OPTIMAL VLSI ARCHITECTURE FOR DISTRIBUTED ARITHMETIC-BASED ALGORITHMS

Kamal NOURJI and Nicolas DEMASSIEUX
Télécom Paris, E.N.S.T, 46 rue Barrault, 75634 PARIS Cedex 13, FRANCE
Email : nourji@elec.enst.fr

Abstract - Digital signal processing algorithms often use inner product as basic computation. In this paper we propose a new design methodology for synthesizing an optimal VLSI architecture implementing a real-time Distributed Arithmetic-based inner product. Our design methodology considers the design space as a bidimensional one. In the first dimension we consider all possible input data parallelisations : from bit-serial to bit-parallel. In the second dimension we consider all possible lookup-table partitioning. Using a new ROM generic model, expressions are developed for area and maximum input data bandwidth, which allows to have an explicit formulation of the area-bandwidth tradeoff. Finally, for a given set of application constraints (inner product size and data bandwidth), we exhibit the optimal architectural parameters that provide the smallest chip area.

1. INTRODUCTION

The first description of Distributed Arithmetic was given by Peled and Liu in 1974 [PELE74]. Distributed arithmetic has been used in many digital signal processing applications where one of the inner product vector operands is fixed. Among these DSP applications are : DCT [LEGE87], [MOU91], [SUN89], [URAM92], FFT [WHIT89], non recursive filters [ZOHAR73], polynomial filters [BURL90], convolution [BURR77], orthogonal transforms [BURL89]. Compared to "lumped" arithmetic-based architectures, Distributed Arithmetic architectures are competitive both in speed and in hardware requirement. In addition they are extremely regular, which makes them most suitable for VLSI realizations.

In the following section, we review the Distributed Arithmetic theory applied to an inner product of size N, then we introduce the generalization of the parallelisation to L Bits At A Time (BAAT) of the input data. Considering this L BAAT parallelization we have to process, in real-time, NL bits of input data. So we propose, in Section 4, the partitioning of this bidimensionnal (N,L) space to pq packets of input data (p is the partitioning of N and q the partitioning of L). Then, we have to generate, for each time cycle, pq partial products. each partial product corresponding to NL/pq bits of input data. In the Section 5, we propose a generic ROM-based implementation depending on (L,p,q) parameters for a given inner product size N. After that, we propose an area and speed estimation of our generic architecture. For this estimation we develop a generic model of the area and cycle time of ROM memories. At the end we propose an efficient design methodology providing the optimal architecture and we exhibit some estimated results.

2. THEORY

The Distributed Arithmetic theory is widely known [WHIT 89] : in this section, we shall review its main proprieties compared to those of the conventional multiplication. Let us consider the following inner product of size N were a_i are fixed coefficients and x_i are input data words :

$$y=\sum_{i=1}^{N} a_i x_i = a_1 x_1 + a_2 x_2 + .. + a_N x_N \qquad (1)$$

Let the input data words x_i be represented by the 2's complement code as follows :

$$x_i = -x_{i0} + \sum_{j=1}^{B-1} x_{ij} 2^{-j} \qquad (2)$$

Were x_{ij} is the jth bit of the x_i word which is Boolean, B is the number of bits of the each input data word and x_{i0} is the sign bit. Substituting Eq.(2) into Eq.(1) we have :

$$y = \sum_{i=1}^{N} a_i \left[-x_{i0} + \sum_{j=1}^{B-1} x_{ij} 2^{-j} \right] \qquad (3)$$

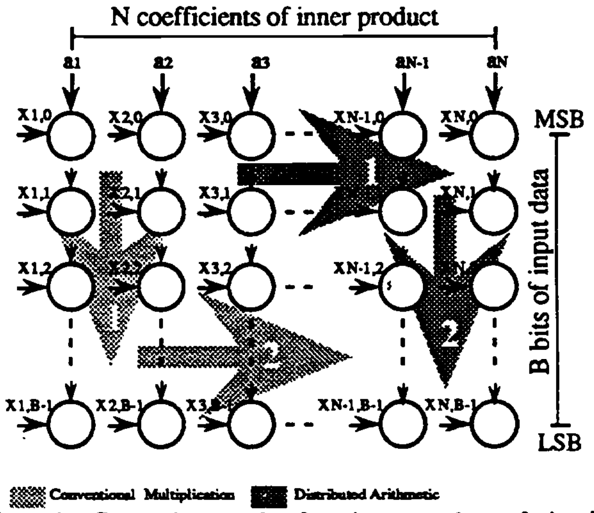

Figure 1 : Computing graph of an inner product of size N with input data words coded in B bits. Pale arrows represent the order of the computation of the conventional multiplication and the bulk ones correspond to Distributed Arithmetic.

Note that Eq.(3) represent the conventional way of calculating the inner product used by standard multiplication. By the way, in the conventional multiplication, we process first the direction of the B bits of the input data and then the direction of the inner product size N. This is illustrated by the computing graph of Figure 1.

Let us interchange the order of the summations of Eq.(3) : this will gives us :

$$y = \sum_{j=1}^{B-1}\left[\sum_{i=1}^{N} x_{ij}\, a_i\right]2^{-j} + \sum_{i=1}^{N} a_i(-x_{i0}) \quad = \sum_{j=1}^{B-1} y_j 2^{-j} - y_0 \qquad (4)$$

This is the formulation of the Distributed Arithmetic computation. In this computation, we process the inner product first in the direction of its size N and then in the direction of the B bits of the input data (cf figure 1).

Eq.(4) corresponds to a bit-serial computation of the input data. We have to generate during the time cycle t_j a partial product y_j corresponding to the input data vector ($x_{1,j}$, $x_{2,j}$, ..., $x_{N,j}$) of size N. Because each bit $x_{i,j}$ may take on values of 0 and 1 only, the partial product y_j may have only 2^N possible values. All the possible values of partial products can be stored in a single ROM memory of size 2^N words of K+$\log_2$(N) bits. K is the nomber of bits which code the fixed coefficient a_i. This K+$\log_2$(N) bits precision of partial products allows us, in this case, to keep the full precision of the computation of the inner product (the final precision will be K+$\log_2$(NB)). The architecture of this bit-serial case is shown in figure 2.

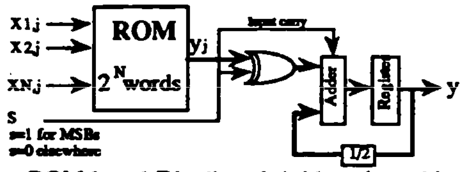

Figure 2 : ROM-based Distributed Arithmetic architecture for bit-serial processing of an inner product of size N.

This architecture requires one ROM, K+$\log_2$(N) XOR gates (this logic allows the subtraction of the partial product y_0 corresponding to MSBs of the input data) and an accumulator (adder + register + shifter by 2^{-1}). To compute the inner product, we need B time cycles.

S.A.White showed how to halve the memory size by using the "Offset Binary Coding" [WHIT 89].
We can easily demonstrate that the OBC technique can also be used in the case of the L BAAT parallelisation of the input data. The memory reduction is at the cost of a small hardware complexity corresponding to NL XOR gates. In this paper we will not use this technique.

3. INPUT DATA PARALLELISATION

We propose now to parallelise the input data by considering L BAAT. During each time cycle we will receive NL bits of input data and we have to generate a corresponding partial product. The computing of the inner product will take B/L time cycles (cf Figure 3).

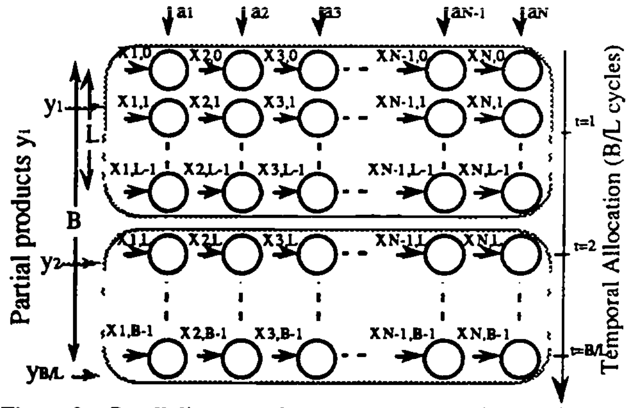

Figure 3 : Parallelisation of the input data to L BAAT. The computation takes B/L time cycles. L=1 correspond to a bit-serial computation and L=B to a parallel one.

If we try to implement this L BAAT parallelisation case as we did it for the bit-serial one (L=1), 2^{NL} words ROM memory will be needed. This size of memory can became prohibitive if the product NL increases. For this reason we propose the partitioning of the memory size.

4. MEMORY PARTITIONING

In this section we propose the partitioning of the single ROM memory of size 2^{NL} words. We can partition the NL address bits of the ROM in two dimensions : by the parameter p in the dimension of the inner product size N and by the parameter q in the dimension of the parallelisation L (cf figure 4). This means that we will use pq ROMs, each ROM of size $2^{NL/pq}$ words being addressed by NL/pq bits of input data.Then during each time cycle we generate and accumulate pq partial products.

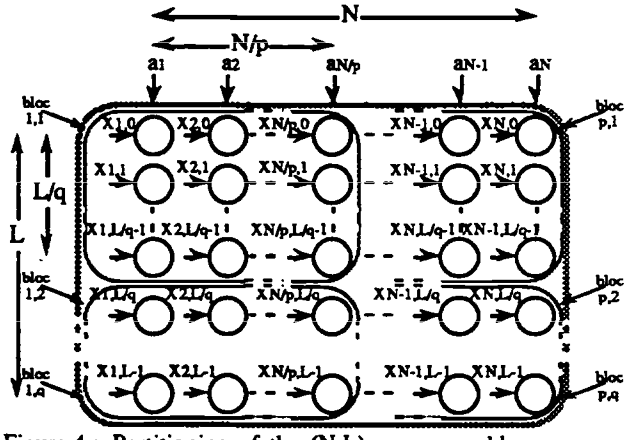

Figure 4 : Partitioning of the (N,L) memory addresses space by (p,q).

This ROM partitioning reduces the global memory size ($pq2^{NL/pq}$ words instead of 2^{NL} words) but increases the accumulator complexity (pq partial products to accumulate).

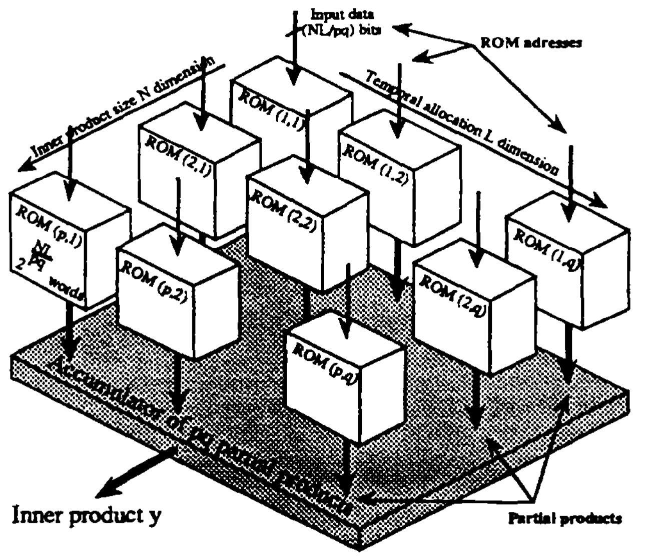

Figure 5: Size N inner product Architecture using a (p,q) partitioning. The pq ROMs are of size $2^{NL/pq}$ words.

In our design methodology we will automatically define a tradeoff between the global memory size and the accumulator complexity in terms of VLSI complexity.

5. VLSI ESTIMATIONS

To define the best tradeoff between the global memory size and the accumulator complexity, we develop, for our generic architecture, the expressins for area and bandwidth (maximum bandwidth of the input data) as follows :

$$A_{Tot} = MA_{ROM} + M\left(K + \log_2\left(\frac{NL}{M}\right)\right)A_{FA} + 2(K + \log_2(NL))A_{Reg} \tag{5}$$

$$\text{Bandwidth} = \frac{L}{T_{ROM} + \log_{3/2}\left(\frac{M+2}{2}\right)T_{FA} + T_{XOR} + T_{Reg}} \tag{6}$$

were M=pq.

In these estimation we assume that our architecture uses a carry save accumulator. Such an increase in accumulation speed is gained at a modest cost of hardware.

For the above expressions we need an efficient model of the area and the cycle time of the used ROM. The proposed generic model takes into account the widely used internal structures of ROMs. Thus, the area and speed of one ROM storing N words of p bits can be estimated by:

$$A_{ROM} = A_0Np + A_1\left(N_{col}\text{Log}_2\left(\frac{N_{col}}{p}\right) + N_{row}\text{Log}_2(N_{row})\right) + A_2p + A_3 \tag{7}$$

$$T_{ROM} = T_1 N_{col} + T_2 N_{row} + T_3 = T_1 N_{col} + T_2 \frac{Np}{N_{col}} + T_3 \tag{8}$$

This proposed generic model of a ROM was validated for the following ROM compilers : VLSI Technology (0.8, 1.0, 1.5 micron) and ES2 (ECPD12 and ECPD15). The maximum area and cycle time errors of our model compared to real data provided by compilers are (-8% , +4%) for the area and (-9% , +10%) for the cycle time.

6. OPTIMAL ARCHITECTURE

For a given inner product size N and parallelisation L, we try to find an optimal (p,q) partitioning which provides the best bandwidth/area tradeoff. In order to demonstrate the validity of this approach, let us consider the example of the implementation of an inner product of size N=16 using the VLSI Technology 0.8micron standard-cell library. The Figure 6 shows that for each parallelisation L we have an optimal (p,q) partitioning that maximizes the bandwidth/area ratio.

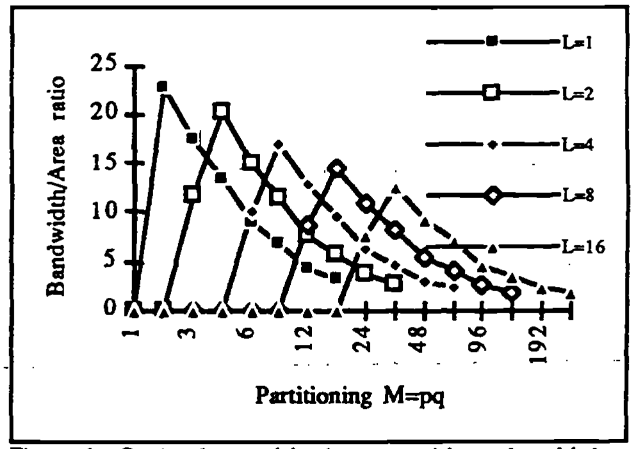

Figure 6: Optimal partitioning provides the highest Bandwidth/Area ratio. N=16, B=16, K=16.

For each parallelisation L the optimal partitioning correspond to the optimal architecture. We call this architecture *"Index L Basis Architecture"* (ILBA). To each ILBA corresponds an $A_{basis,L}$ area and a maximal input data bandwidth $B_{basis,L}$.

For a given application with Bdata as input data bandwidth, we propose to multiplex "n_L" ILBA with :

$$n_L = \text{Ent}\left(\frac{B_{data}}{B_{basis,L}}\right) + 1 \tag{9}$$

with Ent being natural part of one division.

The total area of each basis L architecture is $A_L = n_L A_{basis,L}$. All basis L architectures can process in real-time the B_{data} input data bandwidth. Among them, the optimal architecture is the one which provides the smallest area. For this optimal architecture the 4 architectural parameters (L,p,q,n_L) are produced.

Note that to find this optimal architecture all possible architectures (from bit-serial ($L=1$) to bit parallel ($L=B$) and for all possible memory partitioning) are considered.

7. PROPOSED DESIGN METHODOLOGY

The above ideas are summarized in the following 4-step design methodology :

- **Step 1** : Definition of the application by the specification of the inner product size N and of the input data bandwidth B_{data} (Mbits/s). The input data bandwidth will be defined in the case of a real-time processing.
- **Step 2** : Providing, for the chosen architecture, a generic formula for the estimation of area and speed with the following parameters : the parallelisation L of the input data and the partitioning coefficients p and q.
- **Step 3** : For each parallelisation L, we define the optimal (p,q) partitioning value which provides the best bandwidth/area trade-off (or the highest bandwidth-area ratio). We call the corresponding architecture the "*Index L Basis Architecture*". To each ILBA correspond an $A_{basis,L}$ area and a maximal input data bandwidth $B_{basis,L}$.
- **Step 4** : $B_{basis,L}$ is the maximal input data bandwidth that the ILBA can process at real-time. Then, for the implementation of an application requiring B_{data} input data bandwidth, we have to multiplex "n_L" ILBA (Eq. 9). The total area of each basis L architecture is $A_L=n_LA_{basis,L}$. All basis L architectures can process in real-time the B_{data} input data. Among them, the optimal architecture is the one which provides the smallest area. For this optimal architecture the 4 architectural parameters (L,p,q,n_L) are produced.

As an example, we use the 0.8micron VLSI Technology standard-cell library to implement a filter of size $N \in \{2,8,16,30\}$ for different TV and HDTV real-time applications. We present in Table 1 the area variation and in table 2 architectural parameters (L, $M=pq$, n_L).

Image	Bandwidth	N=2	N=8	N=16	N=30
CCIR601	82,88	0,66	1,98	4,42	8,36
EDI	110,4	1,16	2,64	5,9	10,45
EDP	221,12	1,98	5,28	10,62	18,81
HDI	442,4	3,3	10,56	20,06	37,62
HDP	884,8	6,6	21,12	40,12	75,24

Table 1: Chip Area (mm^2) for different standard TV and HDTV.

Image	Bandwidth	N=2	N=8	N=16	N=30
CCIR601	82,88	4,1,1	1,1,3	2,4,2	1,4,4
EDI	110,4	2,1,2	1,1,4	1,2,5	1,4,5
EDP	221,12	4,1,3	1,1,8	1,2,9	1,4,9
HDI	442,4	4,1,5	1,1,16	1,2,17	1,4,18
HDP	884,8	4,1,10	1,1,32	1,2,34	1,4,36

Table 2: 3 architectural parameters ($L,M=pq,n_L$) (parallelisation, partitioning, number of multiplexed ILBA).

8. CONCLUSION

A new and efficient design methodology for synthesizing optimal real-time VLSI architecture for ROM-based Distributed Arithmetic-based inner product was presented. In the first steps of our design methodology we produce, for a given inner product size N, a generic architecture which depends on the parallelisation L and (p,q) partitioning parameters. Then, for a given input data bandwidth the optimal architecture is the one which requires the smallest area.

For the area and bandwidth estimations of the generic architecture, an efficient generic model for area and time cycle of ROM memories was presented. This ROM generic model can be used in all optimization studies of ROM-based architectures in general.

REFERENCES

[BURL89] W.P.Burlesson , L.L.Scharf, A.R.Gabriel, N.H.Endsley, "Asystolic VLSI Chip for Implementing Orthogonal Transforms", IEEE Journal of Solid-State Circuits, vol 24, no 2, pp. 466-469, (April 1989).

[BURL90] W.P.Burlesson, "Polynomial Evaluation in VLSI Using Distributed Arithmetic", IEEE Trans on ASSP, vol 37, no 10, pp. 1299-1304, (October 1990).

[BURR77] C.S.Burrus, "Digital Filter Structures Described by Disributed Arithmetic", IEEE Trans on Circuits ans Systems, vol 24, no 12, pp. 674-680, (Dec 1977).

[LEGE87] A.Leger, JP. Duhamel, JL. Sicre, G. Madec, JM. Knoepfli, "Distributed Arithmetic Implementation of the DCT for Real Time Photovideotex on ISDN", SPIE vol 804 Advances in Image Processing, pp. 364-369, (1687).

[MOU91] Z.Mou, F.Jutand, "A high-speed Low-cost DCT Architecture for HDTV Applications", IEEE, pp. 1153-1156, (1991).

[PELE74] A.Peled and B.LIU, "A New Hardware Realisation of Digital Filters", IEEE Trans on ASSP, vol ASSP-22, no 6, pp. 456-461, (December 1974).

[SUN89] M.T.Sun, TC. Chen, A.M. Gottlieb, "VLSI Implementation of 16x16 Discrete Cosine Transform", IEEE Trans on Circ and Syst, vol 36, no 4, pp. 610-617, (April 1989).

[URAM92] S.I.Uramoto, Y.Inoue, A.Takabatake, J. Takeda, Y. Yamashita, H. Terane, M. Yoshimoto., "A 100-MHz 2_D Discrete Cosine Transform Core Processor", IEEE Journal of Solid-State Circuits, vol 27, no 4, pp. 492-499, (April 1992).

[WHIT89] S.A.White, "Application of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", IEEE ASSP Magazine, pp. 4-21, (July 1989).

[ZOHA73] S.ZOHAR, "New Hardware Realizations of Nonrecursive Digital Filters", IEEE Trans on Comput, Vol C-22, No 4, pp. 328-338, (April 1973).