ON THE DESIGN AUTOMATION OF THE MEMORY-BASED VLSI ARCHITECTURES FOR FIR FILTERS

Hwan-Rei Lee,† Chein-Wei Jen,† and Chi-Min Liu‡

ABSTRACT

The design automation of the memory-based VLSI architectures for FIR filters is investigated. This paper intends to give a thorough discussion about the design space and schemes for this subject. The discussed topics include the conditions leading to efficient memory replacement, the formulation space of FIR filters, the considerations in architecture design, and the methods to evaluate architectures. With the research results of these topics, we present a parameterized memory-based architecture and the hardware-speed evaluation formulae. This architecture is characterized by three parameters into which various memory-based architectures can be generated by substituting different values. The presented formulae are formulated as functions of the three parameters and implementation technology. Using the evaluation formulae and through the parameterized architecture, an area-minimized architecture can be synthesized under a speed specification. Based on these results, we develop an automatic synthesis tool with an illustrating example.

1. INTRODUCTION

Finite-impulse-response (FIR) filters are basic processing elements in applications such as video signal processing and audio signal processing. The automatic synthesis of an optimal integrated circuit to handle diverse filtering speed requirements is among the most challenging objectives in CAD research today. To achieve this objective, two problems must be resolved. The first one concerns the selection of architecture types. Various architecture types, such as systolic arrays [1], transversal structure [2], stored-product structure [3], have been proposed for FIR filters of various lengths. Given these various architectures, a suitable optimization criterion is needed to help in the selection of an appropriate architecture for a specified filter length. One conventional optimization criterion is the cost-speed product, such as $A \cdot T$ or $A \cdot T^2$ complexity, where A and T are the area and computing time, respectively. Such a criterion gives a general and theoretical analysis of a prototype of a VLSI architecture, but is not practical or precise enough for use in designing the application-specific integrated circuits (ASICs). A more practical criterion, which is adopted in this paper, is to select a VLSI architecture with minimum hardware cost under a certain speed specification. However, an accurate estimation of the hardware and speed cost of VLSI architectures must be given to support the use of this selection criterion. In fact, the hardware and speed cost vary with the technologies provided by different foundries, such as routing technology and available implementations of functional units. For example, it is hard to estimate whether an architecture with higher routing complexity and fewer arithmetic units is more or less costly than one with lower complexity and more arithmetic units. Hence, it is very difficult to give a fair evaluation of the cost of various types of architectures.

Once a certain type of architecture is selected, the second problem concerns the speed-specific configurations. For example, to reduce hardware costs, fewer multipliers are used in an architecture design for low speed requirements than in one designed for high speed requirements. Hence, an automatic design tool should be able to synthesize different architecture configurations to suit diverse speed requirements. This may entail designing a large number of architecture configurations for an application and then selecting a suitable one among them. However, the synthesis and selection procedure involved in this approach is difficult to carry out.

In this paper, the problems of selecting an appropriate architecture type and of speed-specific configuration are solved in the following ways: *First*, we show that the multipliers in FIR filters can be efficiently replaced by ROMs, hence, the resulting architecture styles, which we call memory-based architectures (MBAs) can be arbitrarily selected without sacrificing general speed and hardware performance. To prove this claim, we show the conditions of linearity and time-invariant input of a linear function can be efficiently implemented by ROMs. Also, it is illustrated that FIR can be formulated to have these two properties and hence MBAs are feasible.

Second, we show that FIR filters can be implemented flexibly using different numbers of memory modules. The hardware and speed cost of each module are lower than those of a multiplier. To achieve the flexibility, we show that a highly efficient memory replacement sometimes causes dif-

[0]This work was supported by the National Science Council of the R.O.C. under grant NSC82-0404-E009-224.

†C.-W. Jen and H.-W. Lee are with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.

‡C.-M. Liu is with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.

Manuscript received June 11, 1993

ferent contents of ROM modules and hence the number of ROM modules is proportional to filter length or wordlength. This paper is intended to show that both highly efficient memory replacement and identical ROM contents can be obtained by a proper algorithm formulation and hence a flexible number of memory modules can be used to realize an FIR filter.

Third, a parameterized MBA is presented. In addition to utilizing the formulation results described above, the design of this architecture takes into account the computing parallelism, memory partitioning and pipelining. The resulting architecture is characterized by three *design parameters*: D, the number of bits to be processed in parallel, K, the memory partition factor, and P, the number of pipeline stages in one processing unit. Different MBA configurations can be obtained by substituting different values for these parameters.

Fourth, hardware-speed evaluation formulae for the parameterized MBA are established for a given speed and hardware cost model of the required elements in an MBA. These elements include memory modules, adders, and shift registers. Since all MBAs have the same types of elements and similar configurations, the selection of an optimized MBA will not be affected much by factors such as routing and placement, and different MBAs can be evaluated using the same hardware-speed evaluation formulae.

Finally, based on the hardware-speed evaluation formulae and the parameterized architecture, an optimized MBA configuration can be easily designed by using computers to search for the best values of the design parameters. Thus, speed-specified configurations can be designed.

The technical content of this paper is organized as follows: In Section 2., efficient memory replacement based on algorithm properties is described. These properties include the time-invariant input and functional linearity. With the results of Section 2., computational units in FIR filters can be efficiently replaced by a flexible number of memory modules. In Section 3., architecture issues involved in implementing the formulation results of Section 2. are considered. These issues include memory partition, adder selection, and pipelining. A parameterized MBA is presented by taking these architecture issues and formulation results into consideration. In Section 4., the evaluation formulae for the speed and hardware cost of the parameterized MBA are established. These formulae can be adapted to different technologies by giving different technology parameters for the components used in the MBA. Therefore, with a given set of technology parameters, different configurations of the proposed parameterized architecture can be evaluated using the same formulae and the optimal configuration which meets the speed requirement can be obtained. A design example and discussion are also given in this section. In section 5., concluding remarks are given and other applications of the technical content of this paper are suggested.

2. ALGORITHM FORMULATION

In this section, the formulation of algorithm for designing MBAs is systematically investigated. Two general conditions that lead to an efficient replacement of ROMs for a combinational circuit are addressed in the first subsection.

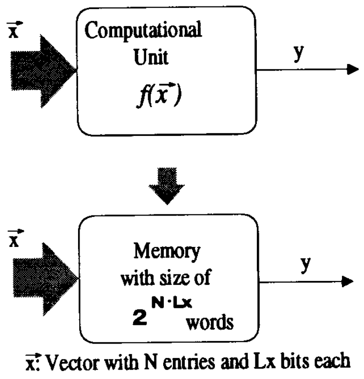

x⃗
Computational Unit
f(x⃗)
y
x⃗
Memory with size of 2^{N·Lx} words
y
x⃗: Vector with N entries and Lx bits each

Figure 1. The basic memory replacement scheme

In the second subsection, two aspects of algorithms that will be used in evaluating algorithm formulations for FIR filters are presented. In the last subsection, various algorithm formulae for FIR filters are derived and analyzed using these conditions and aspects.

2.1. Two Conditions for Memory Reduction

The basic memory replacement model is illustrated in Figure 1, which shows how a combinational circuit with function $y = f(\vec{x})$ can be replaced by a ROM with the addressing lines as the bit lines of $\vec{x}$ and a ROM size of $2^{N \cdot L_x}$ words. In most cases, the ROM size will be too large to be implemented. The ROM size can be reduced, however, whenever one or both of two conditions holds. The first condition is that some inputs of the combinational circuit are time-invariant. To replace a combinational circuit with H input lines, a ROM size of 2^H words is needed. If, however, the signals in some of the H lines, say G lines, are time-invariant, the ROM size can be reduced to 2^{H-G} words.

For example, a multiplier with two operands can be replaced by one ROM with a size of $2^{2 \cdot L}$ words, as shown in Figure 2a, where L is the wordlength of the operands. Such a replacement will induce a ROM area that is larger than that of the original multipliers. However, if one of the operands is time-invariant, as shown in Figure 2b, the ROM size can be reduced to 2^L words.

The other condition for reducing the ROM size is that the function $f(\cdot)$ is linear. In this case, if we represent the input signal as a linear combination of $\vec{x}_i$, i.e., $\vec{x} = \sum_i c_i \cdot \vec{x}_i$, then it follows from the properties of linear functions that

$$y = f(\vec{x}) = f(\sum_i c_i \cdot \vec{x}_i) = \sum_i c_i \cdot f(\vec{x}_i). \quad (1)$$

Different schemes for decomposing $\vec{x}$ will lead to different implementations. For example, we can decompose $\vec{x}$ into the summation of two sub-vectors:

$$\begin{aligned} \vec{x} &= \begin{bmatrix} \vec{x}' \\ \cdots \\ \vec{x}'' \end{bmatrix} = \begin{bmatrix} \vec{x}' \\ \cdots \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \cdots \\ \vec{x}'' \end{bmatrix} = \vec{x}_1 + \vec{x}_2 \\ y &= f(\vec{x}_1 + \vec{x}_2) = f(\vec{x}_1) + f(\vec{x}_2). \end{aligned} \quad (2)$$

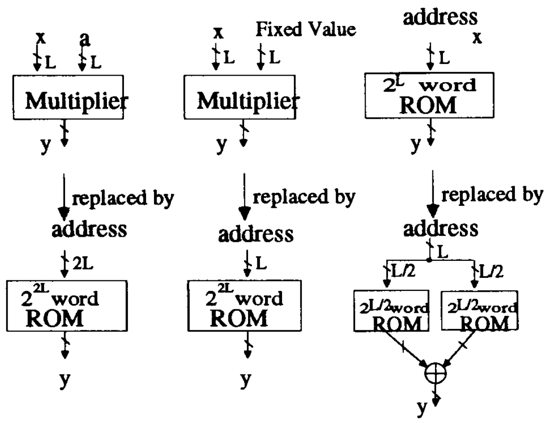


Figure 2. The basic memory replacement schemes applied for multiplications

For this decomposition, two ROM modules are needed to implement $f(\vec{x}_1)$ and $f(\vec{x}_2)$, and the two partial results are summed to get y. In (2), since some entries of $\vec{x}_1$ and $\vec{x}_2$ are zeroes and hence time-invariant, each ROM can be reduced by the scheme described above. In this paper, this scheme is called *memory partition*. To illustrate the scheme, the multiplier in Figure 2 is again taken as an example. In Figure 2c, if we decompose $\vec{x}$ as in (2), the multiplier can be replaced by two memory modules and one adder. The total ROM size is $2 \cdot 2^{\frac{L}{2}}$ words with some overhead in adders. For a value of L of eight, the hardware-speed cost for the two ROMs and the adder has been shown to be more efficient than that of most multiplier architectures [4]. In general, if the original ROM size is 2^L and the ROM module is uniformly partitioned into q submodules, then the total size will be reduced to $q \cdot 2^{\frac{L}{q}}$ words, with some overhead of adders to sum the partial results from each submodule. Therefore, the memory partition scheme can reduce the memory size exponentially.

2.2. Two Aspects for Evaluating Algorithms

In this subsection, *output wordlength* and the *space-time-commutative property* are introduced for use in evaluating the performance of algorithm formulae in the succeeding discussions. Consider an FIR filter with tap weights a_i for an input sequence x_j. This filter can be represented as follows:

$$y_k = \sum_{i=0}^{N-1} a_i \cdot x_{k-i}, \tag{3}$$

where k is the time index starting from zero. In this equation, the number of multiplications required to obtain each output y_k is equal to the tap length N. A direct form implementation of the FIR filter is shown in Figure 3. In most applications, the tap weights are time-invariant. Therefore, the dotted block shown in Figure 3 can be replaced by one ROM with a size of $(2^{N \cdot L_x}) \cdot L_y$, where L_x and L_y are the wordlengths of operands x_i and y_k, respectively. If full

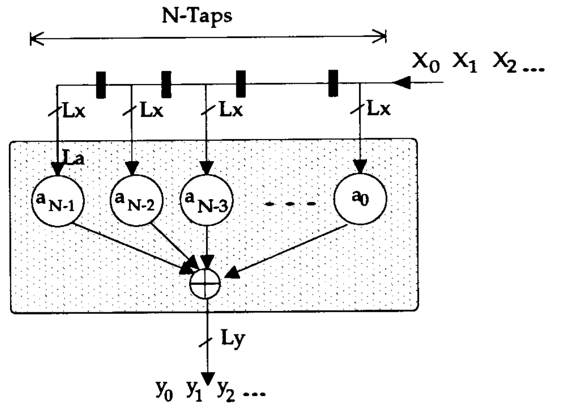


Figure 3. The direct form implementation of an FIR filter

precision of output is necessary, the output wordlength of y would be $(L_x + L_a + \log_2 N)$. Such a ROM size is too large to be implemented in most applications. To reduce the ROM size, a memory partition scheme based on the linear property of FIR filters can be adopted. If the input data sequence $\{x_i\}$ in the N-taps of delays in Figure 3 is decomposed into q subsets, a ROM module can be partitioned into q submodules, and the total ROM size can be reduced from $(2^{N \cdot L_x}) \cdot L_y$ to $(q \cdot 2^{\frac{N}{q} \cdot L_x}) \cdot L_y$. In the reduction process, the output wordlength L_y or the required precision of each ROM is assumed to be a constant. In general, the output wordlength of each submodule can also be reduced through the partitioning. However, the reduction in output wordlength depends on a suitable decomposition of input data $\{x_j\}$. The algorithm formulae described in the next subsection can factor the FIR equation to reduce not only the exponent of $(2^{N \cdot L_x}) \cdot L_y$ but also the L_y of factorized ROMs. This is one reason why we introduce these algorithm formulae in addition to the memory partition scheme.

Another property that may be used to evaluate the algorithm formulae is the *space-time-commutative* property. The space-time-commutative property is the property that a reduction (or an increase) of a factor of hardware results in an increase (or a reduction) of approximately the same factor in computation time. If an algorithm formula has this property, based on a particular type of hardware realization, then the hardware architecture for that algorithm can be flexibly modified to meet different speed requirements. For example, suppose that an algorithm mostly consists of multiplications. If multipliers are used to realize these multiplications, then the number of multipliers required in the architecture varies with different speed requirements. If the required speed is low, fewer multipliers can be used, thereby reducing the hardware cost. If, on the other hand, the required speed is high, a larger number of multipliers should be used. Since one objective of this paper is to design a memory-based architecture that is easily tuned to various speed requirements, this property is essential. The attainment of this property depends on a cautious examination of algorithm formulae and hardware units. The example discussed above shows that a realization based on multipliers possesses the space-time-commutative property. However, ROM tables are not as flexible as multipliers are. For example, the ROM in Figure 2a can be used to realize multiplications with any operand values while that in Figure 2b can only realize multiplications with the operand a. Hence, if the ROM in Figure 2b is adopted as a hardware unit, algorithms have to be formulated to consist of multiplications with only an operand a to attain the space-time-commutative property. This example implies that a highly efficient memory replacement would seriously restrict the range of feasible algorithm formulae. In the next subsection, an algorithm formulation that yields both highly efficient memory replacement and the space-time-commutative property is presented.

2.3. Derivation and Analysis of Various Formulations

In this subsection, time-invariant input and functional linearity are used to derive various formulations for MBAs, and output wordlength and the space-time-commutative property are used to analyze these formulae.

If (3) is represented as an inner-product, it follows that

$$\begin{aligned} y_k &= \sum_{i=0}^{N-1} a_i \cdot x_{k-i} = \vec{a}^T \cdot \vec{x}_k \\ &= [a_0, a_1, \cdots a_{N-1}] \begin{bmatrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-(N-1)} \end{bmatrix} \end{aligned} \tag{4}$$

where $\vec{a} = [a_0, a_1, \cdots a_{N-1}]^T$ and $\vec{x_k} = [x_k, x_{k-1}, \cdots x_{K-(N-1)}]^T$. The superscript, T, denotes the transpose of a vector. Since the function is linear, the input vector can be decomposed to obtain various formulae. One simple approach to doing this is the Stored-Product [3], in which $\vec{x}$ is decomposed at the word level, as follows:

$$\vec{x}_k = \begin{bmatrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-(N-1)} \end{bmatrix} \tag{5}$$

$$\begin{aligned} &= \begin{bmatrix} x_k \\ 0 \\ 0 \\ \vdots \end{bmatrix} + \begin{bmatrix} 0 \\ x_{k-1} \\ 0 \\ \vdots \end{bmatrix} + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ x_{k-(N-1)} \end{bmatrix} \\ &= \vec{x}'_k + \vec{x}'_{k-1} + \cdots + \vec{x}'_{k-(N-1)}. \end{aligned} \tag{6}$$

If the decomposing scheme in (6) is substituted into (4), the output can be obtained from the linear combination of the N separated terms, i.e.,

$$\begin{aligned} y_k &= \vec{a}^T \vec{x}_k \\ &= \vec{a}^T (\vec{x}'_k + \vec{x}'_{k-1} + \cdots + \vec{x}'_{k-(N-1)}) \end{aligned}$$

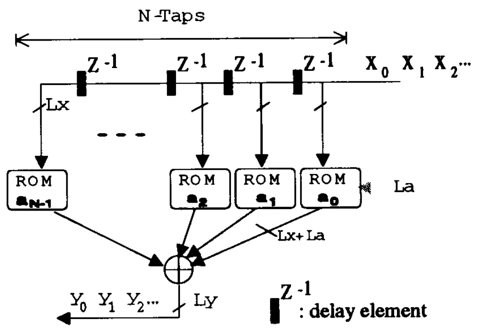


Figure 4. Architecture results from word-level decomposition

$$= \vec{a}^T \begin{bmatrix} x_k \\ 0 \\ \vdots \end{bmatrix} + \vec{a}^T \begin{bmatrix} 0 \\ x_{k-1} \\ \vdots \end{bmatrix} + \cdots + \vec{a}^T \begin{bmatrix} 0 \\ \vdots \\ x_{k-(N-1)} \end{bmatrix}. \tag{7}$$

Each term $\vec{a}^T x_{\vec{k-i}}$ in (7) can be implemented by a ROM module for $i = 0, 1, \cdots, N-1$ as shown in Figure 4. Since only L_x entries of the vector $\vec{x}'_i$ are nonzero, the size of each ROM module is $2^{L_x} \cdot L_y{}'$ bits. Hence, the total ROM size in Figure 3 is $(N \cdot 2^{L_x}) \cdot L'_y$. Also, the full precision length of L'_y is $L_x + L_a$, which is smaller than the $(L_x + L_a + \log_2 N)$ of L_y. This justifies the claim made earlier that proper algorithm formulation can not only exponentially reduce the required ROM size but also reduce the output wordlength. A comparison of Figure 3 and Figure 4 shows that the multiplication on each tap is replaced by one ROM module.

However, the main drawback of this architecture is that it is not space-time-commutative and hence the number of ROM tables must be equal to the tap length N. For considering the space-time-commutative property, (7) is reformulated as

$$y_k = a_0 x_k + a_1 x_{k-1} + \cdots + a_{N-1} x_{k-(N-1)}. \tag{8}$$

From (8), it can be seen that the contents of each ROM module are coded based on $a_i x_{N-i}$ for $i = 0, 1, \cdots, N-1$. Since different ROM tables are dedicated to different tap values a_i, these ROM modules are not space-time-commutative. Consequently, hardware costs cannot be reduced through the space-time-commutative property when the tap length is long or the required speed is low.

Another approach to decompose $\vec{x}$, one that uses the concepts of "Distributed Arithmetic"[5, 6], is to represent the operand $\vec{x}$ at the bit-level, as follows:

$$\vec{x_k} = \begin{bmatrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-(N-1)} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{j=0}^{L_x-1} x_k^j \cdot 2^j \\ \sum_{j=0}^{L_x-1} x_{k-1}^j \cdot 2^j \\ \vdots \\ \sum_{j=0}^{L_x-1} x_{k-(N-1)}^j \cdot 2^j \end{bmatrix}, \tag{9}$$

where x^j denotes the jth bit of data x. Here, x is represented in an unsigned binary form. The extension for two's complement is discussed in the Appendix. We now represent (9) as a linear combination of bit-level vectors,

$$\begin{aligned} \vec{x}_k &= \begin{bmatrix} x_k^{L_x-1} \\ x_{k-1}^{L_x-1} \\ \vdots \\ x_{k-(N-1)}^{L_x-1} \end{bmatrix} \cdot 2^{L_x-1} \\ &+ \cdots + \begin{bmatrix} x_k^1 \\ x_{k-1}^1 \\ \vdots \\ x_{k-(N-1)}^1 \end{bmatrix} \cdot 2^1 + \begin{bmatrix} x_k^0 \\ x_{k-1}^0 \\ \vdots \\ x_{k-(N-1)}^0 \end{bmatrix} \cdot 2^0 \\ &= \vec{x}^{L_x-1} \cdot 2^{L_x-1} + \cdots + \vec{x}^1 \cdot 2^1 + \vec{x}^0 \cdot 2^0 \\ &= \sum_{j=0}^{L_x-1} \vec{x}_k^j \cdot 2^j, \end{aligned} \tag{10}$$

where $\vec{x}^j$ denotes the vector that is composed of the j th bit of each entry of $\vec{x}$. Substituting (10) into (4) and using the linear property yields

$$\begin{aligned} y_k &= \vec{a}^T [\sum_{j=0}^{L_x-1} \vec{x}_k^j \cdot 2^j] \\ &= \sum_{j=0}^{L_x-1} [\vec{a}^T \vec{x}_k^j] \cdot 2^j. \end{aligned} \tag{11}$$

The resulting architecture, which realizes each $\vec{a}^T \vec{x}_k^j$ by one ROM, is shown in Figure 5. In this figure, each input sample, x_k, is decomposed and fed to each ROM module to produce L_x results. These results are weighted by 2^j and summed to get the outputs, y_k. Note that the weights are all powers of two, so only shifts of bit position are needed. These shifts can be implemented by suitably hardwired lines. In this architecture, there are L_x ROM modules. Therefore, a total of $L_x \cdot 2^N \cdot L_y''$ bits are needed, where the full precision length of L_y'', is $(L_a + \log_2 N)$. The functions of the ROM modules in (11) are all identical and equal to $f(\vec{x}_k) = \vec{a}^T \vec{x}_k$ in (11). In other words, any ROM module can take the place of any other module, which implies the formula in (11) is *space-time commutative*. Figure 6 shows an architecture that realizes (11) by only one ROM time-serially. If Figure 6 is compared with Figure 5, four primary features can be found. First, the L_x ROM modules are replaced by a single module. Second, since the input of each ROM module like that in Figure 5 consists of the same significant bits of input sequence $\{x_i\}$ and the input is assumed to proceed data-serially and bit-serially,

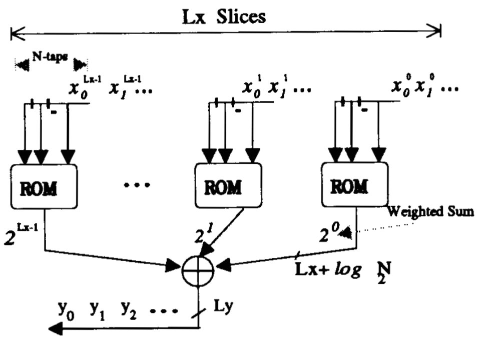
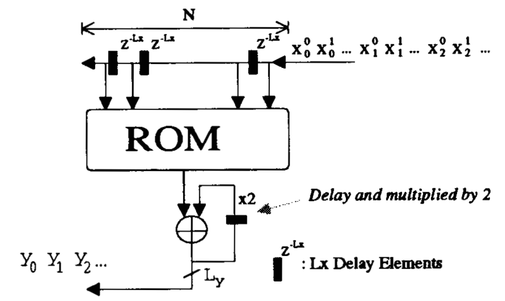


Figure 5. Architecture results from bit-level decomposition



N
$z^{-Lx}\ z^{-Lx}$
z^{-Lx}
$x_0^0\ x_0^1 \ldots x_1^0\ x_1^1 \ldots x_2^0\ x_2^1 \ldots$
ROM
x2
Delay and multiplied by 2
$Y_0\ Y_1\ Y_2 \ldots$
L_y
z^{-Lx} : Lx Delay Elements

Figure 6. Bit-serial architecture results from bit-level decomposition: the DA approach

each black block on the input line of Figure 6 should consist of L_x delay elements to ensure functional correctness. Third, the weighted adder in Figure 5 is now the shift-and-accumulation adder. Fourth, the computation speed of Figure 6 is L_x times slower than that of Figure 5.

The architectures in Figure 5 and Figure 6 are two extreme cases, with the maximal and minimal number of memory modules, respectively. In general, the number of ROM modules for different hardware-speed requirement is flexible. If we let the structure in Figure 6 represent a slice of a memory module, then if D slices of ROM modules are used, we can set j to be equal to $\frac{L_x}{D} \cdot j' + j''$, where $j' = 0, 1, \cdots, D-1$ and $j'' = 0, 1, \ldots, \frac{L_x}{D} - 1$. Substituting this factorization into (11) yields following equation:

$$y_k = \{\sum_{j'=0}^{D-1} [\sum_{j''=0}^{\frac{L_x}{D}-1} (\vec{a}^T \vec{x}_k^{j' \cdot \frac{L_x}{D} + j''} \cdot 2^{j''})] \cdot 2^{j' \cdot \frac{L_x}{D}}\}. \tag{12}$$

An architecture that realizes (12) is illustrated in Figure 7. In this figure, each ROM module is used to realize one term $\vec{a}^T \vec{x}_k^{j' \cdot \frac{L_x}{D} + j''}$; the modules are identical to those in Figure 5 and Figure 6. The $\frac{L_x}{D}$ terms in the bracket are realized time-serially by one slice, which consists of one ROM module and accumulator, as marked in Figure 7. The D terms in the braces are realized by D slices in parallel.

Equation (12) provides high time-sharing potential and efficient memory replacement. Hence, in the next section,

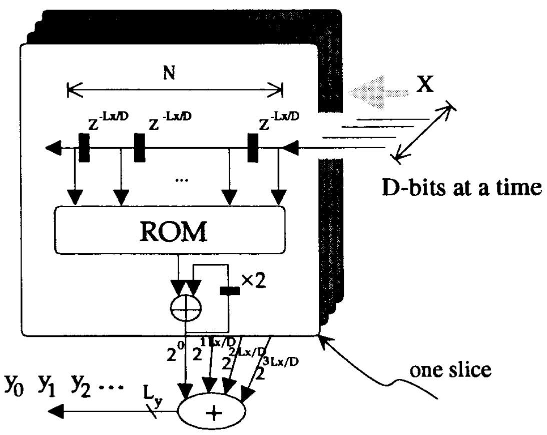


Figure 7. Digit-serial architecture

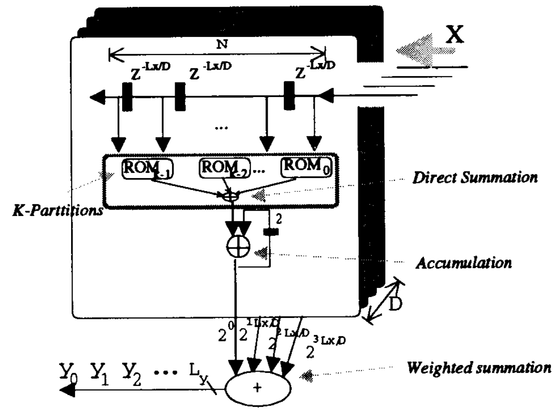


Figure 8. Digit-serial architecture with memory partitions

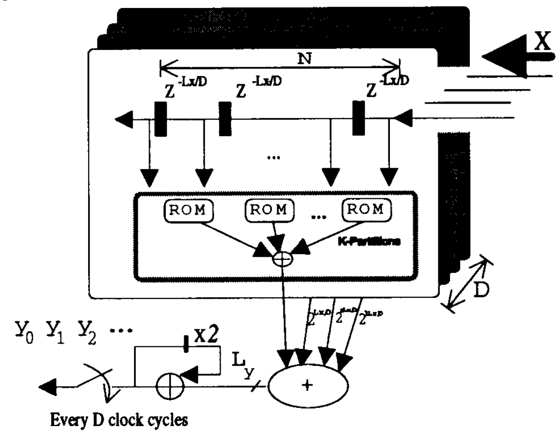


Figure 9. Digit-serial architecture with accumulation rearrangement

we shall adopt it to design a parameterized MBA.

3. ARCHITECTURE DESIGN

In this section, we shall use the results of the last section as a basis for considering three issues involved in the architecture design of MBAs. In the first of the three subsections that follow, a way of further reducing the memory size through a memory partition scheme is presented. In the second subsection, the implementation of additions and the arrangement of the addition sequence are addressed, so that MBAs with low hardware cost and high pipelinability can be designed. In the final subsection, our results concerning algorithm formulation and architecture design are integrated to develop a parameterized MBA that can be easily tuned to various hardware-speed requirements.

3.1. Memory Partitioning

As described in the last section, implementing the inner product term, $\vec{a}^T \vec{x}^{j' \cdot \frac{L_x}{D} + j''}$, in (12) by a ROM module with N input lines results in a memory of size $2^N \cdot L_y''$. Since the function $\sum \vec{a}^T \vec{x}^{j' \cdot \frac{L_x}{D} + j''}$ is linear, the memory partition scheme in Section 2.1. can be applied to reduce the memory size. Applying this scheme to partition each ROM module in Figure 7 into K submodules results in the architecture illustrated in Figure 8. The total ROM size is exponentially reduced from $D \cdot 2^N \cdot L_y''$ to $D \cdot K \cdot 2^{\frac{N}{K}} \cdot L_y''$ at the expense of a multi-operand addition to sum up the partial results from these submodules.

In the discussions above, the number of submodules K is assumed to be a factor of filter length N such that $\frac{N}{K}$ is an integer. In general, K is not necessarily a factor of N. In cases where it is not, the ROM size of each submodule will not be equal. However, since a larger ROM size will result in a longer table lookup time and the ROM size is exponentially proportional to the number of input addressing lines, the memory size of the submodules should be as uniform as possible. If we take $N = 18$ and $K = 5$ as an example, to achieve higher speed and lower cost, N should be partitioned into five sets with line numbers of $(3,3,4,4,4)$ instead of $(3,3,3,3,6)$ or $(3,3,3,4,5)$. In other words, there would be only two kinds of line numbers: $\lceil \frac{N}{K} \rceil$ and $\lfloor \frac{N}{K} \rfloor$, where $\lceil \frac{N}{K} \rceil$ is the smallest integer that is larger than $\frac{N}{K}$, while the number with $\lfloor \frac{N}{K} \rfloor$ is the largest one smaller than $\frac{N}{K}$. The number of submodules with $\lceil \frac{N}{K} \rceil$ is $(N \ mod \ K)$, while the number with $\lfloor \frac{N}{K} \rfloor$ is $(K - N \ mod \ K)$, where $(N \ mod \ K)$ is the residue of $\frac{N}{K}$.

3.2. Implementation and Arrangement of Addition Sequence

In Figure 8, there are three kinds of additions: direct summation, accumulation, and weighted summation. Each slice in Figure 8 involves one accumulator. Since addition is associative, i.e. $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$, all of these accumulators can be moved to the final stage and merged as shown in Figure 9. Such an arrangement would reduce the hardware cost in two ways. First, the wordlength of the hardware implementing the direct summation and weighted summation can be reduced, because the output wordlength increase with the times of accumulations. Second, the number of accumulators is reduced from D to one.

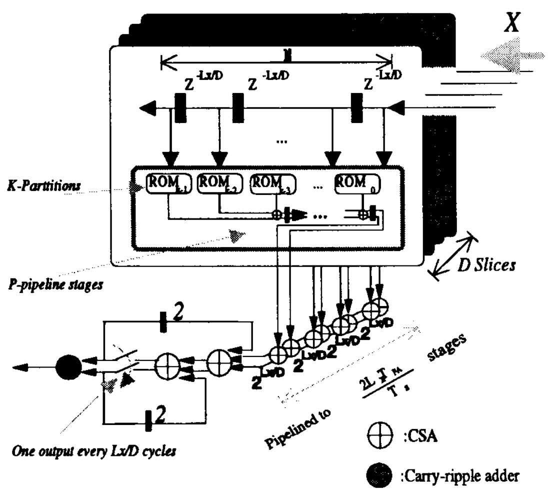


Figure 10. The parameterized MBA

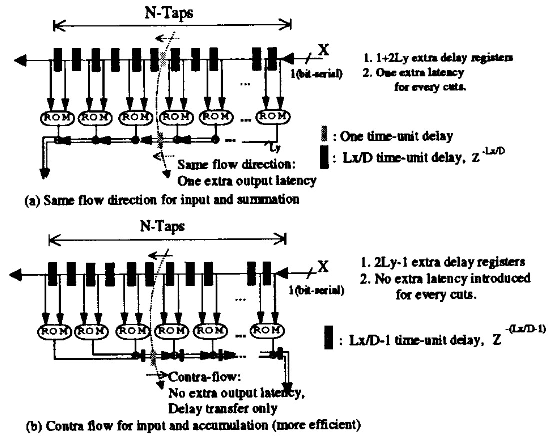

N-Taps
1. 1+2Ly extra delay registers
2. One extra latency for every cuts.
: One time-unit delay
: Lx/D time-unit delay
Same flow direction:
One extra output latency
(a) Same flow direction for input and summation
1. 2Ly-1 extra delay registers
2. No extra latency introduced for every cuts.
: Lx/D-1 time-unit delay
Contra-flow:
No extra output latency,
Delay transfer only
(b) Contra flow for input and accumulation (more efficient)

Figure 11. Dataflow arrangement and cut-set retiming

In Figure 9, aside from the implementation of the ROMs, the implementation of the adders is most critical in determining hardware cost and speed. In this paper, carry-save adders (CSAs) [7] are adopted as the basic addition unit. It is known that managing the carry propagation in an adder dominates the hardware and speed cost of the adder. There are basically three types of adders [7]: carry-ripple adders, carry-look-ahead adders and carry-saved adders. Carry-ripple adders propagate the carry stage-by-stage and hence consume much time to complete an addition. On the other hand, carry-look-ahead adders implement the carry propagation through extra combinational circuits and hence achieve high speed at a high hardware cost. Instead of giving one addition result, as carry-ripple adders and carry-look-ahead adders do, CSAs [7, 8] avoid the carry propagation by giving two partial results, a method that makes them efficient in both hardware and speed cost. For the addition of two operands, CSAs cannot be applied, because the two partial results have to be summed together to get the final result. Since the additions in Figure 9 have multiple operands, they can be implemented by a series of CSAs and only one adder in the final stage is needed to add the two partial results of CSAs for the final result as shown in Figure 10. Also, since one slice in this figure is activated every $\frac{L_x}{D}$ cycles to produce one output datum, the speed requirement of this final addition is not very critical when L_x is greater than D. Therefore, low-speed and low-cost adders like carry-ripple adder may be used in this case as illustrated in Figure 10. In Figure10, accumulations are also implemented by two CSAs. Thus, the clock cycle time is limited by the delay time of the two CSAs within this accumulation, which is independent of the wordlength of the operands.

In Figure 10, the sequence in each slice is arranged to take pipelining into consideration. It can be checked from Figure 8 that the clock cycle time of this architecture is the ROM access time and the addition time of the direct summation, accumulation, and weighted summation. Pipelining can be applied to decrease the cycle time through a scheme called cut-set retiming [9]. The direct summation in Figure 8 can be implemented by the addition sequence of CSAs either in the direction of input data flow or in the opposite direction, as shown in Figure 11(a) and (b). According to the delay transfer rules in [9], the same number of delay elements can be moved from all the inbound edges to the outbound edges of a cut line without modifying the system's behavior. Consider the two data flow arrangements in Figure 11. The cut line applied on Figure 11(a) leads to the insertion of delay elements for all the edges that cross the line. Thus, one extra delay is introduced. In Figure 11(b), the data flow direction for the tapped input, x, and accumulation are arranged contrariwise, so that the delay on the inbound edge (tapped x) is transferred to the outbound edge (accumulation). Therefore, no extra delay time is introduced to obtain pipelining. Thus, the contra data flow is more efficient and so it is adopted in Figure 10. Since that the wordlength of input data is 1 bit while that of accumulation data is $2 \cdot L_y$, an extra $(2 \cdot L_y - 1)$ delay elements will be introduced for each cut-set. Also, the weighted summation sequence should be pipelined to meet the desired sampling period T_s. Since there are $2 \cdot D$ stages of CSAs, the CSAs should be pipelined into L stages to meet the following constraint:

$$\frac{2 \cdot D \cdot T_{FA}}{L} \leq \frac{T_s}{L_x/D}$$

or,

$$L \geq \frac{2 \cdot L_x T_{FA}}{T_s}.$$

Thus, the weighted summation is pipelined into $\lceil \frac{2 \cdot L_x T_{FA}}{T_s} \rceil$ stages.

In Figure 10, for generality and to obtain flexibility in pipelining, the addition sequence is evenly cut into P stages. A higher value of P leads to more pipeline stages and shorter cycle time, but more delay elements are needed. The pipelining factor, P, will serve as a design parameter for our parameterized MBA.

3.3. The Parameterized MBA

On the basis of the foregoing discussion, we now propose the parameterized MBA shown in Figure 10. This parameterized MBA is composed of D slices, each of which processes one bit of input data x in one clock cycle. Thus, D bits are processed at a time and two partial results are obtained from each slice in each cycle. These partial results are produced from the K partitioned ROM submodules and summed through $(K-2)$ stages of CSAs. The $(K-2)$ stages of CSAs are pipelined into P stages, as depicted in the shaded rectangle in Figure 10. These partial results are weighted-summed through a series of CSAs as depicted at the bottom of Figure 10. Since there are two partial results produced in each slice, two stages of CSAs are used for each slice in the series. As for the accumulating section, two stages of CSAs are applied to perform the accumulation efficiently. Finally, the final carry-ripple addition on the lower left of Figure 10, as discussed above, is activated every $\frac{L_x}{D}$ clock cycles.

To sum up, the MBA is characterized by three parameters: the digit size, D; the number of partitioned memory submodules, K; and the number of pipeline stages, P. Various configurations of MBAs can be obtained by substituting different values into the three parameters.

4. HARDWARE-SPEED EVALUATION

In this section, the hardware-speed cost of the MBA introduced above is formulated based on the three parameters and the hardware-speed cost of the basic cells. Through the formulae presented here, the hardware and speed cost of the parameterized MBA can be numerically described in a way that takes into account the architecture structures and implementation technology. Also, a CAD tool is developed in Section 4.3. to search for the values of parameters so that the hardware cost is minimized for a particular a speed specification.

The three basic elements in the parameterized MBA are the ROM, full adders, and delay elements. The delay time and hardware cost of the three elements can be found from the cell library that will be used to implement the MBAs. Here, they are represented as the parameters tabulated in Table 1.

Table 1. Table for technology parameters

Technology Parameters	Delay time	Cost (gates)
ROM	$T_{mem}(n, W_o)^1$	$C_{mem}(n, W_o)$
Full Adder	T_{fa}	C_{FA}
Delay	T_{delay}	C_{delay}

^1n : the number of addressing lines. W_o: output wordlength.

The terms $T_{mem}(n, W_o)$ and $C_{mem}(n, W_o)$ in this table mean that the time and cost of the memory is determined by the number addressing lines and the output wordlength. These terms will be used to develop the hardware-speed formulae in this section.

4.1. Cost Evaluation

To evaluate the cost of the parameterized MBA, we shall discuss the costs of the three basic components individually. We begin with the cost of the memory modules. In Figure 10, there are D slices in each module and each slice has K memory submodules. As described in Section 3.1., the number of addressing lines is either $\lceil\frac{N}{K}\rceil$ or $\lfloor\frac{N}{K}\rfloor$. The number of submodules with lines $\lceil\frac{N}{K}\rceil$ is $(N \bmod K)$ and the number with $\lfloor\frac{N}{K}\rfloor$ is $N-(N \bmod K)$. Therefore, the total cost of the memory modules can be formulated as follows:

$$Cost_{Memories} = D\cdot[n_1\cdot C_{mem}(\lceil\frac{N}{K}\rceil, L_a+\lceil\log_2\lceil\frac{N}{K}\rceil\rceil) \\ +n_2\cdot C_{mem}(\lfloor\frac{N}{K}\rfloor, L_a+\lceil\log_2\lfloor\frac{N}{K}\rfloor\rceil)] \quad (13)$$

where $n_1 = N \bmod K$, $n_2 = K-(N \bmod K)$. The wordlength of each ROM submodule is defined by the coefficient wordlength L_a and its number of addressing lines.

We now consider the cost of the adders. Each CSA consists of WL full adders. The total cost of the full adders in the parameterized MBA can thus be formulated as follows:

$$Cost_{Adders} = [D\cdot(K-2)\cdot WL + 2\cdot D\cdot WL]\cdot C_{FA} \quad (14)$$

where $WL=\min\{L_y, L_a+\log_2 N\}$, L_y corresponds to the desired output wordlength, and $L_a+\log_2 N$ is the maximum possible wordlength for the linear summation in each slice. The smaller one of the two is chosen to fit the requirements of particular specifications. In (14), the first term corresponds to the $(K-2)$ stages of CSAs used for summing the K partial results in each of the D-slices, and the second term corresponds to the CSAs and accumulators in the lower part of Figure 10, which are used to sum the partial results from the D slices. Although it is possible to reduce the wordlength and hence the cost for each adder, we here keep WL a constant to obtain a regular design.

We now turn to the cost of the delay elements. Most of the delays are introduced by cut-set retiming. Originally, there are $[L_x\cdot(N-1)]\cdot D$ delay elements for the tapped delay line in Figure 9. Every cut-set applied in each slice introduces an extra $(2\cdot WL-1)$ elements, for a total of $D\cdot P\cdot(2\cdot WL-1)$. Other delay elements are from the pipelining of the weighted summation. If the weighted summation is pipelined into $\lceil\frac{2\cdot L_x T_{FA}}{T_s}\rceil$ stages, as discussed in Section 3.2., the number of delay elements is $2\cdot WL\cdot\lceil\frac{2\cdot L_x T_{FA}}{T_s}\rceil$. Thus, the total cost of the delay elements becomes

$$Cost_{Delay} = \\ D\cdot L_x\cdot(N-1)+D\cdot P\cdot(2\cdot WL-1) \\ +\lceil\frac{2\cdot T_{FA}\cdot L_x}{T_s}\rceil\cdot 2\cdot WL. \quad (15)$$

The total hardware cost of the parameterized MBA is formulated as follows:

$$Total\_cost = \\ Cost_{Memory}+cost_{Adders}+Cost_{Delay}. \quad (16)$$

4.2. Speed Evaluation

Since each data sample is processed in parallel within D slices and in serial within each slice, the sample period can be estimated as the time for the serial processing, i.e., $\frac{L_x}{D}$

cycles. Therefore, in Figure 10, the clock cycle time is limited by the ROM access time and addition time within a pipelined stage. It then follows that

$$Estimated\ sample\_period = \frac{L_x}{D} \cdot (T_{mem}(\lceil \frac{N}{K} \rceil, \lceil \log_2 \lceil \frac{N}{K} \rceil \rceil) + \lceil \frac{K}{P} \rceil \cdot T_{FA}) \quad (17)$$

where $T_{mem}(\lceil \frac{N}{K} \rceil, \lceil \log_2 \lceil \frac{N}{K} \rceil \rceil)$ is the access time for the memory module and $\lceil \frac{K}{P} \rceil \cdot T_{FA}$ corresponds to the delay time of the CSAs within $\lceil \frac{K}{P} \rceil$ stages.

4.3. Design Example

Equations (15)-(17) were used to develop a CAD tool, and with the inputs being the filter length, N; input wordlength, L_x; coefficient wordlength, L_a; and desired sampling period, T_s. Also, the technology values of the terms in Table 1 must be available.

Table 2. Models for technology parameters

$T_{delay} = 2.2\ ns$	$C_{delay} = 5.25\ gates$
$T_{FA} = 1.1\ ns$	$C_{FA} = 6.5\ gates$
$T_{mem}(n, WL) = 5 + WL$	$C_{mem} = 2^n \cdot \alpha \cdot WL$

where the factor α is the cost ratio between one memory cell and one two-input $NAND$ gate.

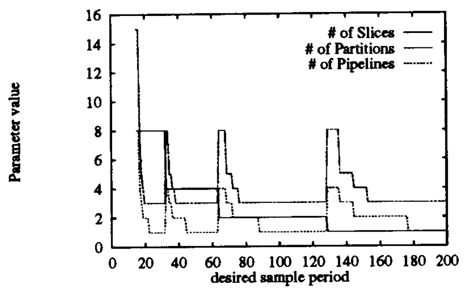
Here, the 1.0μm CMOS gate array library described in [10] and the hardware model in [11] are adopted, as listed in Table 2. The output of the CAD tool includes digit size, D; the number of partitioned memory submodules, K; the number of pipeline stages, P; the input lines of the memory submodules in a slice; the estimated sample period; and the estimated gate counts. The object of the tool is to find the values of these design parameters so that the hardware cost of the architecture is minimized under the specification of the sample period, that is:

For all (D, K, P)
minimize the *Total_cost* in (16)
Subject to
the *Estimated sample_period* in (17) $\leq T_s$

Table 3 gives a design example that shows the input specification and the design results of the tool.

The curves in Figure 12 shows how the optimal values of the parameters vary with different speed specifications. As shown in this figure, most of the design space comes from the bit level parallelism, D. When very high speed is required, the number of memory partitions and the digit size degenerate to 15 and 8, respectively. That is, a fully-extended architecture similar to the bit-level form in [12] results. On the other hand, when cost is the major concern, the digit size is reduced to 1 and the designed architecture is similar to that in Figure 7. Also, in Figure 12, it can be seen that the curve is not continuous and the optimal values of the parameters do not vary with the desired sampling period in some ranges; for example, 50ns - 60ns. This indicates that the same architecture may be generated for different speed requirements.

When different technologies, such as fully custom layout, standard cell and gate arrays, are used, the cost of the memory cells would vary greatly compared with the cost



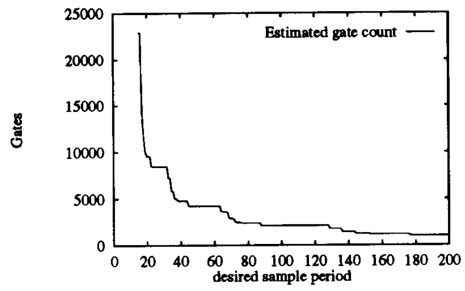


Figure 12. Optimized parameter values for different desired sample periods

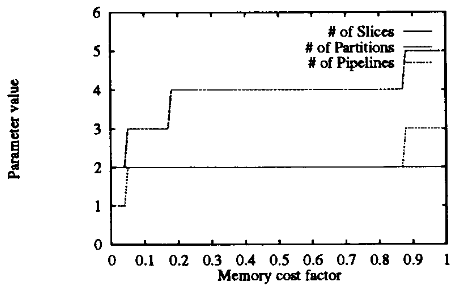
the logic gates. To inspect how the memory cost factor α in Table 1 affects the design parameters, the optimal values are plotted with respect to the cost factor in Figure 13. The memory cost factor (ranging from 0.01 to 1 in the figure) is defined as the ratio of the hardware cost of one memory cell to that of a two-input NAND gate. As shown in the figure, the memory cost factor affects mostly the number of partitions and pipeline stages. A higher memory cost factor results in finer partitions of the memory and more pipeline stages. These curves are drawn under the same specification as that in Table 3.

Filter length:	*15*
Input wordlength:	*8*
Coefficient wordlength:	*8*
Output wordlength:	*18*
Desired sample period :	*60*
Estimated gate counts :	2245.92
D (Number of slices):	2
K (Number of partitions):	2
P (Number of pipeline cuts):	1
The estimated resulting sample period :	55.3

Table 3. Parameters of a design example

5. CONCLUDING REMARKS

The preceding sections have developed an approach to automating the design of memory-based VLSI architectures for FIR filters. The automation is based on the exploration



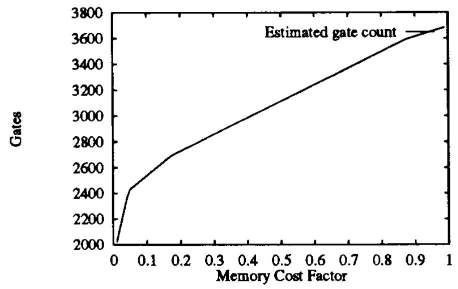


Figure 13. Optimized parameters values for different memory cost factors

of the design space and schemes for efficient memory replacement, algorithm formulation, architecture design, and evaluation method. Various schemes and design considerations were integrated to produce, a parameterized MBA that can easily be tuned to various hardware-speed requirements. This parameterized MBA is characterized by three design parameters. Differently configured MBAs result from specifying different values for these parameters. Also, hardware-speed evaluation formulae were established based on the required elements in MBAs. These elements include ROM, adders and shift registers. These formulae and a cell library of a target technology can be used to design an optimally configured MBA by searching for the best values of the design parameters with the aid of a computer.

The research results can also be extended to IIR and multidimensional filters by decomposing multidimensional filtering into the summation of multiple inner products. Moreover, as we have shown in [13, 14, 15], transformations like Discrete Fourier Transform, Discrete Cosine Transform, and Discrete Sine Transform can be formulated as a convolution form. Since the formulation of convolution is the same as that of FIR filters, the speed-specified design method presented in this paper can also be applied to realize such transformations.

APPENDIX: TWO'S COMPLEMENT CONSIDERATIONS

In (11), $\vec{x}$ is represented in an unsigned binary form. That is, if L_x is eight, the dynamic range of x_k is from 0 to $2^8-1=255$. However, in many applications, two's complement is

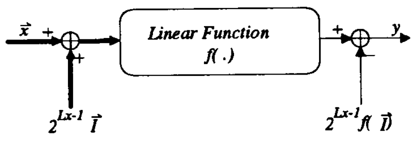


Figure 14. Offsetting inputs for dealing with two's complements

used to represent both positive and negative numbers. To modify the algorithm in this paper for two's complement form, $\vec{x}_k$ can be shifted by adding a value of $2^{L_x-1}\cdot\vec{I}$ so that the dynamic range of each entry of $\vec{x}_k$ is shifted from $(-2^{L_x-2}\sim 2^{L_x-2}-1)$ to $(0\sim 2^{L_x-1})$, where $\vec{I}$ is defined as $[1,1,\cdots,1]^T$. With this formula, $\vec{x}$ is reformulated as $\vec{x}+2^{L_x-1}\cdot\vec{I}-2^{L_x-1}\cdot\vec{I}$ and fed into a linear function $f(\cdot)$:

$$\begin{aligned} y &= f(\vec{x}) \\ &= f(\vec{x}+2^{L_x-1}\cdot\vec{I}-2^{L_x-1}\cdot\vec{I}) \\ &= f(\vec{x}+2^{L_x-1}\cdot\vec{I})-2^{L_x-1}\cdot f(\vec{I}). \qquad (18) \end{aligned}$$

Therefore, each entry of $\vec{x}+2^{L_x-1}\cdot\vec{I}$ is an unsigned binary such that the value of $f(\vec{x}+2^{L_x-1}\cdot\vec{I})$ can be evaluated as discussed in this paper. As for the second term of (18), since it is a constant, it can be evaluated in advance. This process for dealing with two's complement with linear systems that have only unsigned computations is depicted in Figure 14.

If we investigate further, we find that the offsetting of $\vec{x}+2^{L_x-1}\cdot\vec{I}$ is only the inversion of the MSBs of each entry of vector $\vec{x}$. Suppose that x_j is the jth entry of $\vec{x}$; it then follows that

$$\begin{aligned} x_j+2^{L_x-1} &= (-x_j^{L_x-1}\cdot 2^{L_x-1}+\sum_{I=0}^{L_x-2} x_j^i\cdot 2^i)+2^{L_x-1} \\ &= (-x_j^{L_x-1}+1)\cdot 2^{L_x-1}+\sum_{I=0}^{L_x-2} x_j^i\cdot 2^i. \quad (19) \end{aligned}$$

Therefore, the first term in (19) corresponds to only the inversion of the MSB of x_j.

Another overhead in (18) is the subtraction of $f(\vec{I})\cdot 2^{L_x-1}$. However, since this term is a constant for all x's, it can be preloaded in the accumulator of our architecture instead of being subtracted by an extra subtractor.

REFERENCES

[1] M. Sid-Ahmed, "A systolic realization for 2-D digital filters," *IEEE Transcation on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 560–565, April 1989.

[2] C. C. Stearns, D. A. Luthi, P. A. Ruetz, and P. H.Ang, "Design of a 24 MHz 64-tap transversal filter," in *IEEE Conference on Computer Design*, pp. 574–577, 1988.

[3] D. Dubois and W. Steenaart, "High speed stored product recursive digital filter," *IEEE Transactions on Circuits and Systems*, vol. 27, pp. 657–666, August 1982.

[4] M.T.Sun, "VLSI architectures for high-speed video processing," *IEEE Workshop on VSPC*, pp. 307-311, 1991.

[5] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP MAGAZINE*, pp. 4-18, JULY 1989.

[6] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Transcation on Acoustics, Speech, and Signal Processing*, vol. 22, pp. 456–462, December 1974.

[7] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, ch. 4.2, p. 98. John Wiley & Sons, 1979.

[8] T. Noll, "Carry-save arithmetic for high-speed digital signal processing," *Journal of VLSI Signal Processings*, vol. 3, pp. 121–140, 1991.

[9] S. Y. Kung, *VLSI Array Processors*, ch. 4, pp. 208–210. Prentice-Hall, 1988.

[10] Texas Instruments, *TSC700 Series, 1.0-μ CMOS STANDARD CELLS*, 1992.

[11] J. Ward, P. Barton, J. Roberts, and B. Stanier, "Figures of merit for VLSI implementations of digital signal processing algorithms," *IEE Proceedings, Part F*, vol. 131, pp. 156–160, January 1989.

[12] H.-R. Lee and C.-W. Jen, "The design of two-dimensional FIR and IIR filter architectures for HDTV signal processing," in *International Symposium on VLSI Technology System and Architecture*, pp. 307–311, 1991.

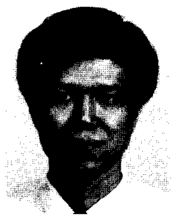
[13] J.-I. Guo, C.-M. Liu, and C.-W. Jen, "The efficient memory-based VLSI array designs for DFT and DCT," *IEEE Transactions on Circuits and Systems*, October 1992.

[14] J.-I. Guo, C.-M. Liu, and C.-W. Jen, "A new array architecture for prime-length discrete cosine transform," *IEEE Transcation on Signal Processing*, vol. 41, pp. 436–442, January 1993.

[15] C.-M. Liu and C.-W. Jen, "On the design of VLSI arrays for discrete fourier transform," *IEE Proceeding Part G*, vol. 139, pp. 541–552, August 1992.

Hwan-Rei Lee was born in Taipei, Taiwan, R.O.C. in 1966. He received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1989.

He is currently working on his Ph.D degree in electronics engineering National Chiao Tung University. His research interests include parallel processing algorithms and architectures and VLSI architecture design and implementation.



Chein-Wei Jen was born in Shanghai, China, in 1948. He received the B.S. degree from National Chiao Tung University in 1970, the M.S. degree from Stanford University in 1977, and the Ph.D degree from National Chiao-Tung University in 1983.

He is a professor in the Department of Electronics Engineering and the Institute of Electronics, National Chiao-Tung University, Hsinchu, Taiwan. From 1985 to 1986 he was a Visiting Researcher at the University of Southern California, USA. He is now the director of the Institute of Electronics, National Chiao-Tung University. His current research interests include VLSI signal processing, VLSI architecture design, design automation, and fault tolerant computing. He is a member of IEEE and Phi Tau Phi.

Chi-Min Liu was born in Pingtung, Taiwan, R.O.C. in 1963. He received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taiwan, R.O.C. in 1985, and the M.S. degree and Ph.D degree in electronics from National Chiao Tung University, Hsinchu, Taiwan, in 1987 and 1991, respectively.

He is currently an Associate Professor in the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. His research interests include parallel processing algorithms, parallel architectures, adaptive signal processing, image signal processing, radar signal processing, neural networks, fast algorithms, and design automation for DSP VLSI architectures.