

A Low-Latency Serial Architecture for the 1-D Discrete Wavelet Transform

Luca Breveglieri, Vincenzo Piuri, Marco Rona
Department of Electronics and Information, Politecnico di Milano
piazza L. da Vinci 32, 20133 Milano, Italy
phone +39-2-23993623, fax +39-2-23993411, email {brevegli,piuri}@elet.polimi.it

Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering, The University of Texas at Austin
Austin, TX 78712, USA
phone (512) 471-5923, fax (512) 471-5907, email e.swartzlander@compmail.com

Abstract

The 1-Dimensional Discrete Wavelet Transform is a powerful DSP technique for several application areas. Dedicated VLSI processors are often required by real-time applications due to the high amount of arithmetic operations; the circuit complexity is usually a relevant constraint for several low-cost applications as well as for complex systems having strong dimensional limits on the implementation (e.g., in aerospace applications). This paper presents the architectural design of a low-latency bit-serial 1-Dimensional Wavelet Transform Processor.

1. Introduction

The Wavelet Transforms (WT's) [13] are nowadays quite popular in several application areas, in particular in signal analysis, digital signal processing, audio and video signal coding, signal compression, network-based video applications (e.g. video-on-demand), and broadcast TV applications. In all these cases, we have very strict real-time constraints in order to achieve attractive results; therefore, dedicated VLSI processors are needed to reach the necessary performance.

In the literature, few dedicated architecture are available. In [6], a solution based on the Mallat transform algorithm [8] has been proposed by using only two filters (one high and one low-pass filter) for any octave number; several large multiplexers are required to recirculate the intermediate results during the subsequent iterations. In [7], the use of Daubechies filters has been considered to create much more compact units than in [6]; however, the efficiency of this architecture is strictly related to the specific characteristics of the Daubechies wavelet, being not efficient for the other wavelets. In [12], a folded structure and a digit-serial architecture have been proposed. The first one requires complex routing and interconnections and does not achieve complete hardware exploitation; conversely, it has a very low latency. The second architecture is slower, but has simple routing and interconnections while exploits all units in every clock cycle. Both of these structures have a minimal number of registers, but scaling with the filter size and the octave number is not easy and regular. In [2], some architectures have been shown effective: a systolic array, a parallel filter architecture, and a SIMD system. The first two solutions are optimal for VLSI implementation because they are very compact and regular; they have also a good performance. The SIMD architecture is very fast, but it has a high circuit complexity which is directly proportional to the input size. In [1,3,5,16], some other examples have been presented for the case of the bi-dimensional DWT, by extending the basic concepts proposed in the previous solutions.

All the above architectures operate on data in a bit parallel way. To reduce drastically the circuit complexity of the dedicated processor, in this paper we explore the use of the bit-serial data treatment, while performance still needs to be quite high. Besides, our architecture needs to be easily scalable with the filter size, the number of wavelet levels, and the number of bits of the input samples. This approach is

interesting for all applications in which the circuit complexity is very critical (e.g., in the case of a macro-cell for the core of a DSP processor), but reduced performance with respect to parallel solutions may be accepted. In particular, we focus our attention on the one-dimensional discrete wavelet transform.

In Section 2, after a brief description of the wavelet transform, we recall the Mallat algorithm for the one-dimensional DWT and introduce the overall architecture of our dedicated processor. Sections 3, 4, and 5 enter in the detailed description of the processor components, namely, the serial multiplier, the serial convolver for the first level of the DWT algorithm, and the convolver structure for the subsequent levels, respectively. In Section 6, the performance (clock cycle, latency, and throughput) and the circuit complexity of our architecture are evaluated and compared to some well-known bit-parallel structures.

2. The Discrete Wavelet Transform and the Processor Architecture

The Discrete Wavelet Transform (DWT) is a signal processing technique suitable for complex signal processing operations [4,8,13,14,15,17]. As Basis Functions, it uses dilated/contracted and translated versions of a single function with compact support, called the Prototype Wavelet, while the traditional transforms (e.g., Fourier Transform, Cosine Transform) adopt functions with infinite support. This allows for creating independent and variable windows, which are centered on every local information of the original signal (e.g., a spike) and avoid to spread such information over the whole frequency axis. Let us consider the sequence $u(i)$ of samples of the original signal, the DWT in the time b and the scale a is:

$$W(b,a) = \frac{1}{\sqrt{|a|}} \sum_{i=b}^{aL+b-1} x(i) h\left(\frac{i-b}{a}\right)$$

where L is the size of the support of the Prototype Wavelet $h(t)$, $h((i-b)/a)$ is the Prototype Wavelet in the scale a and the time b , N is the number of input samples, and m is the number of scales.

Algorithmically, the DWT can be considered as a multilevel filtering process. At each level, the input data are filtered by two related filters, one high-pass (G) and one low-pass (H), in order to produce two data streams. This two data streams are then sub-sampled by two (i.e., decimated), by eliminating the odd samples; therefore the total number of output samples is equal to the number of inputs. The high-pass output of each level is the transform output of such level; the low-pass output is sent to the subsequent level, where it is processed in the same way. And so on, for the required number m of levels. This algorithm, known as the Pyramid Algorithm [8], is shown in Fig. 1 for the case $m = 3$. If it has m levels, it is periodic in $N = 2^m$. We consider an input of N samples at the first level for each period of computation. Each first-level filter produces $N/2$ samples, for a total of N samples; at the second level, each filter produces $N/4$ samples, for a total of $N/2$ samples; generally, at level i , we have a total of $N/2^{i-1}$ samples.

To design our serial solution, we use this Pyramid Algorithm (see Fig. 1), which coincides also with the scheme of the overall processor architecture. At each level, there are couple of filters and decimators. The filter computes a convolution over a given number L of pair of input samples and weights, being L the size (or number of steps) of the filter itself. In the first-level high-pass filter (G) of size L with weights g_i , receiving the input samples $u(t)$ and produces the output samples $v(t)$, the input/output function is:

$$v(t) = g_0 u(t) + g_1 u(t-1) + \dots + g_{L-1} u(t-L+1)$$

The output sequence of this filter is decimated: one sample out of two is eliminated. The same occurs in the low-pass filter (H), whose output is $w(t)$. The sub-sampled output sequence of the first-level H filter is sent to the second level, where it is processed in the same way: $x(t)$ and $y(t)$ are the outputs of the G and H second-level filters, respectively. And so on, the computation is performed level after level.

Our idea is to realize the convolution and the decimation at the same time with a single structure. In other words, instead of computing the complete succession $v(0), v(1), v(2), \dots$ and then decimating it, we directly compute the succession $v(0), v(2), v(4), \dots$, by avoiding the computation of the odd samples $v(1), v(3), v(5), \dots$. Practically, we realize a convolver which doesn't compute the odd outputs. Let us consider, for example, $L = 3$ and $m = 3$. A period of the computation is given by:

Level	High-pass filter (G)	Low-pass filter (H)
I	$v(0) = g_0u(0) + g_1u(-1) + g_2u(-2)$ $v(2) = g_0u(2) + g_1u(1) + g_2u(0)$ $v(4) = g_0u(4) + g_1u(3) + g_2u(2)$ $v(6) = g_0u(6) + g_1u(5) + g_2u(4)$	$w(0) = h_0u(0) + h_1u(-1) + h_2u(-2)$ $w(2) = h_0u(2) + h_1u(1) + h_2u(0)$ $w(4) = h_0u(4) + h_1u(3) + h_2u(2)$ $w(6) = h_0u(6) + h_1u(5) + h_2u(4)$
II	$x(0) = g_0w(0) + g_1w(-2) + g_2w(-4)$ $x(4) = g_0w(4) + g_1w(2) + g_2w(0)$	$y(0) = h_0w(0) + h_1w(-2) + h_2w(-4)$ $y(2) = h_0w(4) + h_1w(2) + h_2w(0)$
III	$z(0) = g_0y(0) + g_1y(-4) + g_2y(-8)$ $z(8) = g_0y(8) + g_1y(4) + g_2y(0)$	$z'(0) = h_0y(0) + h_1y(-4) + h_2y(-8)$ $z'(8) = h_0y(8) + h_1y(4) + h_2y(0)$

Our architecture is thus composed by two convolvers for each level, one for each filter. Each convolver is composed by a cascade of multipliers and adders [10]. Since we are looking for a serial architecture, the external inputs of multipliers and adders need to be serial as well as the internal operation of these arithmetic units. The convolver weights can be presented in parallel to the arithmetic units, since they are constants which can be pre-loaded in the convolvers (e.g. in a ROM memory). In the subsequent Sections, the detailed structure of each component is discussed.

3. The Serial Multiplier

Our multiplier has been derived from [11], by adapting it to our particular requirements. Let us consider two w -bit binary numbers, x and y : $(x_{w-1} x_{w-2} \dots x_0)$ and $(y_{w-1} y_{w-2} \dots y_0)$. The product p of these two numbers is $2w$ bits long: $(p_{2w-1} p_{2w-2} \dots p_0)$. Let us assume that the bits of the factor x arrive in serial mode, starting from the less significant bit, while the other factor is available in parallel mode.

The structure of our serial multiplier is shown in Fig. 2. The FA_{*i*} blocks of Fig. 2 are standard full-adders, while the E_{*i*} and the D_{*i*} blocks are DT latches. The bits of the x factor enter the multiplier from the left side, one at every clock cycle, starting from the least-significant one; the product bits come out from the right side, still in serial mode, starting from the least-significant one. The D_{*i*} latches store the x bits to be used in the subsequent multiplication operations, while the E_{*i*} latches hold the addition carries. The three-input AND gates realize the multiplications $x_i y_j$. The control signals s_i enable the generation of the partial product or force it to zero so as to allow the correct propagation of the carries stored in the E_{*i*} latches towards the multiplier output by considering or avoiding the addition of a new partial product. Similarly, the control signals z_i determine the output and carry propagation in the adder cascade by including the carries at the proper positions. A Control Unit generates all the required control signals.

Due to the particular use of this multiplier, between two subsequent input factors (x_i and x_{i+1}), there is always an interval of w clock cycles, during which no bit enters the multiplier itself. In other words, between the first bit of one serial factor and the first bit of the subsequent serial factor, there are $2w$ clock cycles: this is the time required to complete a serial multiplication. In general, with x and y having w bits, our serial multiplier is composed of $w-1$ full-adders, $2(w-1)$ DT latches, w three-inputs AND gates, and $w-1$ two-inputs AND gates. It needs $2w-1$ control signals and realizes the multiplication in $2w$ clock cycles. Between two subsequent products, there are $2w$ clock cycles. Thus, the latency and the throughput are $2w\tau$ and $1/2w\tau$, respectively, being τ the system clock cycle time.

4. The First-Level Serial Convolver

Let us consider the G convolver at the first level, with samples $u(t)$ as the input and samples $v(t)$ as the output. Let us assume $w = 8$ and $L = 3$. The generic convolution product is given by:

$$v(t) = g_0u(t) + g_1u(t-1) + g_2u(t-2)$$

The overall structure of our convolver, based on our serial multiplier, is shown in Fig. 3. It uses three 8×8 serial multipliers of the type presented in Section 3, one for each convolution component $g_i u(i)$. The

multipliers A, B, and C realize the product $g_2u(t-2)$, $g_1u(t-1)$, and $g_0u(t)$, respectively. The input samples enter the convolver serially, starting from the least-significant one, without any gap between one sample and the subsequent one. As an example, we consider $t = 2$; the corresponding convolution is:

$$v(2) = g_0u(2) + g_1u(1) + g_2u(0)$$

To understand the serial convolver operation, we analyze the computation during the sequence of time steps defined by the system clock cycles, as shown in Table 1. To such a purpose, we take into account the operation of our serial multiplier, as described in the previous Section. In the time interval 0-7, the bits of the sample $u(0)$ arrive to the convolver and the multiplication $g_2u(0)$ is performed by A. This multiplication terminates at the time step 15. In the interval 8-15, the bits of the sample $u(1)$ arrive and B computes the product $g_1u(1)$, which is completed at the time step 23. During such time interval, the multiplication $g_2u(0)$ continues in A. In the interval 16-23, the bits of the sample $u(2)$ arrive; the product $g_0u(2)$ is computed in C up to the time step 31. In this time interval, the multiplication $g_1u(1)$ proceeds in B. The multiplier A is not used from the time step 16 onwards, B from the time step 24, and so on. However, in our particular application, after having computed $v(2) = g_0u(2) + g_1u(1) + g_2u(0)$, we have to evaluate $v(4) = g_0u(4) + g_1u(3) + g_2u(2)$. The sample $u(2)$ is needed to compute both $v(2)$ and $v(4)$, and it is available from the time step 16 onwards. Just at this time, the multiplier A becomes free. Thus, the computation of $g_2u(2)$ for $v(4)$ can be started in A. This computation terminates at the time step 31. At the time step 24, the first bit of $u(3)$ arrives, the multiplier B is free, and the computation of $g_1u(3)$ starts, terminating at the time step 39. At the time step 32, the first bit of $u(4)$ enters, the multiplier C is free, and the computation of $g_0u(4)$ starts and is completed by the time step 47. All the above steps are repeated iteratively. Our convolver has a pipelined structure to avoid sample storing for subsequent computations.

The 16-bit multiplier outputs must be added together and, then, must be truncated to 8 bits by discarding the 8 least-significant bits. The 8 most-significant bits generated by A are available starting from the time step 8, while the corresponding bits produced by B are ready from the time step 16. In order to synchronize these two bit streams, the output of A is sent to the 8 bits serial-input/serial-output (SISO) shift-register SR. At the time step 16, the 8-th bit of $g_2u(0)$ comes out from SR, while the 8-th bit of $g_1u(1)$ comes out from B. These two bits are sent to a serial adder in order to produce the first bit of the sum $g_2u(0) + g_1u(1)$. This sum terminates at the time step 23 and is accumulated in the 8-bits SISO shift-register ACC. The last carry of this sum is ignored, generating an overflow when it is equal to one. The 8 bits of the sum $g_2u(0) + g_1u(1)$ are then stored in ACC. These bits must be added to the 8 most-significant bits of the product $g_0u(2)$, which comes out from C starting from the time step 24 onwards. The bits of $g_2u(0) + g_1u(1)$, produced by the serial adder starting from the time step 16, are available at the output of ACC exactly at time 24. The sum between these bits and the 8 most-significant bits of $g_0u(2)$ is computed in the time interval 24-31. This sum can be computed by the same serial adder which performed the previous sum. Therefore, at the two inputs of this serial adder, there are two multiplexers: the first multiplexer selects the first input either from SR (during the interval 16-23) or from ACC (during the interval 24-31), while the second one selects the second input either from B or C (in the same intervals). This second sum is sent both to ACC and the convolver output. At the time step 32, the 8-th bit of the product $g_2u(2)$ of $v(4)$ comes out from SR, while the 8-th bit of the product $g_1u(3)$ of $v(4)$ is generated by B. With the adder above, we can sum also these bits. And so on, for all convolutions.

The first and the last bits of $v(4)$ are produced by the convolver at the time steps 40 and 47, respectively. The first and the last bits of $v(2)$ have been produced at the time steps 24 and 31, respectively. In the interval 32-39, no bit is produced by the convolver. Therefore, the output rate of samples is half of the input rate. In other words, our convolver realizes not only the convolution, but also the decimation, since only the output samples $v(0)$, $v(2)$, $v(4)$, ..., are produced for the input samples $u(0)$, $u(1)$, $u(2)$, $u(3)$, ..., while the samples $v(1)$, $v(3)$, $v(5)$, ..., are omitted.

The resulting architecture is very regular and compact. Since more convolutions are overlapped, we achieve a complete hardware exploitation and a relevant storage minimization for samples and partial result (only two 8-bits shift-registers). On the other hand, pipelining guarantees a good throughput.

In order to provide the correct data flowing, some multiplexers and the related control signals are required. In particular, a two-input multiplexer is used to select the a input for the full-adder FA between the outputs of the multipliers B and C. Similarly, another two-inputs multiplexer is used to choose the b input of FA between the outputs of the two shift-registers SR and ACC. The operation of these multiplexers is identical: therefore, we can drive both of them with the control signal sel1. At the beginning of a new addition in FA, the carry-in input is forced to zero by the control signal sel2. Between two subsequent transform samples, i.e., when the serial adder is adding the output of SR to the output of B, we must force to zero the convolver output by using the control signal sel1. The three serial multipliers need also properly-synchronized clock signals. The multiplier presented in Section 3 needs 15 clock pulses to complete its operation. Since, A and C work synchronously, the Control Unit must produce only 15 pulses for the multiplier B, while additional 15 pulses are required to complete the multiplications A and C, contemporaneously.

Our serial convolver is composed by 3 8×8 serial multipliers, 1 full-adder, 2 8-bit SISO shift-registers, 1 DT latch, 2 2-inputs AND gates, and 2 two-inputs multiplexers. The latency is 32τ and the throughput is $1/16\tau$. In the general case, the convolver size is L , while the samples and the weights are w bits long. When $L = 3$, there is one input sample in common between two subsequent computations ($v(i)$ and $v(i+1)$). For any L , there are $L-2$ input samples in common. Thus, we must realize $L-2$ additions, contemporaneously. It can be easily shown that such a generalized serial convolver is composed of $L \times w$ serial multipliers, $L-2$ full-adders, $L-2$ DT latches, $L-1$ two-inputs AND gates, $L-1$ w -bit SISO shift-registers, and $2(L-2)$ two-inputs multiplexers. The latency is $w(L+1)\tau$, while the throughput is $1/2w\tau$.

5. The Higher-Level Serial Convolver

The serial convolver scheme, which has been used for the high- and low-pass filtering at the first level, needs a continuous stream of input samples, without any gap between them. Starting from the second level, the continuous data flow it is not possible due to the operation of the first level and to its output characteristics. At the input of the second level, there is a gap between two subsequent samples during which no bit enters the convolvers at this level; the gap has the same length of a sample (i.e., w bits). At the third level, the gap is $3w$ bit long; and so on, for the subsequent levels.

In order to design the complete architecture, we can use the same type of convolver in each level, but the clock signals must be modified at every level. At the first level, we use the original clock signal, i.e., the one which scans the input bit stream. At the second level, we use a clock signal which coincides with the first level one when a transformed sample comes out from the first level, while it is forced to zero when no data come out from the first level. Then, we operate in the same way for the other levels. In other words, we freeze the clock signal in each level when no data come out from the previous level.

As an example, let us consider the second-level G convolver with $w = 8$ and $L = 3$. The entire structure, as well as the control signals timing of this convolver are the ones presented in Section 4 (see Fig. 3): only the clock signal is different. In the time interval 8-15, this convolver receives the sample $w(0)$ and the product $g_2w(0)$ is computed in multiplier A. The last bit of the sample $w(0)$ arrives at the time step 15 and the first bit of the next sample, $w(2)$, arrives at the time step 24. In the interval 16-23, the second level clock signal is forced to zero and our convolver stops to work. In the interval 24-31, the sample $w(2)$ arrives, the clock signal restarts: $g_1w(2)$ is computed in B, while the computation of $g_2w(0)$ continues in A and will terminate at time step 31. Then, the clock signal is forced to zero in the interval 32-39. It is restarted in the interval 40-47 when $w(4)$ arrives and $g_0w(4)$ is computed in C. And so on. In the same way, the structure executes additions. The addition between SR and B outputs is performed in the interval 40-47, while the addition between ACC and C generates outputs in the interval 56-63. Therefore, the first output is available in the interval 56-63, the second sample in the interval 88-95, and so on. This gives us a latency of 56 clock cycles. A similar behavior occurs in each subsequent level.

We always use the same type of convolver in each level, with the same control signals. However, we have to modify the Control Unit in order to have $m-1$ additional control signals as clock signals for the higher levels. The hardware architecture is the same at each level: therefore, the area occupation of the whole architecture is $2m$ times the one of the basic convolver of Section 4. In general, in a m -level architecture, the latency and the throughput of the j -th level convolver are $(2^{j-1}L+1)w\tau$ and $1/2^jw\tau$, respectively. From the second level onwards, the hardware utilization is not complete since the higher-level convolvers are inactive during some cycles.

6. The Architectural Evaluation

The DWT architecture presented in this paper is evaluated from the point of view of the latency, the throughput and the circuit complexity. To provide figures of merit quite independent from the specific integration technology adopted for physical realization, the circuit complexity C is measured as number of MOS [9], due to the direct connection to the silicon area required by the implementation. The abstract value of C in area units can be obtained by multiplying this number by the average MOS area in the specific integration technology. Similarly, the latency L is measured as number of MOS levels along the longest path from the inputs to the outputs [9]; the abstract value of L in time units is the product of this number by the average MOS delay. The throughput T is the inverse of the time between two consecutive complete sets of transform outputs; the abstract value of T in outputs per time units is the product of this number by the inverse of the clock cycle time τ .

The actual physical values of the above figures of merit in a specific integration technology may be different from the abstract values since several physical effects must be taken into account (e.g., the actual fan-in and fan-out, the load capacitances, the interconnection paths). These exact values can be derived only from a detailed analysis of the actual implementation.

As an example, let us consider an m -level architecture having the filter size equal to L and the length of the samples and weights equal to w . Let be α the circuit complexity (in number of MOS) of a full-adder, β the one of an AND logic gate, δ the one of a DT latch, and γ the one of a two-inputs multiplexer. From the analysis performed in Section 4 on the first-level filters and the considerations presented in Section 5 for the higher-level filters, we observe that the basic structure has the same circuit complexity for all levels. Therefore, the total circuit complexity C of our architecture is $2m$ times the circuit complexity of a single convolver; expressed in MOS, it is:

$$C = 2m[(Lw-2)(\alpha+2\delta+2\beta)+\delta w(L-1)+(L-2)(2\gamma-\delta)+3\beta]$$

By considering typical values for the parameters α , β , γ , and δ in number of MOS (e.g., 48, 4, 48, and 14, respectively), we obtain $C = 2m(98Lw+82L-334)$.

As the latency is concerned, we consider the time interval (in clock cycles) between the presentation of the first bit of the first input sample and the generation of the last bit of the last-level transformed sample (for example $z(0)$ and $z(8)$ in the three-level architecture), during one computation period. For the processor operations described in Sections 4 and 5, the latency L of our architecture is:

$$L = 2^{m-1}(L+1)w\tau$$

To evaluate the throughput, we consider the time interval, in clock cycles, between two subsequent outputs generated by the last level ($z(0)$ and $z(8)$ in a three-level architecture). The throughput T is thus:

$$T = 1/2^m w\tau$$

By considering the structure of the computational cycle presented in Fig. 2 and the typical values of the latency for each component of the serial architecture, the clock cycle can be evaluated as

$$\tau = \tau_{AND} + (w-1)\tau_{FA} + \tau_{DT}$$

being τ_{AND} , τ_{FA} , and τ_{DT} the latency of the AND logic gate, the latency of the full adder, and the latency of the DT latch, respectively. By considering typical values for these parameters in MOS delays (namely, 1, 6, and 6 for τ_{AND} , τ_{FA} , and τ_{DT} , respectively), we obtain a clock cycle time equal to

$\tau = 6w + 1$. As a consequence, the latency L in MOS delays becomes $2^{m-1}(L+1)(6w+1)w$, while the throughput T is $1/w(6w+1)2^m$, measured in the inverse of a MOS delay.

The overall evaluation is summarized in Table 2. Some characteristics are exponentially growing with m ; generally, this is not desirable: however, we must take into account that m is rather small in most of the practical cases. In Figs. 4, 5, and 6, all the above characteristic figures are shown. For example, by considering the typical values of the gate parameters as discussed above, in the case of a filter having $m=6$ levels, $L=8$ weights in each convolver, and $w=16$ operand bits, the circuit complexity C is 154392 equivalent MOS gates, the clock cycle τ can be 97 MOS delays, the latency L is 446976 MOS delays, and the throughput T is about $1,007 \cdot 10^{-5}$ inverse of a MOS delay.

In Table 3, we summarize the comparison between our architecture and some of the ones presented in literature. It is important to remind that all architecture available in the current literature are bit-parallel, while our is completely serial. Due to this fact, in the performance of our architecture appears the parameter w (i.e., the samples and the weights length). Besides, it is worth to note that also the coefficients of the considered figures of merit are different, in particular for the circuit complexity. Actually, the constant k in the circuit complexity is quite different between our architecture and the traditional ones, being our architecture completely serial; as a consequence, the circuit complexity of our architecture is definitely smaller than the other architectures.

7. Conclusions

In this paper, a new serial dedicated architecture has been proposed and evaluated to compute the one-dimensional DWT. The main goal has been the minimization of the circuit complexity so as to achieve a very compact structure suitable for inclusion in large computing processors as well as for individual use as reasonably efficient unit. Regularity has been accurately taken into account in the design as well as the scalability: a DWT having more levels can be simply obtained by cascading additional filters of the kind presented in Section 5, while a higher number of filter weights can be used by introducing the corresponding serial multipliers in each convolver.

Further research will be directed to achieve a better exploitation of the units so as to avoid the clock cycles in which some units are inactive, due to data propagation and to decimation. The use of different approaches to design the serial convolver (mainly, to design of the serial multipliers) will be considered.

References

- [1] J.Bae, V.K. Prasanna, "Synthesis of VLSI Architectures for Two-Dimensional Discrete Wavelet Transforms", *Proc. International Conference on Application-Specific Array Processors*, pp. 174-181, 1995
- [2] C. Chakrabarti, M. Vishwanath, "Efficient Realizations of Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers", *IEEE Transactions on Signal Processing*, Vol. 43, n. 3, pp. 759-771, Mar. 1995
- [3] H. Y. H. Chuang, L. Chen, "VLSI Architecture for Fast 2D Discrete Orthonormal Wavelet Transform", *IEEE Journal of VLSI Signal Processing*, Vol. 26, n. 10, pp. 225-236, July 1995
- [4] G. Evangelista, "Comb and Multiplexed Wavelet transforms and Their Applications to Signal Processing", *IEEE Transactions on Signal Processing*, Vol. 42, n. 2, pp. 292-303, Feb. 1994
- [5] C.C. Hsu, J. Ding, M.E. Zaghloul, "An Image Discrete Wavelet Transform and the Hardware Implementation", *Proc. ISCAS'95*, pp. 1315-1319, 1995
- [6] G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform", *Electronics Letters*, Vol. 26, 1990
- [7] A.S. Lewis, G. Knowles, "VLSI Architecture for 2D Daubechies Wavelet Transform without multipliers", *Electronic Letters*, Vol. 27, n. 2, pp. 171-173, Jan. 1991
- [8] S.G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 2, n. 7, pp. 674-693, July 1989
- [9] C. Mead, L. Conway, *Introduction to VLSI System*, Addison-Wesley, 1980
- [10] A.V.O. Oppenheim, R.W. Shafer, *Digital Signal Processing*, Prentice Hall, 1975
- [11] K.K. Parthi, "A Systematic Approach for Design of Digit-Serial Signal Processing Architectures", *IEEE Transactions on Circuits and Systems*, Vol. 38, n. 4, pp. 358-375, Apr. 1991

- [12] K.K. Parthi, T. Nishitani, "VLSI Architectures for Discrete Wavelet Transforms", *IEEE Transactions on VLSI Systems*, Vol. 1, n. 2, pp. 191-202, Jun. 1993
- [13] O. Rioul, M. Vetterli, "Wavelets and Signal Processing", *IEEE Signal Processing Magazine*, Oct. 1991
- [14] O. Rioul, P. Flandrin, "Time-Scale Energy Distributions: A General Class Extending Wavelet Transforms", *IEEE Transactions on Signal Processing*, Vol. 40, n. 7, pp. 1746-1757, July 1992
- [15] M. J. Shensa, "The Discrete Wavelet Transform: Wedding the A Trouss and Mallat Algorithms", *IEEE Transactions on Signal Processing*, Vol. 40, n. 10, pp. 2464-2482, Oct. 1992
- [16] J. Vega-Pineda, S. Cabrera, Y.C. Chang, "VLSI Implementation of a Wavelet Image Compression Technique Using Replicant Coding/Decoding Cells", *Proc. ISCAS'95*, pp. 1173-1176, 1995
- [17] M. Vetterli, C. Herley, "Wavelets and Filter Banks: Theory and Design", *IEEE Transactions on Signal Processing*, Vol. 40, n. 9, pp. 2207-2232, Sept. 1995

Table 1: Operation of the first-level serial convolver

UNIT	TIME STEPS							
	0	8	16	24	32	40	48	56
Data in	u(0)	u(1)	u(2)	u(3)	u(4)	u(5)	u(6)	u(7)
Mul A	$g_2 u(0)$		$g_2 u(2)$		$g_2 u(4)$		$g_2 u(6)$	
Mul B	$g_1 u(1)$		$g_1 u(3)$		$g_1 u(5)$		$g_1 u(7)$	
Mul C	$g_0 u(2)$		$g_0 u(4)$		$g_0 u(6)$			
SR	$(g_2 u(0))^H$		$(g_2 u(2))^H$		$(g_2 u(4))^H$		$(g_2 u(6))^H$	
FA	$(g_2 u(0))^H + (g_1 u(1))^H$		$(g_2 u(2))^H + (g_1 u(3))^H$		$(g_2 u(4))^H + (g_1 u(5))^H$		$(g_2 u(6))^H + (g_1 u(7))^H$	
ACC	$(g_2 u(0))^H + (g_1 u(1))^H$		$(g_2 u(2))^H + (g_1 u(3))^H$		$(g_2 u(4))^H + (g_1 u(5))^H$		$(g_2 u(6))^H + (g_1 u(7))^H$	
Data out			$v(0) = (g_2 u(0))^H + (g_1 u(1))^H + (g_0 u(2))^H$		$v(2) = (g_2 u(2))^H + (g_1 u(3))^H + (g_0 u(4))^H$		$v(4) = (g_2 u(4))^H + (g_1 u(5))^H + (g_0 u(6))^H$	

Table 2: DWT serial architecture evaluation

Latency	$2^{m-1}(L+1)w\tau$	clock cycles
Throughput	$1/2^m w$	clock cycles ⁻¹
Circuit complexity	$2m[(Lw-2)(\alpha+2\delta+2\beta)+\delta w(L-1)+(L-2)(2\gamma-\delta)+3\beta]$	MOS

Table 3: A comparison between our architecture e the ones presented in literature

Architecture	Ref.	Latency	Throughput	Circuit complexity
Our Architecture		$O(wL2^{m-1})$	$1/wN$	$O(kLmw)$
Knowels Architecture	[6]	$O(N)$	$O(1/N)$	$O(kLm)$
Folded Architecture	[12]	$O(LN)$	$1/N$	$O(kLN)$
Digit-Serial Architecture	[12]	$O(wLN)$	$O(1/N)$	$O(kLNw)$
Systolic Array	[3]	$2N$	$1/2N$	$O(kLm)$
Parallel Filter	[3]	$\sim N$	$\sim 1/N$	$O(kLm)$
SIMD Linear Array	[3]	Lm	$1/Lm$	$O(kN)$
SIMD Multigrid ver. 1	[3]	$O(Lm)$	$1/L$	$O(kNm)$
SIMD Multigrid ver. 2	[3]	Lm	1	$O(kNL)$
Daubechies Architecture	[7]	$O(N)$	$O(1/N)$	$O(kLm)$

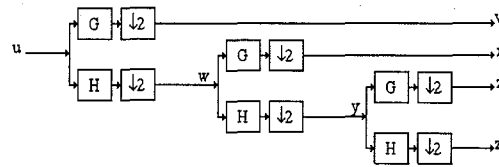


Figure 1: The Pyramid Algorithm for the 1D-DWT (the case of $m=3$)

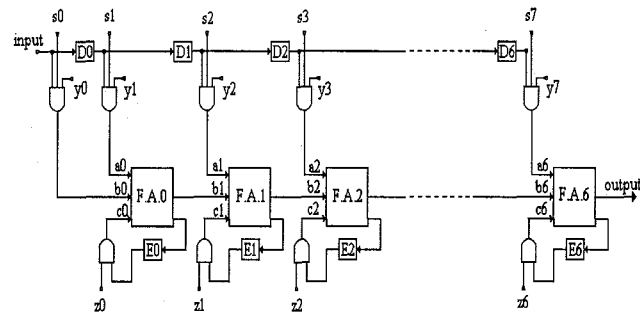


Figure 2: The serial multiplier (the case of $w=8$)

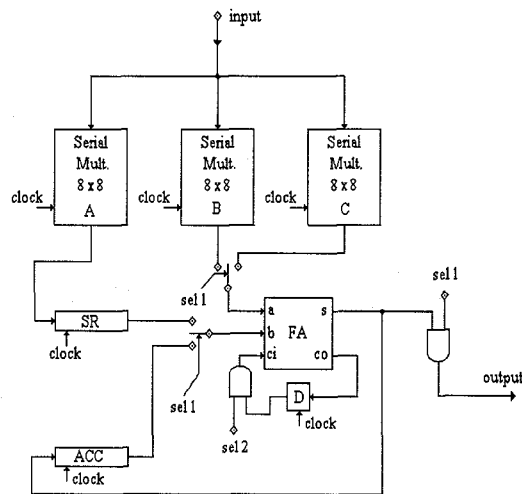


Figure 3: The serial convolver (the case of $L=3$ and $w=8$)

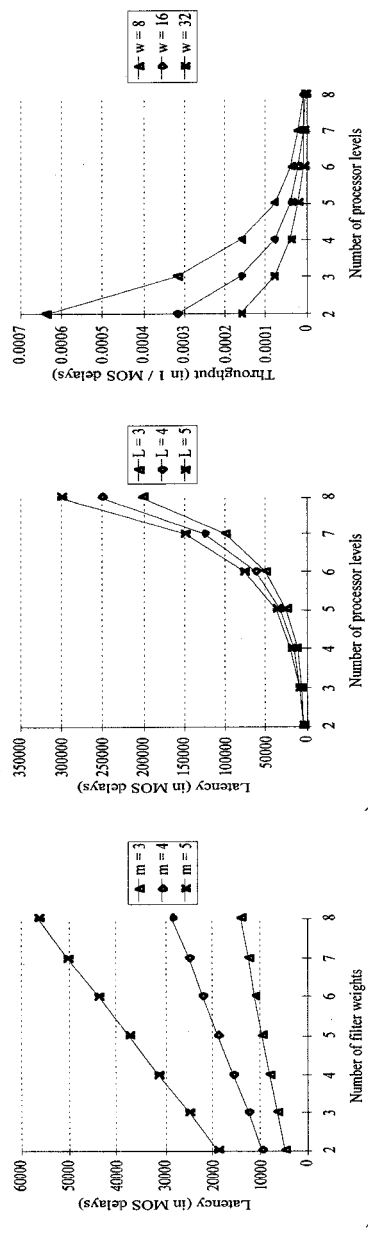


Figure 4: Latency (a) and circuit complexity (b) as functions of the number L of filter weights ($m=3, 4, 5$; $w=8$)

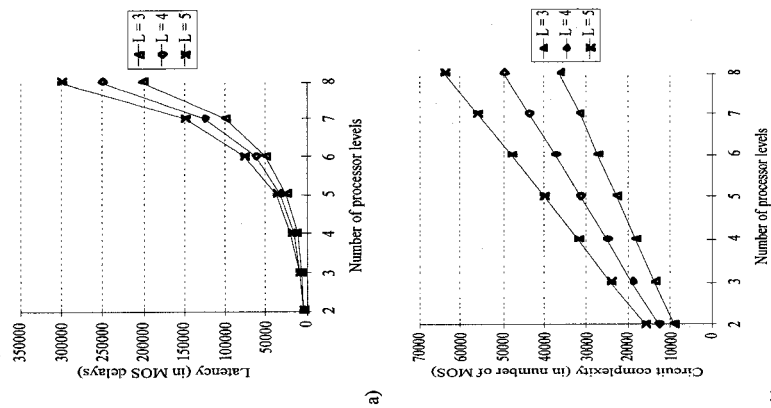


Figure 5: Latency (a) and circuit complexity (b) as functions of the number m of levels ($L=3, 4, 5$; $w=8$)

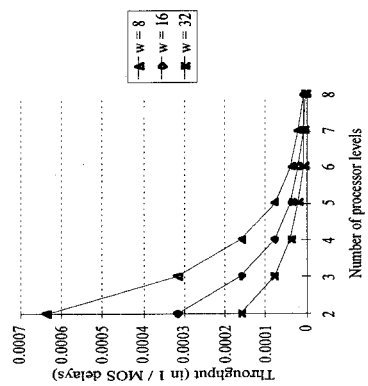


Figure 6: Throughput as function of the number m of levels ($w=8, 16, 32$)