Correspondence

Parallel Implementation of a 4×4-bit Multiplier Using Modified Booth's Algorithm

NARESH R. SHANBHAG, STUDENT MEMBER, IEEE, AND PUSHKAL JUNEJA

Abstract—**Presented in this paper is a design of a 4×4-bit multiplier using the modified Booth's algorithm in 2-μm NMOS technology. The main features of this chip are its 62.5-MHz operating frequency and 31.5-mW power dissipation. The chip occupies an area of 1.37 mm². A novel adder-cum-subtractor (ACS) circuit was designed to realize the arithmetic processing part.**

I. Introduction

Multiplier chips are undeniably a crucial component of any digital signal processing system. Hence the utility of a multiplier chip cannot be understressed.

We have employed a 2-μm NMOS design parameter [1] as it is widely used nowadays. Parallel implementation was resorted to in order to increase the speed of computation. A serial realization was attempted earlier for an 8×8 multiplier and it was found that five clock pulses were required to compute the result. Parallel realization, on the other hand, requires just one clock pulse to generate the result. The inputs and the output of the chip are in signed magnitude form.

While incorporating the modified Booth's algorithm [2] it was found that the arithmetic processing part involved addition as well as subtraction and hence an adder-cum-subtractor (ACS) module was designed. This module is unique in the sense that addition and subtraction are done without taking the two's complement at any stage. Subsequent cascading of the ACS modules was found to generate the result correctly. This module will be dealt with in greater detail later on.

II. Modified Booth's Algorithm

Multiplication of two numbers basically involves multiple shifts of the multiplier and subsequent addition, depending on the bit pattern of the multiplicand. Without using the modified Booth's algorithm [2] a maximum of four shifts (for a 4×4 multiplier) and four additions are required. On the other hand, through the modified Booth's algorithm, a maximum of three terms have to be evaluated and added up.

We now elaborate upon the facts given in the previous paragraph. Let A and B be a 4-bit multiplicand and multiplier, respectively. Therefore, A can be written as

$$A = a_3 \times 2^3 + a_2 2^2 + a_1 \times 2^1 + a_0.$$

Hence B has to be multiplied with each of the four terms and subsequently added up. For the modified Booth's algorithm it can easily be verified that A can also be represented as

$$A = a^4 a_3 + 2^2(-2a_3 + a_2 + a_1) + 2^0(-2a_1 + a_0 + 0).$$

Manuscript received October 12, 1987; revised January 4, 1988.

The authors are with the Indian Institute of Technology, New Delhi 110016, India.

IEEE Log Number 8821661.

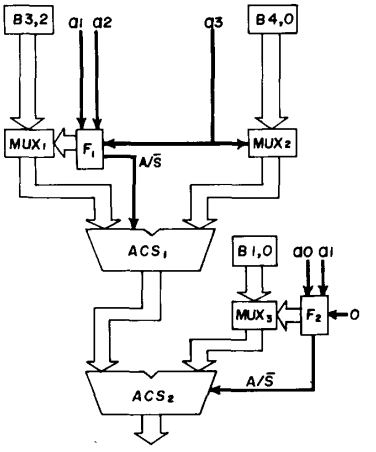


Fig. 1. System architecture.

It is clear that, except for the first term, the remaining two terms can be expressed individually as 2^i multiplied by a function F_i where

$$F_i = (-2a_{i+1} + a_i + a_{i-1}).$$

Now, when B is multiplied with A, depending on the value of the function F_i, the multiplier B will either have to be shifted left $i+1$ or i times and either added or subtracted from the partial result. We also note that for an $n \times n$-bit multiplier N terms have to be evaluated using the modified Booth's algorithm, where

$$N = (n+2)/2$$

This is the reason why the 8×8-bit serial implementation required five clock pulses.

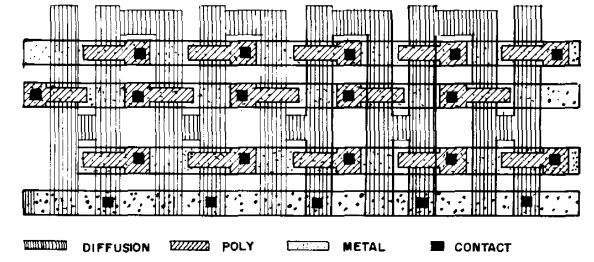
III. System Architecture

The system architecture that we chose inherently simplifies the timing analysis in Section VI because the flow of data is always in one direction, i.e., there are no data feedbacks among any of the modules. Thus we add the individual module delays to get the total system delay.

As mentioned before, the final result of the multiplier is the sum of properly shifted versions of B. The extent of the shift is determined by the value of the function F_i. Therefore, for all of the terms to be evaluated we note that we have to choose between a shift of $i+1$ and i. Hence we decided to multiplex the $(i+1)$th and the ith shifted versions of B. Therefore, in the 4×4-bit case we required three multiplexers.

Also two logic blocks, we call them F blocks, were required to generate the control signals for the multiplexers (for choosing the proper shifts) and for the ACS modules (for adding or subtracting the outputs of the multiplexers). As there were three terms to be added/subtracted, we required two ACS modules. These modules will be described in greater detail later.

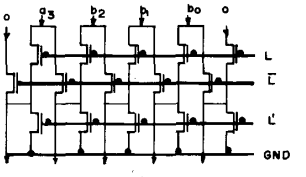
The final system architecture is presented in Fig. 1.



(a)

INPUT		OUTPUT
L	L'	
0	0	(i) SHIFT
0	1	0
1	0	((+1) SHIFT
1	1	0

(b)



(c)

Fig. 2. Multiplexer details: (a) layout, (b) truth table, and (c) logic diagram.

IV. Multiplexers and the F Block

A. Multiplexers

There were two 3×1 multiplexers on the chip. Two control signals L and L' controlled the generation of the output. In reality we had one register which acted as a source of two inputs and a ground line as the third. When $L'=1$ we get a zero as the multiplexer output irrespective of the status of L.

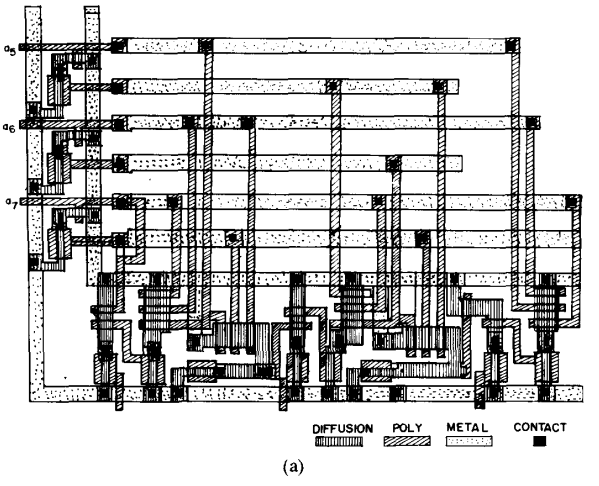
The third multiplexer (2×1) selects between a four shift left version of B and zero. The control signal in this case is a_3.

The layout, the truth table, and the logic diagram are given in Fig. 2.

B. F Block

The inputs to the F block are three bits of A and it generates the control signals L, L', and A/S. The function of the first two has already been explained while the third (A/S) is a signal to be combined with a similar output of another F block to be given to the ACS in order to convert it to either an adder or a subtractor.

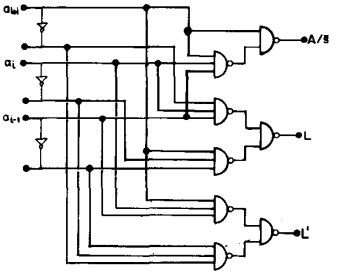
The layout, the truth table, and the logic diagram are shown in Fig. 3.



(a)

INPUT			OUTPUT		
a_{i+1}	a_i	a_{i-1}	L	L'	A/S
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

(b)



(c)

Fig. 3. F-block details: (a) layout, (b) truth table, and (c) logic diagram.

V. Adder-Cum-Subtractor Module

As mentioned earlier the arithmetic processing part would require addition as well as subtraction. Thus the ACS module has two modes of operation, one as an adder and the other as a subtractor. At no point is a two's complement required in this application of the ACS, and the final result is in signed magnitude form.

The adder is a conventional ripple carry adder while for the subtractor a novel design was implemented. The subtraction algorithm developed is given below. We first of all define a bit pair (BP) as the pair (a_i, b_i) of the two operands where B is the subtrahend. The steps are:

1) start from the unit's BP;
2) if the current BP is (0,1) then do step 3 onwards; otherwise take the XOR of the current BP as the output and repeat step 2 for the next BP unless the current BP is the last one;
3) generate 1 as the output of the current BP;
4) make the next BP the current BP;
5) transform the current BP to $(\bar{a}_i, b_i)$;
6) if $a_i = 0$ then do step 2 onwards; otherwise take the XOR of the bits of the current BP and do step 4 onwards.

The algorithm above assumes that for an n-bit subtraction there are $(n+1)$-bit pairs (including the sign-bit pair) plus a final bit pair which equals (1,0). The actual answer is given by the output of the right-most $(n+1)$-bit pairs.

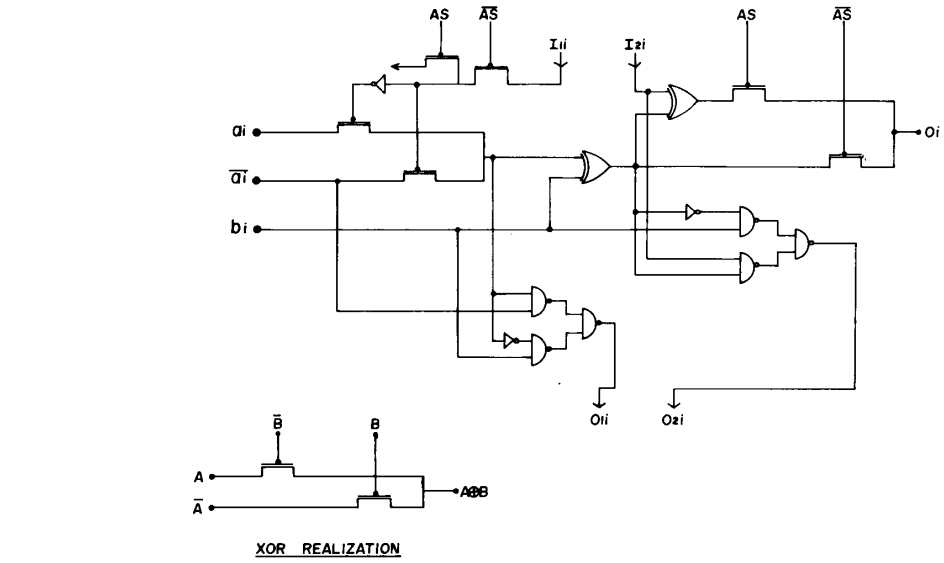
Upon realization of the circuit it was found that one bit slice of the ACS module contained 40 transistors. The design of the XOR gates was done using two pass transistors following the formal design procedure given in [3].

The logic diagram of a bit slice of the ACS module is shown in Fig. 4(a). It also shows the pass-transistor realization of XOR's. The layout is shown in Fig. 4(b).

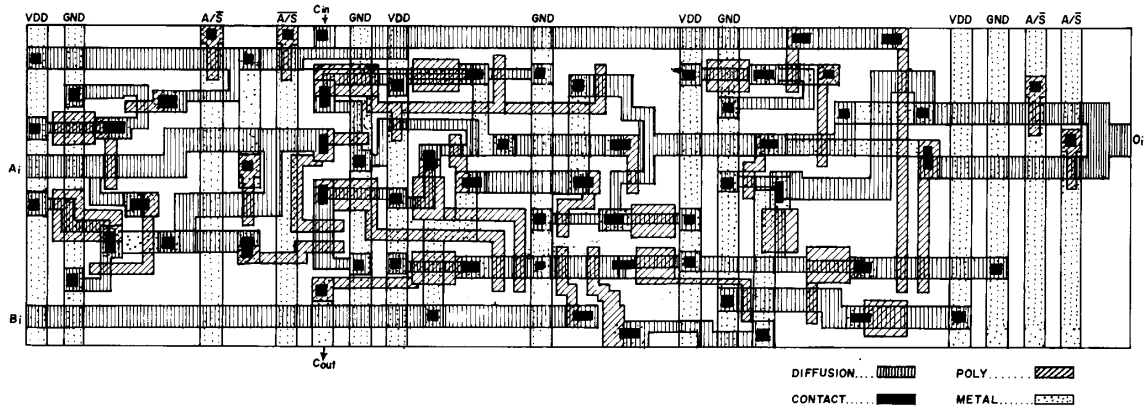
VI. Timing Analysis

It is quite reasonable to neglect the RC effects of wires if the wire resistance is much smaller than the ON resistance of a MOS transistor [4].

As nowhere on the chip do we have wire lengths greater than 2000λ (only then does the wire resistance become comparable to the ON resistance of a MOS transistor), the RC effects were ignored. Thus we modeled the wires as capacitive loads.



(a)



(b)

Fig. 4. ACS module: (a) logic diagram of bit slice showing pass-transistor realization of XOR's; and (b) layout.

The timing analysis was done by first identifying a critical path passing from the input pads through certain logic blocks to the output pads. This path was quite obvious in our architecture and is shown in Fig. 5.

For each of the modules lying in the critical path the delay was estimated and finally added up. If at any stage any module was found to delay the processing by a substantial amount, we identified the exact location of the problem area within the module and redesigned that part.

Within the modules themselves the delay was estimated by identifying critical paths. This identification was based on two considerations: 1) the number of gates along the path, and 2) the cumulative length of wires (or output capacitance faced by each gate in the path). The maximum of all the path delays within the module represented the delay of the module as a whole. Delays for each of the modules are tabulated in Table I. It can be seen that the ACS module is responsible for 81 percent of the total delay.

We would like to point out that no layout optimization was carried out on any of the modules and each one of them was a one-go affair. Appreciable reduction in the delays can be expected if such an optimization is carried out.

VII. Conclusions

A multiplier chip using the modified Booth's algorithm was designed and simulated. The whole chip contained 250 gates and occupied an area of 1.37 mm^2. It was designed for a supply voltage of 5 V.

The performance of the chip in terms of area and power dissipation is quite comparable to the one given in [5] though it lags far behind in the domain of execution time. The reason for this is the inherently slow NMOS technology and the absence of any layout optimization. Comparison with other multipliers is given in Table II. The data were culled from [5] and [13].

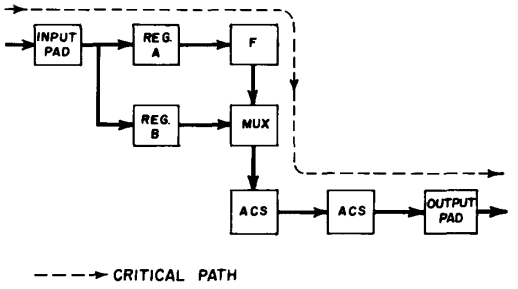


Fig. 5. Critical path selection.

TABLE I
DELAY IN NANOSECONDS

MODULE	DELAY
PAD	0·961
REGISTER	0·456
F_BLOCK	1·653
MUX	0·06
ACS	6·517

TABLE II
COMPARISON OF DIFFERENT MULTIPLIERS

S.NO	TIME (ns)	AREA (mm²)	POWER (mw)	SIZE	TECHNOLOGY	REFERENCE
1	120	5·8X6·3	-	16X16	2·7um, nE/D MOS	[6]
2	9·5	0·61X0·58	600	8X8	1um, NMOS	[7]
3	1·1	0·082 X 0·132	31	16X16	2·5um, JOSEPHSON	[8]
4	3·03	2·1X2·1	1500	8X8	1·0um, NMOS	[9]
5	0·21	-	3	4X4	2·5um, JOSEPHSON	[10]
6	2·7	1·7X2	900	8X8	SIOCS	[11]
7	14·3	-	250	8X8	2·5um, CMOS	[12]
8	6·0	42X37	876	8X8	1um, JFET DCFL	[13]
9	16·0	13X10·5	31·5	4X4	2um NMOS	This one

When extended to 8×8-bit systems we require $N\ (=5)$ F blocks, $N-1\ (=4)$ ACS modules, and $N\ (=5)$ multiplexers. Thus we anticipate an increase of 13 ns in the system delay and an increase of 31.9 mW in the total power dissipation. The ACS module accounts for nearly 76.6 percent of the increment in the power dissipation and it is responsible for all of the incremental delay of 13 ns.

ACKNOWLEDGMENT

The authors would like to acknowledge the constructive criticisms, suggestions, and the whole-hearted encouragement offered by Prof. D. Nagchoudhuri of the Indian Institute of Technology, Delhi. They also wish to thank the reviewers for their helpful suggestions. The heroic efforts of P. Ram, S. M. Pall, and V. S. Rao in the preparation of the manuscript and the figures is appreciated.

REFERENCES

[1] L. A. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*. Reading, MA: Addison-Wesley, 1985.
[2] O. L. Mac Sorley, "High speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, Jan. 1961.
[3] D. Radhakrishnan, S. R. Whitaker, and G. K. Maki, "Formal design procedures for pass transistor switching circuits," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 2, pp. 531–536, Apr. 1985.
[4] N. Weste and K. Eshraghian, *CMOS VLSI Design*. Reading, MA: Addison-Wesley, 1985.
[5] E. Delhaye, C. Rocher, J. C. Baelde, J. M. Giabereau, and M. Rocchi, "A 2.5-ns, 40-mW, 4×4 GaAs multiplier in two's complement mode," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 3, pp. 409–414, June 1987.
[6] Y. Harata, Y. Nakamuya, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 1, pp. 28–34, Feb. 1987.
[7] J. Y. Lee, H. L. Garvin, and C. W. Slayman, "A high-speed high-density silicon 8×8-bit parallel multiplier," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 1, pp. 35–40, Feb. 1987.
[8] S. Kotani, N. Fujimaki, S. Morohashi, S. Ohara, and S. Hasuo, "Feasibility of an ultra-high-speed Josephson multiplier," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 1, pp. 98–103, Feb. 1987.
[9] T. G. Noll, D. Schmitt-Landsiedel, H. Klar, and G. Enders, "A pipelined 330-MHz multiplier," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 3, pp. 411–416, June 1986.
[10] Y. Hatano, Y. Harada, K. Yanaashita, Y. Tarutani, and U. Kawabe, "A 4-bit$\times$4-bit multiplier and 3-bit counter in Josephson threshold logic," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 4, pp. 606–612, Aug. 1987.
[11] K. Washio, K. Nakazato, and T. Nakamura, "2.7-ns 8×8-bit parallel array multiplier using sidewall base contact structure," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 4, pp. 613–614, Aug. 1987.
[12] M. Hatamian and G. L. Cash, "A 70-MHz 8-bit$\times$8-bit parallel pipelined multiplier in 2.5-μm CMOS," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 4, pp. 505–513, Aug. 1986.
[13] K. Gonoi, I. Honbori, M. Wada, K. Togashi, and Y. Kato, "A GaAs 8×8-bit multiplier/accumulator using JFET DCFL," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 4, pp. 523–529, Aug. 1986.

Implementation of Iterative Networks with CMOS Differential Logic

SHIH-LIEN LU

Abstract—**Differential CMOS logic has many advantages over the static CMOS implemented NAND/NOR logic families. It has the ability to evaluate a complex combinational function in a single gate delay. It also provides high layout density and requires no static power consumption. In this paper we present an improved differential CMOS logic family—Enabled/disabled CMOS Differential Logic (ECDL). We also present an extension to this logic technique which enables the implementation of iterative networks. Iterative networks are useful in realizing some important logic functions. The simplicity of design, the ease of testing, and the small area–time product make iterative networks a desirable VLSI implementation method of logic functions. Two simple logic functions are implemented to demonstrate this methodology.**

I. INTRODUCTION

Differential CMOS logic family has many advantages. Heller *et al.* [1] proposed a differential CMOS logic family, cascode voltage switch logic (CVSL). They have shown, with examples, that both n-type and p-type transistor counts are reduced over the two-level NAND/NOR implementation. Since CVSL is capable

Manuscript received October 27, 1987; revised January 26, 1988. This work was supported in part by the Defense Advanced Research Project Agency under Contract MDA903-81-C-0335.

The author is with the Information Science Institute, University of Southern California, Marina del Rey, CA 90292 and with the Department of Computer Science, University of California, Los Angeles, CA 90024.

IEEE Log Number 8822143.