Correspondence

Correction to "A Carry-Free 54 b × 54 b Multiplier Using Equivalent Bit Conversion Algorithm"

Y. Kim, B.-S. Song, J. Grosspietsch, and S. F. Gillig

The algorithm in the above paper[1] is incorrect. When, as stipulated, it is implemented without carry-propagating adders, the circuit produces incorrect results. The actual fabricated multiplier does produce correct results with the carry-free conversion block. However, carry propagation was inadvertently introduced to the StageD RB adders.

The authors regret the error and thank the readers who have pointed out the need for carry propagation. In particular, the authors wish to acknowledge the efforts of Prof. W. Rülling for pinpointing the flaw in the implementation.

Manuscript received September 27, 2002.

Y. Kim, J. Grosspietsch, and S. F. Gillig are with Motorola, Inc., Schaumburg, IL 60196 USA.

B.-S. Song is with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA.

Digital Object Identifier 10.1109/JSSC.2002.806268

[1] *IEEE J. Solid-State Circuits*, vol. 36, pp. 1538–1544, Oct. 2001.

A Remark on Carry-Free Binary Multiplication

Wolfgang Rülling

***Abstract*—It is shown that any binary multiplier needs some mechanism for carry propagation. As a consequence, the carry-free multiplier presented in the paper by Kim *et al.* cannot work correctly. To demonstrate that fact, implementation-independent test patterns are constructed.**

***Index Terms*—Adders, algorithms, encoding, multiplication, redundant number systems.**

I. Introduction

Recently, in [1], a very fast 54 b × 54 b multiplier with low power consumption was introduced. It uses a redundant number representation (RB) for summing up partial products without using carry chains. Finally, the redundant result is converted into the normal binary representation (NB) using a carry-free equivalent bit conversion algorithm (EBCA).

In the following, it will be shown that, for large data lengths, any binary multiplier needs some mechanism for carry propagation. This is demonstrated by constructing general test patterns. Simulating the circuit from [1], it turns out that, indeed, incorrect results are computed. This is due to an error in the basic cell for carry-free RB-to-NB conversion. Unfortunately, correcting the basic cell means introducing a carry chain, thus increasing the computation time.

Manuscript received December 11, 2001; revised September 27, 2002.

The author is with the Fachhochschule Furtwangen, D-78120 Furtwangen, Germany (e-mail: ruelling@fh-furtwangen.de).

Digital Object Identifier 10.1109/JSSC.2002.806267

TABLE I
TRUTH TABLE FOR CORRECT RB-TO-NB CONVERSION

		$ENI=0$			$ENI=1$		
X	Y	Z_1	Z_2	ENO	Z_1	Z_2	ENO
0	0	0	0	0	1	1	**1**
0	$\overline{1}$	1	1	1	1	0	1
0	1	0	1	0	0	0	0
$\overline{1}$	0	1	0	1	0	1	1
$\overline{1}$	$\overline{1}$	0	1	1	0	0	1
$\overline{1}$	1	1	1	1	1	0	1
1	0	1	0	0	0	1	0
1	$\overline{1}$	0	1	0	0	0	0
1	1	1	1	0	1	0	0

II. RB-to-NB Conversion

The essential idea of converting redundant numbers to normal binary numbers is to replace every pair (X, Y) of adjacent RB digits by two bits (Z_1, Z_2). For that conversion, one has to consider an incoming carry ENI from the right and has to produce a carry ENO to the left. Table I specifies the truth table of the elementary converter cell satisfying the following equation:

$$2 \cdot X + Y - \mathrm{ENI} = 2 \cdot Z_1 + Z_2 - 4 \cdot \mathrm{ENO}.$$

In the equation, bits (0,1) and RB symbols $(\overline{1}, 0, 1)$ are interpreted as integers (using $\overline{1} = -1$). This way, any input vector is replaced by a numerically equivalent output vector. For example, the RB number $01\,00\,00\,\ldots\,00\,00\,0\overline{1}$ is replaced by the output $00\,11\,11\,\ldots\,11\,11\,11$, demonstrating the effect of a carry chain.

In [1], a slightly different truth table is used. For $(X, Y) = (0, 0)$ and $\mathrm{ENI} = 1$, the output ENO is changed from 1 to 0. As a consequence, the output ENO becomes independent from the input ENI and the carry chain is broken. Thus, whenever the multiplier produces a redundant result containing a long sequence of type $\ldots 000\overline{1}\ldots$, the RB-to-NB conversion will fail.

III. Constructing Test Patterns

To demonstrate the fault by an easy-to-understand example, we use the fact that a multiplier computing $y = u * v$ can also be used to add two numbers. For any binary numbers a and b at most $n = 26$ in length, we construct the input vectors $u = a \cdot 2^n + b$ and $v = 2^n + 1$. Then, the correct result $y = u * v$ becomes $y = a \cdot 2^{2n} + (a + b) \cdot 2^n + b$ and the binary representation of y is the concatenation of the binary numbers a, $a + b$ and b. Especially, the output bits $(y_{2n-1}, y_{2n-2}, \ldots, y_n)$ are the binary representation of $a + b$. Please also note that there are only two bits of v set to one. The Booth coding of v contains two ones and all other Booth digits are zero. As a consequence, only two partial products are needed to compute the result y. The other partial products and the error correction vector do not depend on a. This way, we can study how the multiplier computes $a + b$ for arbitrary a and b. Since binary addition needs some mechanism for carry propagation like the carry