ temporary buffer

$$\approx 2(K-2)\times N + (2^D - 1)\times(2K-4) + 4^{D-1} \text{ Units.}$$

When N is large enough, such as 512 or 1024, the total storage size need only $2N(K-2)$ units.

Based on the proposed 2-D IDWT architecture, we compare, in Table I, the characteristics of various architectures with respect to the number of multipliers, the number of adders, storage size, storage type, and computing time. Let the filter tap be K and the image size be $N\times N$, it can be seen that the tree-block architecture we proposed is the best one in computation time compared with other architectures. Besides, both column filter and row filter achieve the 100% hardware utilization.

7.RESULTS

In this paper, we proposed a tree-block scheduling scheme for 2-D separable IDWT. The advantage of this scheme is that the required buffers stored the temporary subband can be greatly reduced because wavelet coefficients are processed block by block and that the scheduling make the data flow tight and regular to meet high speed and low complexity. Based on this technique, an efficient VLSI architecture have been designed and implemented. To minimize the hardware cost, the architecture utilized two efficient filter structures with reduced width multipliers to accomplish the computation of all octave levels. Moreover, each filter in the architecture is designed regularly and modularly, so it is easily scalable for different filter lengths and different IDWT octave levels by adding some extra modules. Additionally, both column and row filters achieve the 100% resource utilization. Due to its small storage size, regularity, and high performance, the architecture is suited to time-critical applications, such as MPEG-4 and JPEG-2000.

Architectures	Multipliers	Adders	Storage Size	Storage Type	Computing Time	Control unit
Direct	K	K	N^2	RAM	$4N^2$	Simple
[8]	$3\frac{1}{2}K$	$7(\frac{K}{2}-1)+4$	NK	RAM	N^2	Complex
[11]	$4K$	$4K$-4	$NK/2$	RAM	N^2	Simple
Tree-Block	$4K$	$4K$-2	$2N(K$-2)	Register	$\frac{2}{3}N^2$	Simple

Table I. The comparison of various IDWT architectures.

8. CONCLUSION

In this paper, we proposed a tree-block scheduling scheme for 2-D separable IDWT. The advantage of this scheme is that the required buffers stored the temporary subband can be greatly reduced because wavelet coefficients are processed block by block and that the scheduling make the data flow tight and regular to meet high speed and low complexity. Based on this technique, an efficient VLSI architecture have been designed and implemented. To minimize the hardware cost, the architecture utilized two efficient filter structures with reduced width multipliers to accomplish the computation of all octave levels. Moreover, each filter in the architecture is designed regularly and modularly, so it is easily scalable for different filter lengths and different IDWT octave levels by adding some extra modules. Additionally, both column and row filters achieve the 100% resource utilization. Due to its small storage size, regularity, and high performance, the architecture is suited to time-critical applications, such as MPEG-4 and JPEG-2000.

REFERENCE

[1] A. S. Lewis and G. Knowles, "VLSI Architecture for 2-D Daubechies Wavelet Transform Without Multipliers". *Electronics Letters*, pp. 171-173, Jan. 1991.

[2] M. Vishwanath, R. M. Owens and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits and Systems*, vol. 42, no. 5, pp. 305-316, 1995.

[3] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to map-

pings on SIMD array computers," *IEEE Trans. Signal Processing*, vol. 43, no. 5, pp. 759-771, 1995.

[4] H. Y. H. Chuang and L. Chen, "VLSI architecture for the fast 2-D discrete orthonormal wavelet transform," *Journal of VLSI Signal Processing*, vol. 10, pp.225-236, 1995.

[5] Jijun Chen and M. A. Bayoumi, "A Scalable Systolic Array Architecture for 2-D Discrete Wavelet Transform", *IEEE VLSI Signal Processing*, pp. 303-312, 1995.

[6] A. Grzeszczak, M. K. Mandal, S. Panchanathan, "VLSI implementation of discrete wavelet transform", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 421-433, Dec. 1996.

[7] M. Vishwanath and R. M. Owens, "A Common Architecture for the DWT and IDWT". *Proceedings of International Conference on Application Specific Systems, Architectures and Processors*, pp. 193–198. 1996.

[8] X. Chen, T. Zhou, Z. Qianlin, and M. Hao. "2-D DWT/IDWT Processor Design for Image Coding". *2nd International Conference on ASIC*, pp. 111–114, 1996.

[9] M. H. Sheu, M. D. Shieh and S. F. Cheng. "A Unified VLSI Architecture for Decomposition and Synthesis of Discrete Wavelet Transform". *IEEE 39th Midwest symposium on Circuits and Systems*, pp. 113–116, 1996.

[10] J. T. Kim, Y. H. Lee, T. Isshiki, and H. Kunieda, "Scalable VLSI Architectures for Lattice Structure-Based Discrete Wavelet Transform". *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, pp. 1031-1043, Aug. 1998.

[11] T. Acharya and P. Y. Chen. "VLSI Implementation of a DWT Architecture". *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 272–275, 1998.

[12] Y. Chu and S. J. Chen, "Efficient VLSI Architecture for 2-D Inverse Discrete Wavelet Transforms". *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 524-527, July 1999.

[13] S. A. Martucci, I. Sodagar, T. Chiang, and Y. Q. Zhang, "A Zerotree Wavelet Video Coder". *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 109-118. Feb. 1997.

[14] C. Chakrabarti, and C. Mumford, "Efficient Realizations of Encoders and Decoders Based on the 2-D Discrete Wavelet Transform", *IEEE Transactions on VLSI Systems*, pp. 289-298. Sep. 1999.

[15] J. M. Jou and S. R. Kuang, "Low-error Design of a Fixed-width two's Complement multiplier for DSP applications" *IEEE Transactions on Circuits & Systems* Part II., pp.836-841. Sept. 1999.

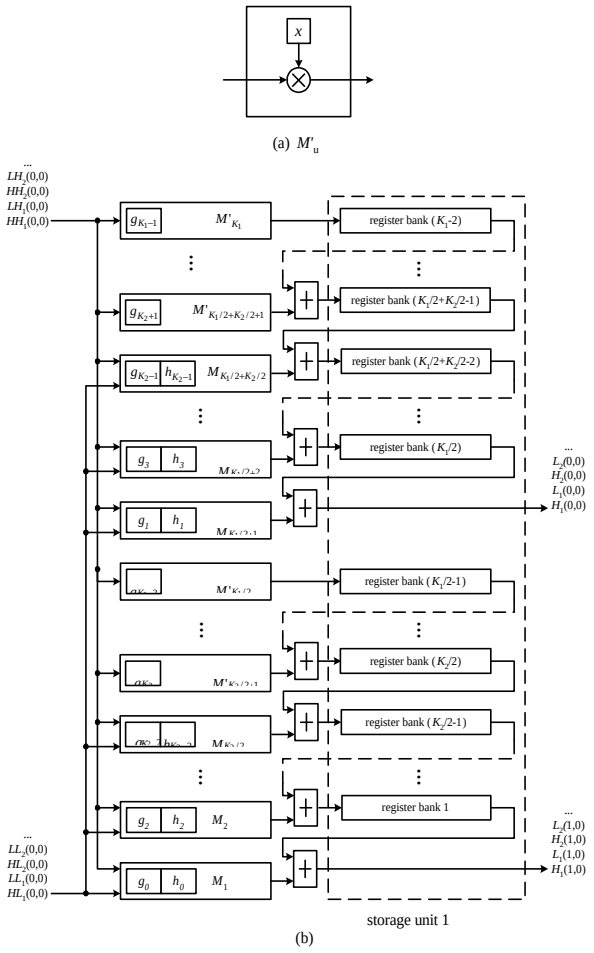


Fig. 12. (a) The structure of $M'(u)$ and (b) The block diagram of the column filter for scalable Architecture.

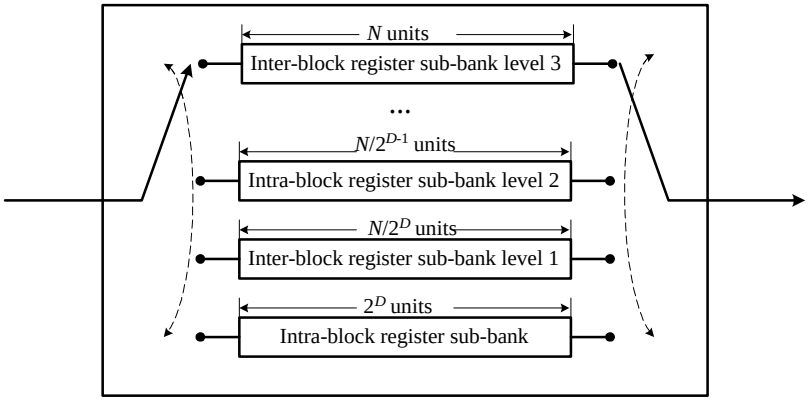


Fig. 13. The routing networks of the registers in storage unit 1 for scalable Architecture.

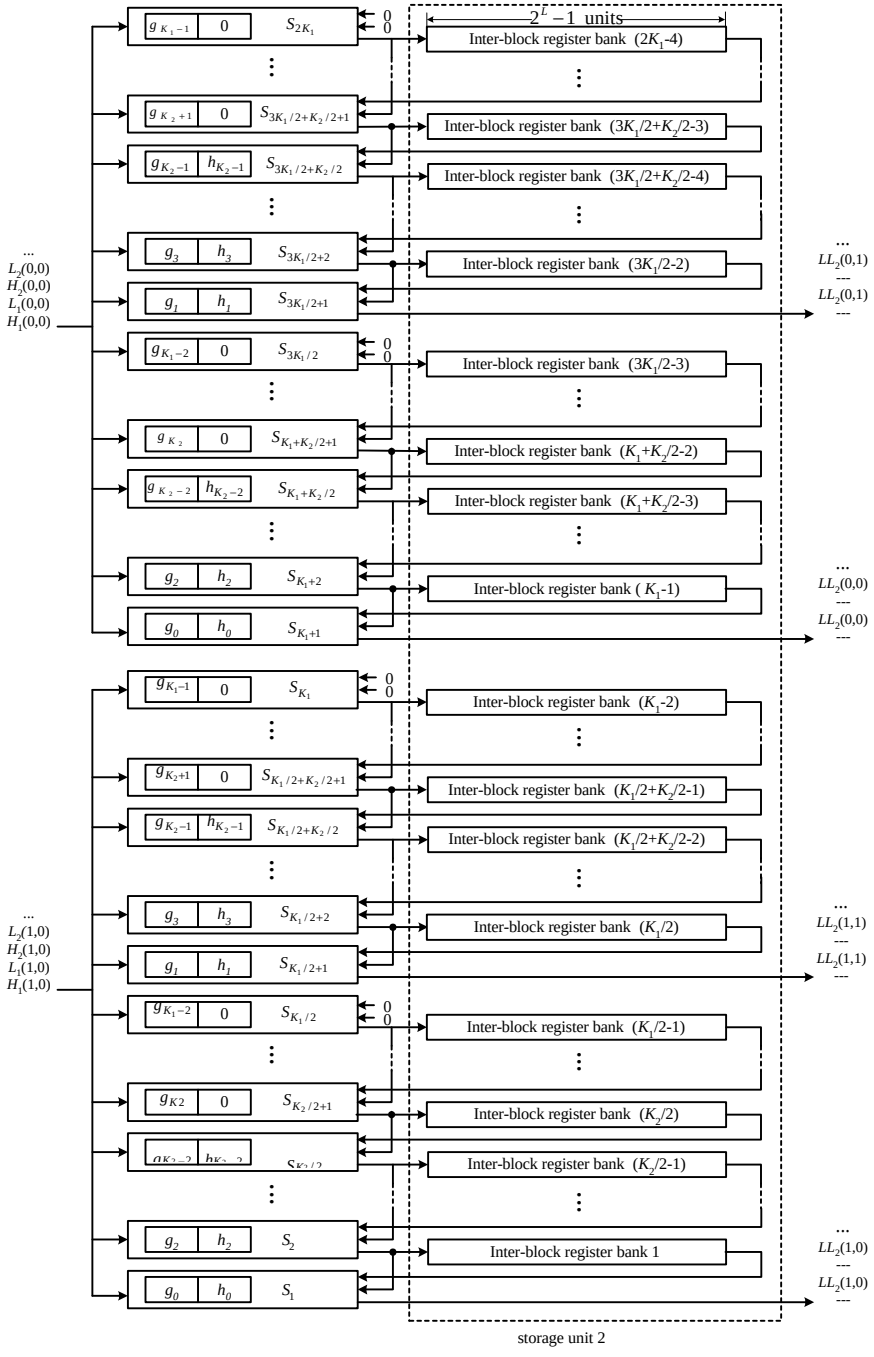


Fig. 14. The block diagram of the row filter for scalable Architecture.