A VLSI ARITHMETIC UNIT FOR A SIGNAL PROCESSING NEURAL NETWORK

V. Rodellar[1], M. Hermida[1], A. Díaz[1], A. Lorenzo[1], P. Gómez[1], P. Aguayo[2], J. C. Díaz[2] and R. W. Newcomb[3]

[1] **Dept. Arquitectura y Tecnología de Sistemas Informáticos**
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
Boadilla del Monte
28660 Madrid **SPAIN**

[2] **CIEMAT**
Avda. Complutense, 22, Edificio 22
28005 Madrid **SPAIN**

[3] **Microsystems Lab**
Electrical Engineering Department
University of Maryland
College Park, MD, 20742 **USA**

Abstract

Through the present paper, the design of a *VLSI Arithmetic Unit* for the support of *Signal Processing* and *Neural Network Algorithms* is being discussed. The *Arithmetic Unit* is conceived to allow the execution of *Inner Product Operations* maintaining the silicon area and development costs under reasonable limits, and allowing the parameterization of the design in a modular way. For such, a *Serial-Bit Pipeline Multiplier* and a *Fixed-Point Number Format* have been successfully used. These aspects are justified by a study on the accuracy required when implementing certain *Signal Processing* and *Neural Network Algorithms*, for which some numerical examples are given. A panoramics of the design of a whole chip incorporating such *Arithmetic Unit* and the structure of the required *Control Unit* are also given.

Introduction

Nowadays the application of Signal Processing [1] and Neural Net [2] techniques is knowing an everyday raising interest and proliferation. This is in part being so because of the spread of VLSI Design techniques, which allow the design of more and more complex chips amenable of supporting end-user products on Application Specific Integrated Circuits (ASIC's). VLSI Design Tools have opened the possibility of designing chips on general purpose workstations affordable by most laboratories in factories and universities, spreading the knowledge and easing the process. Among the different types of ASIC's, one of the most important is that devoted to Numerical Applications, under which we should classify these based both in Signal Processing and Neural Network techniques. Numerical Application ASIC's (NAASIC's) share certain features in their architecture, because of the common structure of the algorithms used in both fields, and this fact introduces certain similarities in their architectures which make them differ from other VLSI processors devoted to numerical applications, such as Digital Signal Processors (DSP's). In this sense, Fig. 1 shows a general structure of a NAASIC, where its main characteristics are made explicit. The general structure is divided classically in the Functional Structure, Data Paths and Control. Regarding the Functional Structure NAASIC's are composed of an Arithmetic Unit specialized in addition-subtraction, products and division, and tables to implement special functions (as for example nonlinear transformations or trigonometric expressions), a Data Memory, to hold coefficients and sampled signals in array-like data-structures, temporary registers, and an interface with the outer world, which in many cases may include Analog/Digital circuitry. Data paths, on their turn are designed to produce flexible connections among the different functional resources before mentioned, allowing different schemes of computation to be carried out. To simplify the design, and render it less expensive both in development time and in silicon area, they use simple data formats (byte, digit or bit) for routing. Data switching is provided by blocks of multiplexers or switching matrices.

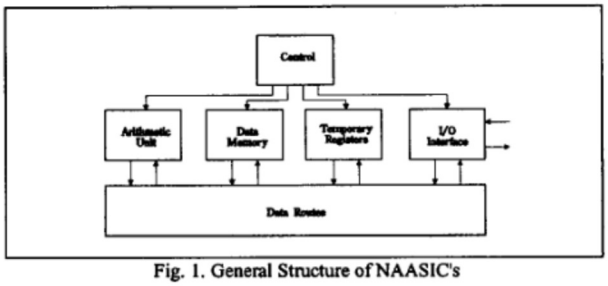

Fig. 1. General Structure of NAASIC's

Usually the Control is not software-programmable, in the general sense, to avoid the cost in resources and clock cycles which has to be devoted to instructuion fetching and interpretation, and in this sense, they operate more like Finite State Machines than general purpose processors, this feature being one of the most important differencies characterizing NAASIC's. Instead, programming is done by a PLA, a ROM memory or other related structure, which has to be personalized during the design process. From this preliminary study, we may infer that the Arithmetics, Data Path and Memory resources take most of the Silicon Area in the chip, and are an important factor on the final cost. As these functional structures are completely dependent in their complexity and size on number formats, it will be determinant to study the numerical behaviour of the algorithms under consideration to determine the best candidates among number formats for an optimized design of a given NAASIC . Based on this study we propose a general structure of such a NAASIC being currently designed to give support to Neural Network and Signal Processing Algorithms in Speech Processing [1, 2, 3] and Spectral Characterization applications [4].

Common Algorithmic Structure and Arithmetic Requirements

When comparing the general structure of both Signal Processing and Neural Newtork algorithms certain most common features come to light, as that both work on array-like data structures, that data may be divided into varying objects (signals or states) and more permanent objects (coefficients, weights), and that number representation is a crucial fact. According to the kind of operations performed, these may be reduced most of the times to the well-known *Inner Product* between array-like data structures, without a loss of generalization. Besides, other not so frequent operations, such as divisions, factorials, square roots and trigonometric (non linear in general) operations, have also to be performed. The *Inner Product* between two given vectors, a and b is a common operation which may be defined recursively as follows:

$$c_n = c_{n-1} + a_n * b_n \qquad (1)$$

0-7803-0510-8/92$03.00 ©1992IEEE

where a_n and b_n are the n-th elements of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and c_n is the n-th update of an accumulator holding the value of the *Inner Product* being evaluated. Under the arithmetic point of view, the operation in (1) presents interesting properties when a given number format is chosen. Without a loss of generality we will focus our atention in m-bit two's-complement fixed-point number formats ($m=p+q$) in which a given number x may be expressed with p bits for its integer part and q bits for the fraction as follows:

$$x = (-x_{m-1}\, 2^{m-1} + \sum_{i=0}^{m-2} x_i\, 2^i)\, 2^{-q} \qquad (2)$$

Then it may be easily shown that the product of two such numbers will produce a *2m-bit* fixed-point number with *2p* bits for the integer part and *2q* bits for the fraction. As our basic number format consists in m bits, we will have to reject m out of the resulting *2m* bits in the product. If we want to keep the same significant bits, this could be done as shown in Fig. 2a, elliminating the first p and the last q bits in the product.

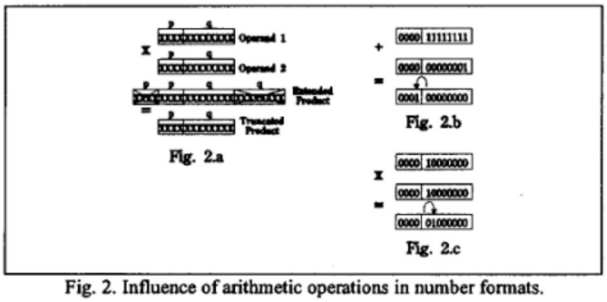

Fig. 2.a

Fig. 2.b

Fig. 2.c

Fig. 2. Influence of arithmetic operations in number formats.

This loss of residual bits may result in a given error. To infer how problematic this process of re-fitting would be, and without a loss of generality, consider that our data are normalized fixed-point numbers. In fact, in certain applications which involve inversion processes, like the solution of systems of equations, the arithmetics with fixed-point numbers works better if a pre-normalization process has been carried on the data. This process consists in determining the highest absolute value among the data, and dividing every data by such value. This produces a data set in which the p bits of the integer part of the format (guard bits) will be all *0's* (positives) or all *1's* (negatives), as they will be defined in the interval [-1, +1]. When two of these numbers are added, the resulting number may be out of the interval if its absolute value is higher that the unity. This will result in the *invasion* of the less significant guard-bit, that is x_q, or in a *displacement* of the significative bits to the left (Fig. 2.b). On the contrary, when two of these numbers are multiplied, the absolute value of the resulting number will be in general lower than the unity, as the absolute value of both factors is. This may produce a *displacement* of the significative bits to the right, having in mind Fig. 2.c. In this sense, both operations exert an oposite action on the bits. As they appear combined in (1), one may infer interesting properties which may be reasonably exploited to build low-cost arithmetics. For example, the optimal value for the number of guard bits, p, may be deducted from the fact that the absolute value in (1) will in general be:

$$|c_N| < N \qquad (3)$$

as a result of the addition of N products of numbers with absolute value under the unity, when *N-dimensional* vectors are considered. Of course, this will be the theoretical upper limit, because in real applications the result will be much under that limit. Experimenting with randomly generated numbers, as commented in the results, it may be established that a rather conservative value for that limit should be around *N/3*. This fact gives us a clear hint to establish p:

$$p > \log_2 N/3 + 1; \qquad \text{with } p \text{ integer} \qquad (4)$$

On its turn, the number of bits of the fraction q, will have to be determined according to the absolute error desirable in the estimation of the *Inner Product*. If we consider this operation as the addition of N partial products of type $a_n * b_n$, and we call these:

$$y_n = a_n * b_n \qquad (5)$$

then considering the error implied in the truncation of the *2m-bit* number y_n to a number in m bits y'_n, and calling this error ε_n:

$$y_n = y'_n + \varepsilon_n \qquad (6)$$

it may be established that:

$$|\varepsilon_n| < 2^{-q} \qquad (7)$$

We will assume that statistically, when adding up y'_n, $1 \le n \le N$, this error will accumulate to produce a total error ε_T limited as:

$$|\varepsilon_T| < N\,|\varepsilon_n| \qquad (8)$$

Then, from (4):

$$N < 3\, 2^{p-1} \qquad (9)$$

with which a simple expression may be given for (8) in terms of p and q:

$$|\varepsilon_T| < 3\, 2^{p-1}\, 2^{-q} = 3\, 2^{p-q-1} \qquad (10)$$

Expressions (4) and (10), should be taken as approximate rather than as exact rules for choosing p and q, but at least may give a hint on how better determine their approximate values, althouth a final decision should be taken on the basis of simulation results. In a practical case discussed in [3], with *N=24*, a format of *p=6* and *q=10* would yield a total expected absolute error lower than 0.09375. As the lowest absolute limit of the *Inner Product* may tend to zero, the definition of the worst relative error does not have any meaning in this case. As an example, which will help us to fix most of the ideas we have commented above we will consider the case of the *Mixture Problem* [4], defined as the determination of $\boldsymbol{c}$ assuming that $\boldsymbol{R}$ and $\boldsymbol{x}$ are known from the following expression:

$$\boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{c} \qquad (11)$$

where $\boldsymbol{R}$ is a *NxK* matrix composed of *K N-dimensional* reference vectors, $\boldsymbol{c}$ is the *K-dimensional* vector of contributions or weights of the linear combination, and $\boldsymbol{x}$ is the *N-dimensional* vector resulting from the linear combination given in (11). Obviously, this problem, when $K<N$ is overdetermined, and it will not have an exact solution if more than K column vectors in $\boldsymbol{R}$ are linearly independent to each other. In the case that both $\boldsymbol{x}$ and $\boldsymbol{R}$ were composed by radiation spectra, their elements would be energy counts, whose values are typically well above the thousands, and as such, a fixed-point format would require many bits for p. It may be seen easily that if M is the highest absolute value among the elements of both $\boldsymbol{x}$ and $\boldsymbol{R}$, the following set of expressions, when inverted would yield the same value for the contribution vector $\boldsymbol{c}$:

$$\boldsymbol{x}' = \boldsymbol{R}'\,\boldsymbol{c} \qquad (12)$$

$$\boldsymbol{x}' = \frac{\boldsymbol{x}}{M} \qquad (13)$$

$$\boldsymbol{R}' = \frac{\boldsymbol{R}}{M} \qquad (14)$$

with the important that both $\boldsymbol{x}'$ and $\boldsymbol{R}'$ are now composed of normalized numbers, much more suited to a fixed-point format as assumed in the present work. This is a typical example of a linear problem re-scaled for its solution in a much simpler hardware than is usual in general purpose DSP's, a set of simulation results of which is presented later. Nevertheless there are

other even better examples to be considered in favour of this methodology, as is the case of Nonlinear Neural Networks composed of layers of *Inner Product* evaluators, followed by a *Nonlinear Function*, whose role is to map the result in the interval [-1, +1]. These subsystems are especially suited for their implementation in a fixed-format arithmetics, as the *Nonlinear Function* is a true *Normalizing Device* of numbers, in the sense described here. Most of the conclusions drawn from this study can be easily generalized to other number formats, such as floating-point ones.

Proposed Architecture

Based on the considerations exposed in the preceding section, an Architecture for a NAASIC devoted to support both Signal Processing and Neural Network Algorithms has been conceived for its modular parameterization, in the sense that the structures of the arithmetics being sensitive to format-number parameters, have been designed for their modular automatic re-sizing. Figure 3 gives a general idea of the structure being considered.

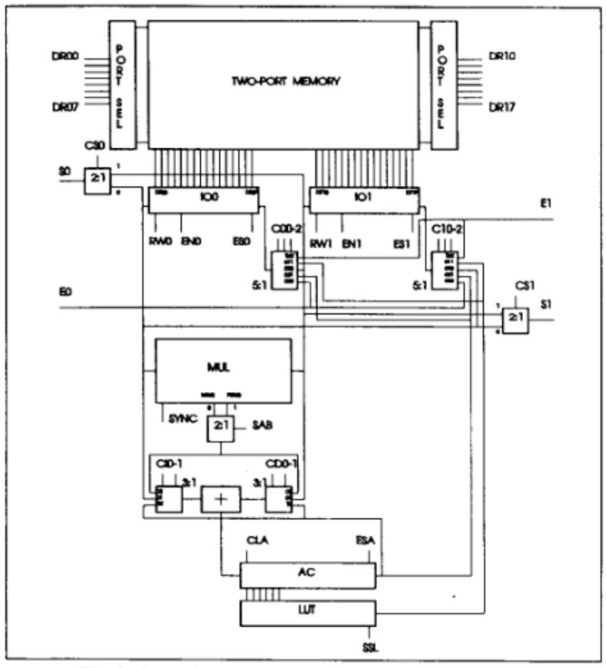

Fig. 3. General Structure of the NAASIC being designed.

The general structure shown is composed of several parameterizable modules, in function of the arithmetics being used, among which we find a Two-Port Memory which stores data structures in a $p+q$ fixed-point format, plus an Arithmetic Unit, and general Data Routes. The Arithmetic Unit is designed as an *Inner Product Unit*, although it may perform several other operations. It is composed of a *Serial Multiplier* (MUL), a *Serial Adder* (+), an *Accumulator* (AC) and a *Look-Up Table* (LUT) to implement different nonlinear mappings and functions. The *Arithmetic Unit* uses *on-line serial arithmetics*, because in this sense the *Multiplier* may be very much reduced in size, keeping the cost in area well under 1 mm² for a CMOS Technology of 1.5 μm and $m=16$. The *Serial Multiplier* is based on *Radix-4 Recoding* [5], using modular substructures, which are easily parameterizable. Three types of macrocells are used in its design, named *initial*, *middle* and *final*. For a given fixed-point arithmetics with $p+q=m$ bits (m even), it requires an initial cell, k middle cells and a final cell, with $k=m/2-2$. With this simple rule a multiplier for any data size may be easily synthesized using a *Structure Compiler*. In this was the productivity factor, defined as the ratio between the total amount of transistor gates used and the real amount being actually designed, is very high.

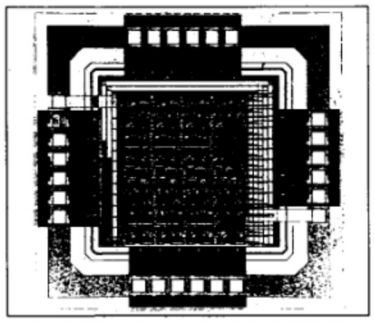

Fig. 4. Microphotograhy of the Multiplier

One of these multipliers for m=16 was designed, and is shown in Fig. 4., where the 8 cells composing it may be clearly distinguished. This kind of multipliers may be easily folded following simple rules, to keep their aspect square. The *Serial Adder* has been designed to accept data coming from the *Multiplier*, the *Two-Port Memory*, the *Accumulator* or the *Input-Output Ports* (S0, S1, E0 and E1) under the control of two 3:1 multiplexers (CI0-1 and CD0-1). The *Serial Accumulator* (AC) uses its 6 most significant bits to read data in serial format from the *Look-Up Table* (LUT) implemented as a ROM. The Data Routes are composed by two *Serial-Parallel Interface Registers* (IO0 and IO1), which allow the bidirectional flow of data between the *Two-Port Memory* and different origins and destinations by means of two 5:1 multiplexers (C00-2 and C10-2).

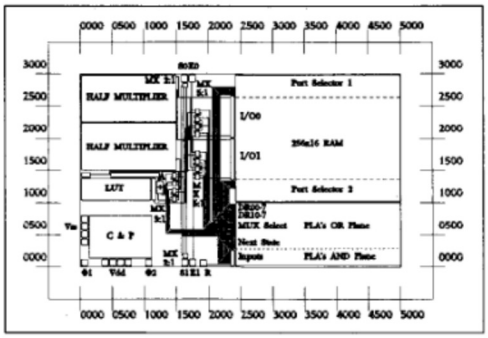

Fig. 5. Floorplan of the NAASIC.

The described structure is being designed for its integration under the EUROCHIP fabrication programme, a floorplan of which is given in Fig. 5., showing other complementary structures as the Control PLA, and the Clock&Power block.

Programming

The programmability of this structure is based in the Control Unit shown in Fig. 6, consisting in a Finite State Machine supported by a PLA.

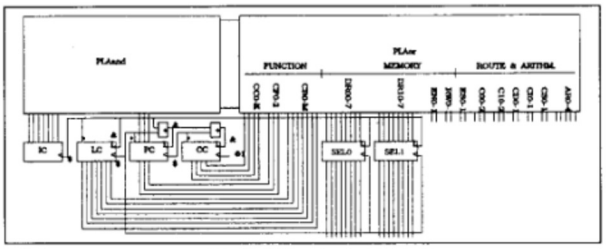

Fig. 6. Control structure.

As data formats are routed serially, the timing of each data transfer is more complex than with parallel formats. For such, the basic timing is divided in *Transfer Cycles*, *Phases*, *Loops* and *Macros*. The basic clock timing for

each transfer is controlled by an external *Cycle Counter* (CC), which counts the number of clock cycles contained in a given *Transfer Phase*. This is the basic time-interval for which the output word of the PLAor Plane has to be maintained without any change and is made of m clock cycles for transfering a serial m-bit datum from one resource to another, although there are *Transfer Phases* with other durations. The *Transfer Loop* counts the times a basic *Arithmetic Operation* has to be repeated within a *Transfer Macro*. A list of these may be used to specify a given *Signal Processing Algorithm*. The relations among the different timing levels is controlled by a battery of external counters (CC, PC, LC and IC), whose contents are in part used to address inside the PLA. The *Control Word* has several fields, the first one being the *Function Field*, used to define the contents of the *Timing Counters* when a new *Transfer Macro* is initiated or at the end of a *Transfer Phase or Loop*. The *Memory Field* is used to address the *Two-Port Memory* (SEL0 and SEL1) and to determine the sense of a *Memory Transfer*. Finally, the *Field of Route and Arithmetics* controls the routing of data to and from the arithmetic resources, and synchronizes the operations in the *Arithmetic Unit*. A specification of the *Arithmetic Unit, Data Routes,* and *Control Unit* is being carried out in VHDL for its evaluation.

Practical examples

To illustrate the behaviour of such arithmetics with real cases, several simulation processes have been carried out. In a first approach, the possibilities of $p+q$ fixed-point arithmetics to support *Inner Product* algorithms have been tried. For such, a set of sixteen 48-element vectors were produced using a random-number generator. The vector elements were normalized and coded in different $p+q$ fixed-point formats. Taking all possible combinations of 2 vectors out of 16, up to 128 inner-products were evaluated. This was done using alternatively a 64-bit floating-point and a $p+q$ fixed-point format. The first fact checked was the ability of the format to support guard-bit invasions during additions, and the second one was precision.

Table 1. Maximum number of sign bit invasions in a typical example.

Sign Bit	*0*	*1*	*2*	*3*	*4*	*5*
# Invas.	*0*	*1*	*1*	*2*	*6*	*8*

Sign invasions were listed in Table 1 for $p=6$ and $q=10$. Having in mind that sign bit *0* in Table 1 is the most significant one, it may be seen that using such format with $N=48$, there are no fatal invasions of the most significant bit.

Table 2. Maximum absolute error in a typical inner product simulation

Floating-point result	*Fixed-point result*	*Maximum abs. error*
13.788	13.747	0.041
14.162	14.118	0.043
15.814	15.773	0.040
16.949	16.900	0.049
14.751	14.707	0.044
15.680	15.639	0.041
16.816	16.768	0.048
15.514	15.475	0.039

Regarding precision, Table 2 lists the results for several maximal-norm vectors. As expected the *Inner Products* yielding highest values were those produced when operating a vector with itself. It may be seen that the maximum absolute error listed is about one half the value predicted using expression (10), which is of 0.09375.

Table 3. Typical results for the *Mixture Problem*

Weight	*Float (64 bits)*	*Fixed (8+10)*	*Rel. Error (%)*
c1	3.404786	3.413901	0.3478
c2	-0.001420	-0.005679	0.1421
c3	0.000492	-0.003873	0.0969
c4	2,100587	2.093908	0.1524
c5	0.001761	0.006941	0.1736
c6	0.002730	-0.002231	0.0558

Finally, to illustrate the performance of such arithmetics in a real problem, Table 3 shows the results of solving the *Mixture Problem* as described by equations (12-14) applying the method known as the *Pseudoinverse Operator* and Widrow-Hoff's Algorithm as stated in [4]. Given a base of six reference spectra $[r_{1,4}]$ with $N=128$ and the contribution vector $c = [3.4, 0, 0, 2.1, 0, 0]^T$, the results using a 64-bit floating-point format and a $p=8$ $q=10$ fixed-point format are compared. The right-most column lists the relative error between them. It can be seen that the resolution obtained is quite good having in mind that this algorithmic variant is very intensive in computations (1787648 products and 908436 additions were required to evaluate the *Pseudoinverse Operator*), and in this sense, the results show the possibility of trading silicon area (simplicity of arithmetics) vs. computational complexity (execution time). In some applications this trading may reduce strongly the costs of the final NAASIC.

Discussion

Through the present work the definition of a methodology to use simpler number formats for the realization of Arithmetic Units for Signal Processing and Neural Network algorithms has been explored. It has been shown that the parameters of fixed-point formats may be inferred from the dimensionality of the data structures and from the desired accuracy on the representation adopted, thus allowing the parameterization of important hardware structures, such as Memory, Arithmetics and Data Routes. One of the most important conclusions derived from this fact is that a modular design of an Arithmetic Unit may be realized. The same principles may be extended to Memories and Data Routes, thus permiting the semi-automatic synthesis of the NAASIC. The accuracy of the approach in specific problems have been tested, showing that simple data formats may produce rather acceptable results, when considering that other perturbation factors, such as noise or instrumental drifts may produce higher levels of error. This *Arithmetic Unit* is presently being incorporated in a general purpose design to be used in *Speech Processing Applications* [1, 2].

Acknowledgements

This work is being carried out under Grants Nos. TIC90-0109-C01 and C02, and TIC92-1239-E from the CICYT, and Nos. MIC88-0398-E and MIC-90-1219-E from the Programa Nacional de Microelectrónica, and under a Cooperation Agreement between the CIEMAT and the UPM.

References

[1] "A VLSI Architecture for the support of an Auditory Model for Hearing and Speech Processing", V. Rodellar, P. Gómez, M. Hermida, A. Díaz and R. W. Newcomb, Proc. of the 33rd Midwest Symposium on Circuits and Systems, Calgary, Alberta, Canada, August 12-15, 1990, pp. 787-790.

[2] "A Neural Network for the Extraction and Characterization of the Phonetic Features of Speech", V. Rodellar, F. Naharro, C. García, S. Martín, M. L. Muñoz and P. Gómez, Proc. of the Fourth Int. Conf. on Neural Networks and Applications, NEURO-NIMES'91, Nimes, France, November 4-8, 1991, pp. 203-212.

[3] "A Specific Processor for the Computation of TDNN Algorithms with application to Phonetic Coding", M. Pérez, V. Rodellar, V. Peinado, A. Díaz and P. Gómez, Proc. of the European Sign. Proc. Conf. EUSIPCO'92, Brussels, August 24-27, 1992.

[4] "An Associative Memory to Solve the Mixture Problem in Composite Spectra", J. C. Díaz, P. Aguayo, P. Gómez, V. Rodellar and P. Olmos, 35th Midwest Symposium on Circuits and Systems, Washington DC, August 9-12, 1992, (in these same Proceedings).

[5] "A Nonredundant-Radix-4 Serial Multiplier", K. K. Primlani and J. L. Meador, IEEE Journal of Solid-State Circuits, Vol. 24, No. 6, December 1989, pp. 1729-1736.