Array Multiplier with an improved propagation

A benefit of the Binary Stored-Carry-or-Borrow representation

Daniel Torno, *Member, IEEE Computer Society*
Exorand Technology, http://www.exorand.com
d.torno@computer.org

***Abstract*— This paper introduces an implementation of an array multiplier with an improved propagation. The well-known initial AND matrix is transformed and expressed only with +1 and -1 values. Each line of the resulting matrix is processed by an accumulation operator using the binary stored carry-or-borrow (BSCB) representation and the digit set {-1, 0, +1, +2}. Thanks to the use of pre-computed assumptions, this operator improves the propagation delay by generating carries and borrows which reduce the propagation. Propagation delay and complexity in terms of gates are estimated and compared to BSCB known design.**

Keywords—array multiplier; redundant binary representation; propagation, binary stored-carry-or-borrow representation; pre-computed assumptions.

I. INTRODUCTION

In [1] Avizienis presented redundant number representations to allow fast addition by eliminating the carry propagation. Redundant number representations were later studied by Parhami [2], Phatak and Koren [3], Takagi [4]. These representations are extensively used in adder and multiplier implementations especially the stored-carry form. In [5] Takagi, Yasuura and Yajima introduced a multiplication algorithm using a redundant number expression with a radix 2 and the digit set {-1, 0, +1}. In this paper we use a redundant number expression with a radix 2 called binary stored-carry-or-borrow (BSCB) representation by Parhami and the digit set {-1, 0, +1, +2}. This representation is similar to the representation called $SD3^{(+)}$ by Phatak, Goff and Koren [6] but with a different encoding of the digit values. Various BSCB arithmetic operator implementations with the digit encoding of the present paper are described by Torno and Parhami [7] and are suitable for design of array multipliers. A first design of a BSCB multiplier was proposed showing that the propagation of carries and borrows leads to similar complexity and performances than the one of stored-carry implementation.

Thanks to their regular structure array multipliers are suitable to VLSI implementations. The computation delay is proportional to the bit length of the operands but this drawback is counterbalanced by the ability to support pipelining. Implementations using the stored-carry representation are described in the literature (see Hwang [8]). In all these implementations, the carry is propagated from one position (*n*) in a line (*i*) to the next position (*n+1*) in the next line (*i+1*) as shown by Fig.1. In the design proposed in [7], carries and borrows are propagated in the same way.

The method described in this paper shows that a more efficient way of generating the propagation signals can improve the speed performances.

First in section II we show that the partial product matrix generated by the AND operation over the multiplier and multiplicand can be transformed in another matrix expressed using the digit set {-1, +1}. Then in section III we describe the accumulation process which is similar to the standard stored-carry one but results are expressed in BSCB representation using the digit set {-1, 0, +1, +2}. We demonstrate that thanks to the use of pre-computed assumptions the accumulation process generates carries or borrows for position (*n+1*) in the line (*n+2*) thus accelerating the propagation. In section IV we deduce Boolean equations for the output signals of the accumulator cell and we obtain a rather direct implementation in EXOR gates and AND gates. Finally in section V we estimate the maximum propagation delay and the gate complexity of this design and we compare the results to those from the BSCB array multiplier described in [7].

II. TRANSFORMATION OF THE INITIAL AND MATRIX

Capital letters are used to express integer variables and small letters are used to express Boolean variables.

Let the multiplier and multiplicand be two N-bit binary numbers denoted by X and Y respectively:

$$X = x_{N-1}.2^{N-1} + x_{N-2}.2^{N-2} + ... + x_1.2^1 + x_0.2^0$$
$$Y = y_{N-1}.2^{N-1} + y_{N-2}.2^{N-2} + ... + y_1.2^1 + y_0.2^0$$

Let the product be a 2N-bit binary numbers denoted by Z:

$$Z = z_{2N-1}.2^{2N-1} + z_{2N-2}.2^{2N-2} + ... + z_1.2^1 + z_0.2^0$$

For N= 8: $Z = X.Y = \left[\sum_{n=0}^{n=7} x_n.2^n\right].\left[\sum_{i=0}^{i=7} y_i.2^i\right]$

Let be a_n^i be the value of the partial sum for the line (*i*) and column (*n*) : $a_n^i = x_{n-i}.y_i$. Fig. 2 shows an example of a 8x8 bits matrix named A composed of partial sum values a_n^i with input values $x_7...x_0 = 11100101$ and $y_7...y_0 = 10111001$.

This AND matrix A (upper part) is transformed into an EXOR matrix named E (lower part).

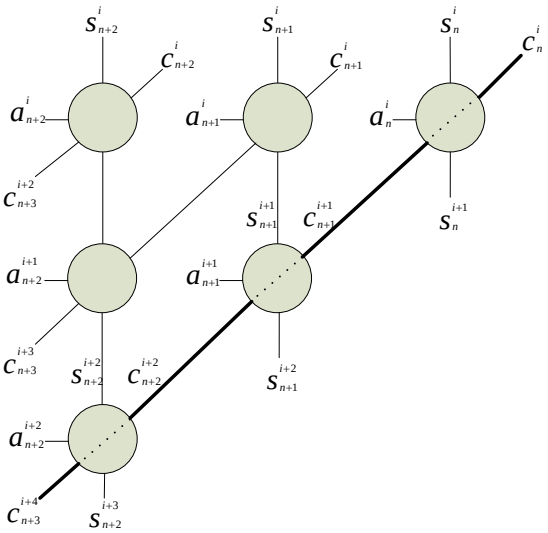

Fig. 1. Propagation in standard array multiplier

This transformation is realized through two steps:

- Horizontal propagation of +1 values of the matrix A across 0 values (see Fig. 4);
- Diagonal propagation of resulting +1 and -1 across 0 values (see Fig.5 and Fig. 6).

AND matrix A

	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
y_0	1	1	1	0	0	1	0	1
y_1	0	0	0	0	0	0	0	0
y_2	0	0	0	0	0	0	0	0
y_3	0	0	0	1	0	1	0	0
y_4	0	0	1	0	1	0	0	0
y_5	0	1	0	1	0	0	0	0
y_6	1	0	1	0	0	0	0	0
y_7	0	1	0	0	0	0	0	0

EXOR matrix E

0	0	0	0	0	0	0	-1	-1	-1	-1	1	1	-1	1
0	0	0	0	0	0	-1	-1	-1	-1	1	1	-1	1	0
0	0	0	0	0	1	1	1	1	-1	-1	1	-1	0	0
0	0	0	0	1	1	1	1	-1	-1	1	-1	0	0	0
0	0	0	1	1	1	1	-1	-1	1	-1	0	0	0	0
0	0	-1	-1	-1	-1	1	1	-1	1	0	0	0	0	0
0	1	1	1	1	-1	-1	1	-1	0	0	0	0	0	0
1	1	1	1	-1	-1	1	-1	0	0	0	0	0	0	0

Fig. 2. Transformation of a AND matrix (example)

Values x_8 and y_8 (outside of the X and Y ranges) are assumed to be 1.

Let be E_n^i the value of the matrix E for the line (*i*) and column (*n-i*) (see Fig. 6 below).

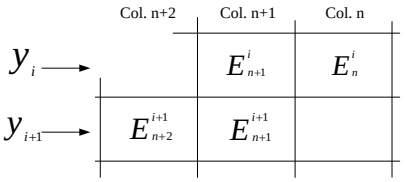

Fig. 3. Values of matrix E

Fig.3, Fig.4 and Fig.5 below shows examples of such propagations.

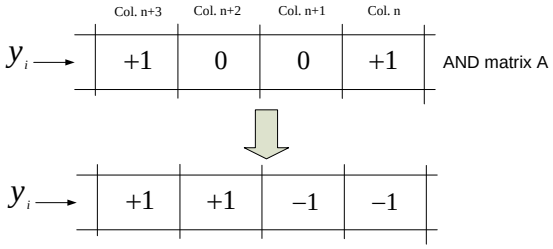

Fig. 4. Horizontal propagation (example)

In the above example, -1 is propagated until the position $x_{n-i+3} = 1$:

$$a_n^i.2^n = x_{n-i}.y_i.2^n = \left(x_{n-i+2}.2^{n+2} - x_{n-i+1}.2^{n+1} - x_{n-i}.2^n\right).y_i$$

Or : $a_n^i.2^n = \left((-1)^{x_{n-i+3}}.2^{n+2} + (-1)^{x_{n-i+2}}.2^{n+1} + (-1)^{x_{n-i+1}}.2^n\right).y_i$

In the same way, resulting -1 and +1 are propagated across 0 values in diagonal direction, see Fig.4 and Fig.5 below.

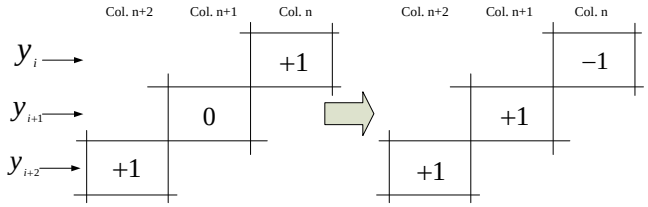

Fig. 5. Diagonal propagation of +1 value

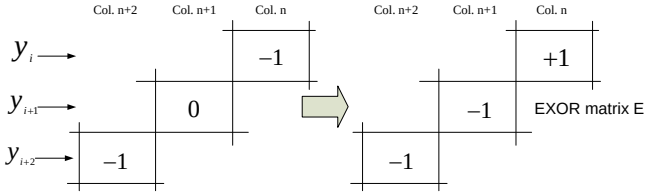

Fig. 6. Diagonal propagation of -1 value

Resulting value at position (*n*, *i*) is: $(-1)^{x_{n-i+1}}.2^n.y_i$

$$(-1)^{x_{n-i+1}}.2^n.y_i = \left((-1)^{y_{i+2}}.2^{n+1} + (-1)^{y_{i+1}}.2^n\right).(-1)^{x_{n-i+1}}$$

$$(-1)^{x_{n-i+1}}.2^n.y_i = (-1)^{(y_{i+2} \oplus x_{n-i+1})}.2^{n+1} + (-1)^{(y_{i+1} \oplus x_{n-i+1})}.2^n$$

At the end of the transformation, resulting value E_n^i of the matrix E at position (*n-i, i*) is depending of values x_{n-i+1} and y_{i+1}. E_n^i can be expressed from the digit set {-1, +1} according to following equation: $E_n^i = (-1)^{(x_{n-i+1} \oplus y_{i+1})}$.

III. COMPUTATION OF THE SUM

We consider in this section the addition of accumulated values L_n^i expressed in the digit set {-1, 0, 1, 2} with values E_n^i of matrix E expressed in the digit set {-1, 1}.

Let be L_n^i and L_n^{i+1} the accumulated values respectively at position (*n, i*) and at position (*n, i+1*).

Let be P_n^i and P_{n+1}^{i+2} the incoming propagated values respectively at position (*n, i*) and at position (*n+1, i+2*).

Fig. 7 shows these values in matrix E.

Let be: $L_n^{i+1} = L_n^i + E_n^i + P_n^i - 2 * P_{n+1}^{i+2}$ Eq. 1

Eq. 1 represents the accumulation process of all E_n^i values of the column *n*.

With $L_n^i \in \{-1, 0, +1, +2\}$, $E_n^i \in \{-1, +1\}$ we demonstrate that if $0 \le L_n^i + E_n^i \le 1$, $\forall P_n^i \in \{-1, 0, +1\}$,

$\exists P_{n+1}^{i+2} \in \{-1, 0, +1\}$ such as: $0 \le L_n^{i+1} + E_n^{i+1} \le 1$ thus keeping Eq. 1 true for next iteration.

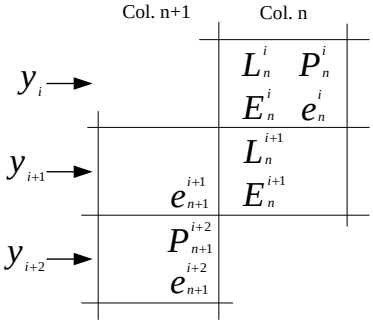

Fig. 7. Accumulation and propagation of matrix E

Let e_n^i, e_{n+1}^{i+1} and e_{n+1}^{i+2} be three Boolean variables such that: $e_n^i = \overline{x_{n-i+1} \oplus y_i}$ $e_{n+1}^{i+1} = \overline{x_{n-i+1} \oplus y_{i+1}}$ $e_{n+1}^{i+2} = \overline{x_{n-i} \oplus y_{i+2}}$.

According to Eq.1: $E_n^i = (-1)^{(x_{n-i+1} \oplus y_{i+1})} = (-1)^{\overline{e_{n+1}^{i+1}}}$

And: $E_n^{i+1} = (-1)^{\overline{e_{n+1}^{i+2}}}$

We assume that E_n^{i+1} and P_{n+1}^{i+2} have the same sign.

If $e_{n+1}^{i+2} = 0$ then $P_{n+1}^{i+2} \in \{-1, 0\}$ else $P_{n+1}^{i+2} \in \{0, +1\}$.

We have the same assumption for P_n^i :

If $e_n^i = 0$ then $P_n^i \in \{-1, 0\}$ else $P_n^i \in \{0, +1\}$.

Assuming $0 \le L_n^i + E_n^i \le 1$, let be l_n^i a Boolean variable such that: $l_n^i = (L_n^i + E_n^i)$.

According to all the above assumptions, Tab. I. displays possible values of: $(L_n^i + E_n^i + P_n^i + E_n^{i+1})$ related to e_n^i, e_{n+1}^{i+2}, l_n^i and P_n^i .

TABLE I. $(L_n^i + E_n^i + P_n^i + E_n^{i+1})$

		$l_n^i = (L_n^i + E_n^i)$ 0			$l_n^i = (L_n^i + E_n^i)$ 1			
		P_n^i			P_n^i			
e_n^i	e_{n+1}^{i+2}	−1	0	+1	−1	0	+1	
0	0	−2	−1		−1	0		$E_n^{i+1} = -1$
0	1	0	+1		+1	+2		$E_n^{i+1} = +1$
1	0		−1	0		0	+1	$E_n^{i+1} = -1$
1	1		+1	+2		+2	+3	$E_n^{i+1} = +1$

A first part named H_{n+1}^{i+2} of the propagation signal P_{n+1}^{i+2} can be applied on all cases surrounded by dotted lines in Tab. I.

If $e_n^i \oplus e_{n+1}^{i+2} = 1$ then $H_{n+1}^{i+2} = 0$,

else [if $e_{n+1}^{i+2} = 1$ then $H_{n+1}^{i+2} = 1$ else $H_{n+1}^{i+2} = -1$].

TABLE II. $(L_n^i + E_n^i + P_n^i + E_n^{i+1} - 2 * H_{n+1}^{i+2})$

		$l_n^i = (L_n^i + E_n^i)$ 0			$l_n^i = (L_n^i + E_n^i)$ 1		
		P_n^i			P_n^i		
e_n^i	e_{n+1}^{i+2}	−1	0	+1	−1	0	+1
0	0	0	+1		+1	+2	
0	1	0	+1		+1	+2	
1	0		−1	0		0	+1
1	1		−1	0		0	+1

Tab. II. displays resulting values of $(L_n^i + E_n^i + P_n^i + E_n^{i+1} - 2 * H_{n+1}^{i+2})$. A second part named K_{n+1}^{i+2} of the propagation signal P_{n+1}^{i+2} can be applied on all cases surrounded by dotted lines.

If $P_n^i = 1$ then $K_{n+1}^{i+2} = 0$,

else [if $e_n^i \oplus l_n^i = 1$ then $K_{n+1}^{i+2} = 1$ else $K_{n+1}^{i+2} = -1$].

Let be $P_{n+1}^{i+2} = H_{n+1}^{i+2} + K_{n+1}^{i+2}$, then Tab. III shows resulting values of l_n^{i+1} such that:

$l_n^{i+1} = L_n^{i+1} + E_n^{i+1} = L_n^i + E_n^i + P_n^i + E_n^{i+1} - 2 * P_{n+1}^{i+2}$

with $0 \le L_n^{i+1} + E_n^{i+1} \le 1$.

To complete the validation of the initial assumptions, it can easily be verified that:

If $e_{n+1}^{i+2} = 0$ then $P_{n+1}^{i+2} \in \{-1, 0\}$ else $P_{n+1}^{i+2} \in \{0, +1\}$, Eq. 2

IV. LOGICAL DESIGN OF THE ARRAY

In this section we convert the multiplication process described in the previous sections into Boolean equations and we detail the logic implementation.

Boolean variables e_n^i and e_{n+1}^{i+2} are expressing the signs of the input propagation signals at positions (*n, i*) and respectively (*n+1, i+2*).

Let be h_{n+1}^{i+2} a Boolean variable expressing an assumption of the propagation signal at position (*n+1, i+2*) such that: $h_{n+1}^{i+2} = \overline{e_n^i \oplus e_{n+1}^{i+2}} = \overline{x_{n-i+1} \oplus y_i \oplus x_{n-i} \oplus y_{i+2}}$.

It can be seen on Tab.I that part named H_{n+1}^{i+2} of the propagation signal P_{n+1}^{i+2} can be computed from variable h_{n+1}^{i+2} and e_{n+1}^{i+2}. Following figure Fig.8 shows the logical implementation of $\overline{h_n^i}$ and e_n^i computation at position (*n, i*):

$\overline{h_n^i} = e_n^i \oplus e_{n-1}^{i-2} = x_{n-i+1} \oplus y_i \oplus x_{n-i+2} \oplus y_{i-2}$

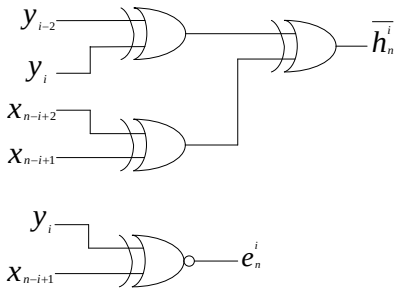

Fig. 8. Computation of assumption $\overline{h_n^i}$ and sign e_n^i

Let be p_n^i a Boolean variable such that:

if $P_n^i = 0$, then $p_n^i = 0$ else $p_n^i = 1$.

According to Tab.II, let be k_{n+1}^{i+2} a Boolean variable expressing the correction K_{n+1}^{i+2} of the propagation signal P_{n+1}^{i+2} at position (*n+1, i+2*) such that:

if $K_{n+1}^{i+2} = 0$, then $k_{n+1}^{i+2} = 0$ else $k_{n+1}^{i+2} = 1$.

k_{n+1}^{i+2} can be computed from variables l_n^i , e_n^i and p_n^i (cases surrounded by dotted lines in Tab. II.):

$k_{n+1}^{i+2} = \left(l_n^i \oplus e_n^i\right).\overline{p_n^i}$. Eq. 3

As H_{n+1}^{i+2} and K_{n+1}^{i+2} and are integer variables such that $-1 \le H_{n+1}^{i+2} \le +1$, $-1 \le K_{n+1}^{i+2} \le +1$ and $-1 \le H_{n+1}^{i+2} + K_{n+1}^{i+2} \le +1$, then $P_{n+1}^{i+2} = H_{n+1}^{i+2} + K_{n+1}^{i+2}$ implies: $p_{n+1}^{i+2} = h_{n+1}^{i+2} \oplus k_{n+1}^{i+2}$.

By substituting in Eq. 3 at position (*i,n*), we obtain:

$k_{n+1}^{i+2} = \left(l_n^i \oplus e_n^i\right).\left(h_n^i \oplus k_n^i\right)$ Eq. 4

From Tab. II. displaying resulting values of l_n^{i+1} (cases surrounded by solid line), we can deduce :

$l_n^{i+1} = l_n^i \oplus p_n^i = l_n^i \oplus h_n^i \oplus k_n^i$ Eq. 5

Fig.9 shows an implementation of the multiplication cell at position (*n, i*) according to Eq. 4 and Eq. 5.

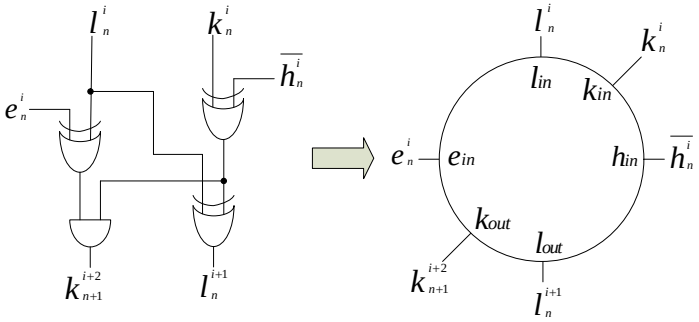

Fig. 9. Multiplication cell at position (*n, i*)

Fig. 10 shows how the array multiplier is designed with the multiplication cell symbolized in Fig. 9.

Propagation between cell at position (*i,n*) and cell at position (*i+2, n+1*) is put in bold line.

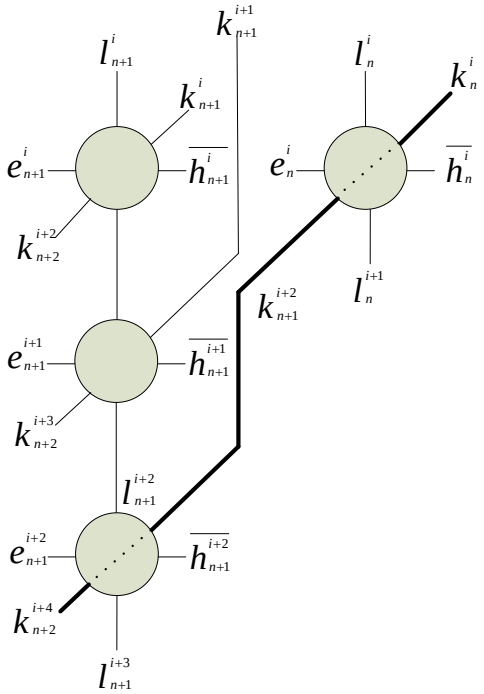

Fig. 10. Architecture of the array multiplier

For the final stage the redundant form has to be converted into the standard binary form.

First we consider the conversion over the first non-null diagonal (see Fig. 11 below).

At position (*n*,*i*), integer variables $\left(L_n^i + E_n^i\right)$ expressed by l_n^i is added to integer variable P_n^i expressed by $p_n^i = h_n^i \oplus k_n^i$.

Let r_{n+1} and u_n be two Boolean variables, the sum value $D_n = L_n^i + E_n^i + P_n^i$ of the diagonal is expressed according to the coding of Table III (BSCB coding used in [8]).

As $0 \le L_n^i + E_n^i \le 1$ and according to Eq. 3:

if $e_n^i = 0$ then $D_n^i \in \{-1, 0, 1\}$ else $D_n^i \in \{0, +1, +2\}$, Eq. 6

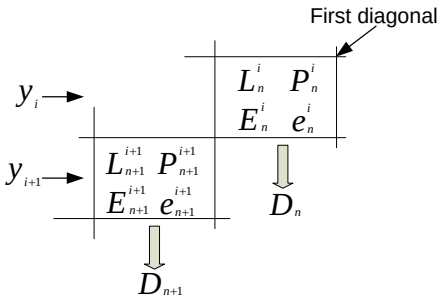

Fig. 11. Computation over the first diagonal

TABLE III. BSCB CODING OF DIAGONAL VALUES

D_n	−1	0	+1	+2
r_{n+1}	1	0	0	1
u_n	1	0	1	0

According to Tab. IV., sum $D_n = L_n^i + E_n^i + P_n^i$ is expressed in following way (r_{n+1} in dotted lines):

$$D_n : \begin{cases} u_n = l_n^i \oplus p_n^i = l_n^i \oplus h_n^i \oplus k_n^i \\ r_{n+1} = \left(l_n^i \oplus \overline{e_n^i}\right).p_n^i = \left(l_n^i \oplus \overline{e_n^i}\right).\left(h_n^i \oplus k_n^i\right) \end{cases}$$

Computation of r_{n+1} and u_n can easily be implemented with the same multiplication cell as the one described in Fig. 9 by simply complementing variables h_n^i and e_n^i of the first diagonal.

TABLE IV. SUM OF FIRST DIAGONAL $D_n = L_n^i + E_n^i + P_n^i$

	$l_n^i = \left(L_n^i + E_n^i\right)$ 0			1		
e_n^i	P_n^i −1	0	+1	P_n^i −1	0	+1
0	(−1)	0		0	+1	
1		0	+1		+1	(+2)

If we consider values of e_n^i we see that some cases never happen: $e_n^i = 0 \Rightarrow D_n^i \neq +2$ and $e_n^i = 1 \Rightarrow D_n^i \neq -1$.

Forbidden cases can be expressed by following equation:

$$f_n = \left(\overline{u_n} \oplus e_n^i\right).r_{n+1} \qquad \text{Eq. 7}$$

We consider now the conversion over the last row (N-1), see Fig. 12 below. An additional row with $y_N = 0$ is required to compute the final result due to k_n^{N+1} generated at position (*n-1, N-1*). The related value h_n^{N+1} is computed with $y_{N+1} = 1$ thus implying: $e_n^{N+1} = \overline{e_{n-1}^N}$.

Let $V_{n-1}^{N+1} = L_{n-1}^N + E_{n-1}^N + P_{n-1}^N$ be the sum value of the additional row. According to Eq. 6, if $e_{n-1}^N = 0$ (or $e_n^{N+1} = 1$) then $V_{n-1}^{N+1} \in \{-1, 0, 1\}$ else $V_{n-1}^{N+1} \in \{0, +1, +2\}$.

According to Eq. 2, if $e_n^{N+1} = 0$ then $P_n^{N+1} \in \{-1, 0\}$ else $P_n^{N+1} \in \{0, +1\}$.

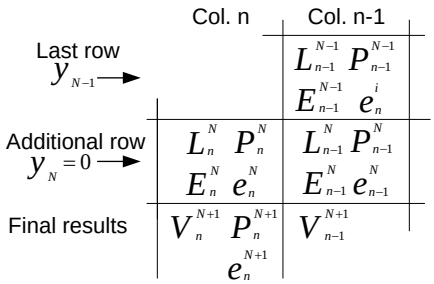

Fig. 12. Computation over the last row

Table VI. shows resulting values of sum R_n related to values of e_n^{N+1}, V_{n-1}^{N+1} , V_n^{N+1} and P_n^{N+1}.

TABLE V. SUM OF LAST ROW R_n

			$V_n^{N+1} \in \{0, +2\}$ $v_n = 0$			$V_n^{N+1} \in \{-1, +1\}$ $v_n = 1$		
e_n^{N+1}	V_{n-1}^{N+1}	w_{n-1}	P_n^{N+1} −1	0	+1	P_n^{N+1} −1	0	+1
0	+2	1	0	+1		(+1)	(+2)	
	{0, +1}	0	(−1)	0		0	+1	
1	−1	1		(−1)	0		0	(+1)
	{0, +1}	0		0	+1		(+1)	(+2)

Let w_{n+1} and v_n be two Boolean variables expressing the integer value V_n^{N+1} and let r_{n+1} and u_n be two Boolean

variables expressing the integer value R_n according to the coding of Table IV.

V_n^{N+1} and R_n (r_{n+1} being the intersection of dotted line boxes with solid line boxes) are expressed in following way:

$$V_n^{N+1}:\begin{cases} v_n^{N+1} = l_n^N \oplus p_n^N = l_n^N \oplus h_n^N \oplus k_n^N \\ w_{n+1} = \left(l_n^N \oplus \overline{e_n^N}\right).p_n^N = \left(l_n^N \oplus \overline{e_n^N}\right).\left(h_n^N \oplus k_n^N\right) \end{cases}$$

$$R_n:\begin{cases} u_n = w_n^{N+1} \oplus v_n^{N+1} \oplus p_n^{N+1} \\ r_{n+1} = \left(\overline{w_n^{N+1}} \oplus v_n^{N+1} \oplus e_n^{N+1}\right).\left(p_n^{N+1} \oplus v_n^{N+1} \oplus e_n^{N+1}\right) \end{cases}$$

By substituting $p_n^{N+1} = h_n^{N+1} \oplus k_n^{N+1}$ in both equations and rewriting :

$$R_n:\begin{cases} u_n = v_n^{N+1} \oplus w_n^{N+1} \oplus h_n^{N+1} \oplus k_n^{N+1} \\ r_{n+1} = \left(v_n^{N+1} \oplus w_n^{N+1} \oplus \overline{e_n^{N+1}}\right).\left(w_n^{N+1} \oplus h_n^{N+1} \oplus k_n^{N+1}\right) \end{cases}$$

Computation of r_{n+1} and u_n of the last row can also easily be implemented with the same multiplication cell as the one described in Fig. 9 by simply computing intermediate variables $\left(w_n^{N+1} \oplus h_n^{N+1}\right)$ and $\left(w_n^{N+1} \oplus \overline{e_n^{N+1}}\right)$.

Final result in binary standard form can be computed over u_n and r_{n+1} values either by using for example a BSCB ripple-carry adder as the one described in [7] and displayed Fig. 13 below or by converting it in stored-carry form and then using a standard ripple-carry adder or a carry-look-ahead adder.

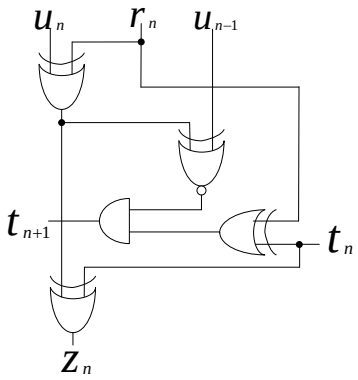

Fig. 13. BSCB Ripple-Carry-Adder

If we now consider values of e_n^N and V_n^{N+1}, as for the diagonal summation some cases never happen:

$e_n^N = 0 \Rightarrow V_n^{N+1} \neq +2$ and $e_n^N = 1 \Rightarrow V_n^{N+1} \neq -1$.

Forbidden cases can be expressed by following equation:

$$f_n = \left(\overline{v_n} \oplus e_n^N\right).w_{n+1} \qquad \text{Eq. 8}$$

At the end of the multiplication process, a reliability check can be implemented by verifying that Eq. 7 and Eq. 8 give a null result over the first diagonal and last row.

V. RESULTS

In this section we evaluate the maximum propagation delay, we estimate the complexity of the proposed design in terms of gates and we compare them to the BSCB design of [7].

The different propagation paths and their respective delays through the multiplication cell are displayed Fig. 14. As the propagation delays of XOR gates and AND gates are depending of the targeted technology, we express the propagation delays in this way:

$$t_{lk} = t_{ek} = t_{kk} = t_{hk} = t_{XOR} + t_{AND}$$
$$t_k = t_{hl} = 2*t_{XOR} \quad and \quad t_{ll} = t_{XOR}$$

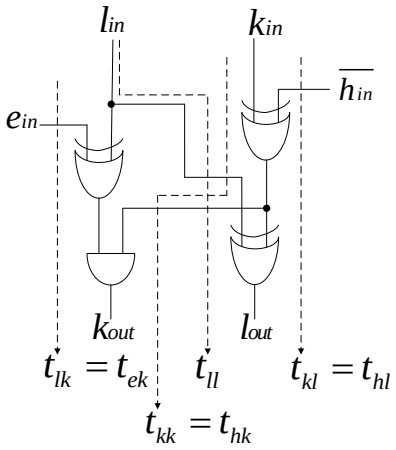

Fig. 14. Propagation delays through the multiplication cell

Fig. 15 shows the propagation delays through the three fist stages of the multiplication cells and the initialization gates (some extra gates are required to handle cases such as zero values at lsb positions).

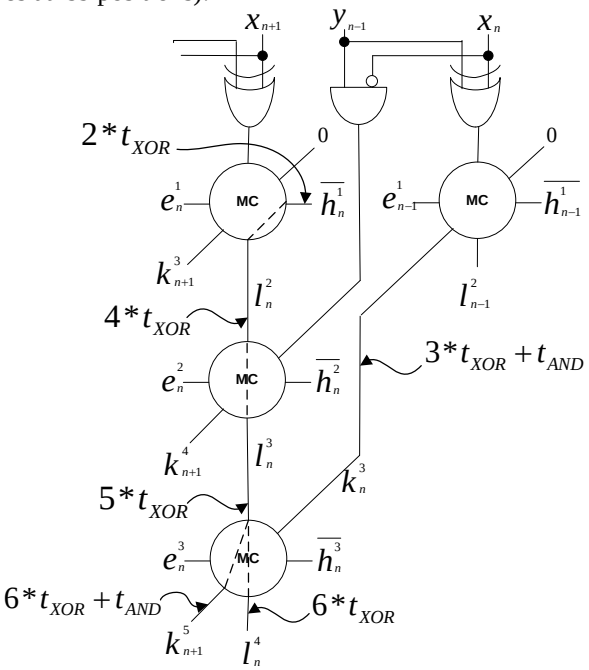

Fig. 15. Propagation delays through the initialization stages

We assume that in general case: $t_{XOR} > t_{AND}$. Due to the improved propagation path of the carry or borrow signal, the maximum propagation delay is directly depending on the propagation delay from input l to ouput l of the multiplication cell, that is to say $t_{ll} = t_{XOR}$. At the output of the three first stages of multiplication cells we obtain: $t_{l4\,MAX} = 6 * t_{XOR}$.

By taking into account the final stages (see Fig. 16), we can deduce following propagation delays for the outputs of a N bits multiplication array in BSCB representation form:

$$\begin{cases} t_{r\,MAX} = (N+6) * t_{XOR} + 2 * t_{AND} \\ t_{u\,MAX} = (N+7) * t_{XOR} + t_{AND} \end{cases}$$

These results can be compared to:

$$\begin{cases} t_{r\,MAX} = (N+1) * t_{XOR} + N * t_{AND} \\ t_{u\,MAX} = (N+2) * t_{XOR} + (N-1) * t_{AND} \end{cases}$$

which are the maximum propagation delays of the BSCB multiplier described in [7].

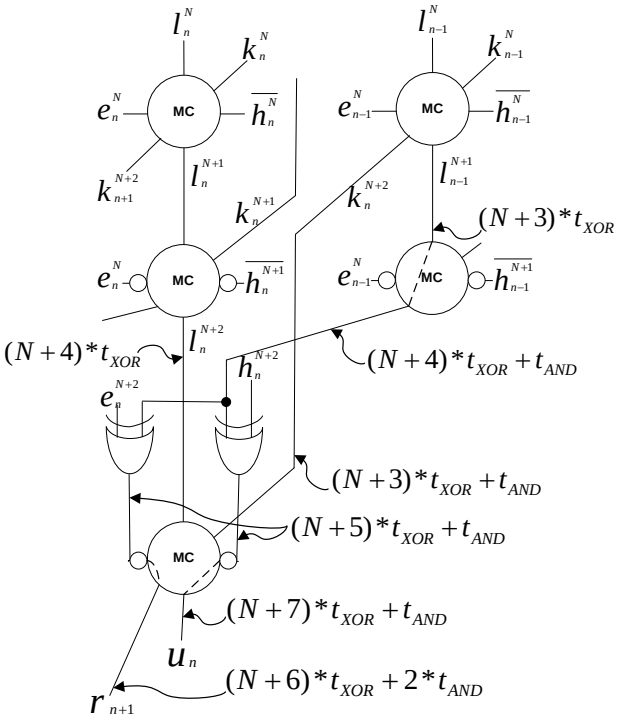

Fig. 16. Propagation delays through the final stages

If: $(N-2)*t_{AND} > 5*t_{XOR}$ then the proposed design improves the propagation delay by: $(N-2)*t_{AND}$.

We now estimate the complexity of the design. Table VI. below details the amount of XOR gates, AND gates and multiplication cells MC of a design implementing a M*N multiplication. The overall gate count is approximately: $4*N*(M+3)*XOR\ gates + N*(M+1)*AND\ gates$.

TABLE VI. COMPLEXITY ESTIMATION IN TERMS OF GATES

	AND	XOR	MC
h computation	0	(M+2) *(N+1)	0
e computation	0	M*N	0
Initialisation of inputs	N	N	0
Matrix of multiplication cells	0	0	M*N
Final stages	0	2*N	2*N

Gate count of the BSCB design described in [7] is: $4*N*M*XOR\,gates + N*M*AND\,gates$. The proposed design increases this gate complexity by adding roughly: $12*N*XOR\,gates$.

As already mentioned, all these comparisons are highly depending on the technology, especially on the performances of XOR gates.

VI. CONCLUSIONS

We have presented an array multiplier based on Binary Stored-Carry-or-Borrow representation with an improved propagation signal. Instead of starting from a dot-matrix, the process of accumulation is performed over a matrix of values in the digit set {-1, +1}. Propagation is theoretically improved by the use of pre-computed assumptions and accumulation values expressed in the digit set {-1, 0, +1, +2} but at the cost of design complexity which increase as a consequence. These results must consider some conditions related to the performances of XOR and AND gates in the targeted technology.

References

[1] Algirdas Avizienis, " Signed-Digit Number Representations for fast Parallel Arithmetic", I*RE Transactions on Electronic Computer*, Vol EC-10, pp 389-400, 1961.

[2] Behrooz Parhami, "Generalized Signed-Digit Number Systems: A unifying Framework For Redundant Number Representation", *IEEE Transactions on Computer*, Vol 39, no. 1, pp 89-98, January 1990.

[3] Dhananjay S. Phatak and Israel Koren, "Hybrid Signed-Digit Number Systems: A unified Framework for redundant Number Representations With Bounded Carry Propagation Chains", *IEEE Transactions on Computer*, Vol 43, no. 8, pp 880-891, August 1994.

[4] Naofumi Takagi,"Multiple-Value-Digit Number Representations in Arithmetic Circuit Algorithms", *32th IEEE International Symposium on Multiple-Value Logic*, May 2002.

[5] Naofumi Takagi, Hiroto Yasuura, Shuzo Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", *IEEE Transactions on Computer*, Vol c-34, no. 9, pp 789-796, September 1985.

[6] Dhananjay S. Phatak, Tom Goff and Israel Koren, "Constant-Time Addition and Simultaneous Format Conversion based on redundant Binary representations", *IEEE Transactions on Computer*, Vol 50, no. 11, pp 1267-1278, November 2001.

[7] Daniel Torno and Behrooz Parhami, "Arithmetic Operators Based on the Binary Stored-Carry-or-Borrow Representation," *Proc. 44th Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, pp.1148-1152, 7-10 November 2010.

[8] Kai Hwang, *Computer Arithmetic: Principles, Architecture and Design*, John Wiley and Sons, pp. 69-96, 1979.