A Serial-Parallel Architecture for Two-Dimensional Discrete Cosine and Inverse Discrete Cosine Transforms

Hyesook Lim, *Member*, *IEEE*, Vincenzo Piuri, *Senior Member*, *IEEE*, and Earl E. Swartzlander Jr., *Fellow*, *IEEE*

Abstract—The Discrete Cosine and Inverse Discrete Cosine Transforms are widely used tools in many digital signal and image processing applications. The complexity of these algorithms often requires dedicated hardware support to satisfy the performance requirements of hard real-time applications. This paper presents the architecture of an efficient implementation of a two-dimensional DCT/IDCT transform processor via a serial-parallel systolic array that does not require transposition.

Index Terms—Application specific processor architecture, Discrete Cosine Transform, Inverse Discrete Cosine Transform, image compression, serial-parallel processor, systolic array.

1 Introduction

The Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT) have been widely used in digital signal and image processing [3]. The two-dimensional (2D) DCT/IDCT is a standard data compression/decompression technique for image coding standards such as H.261, MPEG-1, and MPEG-2.

A popular approach for implementing the 2D DCT/IDCT is the row-column decomposition method [3], [4], in which the 2D transform is computed by applying the 1D DCT/IDCT by rows and, then, by columns. The 1D DCT/IDCT may be computed either by using a direct approach based on the Fast Cosine Transform (FCT) method [5], [6], [7] or an indirect strategy evaluating the DCT by means of the Discrete Fourier Transform [8], [9]. In [10], high performance is achieved by using efficient on-line arithmetic and an FCT algorithm. Even though the indirect approach usually requires a higher number of operations than the FCT technique, it has been shown [11] that efficient algorithms can be obtained by transforming two input data streams simultaneously. These approaches need to transpose the intermediate results by using a memory array, thus leading to a high circuit complexity and a long time for loading and unloading.

Another approach was recently proposed to directly compute the 2D DCT [12], [13], [14], without decomposing it into two successive 1D DCTs. Although this approach requires the least number of multipliers and adders, the structure of the resulting architecture is very complicated and the interconnection complexity is high.

An alternative technique is based on the use of systolic architectures [2], [15], [16], [17], [18], [19], [20]. In this case, the transformation is obtained by applying the 1D DCT twice. In [15], [16], [17], a transpose memory is necessary to reorder the data between the two 1D DCT computations, leading to a large circuit complexity. In [2], [18], [19], [20], unified architectures are presented to implement both the DCT and the IDCT without any transpose memory since the intermediate results stored in the array cells are in the order suited for the subsequent 1D transposition. These architectures consider bit-parallel and bit-serial data presentation and manipulation. Bit-parallel processing leads to a high-speed structure having a high circuit complexity; bit-serial processing reduces both the performance and the complexity.

In this paper, we propose an intermediate approach that allows us to design an architecture having throughput and circuit complexity between the extremes of the previous cases. We use bit-serial data interfaces at the borders of each processing element in the array processor to minimize the interconnection complexity. This creates a modular structure that can be easily expanded by cascading chips each containing a limited number of processors. Also, it allows us to easily introduce fault tolerance features such as reconfiguration [21], due to the limited number of interconnections that need to be rerouted. Pipelined serial-parallel execution of the arithmetic operations limits the performance degradation resulting from the use of serial data interfaces. Similar approaches to the design of complex arithmetic structures have been successfully applied to the implementation of multipliers and convolvers [22] as well as of FFT processors [23].

In Section 2, the DCT and the IDCT transforms are briefly summarized and the high-level structure of a unified

- *H. Lim is with Lucent Technologies, Cisco Systems, 3550 Cisco Way, San Jose, CA 95134. E-mail: hslim@cisco.com.*
- *V. Piuri is with the Department of Electronics and Information, Politecnico di Milano, piazza L. da Vinci 32, 20133 Milano, Italy. E-mail: piuri@elet.polimi.it.*
- *E.E. Swartzlander Jr. is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712. E-mail: e.swartzlander@compmail.com.*

Manuscript received 26 Nov. 1996; revised 14 Apr. 1999; accepted 6 June 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102366.

processor computing both of them is shown. In Section 3, the bit serial-parallel implementation of this architecture is presented and analyzed. Section 4 evaluates the throughput, the latency, and the circuit complexity.

2 The DCT/IDCT Transforms

Let $x(n)$ $(n = 0, 1, \ldots N-1)$ be a time domain data sequence and $y(k)$ $(k = 0, 1, \ldots N-1)$ be the corresponding transform-domain sequence. The Discrete Cosine Transform and the Inverse DCT are defined by [3]:

$$y(k) = \sqrt{\frac{2}{N}}\, u(k) \sum_{n=0}^{N-1} x(n) \cos\frac{(2n+1)k\pi}{2N} \quad \text{for } k = 0, 1, \ldots N-1 \tag{1}$$

and

$$x(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} u(k)y(k) \cos\frac{(2n+1)k\pi}{2N} \quad \text{for } n = 0, 1, \ldots N-1, \tag{2}$$

where

$$u(k) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0 \\ 1, & \text{otherwise.} \end{cases}$$

Since

$$\sqrt{\frac{2}{N}}$$

is a simple scaling of the input sequence, we ignore the scaling factor and consider the normalized DCT/IDCT computation. Equation (1) can be rewritten in matrix form as $(y_N) = [C_N](x_N)$, where (x_N) and (y_N) are N-dimensional input and output column vectors, respectively, and $[C_N]$ is the $N \times N$ DCT coefficient matrix where

$$[C_N]_{kn} = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } k = 0 \\ \cos\frac{(2n+1)k\pi}{2N}, & \text{otherwise} \end{cases} \quad \text{for } k, n = 0, 1, \ldots N-1. \tag{3}$$

For the computation of the 2D DCT, the conventional row-column approach is generally used. The row-column approach can be expressed in matrix form as:

$$[Z_N] = [C_N][X_N][C_N]^T. \tag{4}$$

Equation (4) shows that 2D $N \times N$ DCT is computed by N N-point DCTs along the rows of the input $[X_N]([Y_N] = [X_N][C_N]^T)$, followed by N N-point DCTs along the columns of the matrix obtained from the row transform $([Z_N] = [C_N][Y_N])$. Similarly, the 2D IDCT computation can be expressed in matrix form as:

$$[X_N] = [C_N]^T[Z_N][C_N]. \tag{5}$$

Using the row-column approach, the systolic array performs two consecutive matrix multiplications in sequence. Depending on which coefficient matrix is provided first ($[C_N]$ or $[C_N]^T$), the 2D DCT or the 2D IDCT computation can be selected and, thus, the operation in the systolic array is identical for the 2D DCT and the 2D IDCT.

Definitions of two types of semisystolic arrays for the multiplication of two $N \times N$ matrices have been proposed by Kung [24]. Many other examples are available in the literature, e.g., in [25], [26]. An array is semisystolic if the output data are not produced in the boundary cells of the array (Type 1) or if input data need to be preloaded into the cells of the array (Type 2). The Type 1 array, the Type 2 array, and their PE structures are shown in Fig. 1 for matrix multiplication $C = AB$ with $N = 4$, where H_{in} and H_{out} represent the horizontal input and the horizontal output, respectively, while V_{in} and V_{out} represent the vertical input and the vertical output, respectively. *Rij* is a value saved in a register of the (i, j)th PE.

In the Type 1 semisystolic array shown in Fig. 1a, each PE performs a multiply-accumulate operation where the horizontal and the vertical inputs are multiplied and added to the register value and the result is saved in the register. The initial value of the register is zero. Hence, in N cycles, each PE computes an inner product of a row of the horizontal input and a column of the vertical input. The latency is defined as the time from the first data entry until the output data is available. Cycles per datum (CPD) is defined as the number of clock cycles to compute each point of the transform which is an indication of the average latency. This $N \times N$ systolic array has a latency of $3N - 2$ cycles and a CPD of N, where the cycle time is the time to perform a multiply-accumulate operation. This indicates that it takes $3N - 2$ cycles to compute the first result and N cycles for each subsequent result. This array is semisystolic since the output data is not produced in the boundary cells of the array. This array has overhead for the output to be shifted out of the array.

In the Type 2 semisystolic array shown in Fig. 1b, to compute $C = AB$, each component of matrix B is preloaded into the array with one element of the matrix in a register within each PE, while matrix A is fed into the array. Each PE multiplies the horizontal input times the register value and adds this to the vertical input to produce the vertical output. The inner product of a column of the input matrix and a column of the stored matrix is computed every N cycles.

From Roziner and Karpovsky's point of view [27], there are two space coordinates, (n_1, n_0), in the index set and we have three variables A, B, and C to move along the coordinate axes. There are three possibilities corresponding to the case when one of the variables does not move in the array. In the Type 1 semisystolic array, A moves by n_0, B moves by n_1, and C is accumulated in the PEs. In the Type 2 semisystolic array, B (input data) does not move; this means it is preloaded into the array, A moves by n_0, and C moves by n_1.

Based on these two types of semisystolic arrays, a true systolic array for two-dimensional DCT was presented previously [2]. The key idea is to combine the two semisystolic arrays for the matrix multiplication into one array so that input and output move along axes and the intermediate result does not move. In this way, the systolic array does not require any transposition, which is required

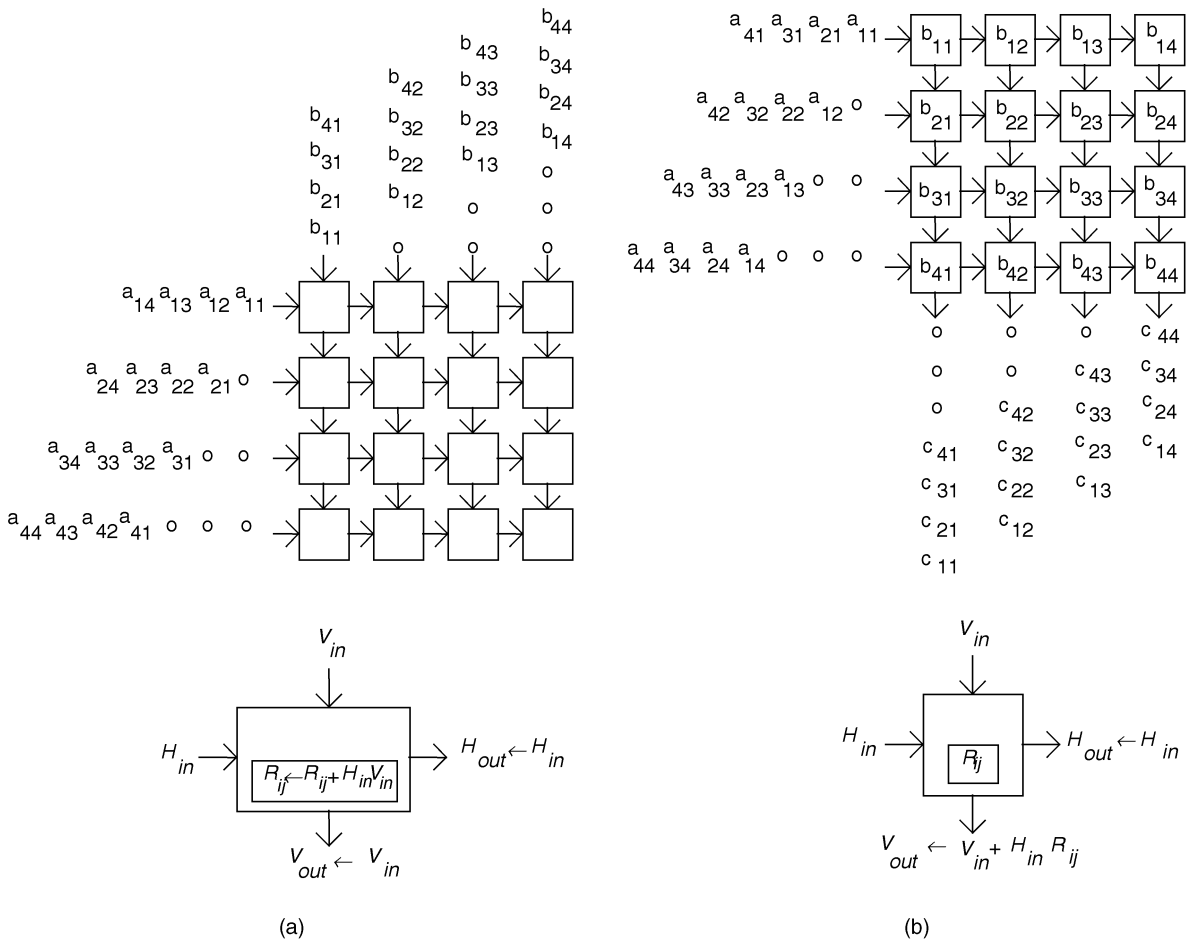

Fig. 1. Semisystolic arrays for matrix multiplication: (a) Type I array and (b) Type 2 array.

in other architectures implementing the row-column approach to 2D transform computation.

In the systolic array, the first 1D DCT ($[Y_N] = [X_N][C_N]^T$) is computed using the Type 1 semisystolic method and then $[Y_N]_{n_1 k_0}$ are computed and saved in the PEs. Using these results, the second 1D transform ($[Z_N] = [C_N][Y_N]$) is computed using the Type 2 semisystolic method of matrix multiplication. In this systolic array for the 2D DCT, $[X_N]$ (input data) and $[C_N]$ (the coefficient matrix for the second 1D transform) move by n_0, while $[C_N]^T$ (the coefficient matrix for the first 1D transform) and $[Z_N]$ (output data) move by n_1. Only $[Y_N]$ (the intermediate spectrum) does not move.

The systolic array and the PE structure for $N = 4$ is shown in Fig. 2 and Fig. 3, respectively, where

$$c_{ji} = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } j = 0 \\ \cos\frac{(2i+1)j\pi}{2N}, & \text{otherwise.} \end{cases}$$

PEs in Fig. 3 perform two different functions, alternately: For the first N cycles, the PE operates as a Type 1 semisystolic array; for the next N cycles, it operates as a Type 2 semisystolic array; then, the process repeats; etc. At the system level, this array is a true systolic array since it does not require any preloading of input data and it generates output data only from the bottom boundary cells of the array. A detailed explanation of this systolic array is found in [2].

For the 2D IDCT computation in (5), $[Y_N] = [Z_N][C_N]$ is performed first and then $[X_N] = [C_N]^T[Y_N]$ is performed. Once the proper coefficient matrix is selected, the remaining processing is identical for the 2D DCT or the 2D IDCT computation. We select the 2D DCT or the 2D IDCT computation by providing an external control signal for mode selection.

3 The Serial-Parallel Architecture

A detailed serial-parallel architecture implementing the DCT/IDCT array processor introduced in Section 2 is shown in Fig. 4. The external interfaces to the individual PEs are serial to minimize the number of interconnections. The internal data manipulation is serial-parallel to increase the speed of the computation with respect to the purely serial case. Data are presented and treated starting from the least significant bit.

The H shift register on the horizontal interconnection and the H register are used to convert the data to a bit-parallel format within each PE during both phases of the PE operation (Type 1 and Type 2). The V shift register on the vertical interconnection provides delay for PE synchronization. Data and possible computation results are propagated

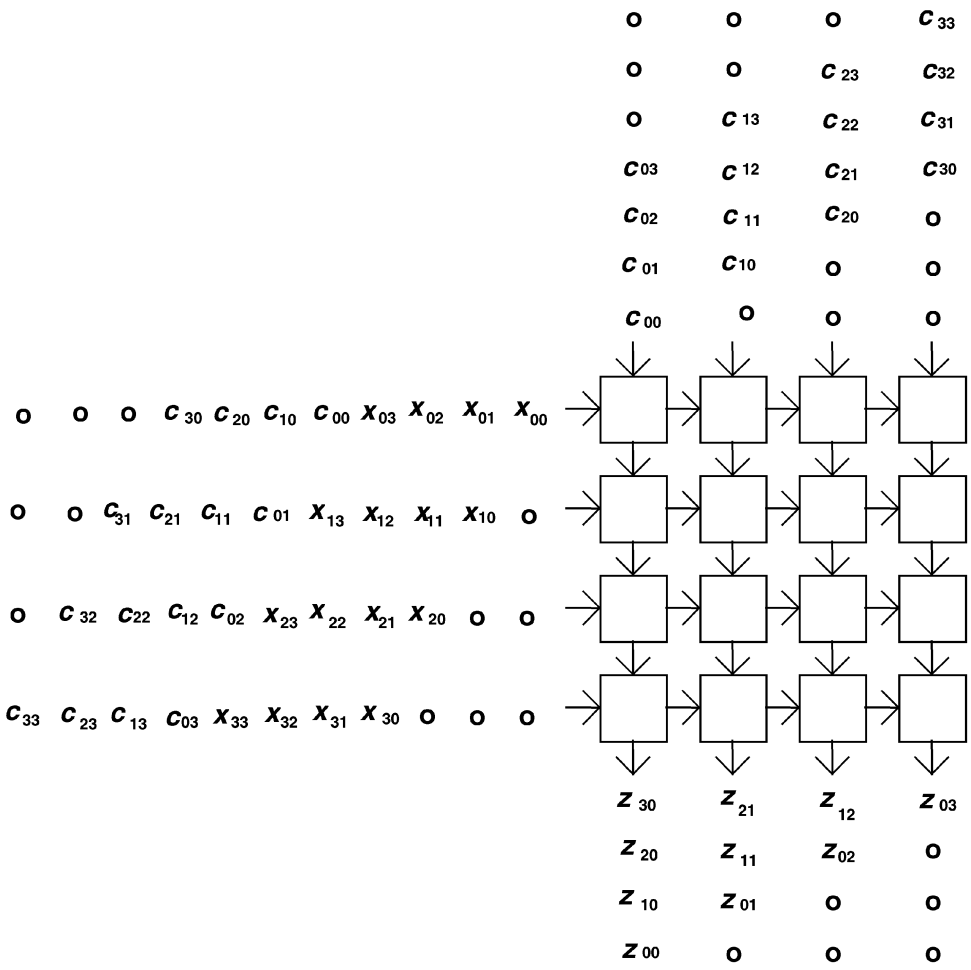

Fig. 2. The systolic array for the computation of the 2D DCT/IDCT [13].

on a diagonal wavefront from the top left PE to the bottom right PE of the array.

The partial product generator, the carry-save adder, the hardwired rotator, the hardwired shifter, the sum register, and the carry register implement the serial-parallel multiplication used during the first and the second PE phases.

Addition to the results of the previous multiplications in the first PE phase (as required by the DCT and the IDCT algorithms) is performed by the carry-save adder since the partial result of the previous multiplication is held in the sum and carry registers as the initialization values for the subsequent multiplication.

Note that no carry propagation is necessary during the whole first PE phase since partial results are accumulated. Conversely, to reuse the same serial-parallel multiplier during the second PE phase, a serial presentation of the result computed in the first PE phase is required.

At the end of the first PE phase, the result (held in sum and carry registers) is truncated and moved to the sum and the carry shift registers to complete the carry propagation and to generate a serial result for the multiplications in the second PE phase. These shift registers, the serial adder SA1, and the master-slave memory element D1 perform this operation during the first multiplication in the second PE phase. The R shift register holds and recirculates the result of the first PE phase during the subsequent multiplications of the second phase.

At the end of each multiplication in the second PE phase, the sum and the carry registers hold the result that needs to be added to the partial result coming from the PE in the same column of the previous row. The product is therefore truncated and moved in the sum and in the carry shift registers as in the last multiplication of the first PE phase. Serial adder SA1 and memory element D1 provide carry propagation within the multiplication result, while serial adder S2 computes the accumulated partial summation that must be propagated to the PE in the same column in the subsequent row.

Since the multiplier and the multiplicand of the first phase are m bit two's-complement fixed-point numbers, the result generated by the multiplication is $2m-1$ bits long. The addition of the results of the first phase is a $2m-1+ceil(log_2N)$ number. Truncation of the m least significant bits leads to operands in the second phase having $m-1+ceil(log_2N)$ bits, while the multipliers are still m-bit long. The result of the multiplication in the second phase is therefore $2m-2+ceil(log_2N)$ bits long and is truncated to $m-2+ceil(log_2N)$ bits by removing the m least significant bits. By adding N of these numbers in the second phase, we produce a final result having $m-2+2ceil(log_2N)$ bits.

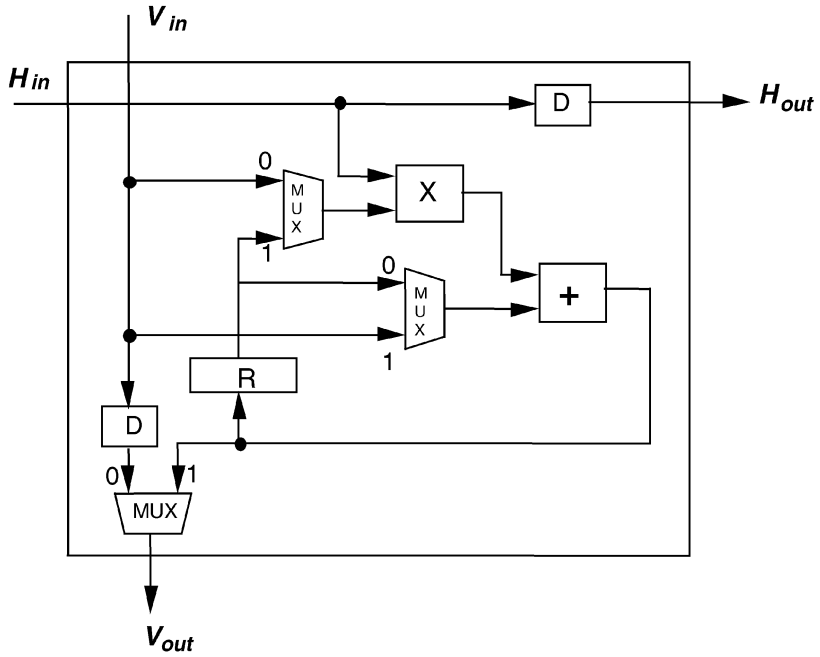

Fig. 3. The high-level design of the processing element for the array of Fig. 2.

The H shift register on the horizontal interconnection holds the data for the interprocessor data transfer along the same row of the array. Its detailed structure is given in Fig. 5. It is a regular shift register composed of master-slave memory elements: One bit is propagated on each clock cycle. The clock signal controlling the shift register, as well as the whole PE, is denoted as $\phi 1$. The H shift register is m-bits long.

When a datum from the horizontal input has been loaded into the H shift register, it is moved in the H register to allow its serial propagation to the subsequent PE in the same row. The H register preserves the data for the operations within the PE in parallel form. This register is a regular latch composed of m memory elements, as shown in Fig. 5. To avoid delay when storing the horizontal datum in the H register from the H shift register, storing in the H register is controlled by the signal $\phi 2$. It becomes active after m cycles of signal $\phi 1$, on the same edge of $\phi 1$, which stores the datum in the slave elements of the H shift register. As a consequence, each new datum stored in the H shift register is moved both to the H shift register and to the H register, simultaneously.

The V shift register on the vertical interconnection has a similar role to that of the H shift register. It is an m-bit master-slave shift register, controlled by clock signal $\phi 1$. For correct operation of the serial-parallel multiplier, the multiplicand is first fed in the PE horizontally, then (when the whole multiplicand is in the PE) the multiplier is fed vertically during the first PE phase. During the second PE phase, the multiplicand is fed horizontally; the multiplier is already stored in the PE, but in a redundant form which is not suitable for direct serial-parallel multiplication. To reuse the same multiplier of the first iteration without wasting cycles for the carry propagation, it is more efficient to "parallelize" the incoming serial multiplicand during the second PE phase, while the redundant (sum and carry) parallel representation of the multiplier is "serialized" while carries are propagated.

The V′ shift register is used during the second PE phase to increase the length of the V shift register. During the second phase $2ceil(log_2 N) - 2$ additional bits are required to accommodate the result, as discussed above. The V′ shift register is controlled by the clock signal $\phi 1$. Multiplexer M2 provides bypassing during the first PE phase. Multiplexer M3 propagates the vertical input during the first PE phase and the result is computed by the PE during the second PE phase.

The partial product generator is a circuit that produces the partial product of the multiplicand stored in the H register by the ith bit of the multiplier arriving from the multiplexer M1, according to the traditional arithmetic rules for serial-parallel multiplication of two numbers in the two's-complement representation. During the first PE phase, the content of the H register is multiplied by the bit coming from the PE in the same column in the previous row through the vertical interconnection. During the second phase, the multiplier bit is the ith bit of the result generated by the first PE phase.

The ith partial product of the current multiplication is added to the accumulated result generated by the previous iterations and stored in the sum and the carry registers.

To assure proper alignment of the partial product bits with respect to the stored result, the partial product should be shifted left by i bits in the current multiplication. To avoid a variable shifting, we have chosen to have the partial product in a standard position and to downshift the result stored in the sum and in the carry registers. In this case, the shifting is always by one bit at each iteration; as a consequence, it can be implemented as an hardwired shifting without any extra hardware.

The carry register is shifted by one bit by the hardwired shifter shown in Fig. 6. Since there is no carry having weight

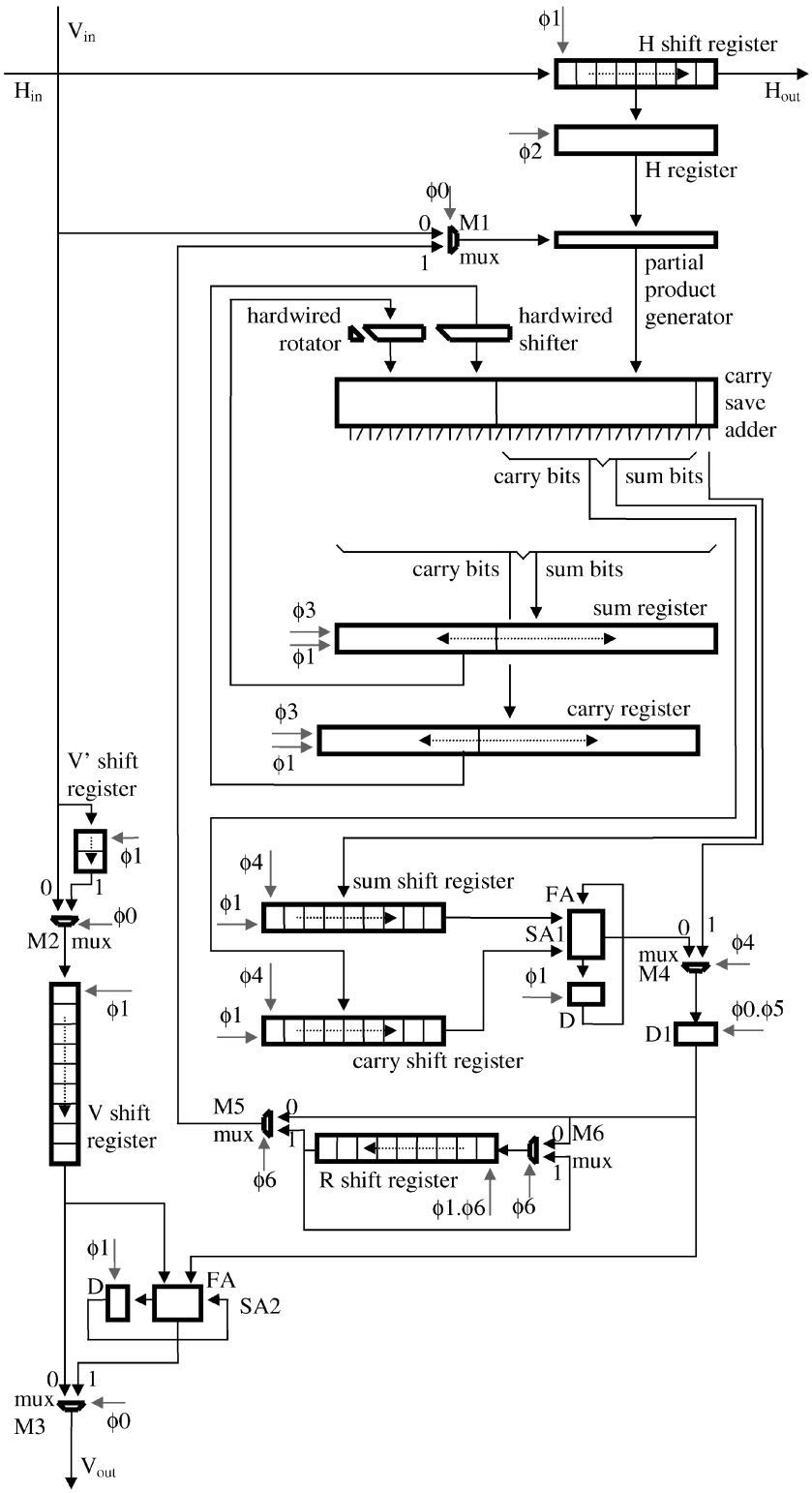

Fig. 4. The detailed design of the processing element for the array of Fig. 2.

2^0, the shifting does not lose any of the stored result. A zero must be introduced in the most significant position of the shifted carry datum since there was no carry in above the most significant digit in the carry register.

To avoid truncation of the result computed in the previous iteration (truncation must be performed only at some specific points of the computation to guarantee a given precision), the least significant bit of the sum register is rotated to the most significant position so that it can be saved again in the sum register itself. Rotation is shown in Fig. 7. The hardwired shifter and the hardwired rotator accommodates operands having $2m - 1 + ceil(log_2 N)$ bits.

The carry save adder consists of $2m - 1 + ceil(log_2 N)$ standard full adders, reducing the three inputs (the rotated

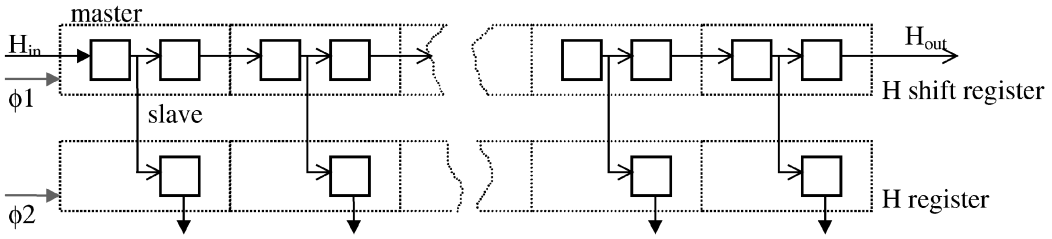

Fig. 5. The H shift register and the H register.

sum register and the shifted carry register generated by the previous iteration and the current partial product) to two values that are stored in the sum and carry registers.

The sum and carry registers each consist of $2m - 1 + ceil(log_2 N)$ master-slave memory elements. Storing in the master and slave elements is controlled by the control signal $\phi 1$. Their structure is shown in Figs. 8 and 9, respectively.

The result stored in the sum register is shifted right by the hardwired rotator, leaving unused room at the most significant positions. This insures that there is no interference between the most significant part of the result that is progressively moving right and the saved least significant part. In other words, it can never happen that a carry propagates from the most significant bit of the result to the least significant bit of the result itself that is temporarily saved on the unused left side of the sum register. Introducing a zero at the most significant position of the hardwired shifter insures that the bits saved in the left part of the sum register are preserved at their original value. The boundary between the temporary storage area in the sum register and the active computing area is progressively moving to the right at each iteration during the individual multiplication.

At the end of the m iterations required to generate the complete product of the multiplicand stored in the H register by the multiplier arriving from the vertical interconnection, the m most significant positions of the sum register hold the m least significant bits of the result, while the remaining positions contain the most significant part of the result. Similarly, the carry register has zeroes in the m most significant positions and the possible carries of the most significant part of the result in the least significant positions.

The result stored in the sum and carry registers is used as the initialization value for the subsequent multiplication of the first PE phase, i.e., we are using these registers as accumulators. As a consequence, before starting the subsequent multiplication, it is necessary to align the accumulated result stored in the sum and carry registers. In each of these registers, this is accomplished by swapping the positions holding the most and the least significant parts of the result. Swapping is performed by multiplexing the outputs of the carry save adders directly and by controlling the storing operation with control signal $\phi 3$, as shown in Figs. 8 and 9. This signal is active only during the last computational cycle related to the last digit in each multiplication except the last one.

Since the hardwired rotator and the hardwired shifter are always present and apply the rotation and shifting during the first iteration of the subsequent multiplication, we need to accommodate this in the interconnections for the swapping operation. In particular, we need to swap the contents of the sum register by leaving the most significant bit of the result in the least significant position of the sum register, while the m least significant bits of the result are moved from the most significant positions of the sum register to the positions starting from weight 2^1; the remaining most significant bits of the result are shifted in the most significant positions. Thus, the hardwired rotator presents the accumulated summation to the subsequent multiplication since it rotates to the right side the value swapped as described above. Processing is similar for the carry register, except that the least significant positions are filled by zeroes instead of swapping the zeroes present in the most significant part.

At the end of the first PE phase, i.e., at the end of all multiplications in the first phase, the outputs of the carry save adder do not need to be swapped since the accumulated sum of products does not need to be realigned for subsequent addition of multiplication results. As a consequence, control signal $\phi 3$ does not need to be activated to perform the swapping and the correct storing in the sum and in the carry registers. Actually, the result generated by the carry save adder does not need to be stored in the sum and in the carry registers. In fact, since a bit-serial presentation of the final result of the first phase has to be generated to execute the multiplication in the second phase, we can truncate the result of the first phase and then use a parallel-to-serial conversion of the bits to control the operation of the partial product generator during the second PE phase.

Therefore, we discard the m least significant bits of the final result of the first PE phase which are located in the m most significant positions of the sum and carry registers at the end of the last multiplication of the first PE phase. The most significant bits of the final result (that have to be used

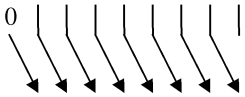

Fig. 6. The hardwired shifter.

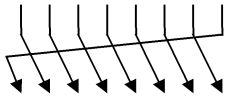

Fig. 7. The hardwired rotator.

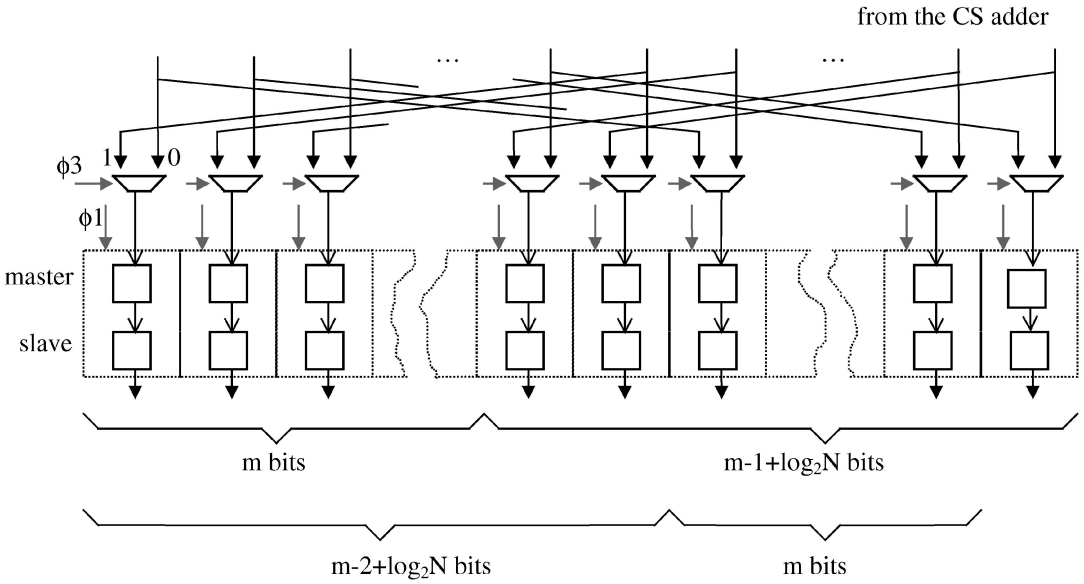

Fig. 8. The sum register.

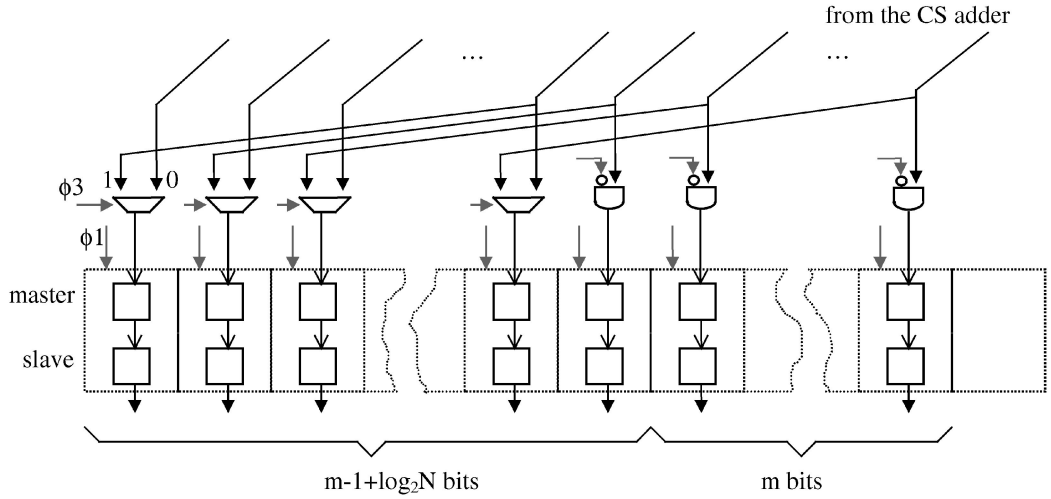

Fig. 9. The carry register.

during the second phase) are located in the remaining positions of the sum and carry registers.

To allow reuse of the serial-parallel multiplier during the second phase as in the first phase and to propagate the carries of the final result itself, the most significant bits of the final result are moved into the sum shift register, the carry shift register, and the master-slave memory element D1. The sum shift register is a $m-2+ceil(log_2N)$ shift register with parallel loading, as shown in Fig. 10; serial propagation is controlled by the clock signal $\phi 1$, while the initial parallel loading is controlled by signal $\phi 4$. It is used to hold and serialize the $m-2+ceil(log_2N)$ most significant bits of the sum result of the first phase. The carry shift register has the same structure and operation as the sum shift register.

Since the bit having weight 2^m in the final result is generated by the rightmost full adder in the carry save adder, there is no carry bit associated with this position. As a consequence, that sum bit is the result bit that has to be used during the first iteration of the multiplication in the second PE phase. To avoid wasting clock cycles, this bit is stored directly in the master-slave memory element D1 instead of being saved in the sum shift register, so it is immediately ready for generating the first partial product of the second PE phase. The use of such a memory element facilitates separating the serialization of the multiplier of the second phase from the partial product generation and accumulation so that both of these operations have about the same latency. If this element was not used, the latency of partial product addition would be approximately doubled.

During the subsequent $m-2+ceil(log_2N)$ clock cycles, the standard serial adder SA1 computes the subsequent bits of the multiplier that are fed (through multiplexer M4, memory element D1, and multiplexer M1) into the partial product generator.

Shift register R copies and holds each bit of the result of the first PE phase (incorporating the propagated carries) for reuse in the subsequent multiplications of the second PE phase. During each multiplication, the contents of the shift register R are recalculated Multiplexer M5 selects shift

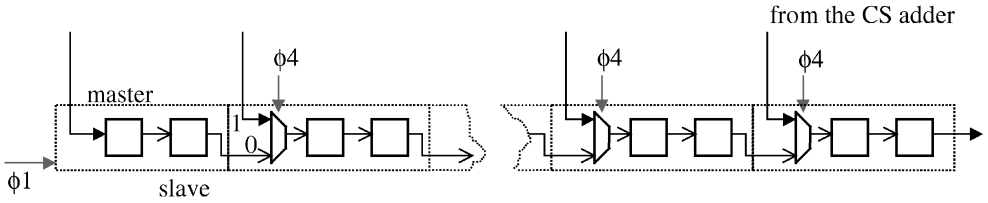

Fig. 10. The sum and carry shift registers.

register R during these multiplications, while multiplexer M6 is used to load and recirculate the multiplier. Shift register R is a standard shift register having $m-1+ceil(log_2N)$ bits and its operation is controlled by signals $\phi 6$ and $\phi 1$.

During the last multiplication of the first PE phase, the multiplicand of the second PE phase has been serially fed into the H shift register and, then, moved into the H register. To guarantee the correct data presentation along the H interconnection, each of the data in the second PE phase (except the last one) needs to be transformed in a representation on $m-2+2ceil(log_2N)$ bits: Only the first m least-significant bits will be used and moved into the H register, while the $2ceil(log_2N)-2$ most-significant bits will be discarded. Representation extension can be easily obtained by holding the most significant bit of every operand fed in the horizontal bus at the primary array input for $2ceil(log_2N)-2$ additional clock cycles. Extending the representation is more efficient than modifying the control signal of the H shift register since generation and propagation of such a signal is more complex.

The multiplication of the second PE phase is performed as the last multiplication of the first phase. The $m-2+ceil(log_2N)$ bits of the result are available at the outputs of the carry save adder at the end of the last iteration of the multiplication. Control signal $\phi 4$ is activated to move the bits in the sum shift register, in the carry shift register, and in memory element D1. At the end of the multiplication in the second PE phase, no swapping for realignment is necessary since only one multiplication is performed in the PE at each iteration and no accumulation is required within the PEs. Therefore, we perform the truncation and serialization of the result with the carry propagation. Note that the most significant bit of the sum and the carry registers are meaningless in this second phase since the result is one bit shorter than the accumulated result of the first PE phase.

The carry propagation through the serial adder SA1 and the overall serialization of the result of the second phase are performed as in the first phase. The serial final result is now added to the result computed by the PE in previous row of the same column and propagated down through the V and the V′ shift registers. This addition is performed by serial adder SA2.

The value contained in D1 is held for $ceil(log_2N)-1$ additional clock cycles in order to extend the representation of the multiplication result to $m-2+2ceil(log_2N)$ bits (as the final value of the second PE phase which propagated vertically) for correct addition in SA2.

4 Architectural Evaluation

The registers that decouple the computation during the different phases allow us to obtain a sort of pipelining between the operations in the PE. That is, the operations in the PE may be overlapped in time to achieve a high performance. The timing of PE (0, 0) is shown in Fig. 11. We denote with s and s' the sum bits generated by carry-save adder during the previous and the current iterations, with c and c' the carry bits generated by carry-save adder during the previous and the current iterations, with r the result of the first PE phase, and with t and t' the partial vertical summations which come in and go out during the second PE phase. In parentheses, we give the bit index when appropriate. The columns are associated with the clock cycles: The time is clocked by the signal $\phi 1$.

From Figs. 4 and 11, it is possible to deduce the performance of the proposed serial-parallel architecture. Due to the internal pipelining of each PE, the minimum clock period, τ, is given by:

$$\tau = \tau_{\mathrm{pp}} + \tau_{\mathrm{FA}} + \tau_{\mathrm{sr}} = \tau_{\mathrm{pp}} + \tau_{\mathrm{FA}} + \tau_{\mathrm{MUX}} + 2\tau_{\mathrm{D}},$$

where τ_{pp} is the latency of the partial product generator, τ_{FA} is the latency of a full adder, τ_{sr} is the latency of the sum register, τ_{MUX} is the latency of a two-input multiplexer, and τ_{D} is the latency of a master-slave register. To have a rough evaluation of the clock cycle independently from the specific integration technology adopted for the physical realization, we assume $\tau_{\mathrm{pp}}=1$, $\tau_{\mathrm{FA}}=6$, $\tau_{\mathrm{MUX}}=2$, and $\tau_{\mathrm{D}}=3$. For these values, the clock cycle τ is approximately 15 equivalent gate delays.

The latency of each PE for the first phase is $(N+1)m\tau$, while, for the second phase, it is

$$[N(m-2+2ceil(log_2N))-1+ceil(log_2N)]\tau.$$

The total latency, l_{PE}, for an individual PE is therefore $(2N+1)(m-1+ceil(log_2N))\tau$. Due to the time skewing of the computation in the PEs by starting from PE(0, 0) down to PE(N, N), the total latency l of the array computation is given by:

$$\begin{aligned} l &= l_{\mathrm{PE}} + m(N-l)\tau \\ &= [N(3m-2+2ceil(log_2N))-1+ceil(log_2N)]\tau, \end{aligned}$$

where $m\tau$ is the skew time between two adjacent PEs.

time → (clock cycle ϕ1)	1	2	...	m	m+1	m+2	...	2m	...
H shift register	load x00 serially				load x01 serially				
H register				load x00	hold x00	hold x00	hold x00	hold x00 | load x01	
V shift register					load c00 serially				
V' shift reg.									
partial product					c00 x00				
CSA					s+c+c00 x00				
sum register					store s'	store s'	store s'	store rotated s'	
carry register					store c'	store c'	store c'	store rotated c'	
sum shift reg.									
carry shift reg.									
D1									
SA1									
SA2									
R shift register									
ϕ0	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive
ϕ2	inactive	inactive	inactive	inactive | active	inactive	inactive	inactive	inactive | active	
ϕ3	inactive	inactive	inactive	inactive	inactive	inactive	inactive	active	
ϕ4	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive
ϕ5	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive
ϕ6	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive

time → (clock cycle ϕ1)	(N-1)m+1	(N-1)m+2	...	Nm	Nm+1	Nm+2	...	(N+1)m
H shift register	load x0N serially				load c00 serially			
H register	hold x0,N-1	hold x0,N-1	hold x0,N-1	hold x0,N-1 | ld x0N	hold x0N	hold x0N	Hold x0N	hold x0N | ld c00
V shift register	load c0,N-1 serially				load c0N serially			
V' shift reg.								
partial product	c0,N-1 x0,N-1				c0N x0N			
CSA	s+c+c0N x0N				s+c+c0N x0N			
sum register	store s'	store s'	store s'	store rotated s'	store s'	store s'	store s'	
carry register	store c'	store c'	store c'	store rotated c'	store c'	store c'	store c'	
sum shift reg.								store s'(1..)
carry shift reg.								store c'
D1								store r(0)=s'(0)
SA1								
SA2								
R shift register								store r(0)=s'(0)
ϕ0	inactive	inactive	inactive	inactive	inactive	inactive	inactive	inactive
ϕ2	inactive	inactive	inactive	inactive | active	inactive	inactive	inactive	inactive | active
ϕ3	inactive	inactive	inactive	active	inactive	inactive	inactive	inactive
ϕ4	inactive	inactive	inactive	inactive	inactive	inactive	inactive	active
ϕ5	inactive	inactive	inactive	inactive	inactive	inactive	inactive	active
ϕ6	inactive	inactive	inactive	inactive	inactive	inactive	inactive	active

time → (clock cycle ϕ1)	(N+1)m+1	...	(N+2)m-2+logN	(N+2)m-1+logN	(N+2)m+logN	...	(N+2)m-2+2logN	(N+2)m-1+2logN	...	(N+2)m-3+3logN-1
H shift register						load c10				
H register							| load c10			
V shift register								load t(0)		...
V' shift reg.	load t(0)	...	load t(m-2+logN)	Load t(m-1+logN)	load t(m+logN)			c10 r(0)		...
partial product	c00 r(0)	...	c00 r(m-2+logN)	C00 r(m-1+logN)				s+c+c10 r(0)		...
CSA	s+c+c00 r(0)	...	s+c+c00 r(m-2+logN)	s+c+c00 r(m-1+logN)				store s'		...
sum register	store s'	store s'	store s'					store c'		...
carry register	store c'	store c'	store c'					shift		...
sum shift reg.	shift	shift	shift	store s'(1..)				shift		...
carry shift reg.	shift	shift	shift	store c'(1..)				store r(1)		...
D1	store r(1)	store r(i)	store r(m-1+logN)	store s'(0)	store t(1)	store t(i)	store t(m-1+logN)	r(1)		...
SA1	r(1)	r(i)	r(m-1+logN)		t(1)	t(i)	t(m-1+logN)			...
SA2					t'(0)	t'(i-1)	t'(m-2+logN)	t'(m-1+logN)		t'(m-2+2logN)
R shift register	store r(1)	store r(i)	store r(m-1+logN)							
ϕ0	active	active	active	active	active	active	active	active		active
ϕ2	inactive	inactive	inactive	inactive	inactive	inactive	inactive | active	inactive		inactive
ϕ3	inactive	inactive	inactive	inactive	inactive	inactive	inactive	active		inactive
ϕ4	inactive	inactive	inactive	active	inactive	inactive	inactive	inactive		inactive
ϕ5	active	active	active	inactive	inactive	inactive	inactive	inactive		inactive
ϕ6	active	active	active	inactive	inactive	inactive	inactive	active		active

Fig. 11. Schedule for processing element (0, 0).

The maximum throughput, T, can be obtained by continuously feeding input matrices into the array. It is given by:

$$T = [2N(m - 1 + ceil(log_2 N))]^{-1}\tau^{-1}.$$

The circuit complexity, C, can be roughly evaluated by considering the equivalent gate count. It is given by:

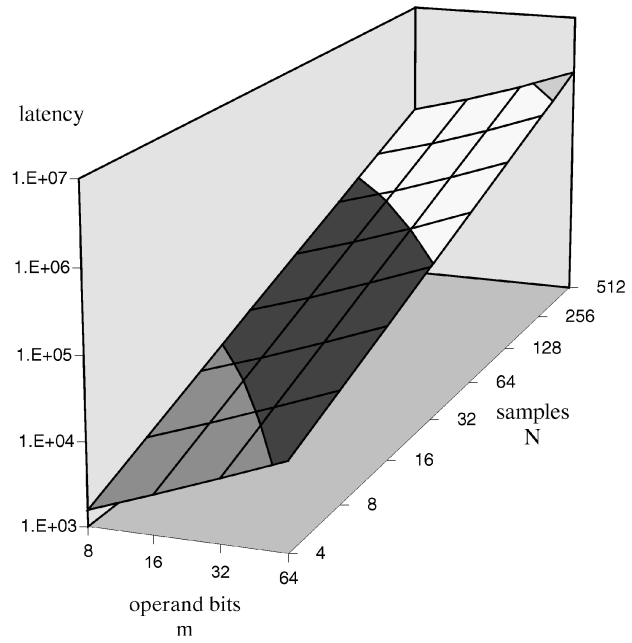

Fig. 12. Estimated latency.

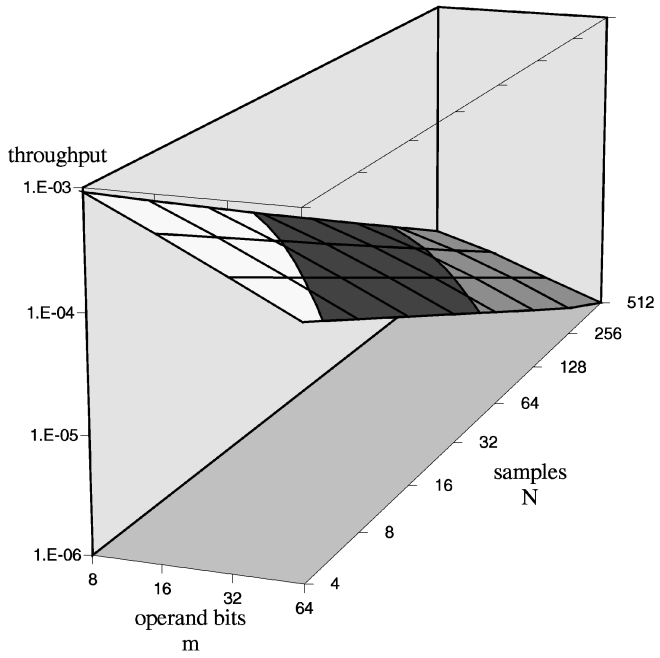

Fig. 13. Estimated throughput.

$$
\begin{aligned}
C &= N^2 C_{\mathrm{PE}} \\
&= N^2[(2m+1+ceil(log_2 N))C_{rmFA} \\
&\quad + (19m-13+12ceil(log_2 N))C_{\mathrm{D}} \\
&\quad + m\, C_{\mathrm{pp}} + (6m+3ceil(log_2 N))C_{rmMUX}],
\end{aligned}
$$

where C_{PE} is the complexity of the individual PE, C_{pp} is the complexity of the partial product generator, C_{FA} is the complexity of a full adder, C_{MUX} is the complexity of a two-input multiplexer, and C_{D} is the complexity of a master-slave register. Typical values for the circuit complexity of these components (measured in equivalent gates), are $C_{\mathrm{pp}} = 1$, $C_{\mathrm{FA}} = 9$, $C_{\mathrm{MUX}} = 3$, and $C_{\mathrm{D}} = 6$. As a consequence, the complexity of the whole array is approximately $N^2[151m - 69 + 90\ ceil(log_2 N)]$ equivalent gates.

Figs. 12, 13, 14, and 15 summarize the above evaluations concerning latency, throughput, circuit complexity, and CL^2 for typical device parameters.

5 Conclusions

In this paper, we have presented the detailed structure implementing the processing element of a unified array for two-dimensional DCT and IDCT transforms. The serial-parallel architecture allows us to realize a compact system

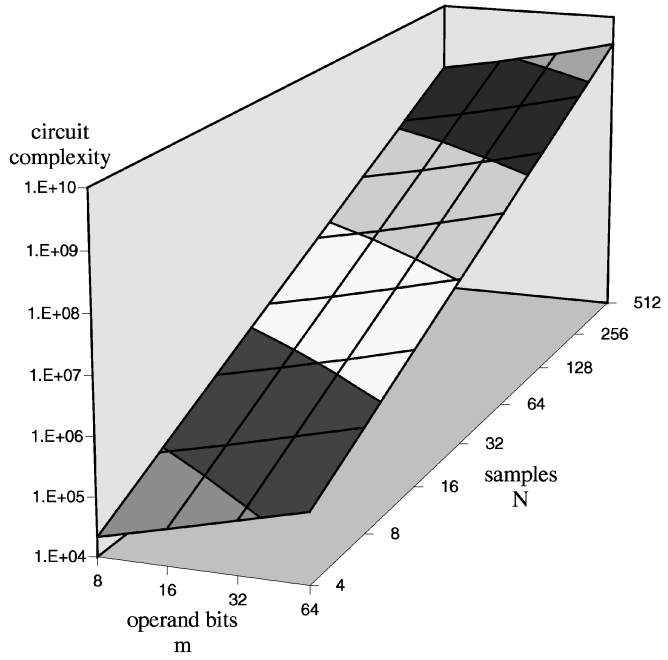

Fig. 14. Estimated complexity.

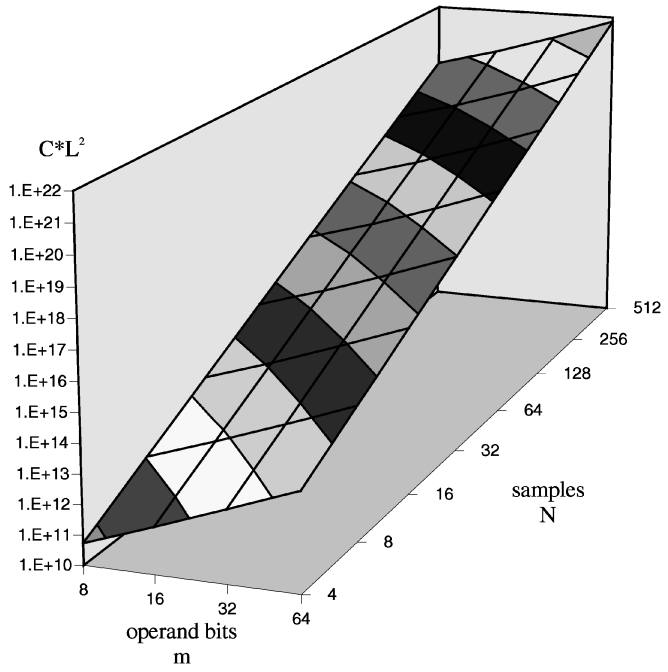

Fig. 15. Estimated computational power ($\text{complexity} * \text{latency}^2$).

having high performance. Serial data transfer reduces the number of interprocessor connections and the overall wiring required to propagate the data. Internal parallel operation with carry-save addition achieves a high performance by avoiding carry propagation. The internal pipelining of the operations provides the maximum exploitation of all components so that operations are at least partially overlapped in time to reduce the overall latency and to increase the throughput.

REFERENCES

[1] H.S. Lim, "Multidimensional Systolic Arrays and Their Implementations for Discrete Fourier Transform and Discrete Cosine Transform," PhD dissertation, Univ. of Texas at Austin, 1996.

[2] H.S. Lim and E.E. Swartzlander Jr., "A Systolic Array for 2-D DFT and 2-D DCT," *IEEE Proc. Int'l Conf. Application Specific Array Processors*, pp. 123-131, 1994.

[3] A.V. Oppenheim and R.W. Shafer, *Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice Hall, 1975.

[4] J. Canaris, "A VLSI Architecture for the Real Time Computation of Discrete Trigonometric Transforms," *J. VLSI Signal Processing*, vol. 5, pp. 95-104, 1993.

[5] B.G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 32, pp. 1,243-1,245, 1984.

[6] H.S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 35, pp. 1,455-1,461, 1987.
[7] M. Sheu, J. Lee, J. Wang, A. Suen, and L. Liu, "A High Throughput-rate Architecture for 8*8 2D DCT," *Proc. Int'l Symp. Circuits and Systems,* vol. 3, pp. 1,587-1,590, 1993.
[8] M.J. Narasimha and A.M. Peterson, "On the Computation of the Discrete Cosine Transform," *IEEE Trans. Comm.,* vol. 26, pp. 934-936, 1978.
[9] J. Makhoul, "A Fast Cosine Transform in One and Two Dimensions," *IEEE Trans. Acoustics, Speech, and Signal Processing.* vol. 28, pp. 27-34, 1980.
[10] J. Bruguera and T. Lang, "2D DCT Using On-Line Arithmetic," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 3,275-3,278, 1995.
[11] M. Vetterli and H.J. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations," *Signal Processing,* vol. 6, pp. 267-278, 1984.
[12] N. Cho and S. Lee, "Fast Algorithm and Implementation of 2D Discrete Cosine Transform," *IEEE Trans. Circuits and Systems,* vol. 38, pp. 297-305, 1991.
[13] P. Duhamel and C. Guillemot, "Polynomial Transform Computation of the 2D DCT," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 1,515-1,518, 1990.
[14] E. Feig and S. Winograd, "Fast Algorithms for the Discrete Cosine Transform," *IEEE Trans. Signal Processing,* vol. 40, pp. 2,174-2,193, 1992.
[15] C. Chakrabarti and J. JáJá, "Systolic Architectures for the Computation of the Discrete Hartley and the Discrete Cosine Transforms Based on Prime Factor Decomposition," *IEEE Trans. Computers,* vol. 39, pp. 1,359-1,368, 1990.
[16] N.I. Cho and S.U. Lee, "DCT Algorithms for VLSI Parallel Implementation," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 38, pp. 121-127, 1990.
[17] P.Z. Lee and F.Y. Huang, "An Efficient Prime-Factor Decomposition of the Discrete Cosine Transform and Its Hardware Realization," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 20.5.1-20.5.4, 1985.
[18] Y. Chang and C. Wang, "New Systolic Array Implementation of the 2D Discrete Cosine Transform and Its Inverse," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 5, pp. 150-157, 1995.
[19] S. Pan and R. Park, "Unified Systolic Arrays for Computation of the DCT/DST/DHT," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 7, pp. 413-419, 1997.
[20] J. Guo, C. Liu, and C. Jen, "The Efficient Memory-Based VLSI Array Designs for DFT and DCT," *IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing,* vol. 39, pp. 723-733, 1992.
[21] R. Negrini, M. Sami, and R. Stefanelli, *Fault Tolerance through Reconfiguration of VLSI and WSI Arrays,* Cambridge, Mass.: MIT Press, 1989.
[22] L. Breveglieri and L. Dadda, "A Serial-Input Bit-Sliced Convolver," *Proc. Int'l Conf. Circuit Design,* 1988.
[23] L. Breveglieri and V. Piuri, "A Fast Pipelined FFT Unit," *Proc. Int'l Conf. Application Specific Array Processors,* pp. 143-151, 1994.
[24] S.Y. Kung, *VLSI Array Processors.* Englewood Cliffs, N.J.: Prentice Hall, 1988.
[25] H.V. Jagadish, S.K. Rao, and T. Kailath, "Array Architectures for Iterative Algorithms," *Proc. IEEE,* vol. 75, pp. 1,304-1,321, 1987.
[26] *Systolic Signal Processing Systems,* E.E. Swartzlander Jr., ed. New York: Marcel Dekker, 1987.
[27] T.D. Roziner and M.G. Karpovsky, "Multidimensional Fourier Transforms by Systolic Architectures," *J. VLSI Signal Processing,* vol. 4, pp. 343-354, 1992.

Hyesook Lim received the BEng degree from the Department of Control and Instrumentation Engineering, Seoul National University, Seoul, Korea, in 1986, the MS degree from the same school in 1991, and the PhD degree in electrical and computer engineering from the University of Texas at Austin in 1996.

From 1986 to 1989, she was a development engineer at Samsung Hewlett Packard, Seoul, Korea. From 1995 to 1996, she held a part-time research fellowship at IBM, Austin, Texas, working on switch chip design. Since 1996, she has been a member of the technical staff in Advanced Video and Distributed Systems Technology Department, Bell Laboratories, Lucent Technologies, Murray Hill, New Jersey. Her research interests include high-level modeling for channel coding algorithms and VLSI designs for SONET and video processing algorithms. She is a member of the IEEE.

Vincenzo Piuri obtained the PhD degree in computer engineering in 1989 from the Politecnico di Milano, Italy. Since 1992, he has been an associate professor in operating systems at the Politecnico di Milano. His research interests include distributed and parallel computing systems, application-specific processing architectures, DSP architectures, arithmetic units, arithmetic codes, fault tolerance, neural network architectures, and theory and applications of neural techniques for industrial applications. This research has been published in more than 150 papers in book chapters, international journals, and proceedings of international conferences.

He is a member of the ACM, IMACS, INNS, and AEI and a senior member of the IEEE. In the IEEE Instrumentation and Measurement Society, he is the founding chair of the Technical Committee on Emerging Technologies and the Technical Committee on Intelligent Measurement Systems. He is a member of the IMACS Technical Committee on Neural Networks. He is an associate editor of the *IEEE Transactions on Instrumentation and Measurement*.

Earl E. Swartzlander Jr. is a professor of electrical and computer engineering at the University of Texas at Austin, where he holds the Schlumberger Centennial Chair in Engineering. Previously, he was with TRW Defense and Space Systems from 1975 to 1990, where he held positions ranging from staff engineer to laboratory manager and, most recently, was the director of independent research and development for the TRW Defense Systems Group.

His research interests are in application specific processing and the interaction between computer architecture and VLSI technology. These areas involve computer arithmetic, VLSI development, and digital signal processor implementation.

He is currently the hardware area editor for *ACM Computer Reviews*, the computer arithmetic editor for the *Journal of Systems Architecture*, and editor of the Calculators column of the *IEEE Annals of the History of Computing*. Previously, he was the editor-in-chief of the *IEEE Transactions on Computers* and was the founding editor-in-chief of the *Journal of VLSI Signal Processing*.

He obtained his doctorate in computer design with the support of a Howard Hughes Doctoral Fellowship. He is a fellow of the IEEE and is a registered professional engineer. He has been recognized as an Outstanding Electrical Engineer and a Distinguished Engineering Alumnus of Purdue University and a Distinguished Engineering Alumnus of the University of Colorado.