VLSI Design, 2002 Vol. 15 (2), pp. 499–506

Taylor & Francis
Taylor & Francis Group

Hardware Architectures for the Orthogonal and Biorthogonal Wavelet Transform

G. KNOWLES*

School of Informatics and Engineering, Flinders University of South Australia, GPO Box 2100, Adelaide 5001, Australia

(Received 5 July 2001; Revised 16 October 2001)

In this note, optimal hardware architectures for the orthogonal and biorthogonal wavelet transforms are presented. The approach used here is not the standard lifting method, but takes advantage of the symmetries inherent in the coefficients of the transforms and the decimation/interpolation operators. The design is based on a highly optimized datapath, which seamlessly integrates both orthogonal and biorthogonal transforms, data extension at the edges and the forward and inverse transforms. The datapath design could be further optimized for speed or low power. The datapath is controlled by a small fast control unit which is hard programmed according to the wavelet or wavelets required by the application.

Example circuits are given, including one for the Daubechies 9-7 wavelet which requires only 2.5 multipliers for each input data value, when the equivalent for lifting is 3.

Keywords: Orthogonal wavelet transform; Biorthogonal wavelet transform; VLSI; JPEG2000; MPEG4

INTRODUCTION

The use of the wavelet transform in image and video processing is well known [1,2]. One of its main advantages is that there are very efficient software implementations, such as lifting [3]. Lifting has also been used for hardware implementations, however, it is well known that it is optimal only in the case when filter lengths are large [3]. Here we shall show that there are also highly efficient hardware architectures for the orthogonal and biorthogonal wavelet transform. Architectures for the wavelet transform designed to minimize the number of low and high pass convolvers have been given, in the one dimensional case in Ref. [4], and an extended version in Ref. [5]. In Refs. [5,6], architectures for the multi-octave two-dimensional wavelet transform using only three convolvers are presented. However, the approach of Ref. [5] reduces the number of convolvers needed, but has major disadvantages for practical implementation. Namely, the forward and inverse transforms use different architectures, so doubling the circuit area, and the problem of boundary conditions for finite data sets has not been considered.

The basis of our approach is a new convolver circuit that generates low and high pass values simultaneously in the forward transform, and combines low and high pass values in the inverse transform to produce even and odd data values. This is possible because of the symmetry of the orthogonal and biorthogonal wavelet coefficients and the decimation and interpolation by two operators. The results extend that of Ref. [7] by including boundary conditions, and biorthogonal wavelets, and are optimal in the sense of the number of multipliers and adders used. The architectures given here are more efficient than those from lifting, for example in the Daubechies 4 case, lifting requires [3, Table I] 5 multiplications and 4 additions per transformed (H, G) pair, the method here uses 4 multiplications and 4 additions. Similarly, in the case of the Daubechies 9-7 wavelet, lifting requires [3, Table I] 6 multiplications and 8 additions per transformed (H, G) pair, the method here uses 5 multiplications and 8 additions. Note that the designs given here are fully pipelined and so are suitable for high speed implementation. This will suit future video compression standards, such as Motion JPEG2000.

A prototype circuit of a 4 tap Daubechies 2D wavelet transform has been fabricated and tested successfully as part of a wavelet zero-tree video codec project [1,2,8,9], and a single circuit implementing all the wavelets used in the JPEG2000 and MPEG4 image compression standards has been designed and simulated in VHDL.

*E-mail: gknowles@infoeng.flinders.edu.au

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/1065514021000012110

TABLE I Forward wavelet transform—row convolution

in	out0	AS0	AS1	AS2	AS3	out1
d_0		0	$(a+b)$	$(c+d)$	0	
d_1		a	$-b$	c	d	
d_2	G_0	a	b	c	$-d$	H_0
d_3		a	$-b$	c	d	
d_4	G_1	a	b	c	$-d$	H_1
d_5		a	$-b$	c	d	
d_6	G_2	a	b	c	$-d$	H_2
d_7		0	$-(b-a)$	$(c-d)$	0	
n_0	G_3	0	$(a+b)$	$(c+d)$	0	H_3
⋮	⋮	⋮	⋮	⋮	⋮	⋮

ORTHOGONAL WAVELETS

As a example to illustrate the concepts introduced here we firstly consider a circuit for the Daubechies 4 tap orthogonal wavelet from Refs. [7,10], with (sign less) coefficients a, b, c, d. It was shown in there that the multiplications between the filter coefficients and the input data for this wavelet can be performed with a small number of shifts and adds. The filter equations are, for $1 < i < (n/2) - 1$:

$$H_i = ad_{2i-1} + bd_{2i} + cd_{2i+1} - dd_{2i+2} \tag{1}$$

$$G_i = dd_{2i-1} + cd_{2i} - bd_{2i+1} + ad_{2i+2} \tag{2}$$

In order to transform a line of data of length n (n even) we need an extra data value at the start and end of a line. Here we use an even symmetric extension at the ends of the line, however, in hardware it is much simpler to change the filters at the start and end of the line than to extend the input data. Accordingly, we define the forward start and end filters,

$$H_0 = (a+b)d_0 + cd_1 - dd_2 \tag{3}$$

$$G_0 = (c+d)d_0 - bd_1 + ad_2 \tag{4}$$

$$H_{\frac{n}{2}-1} = ad_{n-3} + bd_{n-2} + (c-d)d_{n-1} \tag{5}$$

$$G_{\frac{n}{2}-1} = dd_{n-3} + cd_{n-2} - (b-a)d_{n-1} \tag{6}$$

The row convolver circuit is shown in Fig. 1, the operation of the convolver in the forward transform on a row of length 8 is shown in Table I, and in the inverse transform in Table II. In the MULTIPLY block (not shown) all the filter coefficients are multiplied by the input data value and are then combined in the adder–subtracter units (AS0–AS3). The ZERO block selectively zeros or passes the input value and the DEL block delays its input value by one cycle. We will explain the sequence of operations in Table I and then indicate how these are mapped to the datapath (Fig. 1). Each line in the table represents one clock cycle. For example, in Table I, line 4 the input data value d_3 is multiplied by the filter coefficients a, b, c, d in the MULTIPLY unit, so at the end of the first cycle the output of AS0 is ad_3, the first term of H_2. Simultaneously, in AS3 dd_3 is the output. On the next cycle ad_3 from AS0

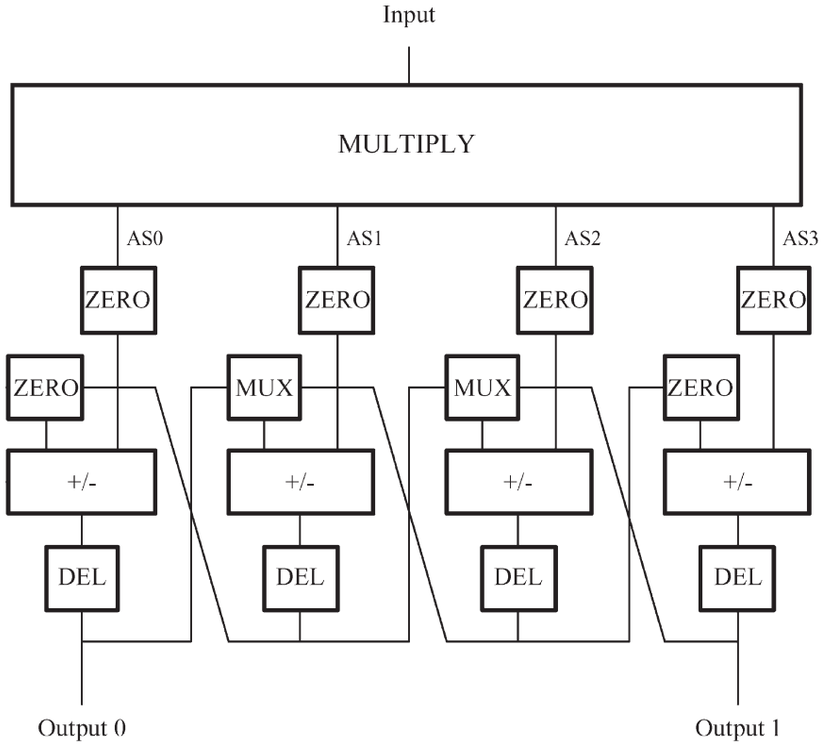

FIGURE 1 Daubecheis4 wavelet—Pipelined Adder–Subtracters.

TABLE II Inverse wavelet transform—row convolution

in	out0	AS0	AS1	AS2	AS3	out1
H_0		c	$b-a$	0	$-d$	
G_0	$d_0/4$	$c-d$	$-b$	a	0	
H_1		c	b	a	$-d$	
G_1	d_2	c	$-b$	a	$-d$	d_1
H_2		c	b	a	$-d$	
G_2	d_4	c	$-b$	a	d	d_3
H_3		$(c+d)$	b	a	0	
G_3	d_6	c	$-(a+b)$	0	d	d_5
$\overline{H}_0$		c	$b-a$	0	$-d$	
$\overline{G}_0$	$d_1/4$	$c-d$	$-b$	a	0	$d_7/4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

is added to bd_3 in AS1 and dd_3 from AS3 is added to cd_4 in AS3. In this way the low pass value H_2 is evaluated in the sequence: AS0 (ad_3), AS1 ($ad_3 + bd_4$), AS2 ($ad_3 + bd_4 + cd_5$), AS3 ($ad_3 + bd_4 + cd_5 - dd_6$). At the same time, the high pass value G_2 is evaluated in the sequence: AS3 (dd_3), AS2 ($dd_3 + cd_4$), AS1 ($dd_3 + cd_4 - bd_5$), AS0 ($dd_3 + cd_4 - bd_5 + ad_6$). The start filters G_0 and H_0 are evaluated in three cycles, for G_0 the order is: AS2 ($(c+d)d_0$), AS1 ($(c+d)d_0 - bd_1$), and AS0; ($(c+d)d_0 - bd_1 + ad_2$), and for H_0: AS1 ($(a+b)d_0$), AS2 ($(a+b)d_0 + cd_1$), AS3 ($(a+b)d_0 + cd_1 - dd_2$). The end filters are evaluated in the same way. Note that since all the multiplications are performed in the same cycle, the computation of $(c+d)d_0$, and $(a+b)d_0$ requires only two extra adders.

The movement of data described above is mapped to the datapath in the following way. If we consider again the calculation of the transformed values G_2 and H_2, as before, the filter coefficients are simultaneously multiplied by the input data d_3 in the MULTILPY block. Thus, ad_3, bd_3, cd_3, dd_3 are passed, in the same cycle, to the AS0, AS0, AS2 and AS3 units. Note that the ZERO block cancels the output from AS1 (AS2) so that the output of the adder in AS0 (AS3) is $ad_3(dd_3)$. On the next cycle, the control signal on MUX1 is switched so that the multiplexor takes its input from the left side, and MUX2 takes it input from the right side. Consequently, at the end of the second cycle the output of AS1 is $ad_3 + bd_3$, and AS2 $dd_3 + cd_4$. For the third cycle, these multiplexors swap their inputs, so that MUX1 takes its input from the right side and MUX2 takes its input from the left side. So now AS1 sums $dd_3 + cd_4$ with $-bd_5$ and AS3 sums $ad_3 + bd_4$ with cd_5, and so on. By changing the control signals on of the multiplexers on each cycle, they act like a *swinging door* pushing the output of the AS units the right on one cycle and to the left on the next cycle, and so the low pass and high pass values (H, G) are calculated simultaneously. In this way a high pass (G) coefficient and a low pass coefficient (H) are output on every second cycle. The high pass value is delayed one cycle and the row convolver outputs one filtered value per cycle in the sequence, $H_0, G_0, H_1, G_1, \ldots$, the same order as in the lifting algorithm.

The inverse row transform is shown in Table II. The usual formula for calculating the inverse wavelet transform is to interpolate each of the H, G values by 2 and sum the results. However, in hardware it is simpler to combine these operations in two filters, one to give the even data values and another to give the odd data values. For the Daubechies 4 wavelet the even and odd reconstruction filters are, for $0 < i < n/2 - 2, i$ even,:

$$d_{2i+2} = -dH_i + aG_i + bH_{i+1} + cG_{i+1} \tag{7}$$

$$d_{2i+1} = cH_i - bG_i + aH_{i+1} + dG_{i+1} \tag{8}$$

The same hardware (Fig. 1) is used for both the forward and the inverse transforms. In this case the H, G filtered pairs from the inverse column convolver are multiplied by the respective coefficients in the MULTIPLY unit and input into the pipelined adder–subtracter unit. It can be shown that perfect reconstruction is obtained with the inverse start and end filters

$$d_0 = 4[(b-a)H_0 + (c-d)G_0] \tag{9}$$

$$d_{n-1} = 4[(c+d)H_{\frac{n}{2}-1} - (a+b)G_{\frac{n}{2}-1}] \tag{10}$$

The start reconstruction filter [Eq. (9)] is calculated in the order AS1 ($(b-a)H_0$) and AS0 ($(b-a)H_0 + (c-d)G_0$), and the end reconstruction filter in the order AS0 ($(c+d)H_3$) and AS1 ($(c+d)H_3 - (a+b)G_3$).

It can be seen that this architecture extends straightforwardly to any even length orthogonal wavelet, once the type of extension at the boundary has been decided, the corresponding start and end forward and reconstruction filters can be used, and a suitable MULTIPLY block designed.

A prototype of this design was simulated in VHDL and synthesized with SYNOPSIS. The gate count was 12 K gates, and easily achieved the target rate of 60 frames a second for an image size of 320×240 for a three octave 2D transform with YUV 4:1:1 video input. The chip was fabricated and successfully tested [9].

BIORTHOGONAL WAVELETS

In this section we consider the extension of the basic hardware architecture to linear phase, odd symmetric biorthogonal wavelets, such as the popular binomial [11] or spline wavelets [12]. These are the most commonly used biorthogonal wavelets, for example, the wavelets used in the standards JPEG2000 and MPEG4 are of this type.

As an example of this type of wavelet we have chosen the well known 9-7 biorthogonal wavelet [12] from the JPEG2000 image compression standard. Denote $h = (h_4, h_3, h_2, h_1, h_0, h_1, h_2, h_3, h_4)$ as the analysis low pass filter and $g = (g_3, g_2, g_1, g_0, g_1, g_2, g_3)$ as the analysis high pass filter. Then using symmetric extension at the edges,

e.g. Refs. [13,14], the forward start and end filters for the first row of the row convolution become:

$$H_0 = h_0d_0 + 2h_1d_1 + 2h_2d_2 + 2h_3d_3 + 2h_4d_4 \quad (11)$$

$$G_0 = g_1d_0 + (g_0 + g_2)d_1 + (g_1 + g_3)d_2 + g_2d_3 + g_3d_4 \quad (12)$$

$$H_1 = h_2d_0 + (h_1 + h_3)d_1 + (h_0 + h_4)d_2 + h_1d_3 + h_2d_4 + h_3d_5 + h_4d_6 \quad (13)$$

$$G_1 = g_3d_0 + g_2d_1 + g_1d_2 + g_0d_3 + g_1d_4 + g_2d_5 + g_3d_6 \quad (14)$$

$$G_{\frac{n}{2}-1} = 2g_3d_{n-4} + 2g_2d_{n-3} + 2g_1d_{n-2} + g_0d_{n-1} \quad (15)$$

$$H_{\frac{n}{2}-1} = h_4d_{n-6} + h_3d_{n-5} + (h_2 + h_4)d_{n-4} + (h_1 + h_3)d_{n-3} + (h_0 + h_2)d_{n-2} + h_1d_{n-1} \quad (16)$$

$$G_{\frac{n}{2}-2} = g_3d_{n-6} + g_2d_{n-5} + g_1d_{n-4} + g_0d_{n-3} + (g_1 + g_3)d_{n-2} + g_2d_{n-1} \quad (17)$$

$$H_{\frac{n}{2}-2} = h_4d_{n-8} + h_3d_{n-7} + h_2d_{n-6} + h_1d_{n-5} + h_0d_{n-4} + h_1d_{n-3} + (h_2 + h_4)d_{n-2} + h_3d_{n-1} \quad (18)$$

The even reconstruction filter is (0, h_3, $-g_2$, h_1, $-g_0$, h_1, $-g_2$, h_3) and the odd reconstruction filter is (0, $-h_4$, g_3, $-h_2$, g_1, $-h_0$, g_1, $-h_2$, g_3, $-h_4$). Perfect reconstruction at the start and end of a line will be obtained with the inverse start and end filters;

$$d_0 = -g_0H_0 + 2h_1G_0 - 2g_2H_1 + 2h_3G_1 \quad (19)$$

$$d_1 = g_1H_0 - (h_0 + h_2)G_0 + (g_1 + g_3)H_1 - (h_2 + h_4)G_1 + g_3H_2 - h_4G_2 \quad (20)$$

$$d_2 = -g_2H_0 + (h_1 + h_3)G_0 - g_0H_1 + h_1G_1 - g_2H_2 + h_3G_2 \quad (21)$$

$$d_3 = g_3H_0 - (h_2 + h_4)G_0 + g_1H_1 - h_0G_1 + g_1H_2 - h_2G_2 + g_3H_3 - h_4G_3 \quad (22)$$

$$d_{\frac{n}{2}-1} = -2h_4G_{\frac{n}{2}-3} + 2g_3H_{\frac{n}{2}-2} - 2h_2G_{\frac{n}{2}-2} + 2g_1H_{\frac{n}{2}-1} - h_0G_{\frac{n}{2}-1} \quad (23)$$

$$d_{\frac{n}{2}-2} = h_3G_{\frac{n}{2}-3} - g_2H_{\frac{n}{2}-2} + (h_1 + h_3)G_{\frac{n}{2}-2} - (g_0 + g_2)H_{\frac{n}{2}-1} + h_1G_{\frac{n}{2}-1} \quad (24)$$

$$d_{\frac{n}{2}-3} = -h_4G_{\frac{n}{2}-4} + g_3H_{\frac{n}{2}-3} - h_2G_{\frac{n}{2}-3} + g_1H_{\frac{n}{2}-2} - (h_0 + h_4)G_{\frac{n}{2}-2} + (g_1 + g_3)H_{\frac{n}{2}-1} - h_2G_{\frac{n}{2}-1} \quad (25)$$

$$d_{\frac{n}{2}-4} = h_3G_{\frac{n}{2}-4} - g_2H_{\frac{n}{2}-3} + h_1G_{\frac{n}{2}-3} - g_0H_{\frac{n}{2}-2} + h_1G_{\frac{n}{2}-2} - g_2H_{\frac{n}{2}-1} + h_3G_{\frac{n}{2}-1} \quad (26)$$

For this wavelet the hardware architecture is given in Figs. 2 and 3. The pipelined convolver is just Fig. 1 increased to 9 AS units to accommodate the 9 tap filter. The multiplier unit (Fig. 3) is where the multiplication of the input data with the filter coefficients is performed. It can be seen that only 5 multipliers and 8 adders are needed, the minimum for a 9 tap symmetric filter, in Fig. 2 the extra adders and muxs are used for the generation of the terms required by the start and end filters. The ZPS block either zeroes and passes or shifts by 2 the input value. For the forward row convolution the data flow through the pipelined convolver is shown in Table III and the inverse in Table IV. This moves through the array as before, in the forward transform the high pass filtered values move diagonally from right to left and the low pass diagonally from left to right. In the inverse transform, the odd data values move diagonally from right to left and the even values diagonally from left to right. For example, H_0 is computed in the order AS4 (h_0d_0), AS5 ($h_0d_0 + 2h_1d_1$), AS6 ($h_0d_0 + 2h_1d_1 + 2h_2d_2$), AS7 ($h_0d_0 + 2h_1d_1 + 2h_2d_2 + 2h_3d_3$), and finally AS8 ($h_0d_0 + 2h_1d_1 + 2h_2d_2 + 2h_3d_3 + 2h_4d_4$). Similarly, G_0 is computed via AS5 (g_1d_0), AS4 ($g_1d_0 + (g_0 + g_2)d_1$), AS3 ($g_1d_0 + (g_0 + g_2)d_1 + (g_1 + g_3)d_2$), AS2 ($g_1d_0 + (g_0 + g_2)d_1 + (g_1 + g_3)d_2 + g_2d_3$), AS1 ($g_1d_0 + (g_0 + g_2)d_1 + (g_1 + g_3)d_2 + g_2d_3 + g_3d_4$), then the zero in AS0 passes this result to the output.

Apart from the start and end filters (which only affect the first and last five values of the line) the filter coefficients in each of the AS cells switch between two values on each cycle. We can see from this that the control unit will be small.

An important example of an odd symmetric biorthogonal filter has been specified in the MPEG4 image compression standard. In this case the high pass analysis filter is $2^{13/2}(3, 6, -16, -38, 90, -38, -16, 6, 3)$ and the low pass synthesis filter is $2^{13/2}(32, 64, 32)$. Although now the high pass is longer than the low pass, we can still use the same architecture by swapping the role of h and g.

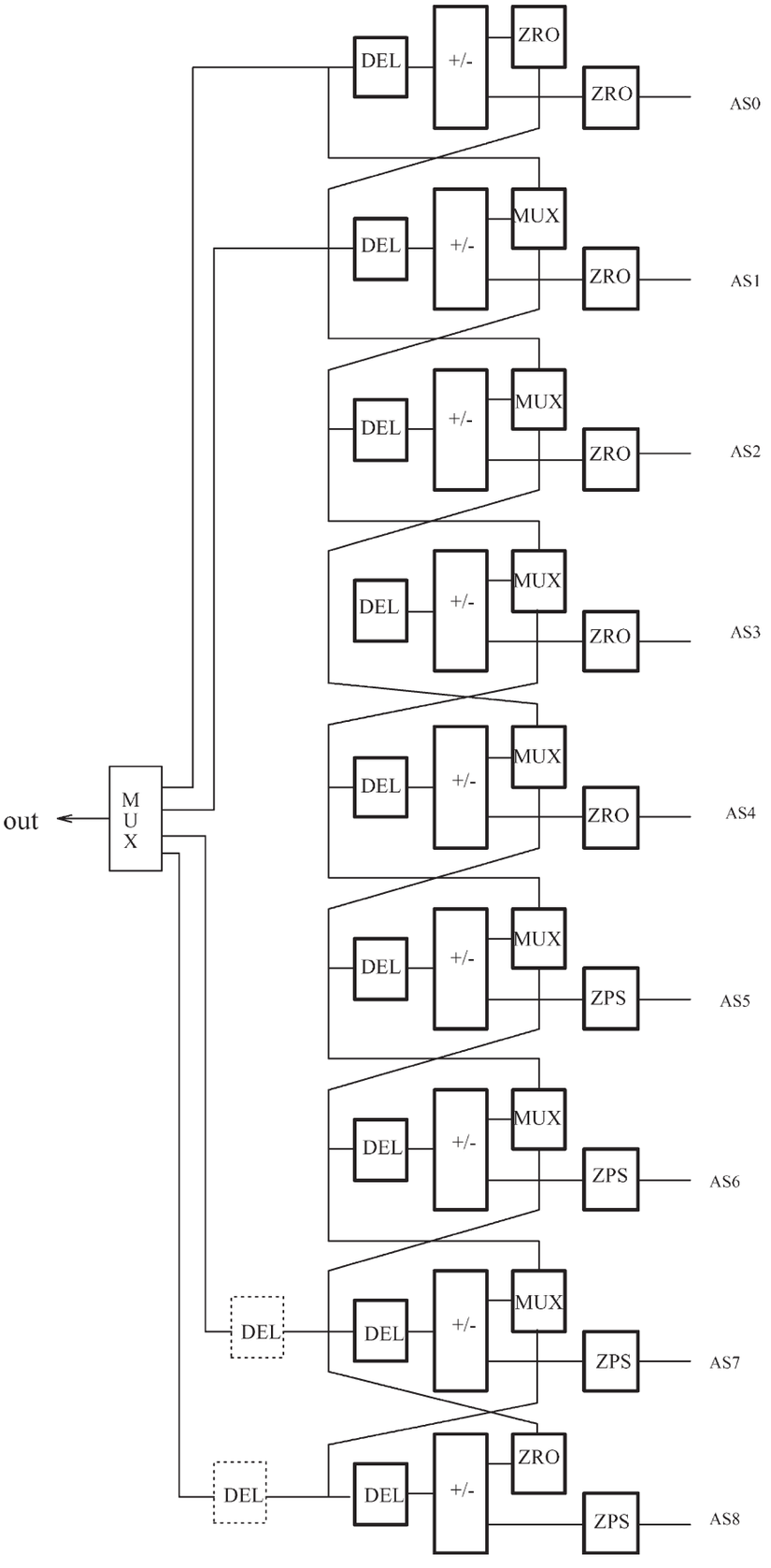

FIGURE 2 Biorthogonal 7-9 wavelet—Pipelined Adder–Subtracters.

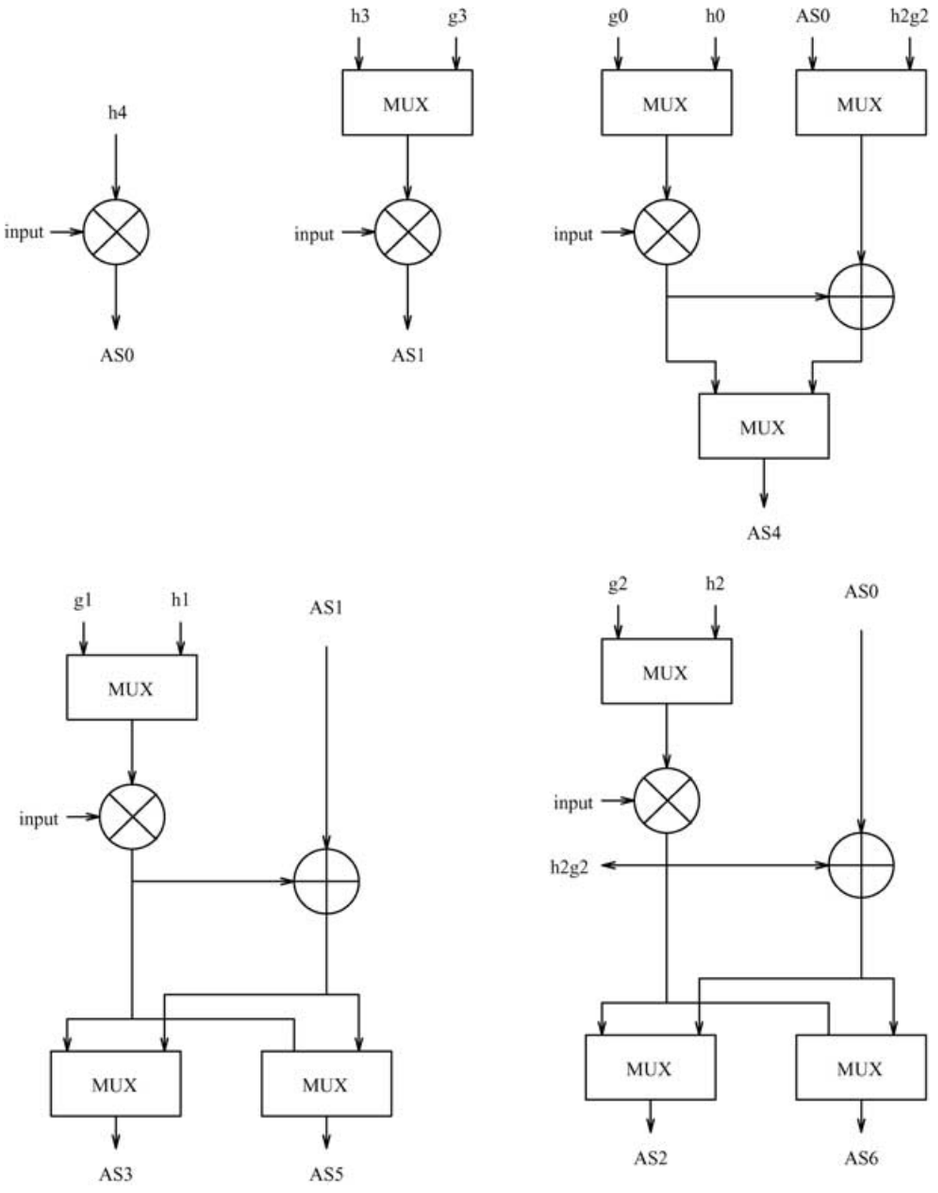

FIGURE 3 Biorthogonal 7-9 wavelet—Multiply Circuit.

In the forward transform, the low pass values appear from AS1, and the high pass from AS8. By changing the settings of the output mux, and by using the optional delay on AS8 we can still write the values to memory consistent with the order output from the lifting algorithm. In the inverse transform the even data values are output from AS0, and the odd from AS7.

To illustrate these designs a one dimensional biorthogonal wavelet transform using the MPEG4 wavelet and the LeGall 5-3 and Daubechies 9-7 wavelets from JPEG2000, were designed in VHDL, and fully simulated. An important point for the efficient operation of this type of design is the size of the control unit. In this case it was very small, using only a two hundred gates.

Finally, note that for the case of odd symmetry, the same architectures applies for an anti-symmetric extension of the data at the edges. Only in the case that h and g are odd symmetric and have the same number of coefficients to use this type of architecture it would be necessary to zero extend the h filter, increasing its length by 2.

CONCLUSION

We have shown here that the criss-cross pipelined convolver unit of Figs. 1 and 2 provides an extremely efficient, regular architecture for the convolver part of the orthogonal and biorthogonal wavelet transform.

TABLE III Forward biorthogonal 7-9 wavelet—row convolution

in	out0	AS0	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	out1
d_0		h_4	0	h_2	0	h_0	g_1	0	g_3	0	
d_1		0	h_3	0	h_1+h_3	g_0+g_2	$2h_1$	g_2	0	0	
d_2		h_4	0	h_2	g_1+g_3	h_0+h_4	g_1	$2h_2$	g_3	0	
d_3		0	h_3	g_2	h_1	g_0	h_1	g_2	$2h_3$	0	
d_4		h_4	g_3	h_2	g_1	h_0	g_1	h_2	g_3	$2h_4$	H_0
d_5	G_0	0	h_3	g_2	h_1	g_0	h_1	g_2	h_3	0	
d_6		h_4	g_3	h_2	g_1	h_0	g_1	h_2	g_3	h_4	H_1
d_7	G_1	0	h_3	g_2	h_1	g_0	h_1	g_2	h_3	0	
d_8		0	g_3	h_2+h_4	g_1	h_0	g_1	h_2	$2g_3$	h_4	H_2
d_9	G_2	0	0	g_2	h_1+h_3	g_0	h_1	$2g_2$	h_3	0	
d_{10}		0	g_3	0	g_1+g_3	h_0+h_2	$2g_1$	h_2+h_4	0	h_4	H_3
d_{11}	G_3	0	0	g_2	0	g_0	h_1	0	h_3	0	
n_0		h_4	0	h_2	0	h_0	g_1	0	g_3	0	H_4
n_1	G_4	0	h_3	0	h_1+h_3	g_0+g_2	$2h_1$	g_2	0	0	
n_2		h_4	0	h_2	g_1+g_3	h_0+h_4	g_1	$2h_2$	g_3	0	H_5
n_3	G_5	0	h_3	g_2	h_1	g_0	h_1	g_2	$2h_3$	0	
n_4		h_4	g_3	h_2	g_1	h_0	g_1	h_2	g_3	$2h_4$	$\overline{H}_0$
n_5	$\overline{G}_1$	0	h_3	g_2	h_1	g_0	h_1	g_2	h_3	0	
n_6		h_4	g_3	h_2	g_1	h_0	g_1	h_2	g_3	h_4	$\overline{H}_1$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

It's principal advantage over the traditional convolver architectures is that it combines the decimation/interpolation by two of the filtered coefficients, the simultaneous generation of the high and low pass filtered values, the start and end filters and the symmetry between the forward and inverse transforms in a single simple architecture. A sample orthogonal wavelet transform chip has been fabricated and tested successfully [9], and a design of the JPEG2000 and MPEG4 biorthogonal wavelets has been implemented in VHDL and fully simulated.

In this work, we have not addressed possible memory architectures for the transform, since this problem has received considerable attention in the literature. The designs given here fit the memory architectures of Refs. [8,9] without modification, or they could use those of Ref. [15].

As mentioned in the introduction, the datapaths used in this note (e.g. Figs.1 and 2) can be optimized for low power or high speed by appropriately sizing the transistors in the datapath. This research is currently being undertaken.

Acknowledgements

The author would like to thank the referee for his invaluable comments.

TABLE IV Inverse biorthogonal 7-9 wavelet—row convolution

in	out0	AS0	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	out1
H_0		0	0	$-g_2$	0	$-g_0$	g_1	0	g_3	0	
G_0		0	h_3	0	h_1+h_3	$-(h_0+h_2)$	$2h_1$	$-(h_2+h_4)$	0	$-h_4$	
H_1		0	0	$-g_2$	g_1+g_3	$-g_0$	g_1	$-2g_2$	g_3	0	
G_1		0	h_3	$-(h_2+h_4)$	h_1	$-h_0$	h_1	$-h_2$	$2h_3$	$-h_4$	
H_2		0	g_3	$-g_2$	g_1	$-g_0$	g_1	$-g_2$	g_3	0	d_0
G_2	d_1	$-h_4$	h_3	$-h_2$	h_1	$-h_0$	h_1	$-h_2$	h_3	$-h_4$	
H_3		0	g_3	$-g_2$	g_1	$-g_0$	g_1	$-g_2$	g_3	0	d_2
G_3	d_3	$-h_4$	h_3	$-h_2$	h_1	$-h_0$	h_1	$-h_2$	h_3	$-2h_4$	
H_4		0	g_3	$-g_2$	g_1	$-g_0$	g_1	$-g_2$	$2g_3$	0	d_4
G_4	d_5	$-h_4$	0	$-h_2$	h_1+h_3	$-(h_0+h_4)$	h_1	$-2h2$	h_3	0	
H_5		0	g_3	0	g_1+g_3	$-(g_0+g_2)$	$2g_1$	$-g_2$	0	0	d_6
G_5	d_7	$-h_4$	0	$-h_2$	0	$-h_0$	h_1	0	h_3	0	
$\overline{H}_0$		0	0	$-g_2$	0	$-g_0$	g_1	0	g_3	0	d_8
$\overline{G}_0$	d_9	0	h_3	0	h_1+h_3	$-(h_0+h_2)$	$2h_1$	$-(h_2+h_4)$	0	$-h_4$	
$\overline{H}_1$		0	0	$-g_2$	g_1+g_3	$-g_0$	g_1	$-2g_2$	g_3	0	d_{10}
$\overline{G}_1$	d_{11}	0	h_3	$-(h_2+h_4)$	h_1	$-h_0$	h_1	$-h_2$	$2h_3$	$-h_4$	
$\overline{H}_2$		0	g_3	$-g_2$	g_1	$-g_0$	g_1	$-g_2$	g_3	0	n_0
$\overline{G}_2$	n_1	$-h_4$	h_3	$-h_2$	h_1	$-h_0$	h_1	$-h_2$	h_3	$-h_4$	
$\overline{H}_3$		0	g_3	$-g_2$	g_1	$-g_0$	g_1	$-g_2$	g_3	0	n_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

References

[1] Lewis, A.S. and Knowles, G. (1992) "Image compression using the 2D wavelet transform", *IEEE Trans. Image Processing* **1**, 244–250.

[2] Lewis, A.S. and Knowles, G. (1991) "A 64Kb/s video codec using the 2-D wavelet transform", IEEE Data Compression Conference (IEEE Computer press), pp. 197–201.

[3] Daubechies, I. and Sweldens, W. "Factoring wavelet transforms into lifting steps", to appear.

[4] Knowles, G. (1990) "VLSI Architecture for the discrete wavelet transform", *Electron. Lett.* **26**, 1184–1185.

[5] Parhi, K. and Nishtani, T. (1993) "VLSI architecture for the discrete wavelet transform", *IEEE Trans. VLSI* **1**, 191–202.

[6] Miyazaki, T. (1992) "A study on a two-dimensional wavelet transform structure for a video codec", *Proc. 1992 Natl Conf. IEICE Jpn*, 7–49.

[7] Lewis, A.S. and Knowles, G. (1991) "A VLSI architecture for the 2D Daubechies wavelet transform without multipliers", *Electron. Lett.* **27**, 171–172.

[8] Knowles, G. (1998) "A single chip wavelet transform zero-tree processor for video compression and decompression", *Proc. DATE*, 61–65.

[9] Knowles, G. "Device for Data Compression/Decompression using a discrete Wavelet Transform", *US Patent* No. 6118902.

[10] Daubechies, I. (1998) "Orthornormal bases of compactly supported wavelets", *Comm. Pure Appl. Math.* **XL1**, 909–996.

[11] Herley, C. and Vetterli, M. (1990) "Wavelets and filter banks: relationships and new results", *Proc. IEEE ICASSP*.

[12] Antonini, M., Barlaud, M., Mathieu, P. and Daubechies, I. (1992) "Image coding using the wavelet transform", *IEEE Trans. Image Processing* **1**, 205–220.

[13] Barnard, H., Weber, J. and Biemond, J. (1993) "Efficient signal extension for subband/wavelet decomposition of arbitrary length signals", *SPIE Vis. Commun. Image Processing* **2094**, 966–975.

[14] ISO/IEC, JTC 1/SC 29/WG 1, JPEG2000, Coding of Still Images.

[15] Lafruit, G., Nachtergaele, L., Bormans, J., Engels, M. and Bolsens, I. (1999) "Optimal memory organisation for scalable texture codecs in MPEG-4", *IEEE Trans. Circuits Syst. Video Technol.* **9**, 218–243.

Professor Greg Knowles has worked in various academic and industrial positions in Australia, USA, Europe and the UK, and is currently Professor of Computer Engineering at Flinders University in Adelaide, Australia. His research interests are in Embedded Systems, Systems on a Chip, Image Processing, and Image Compression.