IMPLEMENTATION OF A VLSI POLYNOMIAL EVALUATOR FOR REAL-TIME APPLICATIONS

Guy Corbaz[1], Jean Duprat[2], Bertrand Hochet[1] and Jean-Michel Muller[2]

[1] Ecole Polytechnique Fédérale de Lausanne
LEG-Ecublens, 1015 Lausanne, SWITZERLAND

[2]CNRS, Laboratoire LIP-IMAG
ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, FRANCE

Abstract

In [5], Duprat and Muller present a new architecture, called a polynomier, able to evaluate quickly polynomials and elementary functions. In this paper, we propose a very regular VLSI implementation of the polynomier, and we extend this architecture to 2's complement arithmetic. Some applications are presented, and the performances enable real-time processing.

Introduction

Fast evaluation of polynomials is a major goal of computer science, since any continuous function may be approximated as accurately as desired by a polynomial. For instance, most part of current computers evaluate elementary functions using polynomial or rational approximations [3].

In [5], Duprat and Muller present a new operator, a *polynomier*, suitable for VLSI implementation, and specifically designed for polynomials computations. This polynomier is composed of two pipe-lined subparts : a *squarer* (i.e. an operator able to compute the square of a number), and a *binomier* (i.e. an element which computes expressions of the form $Ax+B$).

In the first part of this paper, we present some possible applications of the polynomier. Then, we recall briefly the main characteristics of the architecture proposed in [5], in order to give a background for understanding the two following sections. The third part of the paper is devoted to a VLSI implementation of this architecture, designed in the Ecole Polytechnique Fédérale de Lausanne, Switzerland. The final part presents an extension of this architecture to the two's complement computation.

Possible applications

An interesting application of the polynomier is the computation of elementary functions. In 1885, Weiertrass showed that every continuous function may be approximated as accurately as desired, in a given interval [a,b], by a polynomial. Chebyshev gave a characterization of the best polynomial of approximation of degree n of a conti-

TH0382-2/91/0000/0013$01.00 © 1991 IEEE

nuous function f in [a,b], and this characterization has been used in 1934 by Remes [11] in order to give an algorithm which computes this best polynomial approximation. Polynomial approximants of classical elementary functions may be found in [8] and [3]. In [5], Duprat and Muller show that the polynomier computes the sine, the cosine and the exponential in [0,1] with p significant bits, in time $T = \Theta(p \text{ Log } p)$.

Therefore, the polynomier may be used in order to compute any continuous function : it may be viewed as a "programmable coprocessor". The user has to provide to the polynomier the coefficients of a polynomial approximation of the function he wants to evaluate. A possible application is in the field of telecommunication : in data compression techniques, the input data (supposed to be a fixed-point number between 0 and 1), must be transformed following a concave function. This function – and its reverse – may be easily approximated by a polynomial.

Architecture of the polynomier

A recursive strategy

We evaluate polynomials using a recursive strategy : the polynomial $\sum_{i=0}^{n} a_i x^i$ is equal to $\sum_{i=0}^{\lfloor n/2 \rfloor} a_i x^i + x^{\lfloor n/2 \rfloor + 1} \left(\sum_{j=0}^{\lceil n/2 \rceil - 1} a_{j+\lfloor n/2 \rfloor + 1} x^j \right)$. Thus evaluating a polynomial of degree n is equivalent to evaluate two polynomials of degree n/2. These two evaluations may be performed in parallel (in practice, we perform them in pipe-line on the same operator). This kind of decomposition is due to G. Estrin ([6], [10]). For instance, if n is equal to 7, the different computations of the algorithm are described by the tree of Fig.1.

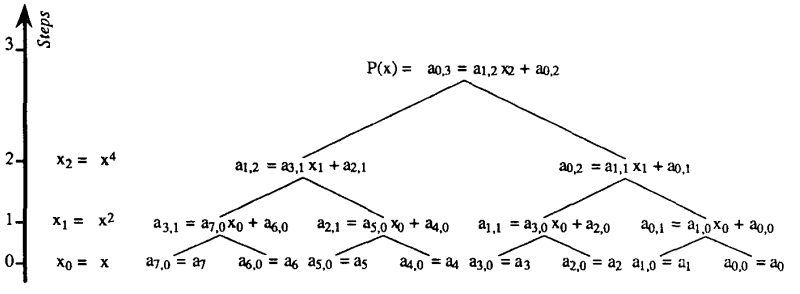

Fig. 1 The computational tree.

Architecture of the polynomier

Our operators are designed starting from Braun's cellular multiplier ([1],[9]). Cellular multipliers seems to be a good compromize between time of computation and circuit area. Fig. 2 presents such a multiplier, whose operands ($A_5A_4A_3A_2A_1A_0$) and ($B_5B_4B_3B_2B_1B_0$) are *unsigned numbers* . The square cells are classical Full-Adder cells.

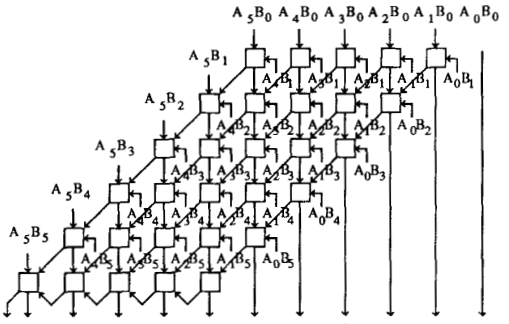

Fig. 2 Braun's multiplier.

The previous algorithm is implemented using two blocks : a *squarer*, which computes the successive squares $x_k = x^{2^k}$, and a pipe-lined *binomier*, which computes the expressions $a_{2m+1,k}x_k + a_{2m,k}$. Since we want to perform easily the multiplications, we shall suppose that all the terms x_k and $a_{i,k}$ have the same binary p-bit fixed point format. In [5] it is assumed that we manipulate fixed-point positive numbers, written $0.b_{-1}b_{-2}\ldots b_{-p} = \sum_{i=1}^{p} b_{-i}2^{-i}$. In such a notation, we have to assume :

- $0 \leq x < 1$
- the coefficients of P are positive
- $\sum_{i=0}^{n} a_i < 1$
- $p > n$ (n is the degree of P).

The third assumption is needed to avoid overflow. In our fixed-point format, we have to assume that P(x) is lower than 1 for $0 \leq x < 1$. Since the coefficients a_i of P are positive, the maximal value of P in [0, 1] is equal to $P(1) = \sum_{i=0}^{n} a_i$. Thus we have to assume that this value is lower than 1. Except for the second, these constraints are not really restrictive : the first and the third need a pre-normalization of data – necessary in order to avoid overflow –, and if P is the polynomial of best uniform approximation of an usual elementary function, this assumption is true (see for instance the approximations presented in [8] or [3]).

The binomier

In Braun's multiplier, the last row propagates the carry. It needs a time proportional to the size of the multiplier, thus this last row is not synchronous with the other rows. In a pipe-line implementation, one must suppress this row, thus the result of the multiplication will be given in a redundant "Carry-Save" notation : each digit of the result will be defined by a couple (carry, sum). A column is added to the left part of the multiplier in order to add the coefficient b of a binomial computation ax+b to the product ax. We shall use the elementary cells of Fig. 2, slightly modified (Fig. 3) in order to have a different topology.

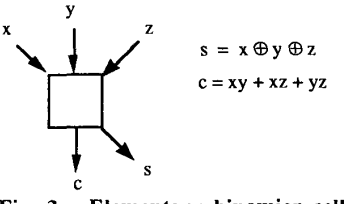

Fig. 3 Elementary binomier cell.

The binomier has the following structure (Fig. 4). The circular cells are AND gates :

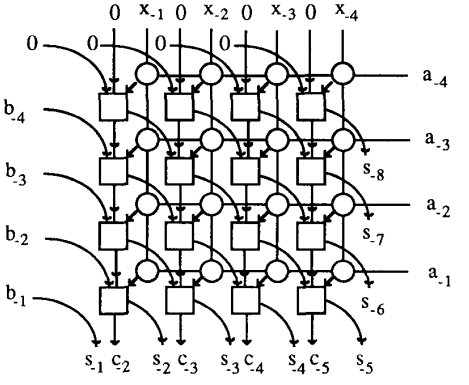

Fig. 4 A 4-bit binomier for unsigned numbers

The propagators

During the computation of ax + b, the outputs of the binomier will be used as coefficient a or b for the next step. Since these outputs are given in a redundant form, it is necessary to propagate a carry in order to obtain them in non-redundant binary form. At each step of the pipe line, only one bit of a and one bit of b are used, starting from the

least significant position. Therefore, the carry propagation may be executed in a triangular systolic propagator synchronized with the binomier (see Fig. 5).

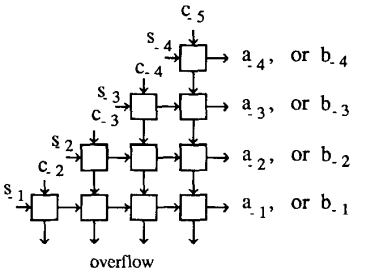

Fig. 5 A 4-bit propagator

The propagators make it possible to start the next iteration of the computation of the binomials during the current one. The high part of the binomier should start some calculations with x_{k+1}, while the lowest part ends some calculations with x_k. Thus, we must be able to segment the bus which carries x using registers at each level of the binomier.

The Squarer

We have to give to the binomier the successive squares x^{2^k}, at the rate of one per p cycles, where a cycle is the time needed by an elementary cell. If we use Braun's multiplier in order to compute $x^{2^{k+1}} = x^{2^k} * x^{2^k}$, the time of calculation is too large (2p cycles), because we cannot avoid the carry propagation : the terms x^{2^k} must not be in redundant form.

Let us consider the binary representation of x : $x = \sum_{j=-p}^{-1} x_j 2^j$. We have :

$$x^2 = \sum_{j=-p}^{-1} x_j 2^{2j} + 2 \sum_{\substack{-1 \le j \le p \\ -1 \le k < j}} x_j x_k 2^{j+k}$$

On Braun's multiplier, the terms x_j^2 are computed on the diagonal part, while the terms $x_j x_k$ are computed two times : on the upper triangular part, and on the lower triangular part. Thus, it is possible to perform the whole computation on the lower triangular part. In order to multiply by 2 the products $x_j x_k$, these products are shifted to the left in the multiplier. Thus it is possible to compute x^2 (in redundant form) in the lower triangular part in p cycles, using during the k^{th} cycle the k least significant bits of x, which can be given by a carry propagator synchronized with the multiplier.

In practice, it is easier to shift to the right the squares of the diagonal part, and to position the results instead of shifting the products $x_j x_k$. Using the cells of Braun's multiplier, we can obtain a triangular squarer. We shall not use such a squarer, since at each time, only one of its rows would be active, we shall use only one row, looped on itself (see Fig. 6). The connections will be different whether the cell is or is not on the diagonal part. We have to distinguish the cells by a control mechanism which shows what cell appears on the diagonal part at the time considered. This control is done by a token running into a shift register (j_{-4} j_{-3} j_{-2} j_{-1} j_0) synchronized with the circuit. The outputs of the shift register control two-ways multiplexers, represented by diamonds in Fig. 6 and 7 (the output is equal to the left input if the command is set to 1, else to the right input). The 3 input and 2 output rectangular cells are full adders, and circular cells are AND gates.

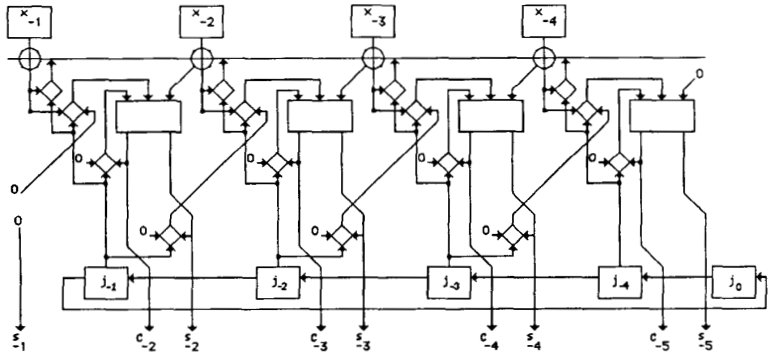

Fig. 6 Linear multiplier of a 4-bit squarer.

We need to convert the output values of this linear multiplier from carry-save representation to binary representation. This conversion is performed using a linear carry propagator depicted in Fig. 7, and is controlled by the bit j_0 of the shift register.

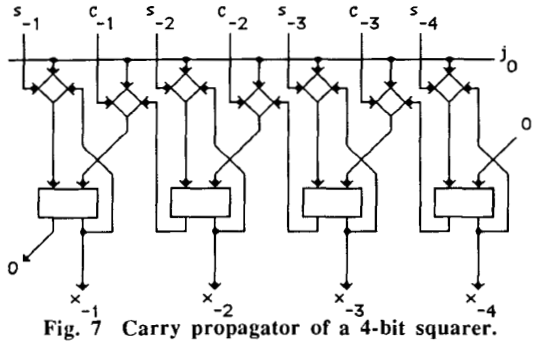

Fig. 7 Carry propagator of a 4-bit squarer.

VLSI implementation

Presentation

The design of the circuit was achieved using a 2 microns twin tub CMOS technology with 2 metal layers. In this version, only static logic has been used, although a fully dynamic version could be realized.

Since the propagators of the binomier have a triangular shape, it is possible to assemble them in such a way that they form a rectangular functional block. This leads to a very regular implementation of the whole circuit. Notice that it would be possible to use only one propagator, because only one of them is used at the same time. However, this would have led to some complication in the sequencer. Furthermore, the price for obtaining a compact regular shape for the remaining triangular propagator would be a loss of regularity. Figure 8 shows the disposition of the functional blocks of the data path. Since the squarer and the propagators have been designed in order to give to the binomier the correct data at the correct time, the control part is very simple.

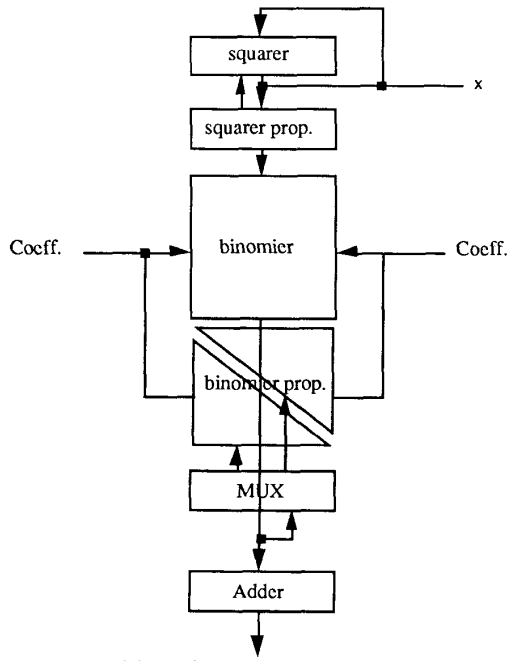

Fig. 8 Disposition of the functional bocks of the circuit

Basic elements

All the cells of the circuit are composed of a master-latch, a combinatorial element and a slave-latch. The most complex element is the combinatorial adder, for which we choose the well known symmetric version described in [13], which leads to a very regular and simple layout. The adder cell is associated with a NAND gate to form the multiplier cell. The elementary cell of the squarer is obtained by adding some passgates to the multiplier cell.

The schematic of the latch is given in Fig. 9. It is a 2-phase clocked latch, thus needing the use of clock amplifiers with inverting and non-inverting outputs. Note that between the clock generator and their destination, all clock signals pass through the same number of amplifiers, in order to equilibrate the clock skews.

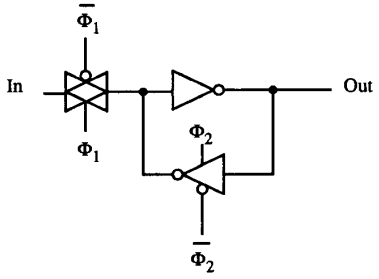

Fig. 9 Schematic of the latch.

Realization and performances estimation

We designed a version of a 12-bit polynomier. The size of the chip is 6.4mm x 3.2 mm. The clock frequency depends closely on the performances of the most complex gate, which is an elementary adder/multiplier cell. Electrical simulations using SPICE have shown that the propagation delay of such a gate is near to 2ns. Thus, taking in account synchronization latches, a 100 MHz clock rate may be easily reached. Previous works on pipelined multipliers design [12] have shown that higher clock rates may be achieved. With such a 12-bit polynomier, a polynomial of degree n may be computed in $120\lceil \log_2 n \rceil + T_{add}$ nanoseconds, where T_{add} is the time needed to convert the final carry-save result in 2's complement notation. In our implementation, T_{add} is near 10 ns.

Fig. 10 presents a floorplan of the circuit.

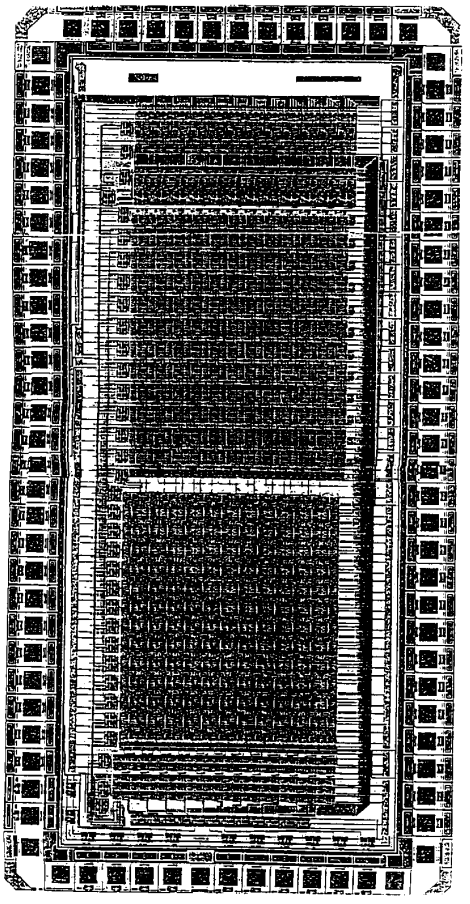

Fig. 10 Floorplan of the circuit.

Extension to 2's complement

In part II.b, we assumed the constraints :

- $0 \leq x < 1$
- the coefficients of P are positive
- $\sum_{i=0}^{n} a_i < 1$
- $p > n$ (n is the degree of P).

As we saw, the only really restrictive constaint was the second one. In order to avoid it, we have to manipulate 2's complement numbers. Then the second constraint vanishes, and the third becomes $\sum_{i=0}^{n} |a_i| < 1$. Now a number x, $|x| < 1$, will be represented by a sequence $x_0.x_1x_2 \ldots x_n$ satisfying : $x = - x_0 + \sum_{i=1}^{n} x_i 2^{-i}$.

In part II, we started from Braun's cellular multiplier. Now, we have to design our operators starting from 2's complement cellular multipliers.

Cellular 2's complement multipliers.

In [2], Baugh and Wooley present a cellular multiplier able to receive 2's complement numbers. This multiplier is presented in Fig. 11 (the elementary cells are the same as in Braun's multiplier).

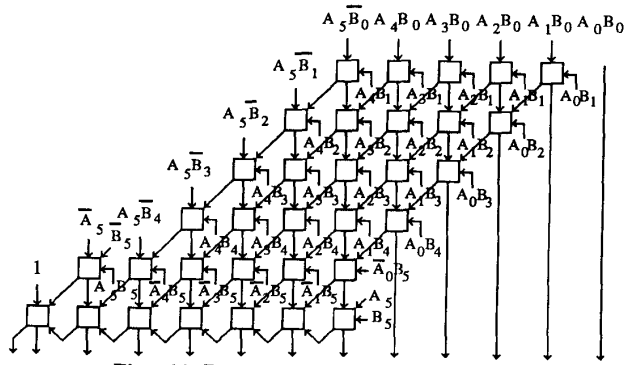

Fig. 11 Baugh and Wooley's multiplier.

More recently, Luiggi Dadda [4] proposed a more regular structure, depicted in Fig. 12.

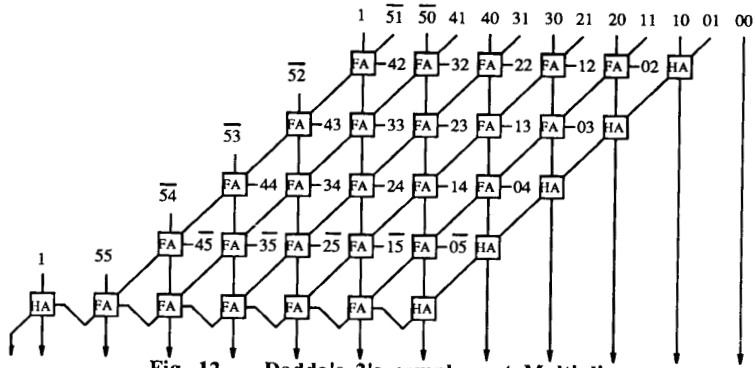

Fig. 12 Dadda's 2's complement Multiplier.

As in part II, from Dadda's multiplier, it it easy to build a binomier. This binomier is depicted in Fig. 13.

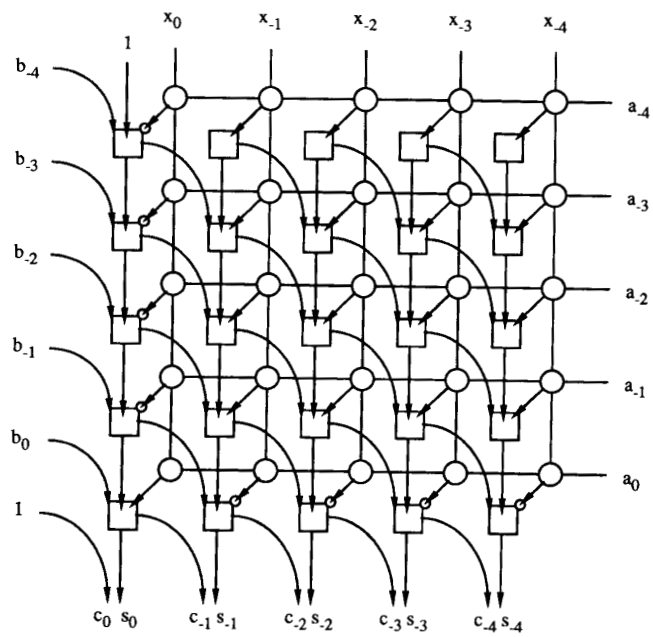

Fig. 13 A 2's Complement binomier.

We use the same squarer as in part II, with a slight modification in the first step (in the other steps, the terms x^{2^k} are positive). During the beginning of this first step, if the input x is negative (i.e. if $x_0 = 1$), then the bits $x_{i\ (i \leq -1)}$ are complemented before entering the squarer.

Conclusion

We have proposed an architecture dedicated to the evaluation of polynomials. This architecture may be used for evaluating any continuous function. Our VLSI realization is only a prototype, but it is sufficient to prove the feasibility of a complete circuit, and to show that high computational rates are achievable. Our circuit manipulates only positive numbers, but the last part of this paper shows that with a few modifications, one can obtain easily a 2's complement polynomier.

References

[1] E.L. Braun, *Digital computer design*, New York Academic, 1963.

[2] C.R. Baugh and B.A. Wooley, *A Two's complement parallel array multiplication algorithm*, IEEE Transactions on Computers, Col. C-22 No 12, December 1973.

[3] W. Cody and W. Waite, *Software manual for the elementary functions*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1980.

[4] Luiggi Dadda, 7thSymposium on Computer Arithmetic.

[5] J. Duprat and J.M. Muller, *Hardwired Polynomial Evaluation*, J. of Parallel and Distributed Computing 5, pp. 291-309, June 1988.

[6] G. Estrin, Proc. Western Joint Computing Conf. 17 (1960), pp 33-40.

[7] P.G. Fontolliet, *Systèmes de télécommunications*, Ecole Polytechnique Fédérale de Lausanne, Ed. Georgi, Presses Polytechniques romandes, Lausanne, Switzerland, 1983 (in French).

[8] J.F. Hart, E.W. Cheney, C.L. Lawson, H.J. Maehly, C.K. Mesztenyi, J.R. Rice, H.C. Tacher and C. Witzgall, *Computer approximations*, Wiley, New York, 1968.

[9] K. Hwang, *Computer arithmetic principles, architecture and design*, New York, J. Wiley & Sons Inc. 1979.

[10] D.E. Knuth, *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, Addison-Wesley, 1981.

[11] E. Remes, *Sur un procédé convergent d'approximations successives pour* déterminer les polynômes d'approximation, C.R. Acad. Sci. Paris, 198, pp. 2063-2065, 1934 (in French).

[12] D. Schmitt-Landsiedel, T.G. Noll, H. Klar, G. Enders, *A pipelined 330 Mhz multiplier*, 11th ESSCIRC, Toulouse, France, Sept. 16-18, 1985.

[13] N. Weste, K. Eshraghian, *Principles of CMOS VLSI design, A Systems Perspective*, Addison-Wesley VLSI systems Series, 1985.