Pipelined Addition, Accumulation and Multiplication of Binary Numbers on Cellular Automata

A. Clementi, G. A. De Biase and A. Massini

Dipartimento di Scienze dell'Informazione, Università di Roma la Sapienza
Via Salaria 113, 00198 Roma, Italy

Abstract

Some arithmetical operations on binary (or 2s complement) numbers performed on Cellular Automata (CA) are presented: *a)* by implementing on the CA the half-adder functions it is possible to perform a pipelined binary addition of binary number pairs which gives results every two machine-state transitions (after the start-up phase); *b)* by implementing the full-adder functions on the CA, the sum of N binary number of size N (i.e. the *accumulation* operation) can be obtained in $O(N)$ time; *c)* the accumulation operation is also used to perform multiplications between two binary numbers in $O(N)$ time. Implementations of these operations are made for the MIT CAM-8 machine.

u / l	0	1		
	0	1	u	sum
0	0	0	l	carry
	1	0	u	sum
1	0	1	l	carry

Table 1: Table derived from the half adder true table. This table acts on a pair of digits; u and l indicate the upper and lower rows respectively. In the output pair the lower digit is shifted left one position.

1 Introduction

The effectiveness of a computing model is related to its practical usability. From this standpoint arithmetic operations play a role of primary importance. The problem of performing addition or algebraic sum on a Cellular Automaton (CA) has been the subject of two recent works, in particular, a method to perform addition on binary numbers in $O(N)$ time has been presented by Sheth *et al.* in [6] and a method to perform addition in constant time using the redundant binary number representation has been presented by Clementi *et al.* in [7].

In this paper, procedures to perform some arithmetical operations on binary (or 2s complement) numbers using cellular automata are presented. The first procedure regards the *pipelined addition*: this procedure gives sums in constant time at every two state transitions of the cellular automaton. Then, a procedure to perform the *accumulation* (i.e. the addition of N numbers) on a CA is presented. This last procedure is based on the implementation of the full adder functions on the CA and requires $O(N)$ time, where N is both the number and the size of the operands. Finally, by using a suitable initial configuration for the cellular automaton, the accumulation operation is also used in order to perform multiplications between two binary numbers of size $N/2$ in linear time.

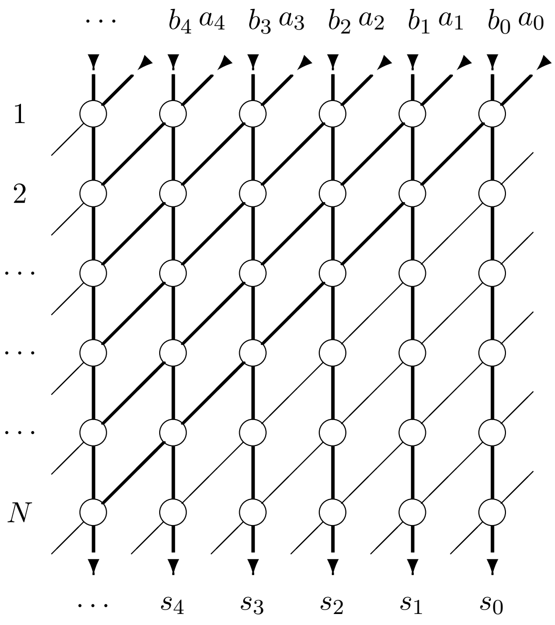
Implementations of these operations are made on the CAM-8 machine, which is the novel version of the Cellular Automaton Machine developed by the MIT laboratories [5].

2 A pipelined binary adder

In the binary number system an unsigned integer D is represented by a digit string $a_{N-1}, \ldots, a_1, a_0$, with $a_i \in \{0,1\}$, where the most significant digit is on the left end of the string. The decimal value of D is given by:

$$D = \sum_{i=0}^{N-1} a_i 2^i \tag{1}$$

If two strings (N bit wide) $a_{N-1}, \ldots, a_0$ and $b_{N-1}, \ldots, b_0$, representing two binary numbers are arranged on two aligned and superposed rows, the sum of these strings can be obtained by an iterative and parallel application of Table 1 (derived from



$\cdots$ $b_4\, a_4$ $b_3\, a_3$ $b_2\, a_2$ $b_1\, a_1$ $b_0\, a_0$
1
2
$\cdots$
$\cdots$
$\cdots$
N
$\cdots$ s_4 s_3 s_2 s_1 s_0

Figure 1: Addition in pipeline of two binary (or 2s complement) numbers. The nodes represent half-adder functions, the arrows indicate the input and output terminals. The values of the inputs and outputs of all other terminals and the data on thin lines are always zero (if no overflows occour). After N steps the sum is given.

the truth table of the popular half adder) [8]. In fact the input of Table 1 is a pair of digits a_i and b_i (having the same weight) and the output is also two digits: an upper digit with position i (sum) and a lower digit with position $i+1$ (carry). After a parallel application of the table on all digits of the input strings, Table 1 gives an upper resulting string which is the string of the *sums* and a lower one which is the string of the *carries*. N subsequent applications of Table 1 are sketched in Fig. 1. It is evident that after N applications of this table (if no overflows occour) the string of carries is always formed by N zeros[1] [8], while the string representing the sum is given. It is also evident (see Fig. 1) that the addition operation can be performed in pipeline.

The same procedure can be used to obtain the algebraic sum on 2s complement numbers. In Fig. 2 an example of addition of two binary integers 6 bit wide, obtained by means of 6 parallel applications of Table 1, is presented.

[1]The use of tables to perform operations (symbolic substitution) is widely employed in optical computing, see Ref. [9].

$(31)_{10}$		0	1	1	1	1	1	1st operand
$(1)_{10}$		0	0	0	0	0	1	2nd operand
$(30)_{10}$		0	1	1	1	1	0	
$(2)_{10}$	0	0	0	0	0	1	$\emptyset$	
$(28)_{10}$		0	1	1	1	0	0	
$(4)_{10}$	0	0	0	0	1	0	$\emptyset$	
$(24)_{10}$		0	1	1	0	0	0	
$(8)_{10}$	0	0	0	1	0	0	$\emptyset$	
$(16)_{10}$		0	1	0	0	0	0	
$(16)_{10}$	0	0	1	0	0	0	$\emptyset$	
$(0)_{10}$		0	0	0	0	0	0	
$(32)_{10}$	0	1	0	0	0	0	$\emptyset$	
$(32)_{10}$		1	0	0	0	0	0	
$(0)_{10}$	0	0	0	0	0	0	$\emptyset$	

Figure 2: Example of addition as a sequence of N parallel applications of Table 1. In this example $N = 6$.

3 Cellular Automata as pipelined binary adders

The addition procedure described can be implemented on a CA to realize pipelined additions of number pairs following the scheme shown in Fig. 1. The half adder functions can be obtained on the CA by distinguishing two phases: in the first phase the needed logical operations XOR and AND are performed, whereas in the second phase the shift operations, required by Table 1, are performed.

3.1 Cellular automaton local rule for pipelined addition

According to the CAM (Cellular Automaton Machine) terminology [2, 4, 5], the state binary components of cells will be denoted *Planes*. Six planes are required to perform the half-adder rule shown in Table 1. Two planes are used for the pairs of input operands and two planes for the current elaborating data. The use of each plane is:

Plane 0: the first row represents the current augend, the remaining rows are used to store the intermediate sums;

Plane 1: the first row represents the current addend, the remaining rows are used to store the carries;

Plane 2: is used to store the N input binary numbers (augends);

Plane 3: is used to store the N input binary numbers (addends);

Plane 4: is used to individuate the first cell row (current operand);

Plane 5: is used to distinguish between the computing phase (application of the logical operation XOR and AND) and the shift phase.

In the computing phase, each cell of Plane 0 takes the value of the XOR between center-0 and center-1; each cell of Plane 1 assumes the value of the logical AND between center-0 and center-1. To perform pipelined operations, a shift phase is necessary to prepare data on Plane 0 and Plane 1 for the subsequent application of the computing phase. In the shift phase the following copy operations are performed: a) on Plane 0 each cell copies the north cell (their content is shifted down), b) on Plane 1 each cell copies the north-east cell (their content is shifted down and left), c) on Plane 2 and Plane 3 each cell copies the south cell (their content is shifted up). In this phase the current operands are copied on the topmost row of Plane 0 and Plane 1 from the topmost row of Plane 2 and Plane 3 respectively. After a start-up time (proportional to the CA size) on the last row of Plane 0 a sum is ready at every two machine state transitions.

The starting configuration is defined in the following way. The first row of Plane 0 contains the augend and the first row of Plane 1 contains the addend (the remaining rows are all 0s). Plane 2 and Plane 3 contain the N successive pairs of augends and addends respectively. All cells of the topmost row of Plane 4 are set to 1 and the cells of the remaining rows are set to 0. All cells of Planes 5 are set to 0.

3.2 Implementation of the pipelined addition on the CAM-8 machine

The formal description of the cellular automaton local rule are given using the CAM-Forth language of the Cellular Automata Machine (CAM) designed by the Information Mechanics Group of the Massachusetts Institute of Technology [2, 4, 5]. The notation *neighbor-k*, where *neighbor* $\in$ {center, south, south-west, west, north-west, north, north-east, east, south-east}, denotes the corresponding neighbor bit of Plane k ($k = 0, \ldots, 5$). According to the use of cells in the described computing and shift phases, the total number of inputs for the CA rule[2] is 10.

For the sake of clarity, the normal order notation is still preserved instead of the Reverse Polish Notation one (adopted by the CAM-Forth) and also the simple construct: *begin ... end* is used.

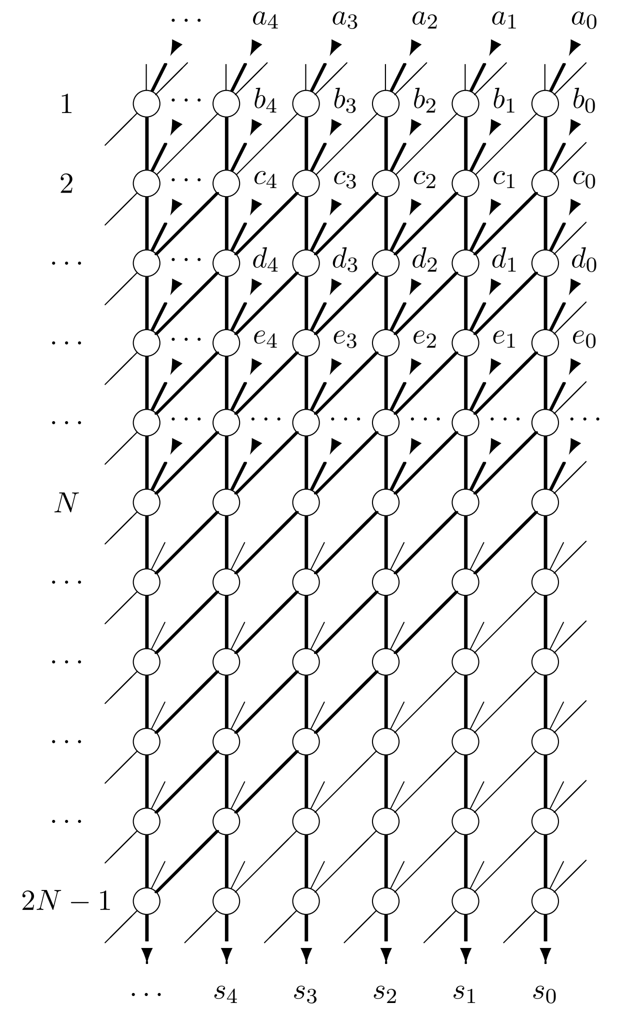
```
new-experiment
n/vonn       {* Neighborhood for planes 0 and 1}
& /centers       {* Neighborhood for the other planes}
pipelined-binary-sum:
begin
if center-5 then begin     {* Computing phase}
   center-0 xor center-1 > pln0;
   center-0 and center-1 > pln1;
   center-2 > pln2;
   center-3 > pln3;
   end;
   else begin     {* Shift phase}
   if center-4 then begin
      center-2 > pln0;
      center-3 > pln1;
      end;
      else begin
      north-0 > pln0;
      north-east-1 > pln1;
      end;
   south-2 > pln2;
   south-3 > pln3;
   end;
center-4 > pln4;
not center-5 > pln5;
end.
make-table pipelined-binary-sum
```

This program calculate the sums of N pairs of binary (or 2s complement) numbers N bit wide. After a start-up phase of $2N$ machine state transitions, results are given every two state transitions.

4 Accumulation by Cellular Automata

To perform the addition of N numbers (accumulation) the scheme presented in Fig. 3 can be used. As one can see, on each node of each row three quantities must be added: the sum and the carry computed in the previous row and one of the N operands. If three strings (N bit wide) $a_{N-1}, \ldots, a_0$, $b_{N-1}, \ldots, b_0$ and $c_{N-1}, \ldots, c_0$ representing three binary numbers are arranged on three aligned and superposed rows, the sum of these strings can be obtained by an iterative and parallel application of Table 2 (derived from the truth table of the full-adder). The input of Table 2 is a triple of digit a_i, b_i and c_i (upper, medium and lower digit having the same weight) and the output is two digits: an upper digit with position i (sum) and a lower digit with position $i+1$ (carry). This fact allows the insertion of one operand (the medium string of digits) before each application of the table. It is evident from Fig. 3 that after N insertions of operands

[2]Observe that in spite of what holds in the CAM-6, the only restriction in the CAM-8 for the structure and the dimension of the CA rule consists on the maximum number of input bits which is 16.



$\cdots$ a_4 a_3 a_2 a_1 a_0
1
2
$\cdots$
$\cdots$
$\cdots$
N
$\cdots$
$\cdots$
$\cdots$
$\cdots$
$2N-1$
b_4 b_3 b_2 b_1 b_0
c_4 c_3 c_2 c_1 c_0
d_4 d_3 d_2 d_1 d_0
e_4 e_3 e_2 e_1 e_0
$\cdots$ s_4 s_3 s_2 s_1 s_0

Figure 3: Accumulation of N binary (or 2s complement) numbers. The nodes represent full-adder functions, the arrows represent the data input and output. The values of the input and the output of all other terminals, and the data on thin lines are always zero (if no overflows occour). After $2N-1$ steps the sum of all operands is given.

u	0	1	0	1		
m	0	0	1	1		
l						
	0	1	1	0	u	sum
0	0	0	0	1	l	carry
	1	0	0	1	u	sum
1	0	1	1	1	l	carry

Table 2: Full-adder action. The table acts on a triple of digits u, m and l (upper, medium and lower respectively) and gives two digits u and l (upper and lower). In the output pair the lower digit is shifted left one position.

and further $N-1$ applications of Table 1 without new operand insertion (i.e. with the medium string formed by all zeros), and if no overflows occour, the string of carries is always formed by N zeros, while the string representing the sum of all numbers is given. In Fig. 4 an example of accumulation of 6 numbers each 6 bit wide, obtained by means of 11 parallel applications of Table 2, is presented. As one can see, in $2N-1$ steps the sum of N numbers is provided: N steps are used to load the N addends, and $N-1$ steps are required to propagate the carries to the most significant bit (see Fig. 3).

A CA is suitable to perform the described accumulation scheme of N binary (or 2s complement) numbers. Full-adder functions are obtained by means of two cascaded half-adders and one OR port whose actions spend three consecutive phases, as shown in Fig. 5.

4.1 Cellular automaton local rule for accumulation

Six planes are required to perform the accumulation procedure. One plane is used for storing the N addends, another for storing the carries, and two further planes to perform computations. Finally, two planes are required to distinguish four addition phases. Three phases are reserved to individuate the full-adder functions, while the fourth one individuates the shift phase which acts on two different planes: by moving up all the rows but the first one of the plane of addends and by shifting left the string of carries. The use of each plane is:

Plane 0: the first and second rows represent the first and second inputs of full-adders, after the action in the first row the output sums are stored;

Plane 1: the first row represents the third input of the full-adders, after the action it is used to store the carries;

Plane 2: is used to distinguish the first row (the top row) from the remaining ones;

Plane 3: is used to distinguish the second row from the remaining ones;

Plane 4 and Plane 5: are used to distinguish the needed four phases.

After the first half-adder phase ($ha1$-phase in Fig. 5) the result of the XOR between the first and the second rows of Plane 0 is stored in the first row of Plane 0; the second row of Plane 2 takes the value of the AND between the first and the second rows of

		0	0	0	0	0	0
$(5)_{10}$ $\rightarrow$		0	0	0	1	0	1
		0	0	0	0	0	0
		0	0	0	1	0	1
$(11)_{10}$ $\rightarrow$		0	0	1	0	1	1
	0	0	0	0	0	0	$\emptyset$
		0	0	1	1	1	0
$(5)_{10}$ $\rightarrow$		0	0	0	1	0	1
	0	0	0	0	0	1	$\emptyset$
		0	0	1	0	0	1
$(2)_{10}$ $\rightarrow$		0	0	0	0	1	0
	0	0	0	1	1	0	$\emptyset$
		0	0	0	1	1	1
$(8)_{10}$ $\rightarrow$		0	0	1	0	0	0
	0	0	1	0	0	0	$\emptyset$
		0	1	1	1	1	1
$(1)_{10}$ $\rightarrow$		0	0	0	0	0	1
	0	0	0	0	0	0	$\emptyset$
		0	1	1	1	1	0
		0	0	0	0	0	0
	0	0	0	0	0	1	$\emptyset$
		0	1	1	1	0	0
		0	0	0	0	0	0
	0	0	0	0	1	0	$\emptyset$
		0	1	1	0	0	0
		0	0	0	0	0	0
	0	0	0	1	0	0	$\emptyset$
		0	1	0	0	0	0
		0	0	0	0	0	0
	0	0	1	0	0	0	$\emptyset$
		0	0	0	0	0	0
		0	0	0	0	0	0
	0	1	0	0	0	0	$\emptyset$
$(32)_{10}$ $\rightarrow$		1	0	0	0	0	0
		0	0	0	0	0	0
	0	0	0	0	0	0	$\emptyset$

Figure 4: Example of accumulation as a sequence of $2N-1$ parallel applications of Table 2. N steps are used to load the N addends and $N-1$ steps are required to propagate the carry. In this example $N = 6$.

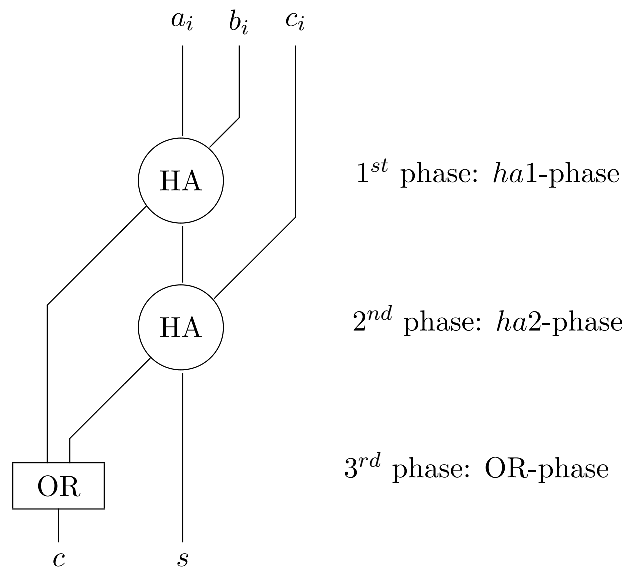


Figure 5: Scheme of a full-adder obtained by means of two half-adder (HA) and one OR port.

Plane 0. During the second half-adder phase ($ha2$-phase in Fig. 5) in the first row of Plane 0 the result of the XOR between the first row of Plane 0 and the first row of Plane 1 is put; the first row of Plane 1 takes the value of the AND between the first row of Plane 0 and the first row of Plane 1. Finally, during the OR-phase, the first row of Plane 1 takes the value of the OR between the second row of Plane 0 and the first row of Plane 1. During the shift phase all the rows but the first of Plane 0 are shifted up (each cell copies the south cell) and a new addend is copied on the second row of Plane 0. Further, the first row of Plane 1 (string of carries) is shifted to the left.

The current state of the generic cell of Plane 0 depends on center-0, south-0, north-0 and center-1. The current state of the generic cell of Plane 1 depends on center-1, east-1, center-0 and south-0. For the remaining four Planes, the neighborhood of each cell consists only of the cell itself.

The starting configuration is defined in the following way. Plane 0 contains the N addends. All cells of Plane 1 are set to 0. All cells of the topmost row of Plane 2 are set to 1 and the cells of the remaining rows are set to 0. All cells of the second row of Plane 3 are set to 1 and the cells of the remaining rows are set to 0. All cells of Plane 4 and Plane 5 are set to 0.

4.2 Implementation of the accumulation on the CAM-8 machine

As in the previous case, the new available version of Cellular Automata Machine, the CAM-8, is used for the implementation. The total number of bits used by the accumulation procedure is 9.

```
new-experiment
accumulation:
begin
center-2 > pln2;
center-3 > pln3;
if not center-4 then
begin     {* Computing phases 1 and 2}
if not center-2 and not center-3 then begin
   center-0 > pln0;
   center-1 > pln1;
   end;
   else begin
   if not center-5 then begin
      if center-2 then center-0
           xor south-0 > pln0;
      if center-3 then center-0
           and north-0 > pln0;
      end;
      else begin
      if center-2 then center-0
           xor south-0 > pln0;
      if center-3 then center-0
           and center-1 > pln1;
      end;
   end;
else
begin     {* Computing phase 3 and shift phase}
if not center-5 then begin
   center-0 > pln0;
   center-0 or center-1 > pln1;
   end;
   else
   if center-2 then begin
      center-0 > pln0;
      east-0 > pln1;
      end;
      else begin
      south-0 > pln0;
      center-1 > pln1;
      end;
not center-5 > Pln5;     {* 4-phase clock}
if center-5 then center-4 > Pln4;
            else not center-4 > Pln4;
end;
end.
make-table accumulation
```

This program calculates the accumulation of N binary (or 2s complement) numbers N bit wide in $2N-1$ steps ($8N-4$ machine state transitions).

5 Multiplication of two N/2 bit wide binary numbers

The accumulation procedure shown in the previous section can be easily adapted in order to perform the multiplication between two binary (or 2s complement), $N/2$ bit wide numbers in $O(N)$ time.

The result of a multiplication of two numbers is given by the sum of all partial products (see Fig. 6a). For this reason, the multiplication procedure is split into two parts: a) generation of partial products, b) accumulation of all partial products. The whole operation is sketched in Fig. 6.

5.1 Cellular automaton local rule for multiplication

Seven planes are required to perform the multiplication procedure. Six planes are used in the same way as in the accumulation procedure, and a further plane is required to store the multiplier. The use of each plane is:

Plane 0 to Plane 5: the same use as in the accumulation procedure;

Plane 6: is used to store the multiplier.

5.1.1 Generation of partial products

The multiplicand and the multiplier are stored on the top row of Plane 0 and on the most left column of Plane 6 respectively. To obtain on Plane 0 all partial products, the following procedure is used: the first row of Plane 0 is copied on all rows of the plane; the most left column of Plane 6 is copied on all columns of the plane. After, by means of an AND operation between all cells of Plane 0 and all cells of Plane 6, a certain number of cells are zeroed according to the multiplier value. At this point all partial products are generated, but their digits have not the correct weights yet.

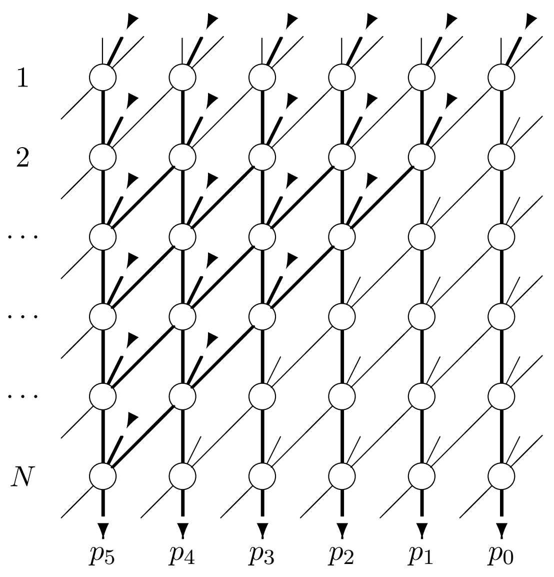
This procedure gives the needed partial products and costs $O(N)$ time. In fact the two copy operations can be performed in parallel and require N steps, and the AND is performed in $O(1)$ time (the total time is $N+1$ machine state transitions). Finally, all the rows but the first of Plane 0 are shifted to the left; this left shift operation is performed in constant time (one state transition each).

5.1.2 Sum of all partial products

The accumulation procedure is here applied with just a little change: to obtain that all the digits of all partial products have the correct weight, and give the correct contribution to the sum, in the shift

	sign extensions			operands		
	a_5	a_4	a_3	a_2	a_1	a_0
	b_5	b_4	b_3	b_2	b_1	b_0
	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
...	a_4b_1	a_3b_1	a_2b_1	a_1b_1	a_0b_1	
...	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
...	a_2b_3	a_1b_3	a_0b_3			
...	a_1b_4	a_0b_4				
...	a_0b_5					
	p_5	p_4	p_3	p_2	p_1	p_0

a)



b)

Figure 6: Accumulation of partial products to perform the multiplication operation: a) partial products used in the procedure, b) accumulation of partial products, the nodes represent full-adder functions. The data inputs, marked by arrows, are loaded with the partial products in the same order as presented in a). In this case the accumulation procedure requires N steps.

phase on Plane 0, each cell of all the rows but the first copies the south-east cell, instead of south.

In the multiplication operation, if a product N bit wide is expected, the maximum length of input operands must be $N/2$, for this reason, to avoid result truncation the two strings representing the operands must be $N/2$ bit wide (right giustified). In the case of 2s complement numbers the most significant part of the strings must contain the sign extension of operands (see Fig. 6a).

The starting configuration of planes from 1 to 5 is the same as in the accumulation procedure and planes have the same function. A further plane (Plane 6) is needed to store the multiplier. Plane 6 is also used to store the partial products.

The formal implementation of this multiplication procedure can be immediately derived from that shown for the accumulation procedure and from the above description.

6 Conclusions

A cellular automaton performing pipelined parallel addition (or algebraic sum), accumulation and multiplication on binary or 2s complement numbers has been presented. The procedure which performs binary addition on cellular automata, presented in [6], is enhanced by using the pipeline technique. Both procedures use the conventional binary number representation, but Sheth *et al.* one requires $O(N)$ time to provide one result, whereas with the pipelined addition, after the start-up time, results are obtained in constant time (at every two machine state transitions) which is the same time complexity obtained in the work of Clementi *et al.* [7] for the addition of two number in redundant binary representation.

Then, a procedure to perform the accumulation of N binary or 2s complement numbers of size N, which runs in $O(N)$ time, has been presented. Finally, this accumulation operation can be easily adapted to perform the multiplication between two numbers of size $N/2$ in $O(N)$ time.

All procedures presented are implemented on the CAM-8 machine, showing that efficient arithmetical operations can be easily and efficiently implemented on cellular automata.

References

[1] G. Bilardi and F. Preparata, “Horizons of Parallel Computation”, *Proceedings of Symposium of* 25th *Anniversary of* INRIA, Springer-Verlag LNCS, **653** (1992) 155.

[2] A. Califano, T. Toffoli and N. Margolus, *CAM-6 User's Guide, versions 2.1*, (1986).

[3] G. Jacopini and G. Sontacchi, "Reversible Parallel Computation: an Evolving Space Model", *Theoretical Computer Science*, **73** (1990) 1.

[4] T. Toffoli and N. Margolus, *Cellular Automata Machines - A New Environment for Modeling*, (Cambridge, MIT Press, 1987).

[5] N. Margolus and T.Toffoli, *STEP: a Space Time Event Processor - Architecture reference*, (Cambridge, MIT Press).

[6] B. Sheth, P. Nag and R. W. Hellwarth, "Binary Addition on Cellular Automata", *Complex Systems*, **5** (1991) 479.

[7] A. Clementi, G. A. De Biase and A. Massini, "Fast Parallel Arithmetic on Cellular Automata", *Complex Systems*, **8** (1994) 435.

[8] G. A. De Biase and A. Massini, "Redundant Binary Number Representation for an Inherently Parallel Arithmetic on Optical Computers," *Applied Optics*, **32** (1993) 659.

[9] K.-H. Brenner, A. Huang and N. Streibl, "Digital Optical Computing with Symbolic Substitution", *Applied Optics*, **25** (1986) 3054–3060.