Fast Parallel Algorithm for Ternary Multiplication Using Multivalued I^2L Technology

Mallika De and Bhabani P. Sinha

Abstract—An algorithm for parallel multiplication of two n-bit ternary numbers is presented in this brief contribution. This algorithm uses the technique of column compression and computes the product in $(2\lceil \log_2 n \rceil + 2)$ units of addition time of a single-bit ternary full adder. This algorithm requires regular interconnection between any two types of cells and hence is very suitable for VLSI implementation. The same algorithm is also applicable to the multiplication of negative numbers.

Index Terms—Balanced ternary logic, column compression, precarry addition, systolic architecture, ternary multiplication.

I. Introduction

During recent years, design of fast multipliers has become an area of growing research interest to the system designers. To improve the speed of computations involving many multiplications, parallel algorithms for multiplication have become increasingly popular. The execution time for parallel multiplication has been decreased substantially by using the technique of partial product matrix reduction or column compression [1]–[4]. Two algorithms for iterative array multiplication have been reported [5], both of which require only n units of time for multiplication of two n-bit numbers and involve almost regular interconnection structures of the multiplier array cell elements which are ideal for VLSI implementation. Authors in [6] have also reported an $O(\log n)$ multiplication scheme using redundant binary trees. Authors in [7] have proposed two parallel algorithms for multiplication using the technique of column compression which require $(n + \lceil \log_2 n \rceil)$ and approximately $3\lceil \log_2 n \rceil$ units of time, respectively, and involve almost regular interconnection between only two types of cells.

Multiple valued logic, in which the number of discrete logic levels are not confined to two, has been the subject of much research over many years. It has been shown mathematically that if the cost or complexity C of the system hardware is proportional to the digit complexity, the radix three would be more economical than radix two [8], [9]. Frieder and Luk [10] used binary coded form for balanced and ordinary ternary operation. The ternary design is easily produced if one selects as supporting technology a ternary version of the I^2L circuits described in [11]–[15]. Also there exist in the literature an ECL version [16] and MOS version [17] of ternary logic.

Recently, new, reliable, and low-cost multistate memory devices have been implemented [18], [19] in a single physical element, called the resonant tunneling transistor (RTT), providing the possibility of low-cost implementation of multivalued logic circuits. It appears that in the near future it will be possible to implement different reliable low-cost circuits such as counters, adders, multipliers, dividers, etc., in ternary logic using these RTT's.

In this brief contribution, we propose a parallel algorithm for a multiplier based on the balanced ternary system and implemented by means of I^2L current-mode threshold logic gates. I^2L logic family facilitates the design of LSI chip because it offers a high packing density (a factor of 10 over the popular TTL), a low speed-power product (a factor of 100 over TTL), and a wide range of operating power supply and current levels. This performance is obtained at a delay per gate on the order of 10 ns [20]. The speed of I^2L is not, in general, as great as that of TTL. But a delay of the order of 5 ns has been achieved using dielectric isolation [21]–[23].

In the balanced ternary system each digit of representation has three possible values $(-1, 0, -1)$. The parallel algorithm for multiplication uses the technique of column compression to increase the speed of execution. For multiplication of two n-bit ternary numbers the algorithm requires $(2\lceil \log_2 n \rceil + 2)$ units of time with $O(n^2(\log n)^2)$ AT^2-value. A systolic architecture using a pyramidal interconnection of elementary processors (full adders) to multiply two 64-bit ternary numbers using the algorithm has been shown as an implementation example. It requires regular interconnection between only two types of cells: four-input single-bit full adder and precarry generators, and hence, it is very suitable for single-chip VLSI implementation. Since in the balanced ternary system, any negative number can be represented only by changing the sign of every bit in its positive representation, the same algorithm, without any change, also applies to the multiplication of negative numbers.

The brief contribution is organized as follows. In Section II, we describe the preliminaries of ternary multiplication and the method of implementing balanced ternary full adder by means of threshold I^2L logic. In Section III, we describe the design of a new ternary precarry adder (to add two ternary numbers) which will be used by our algorithm. In Section IV, we describe the parallel algorithm for multiplication, and in Section V an implementation example is given.

II. Preliminaries

Let U and V be the two numbers to be multiplied whose balanced ternary representations are as follows:

$$U = u_{n-1}u_{n-2}\cdots u_1u_0$$
$$V = v_{n-1}v_{n-2}\cdots v_1v_0,$$

where u_0 and v_0 are the least significant bits and u_i's, v_i's $(i = 0, 1, 2, \cdots, n-1)$ can take the values -1, 0, or 1. (To avoid any confusion, we note that this number representation is different from the base 2 representation of [6] using -1, 0, 1 digit set.) Let $W = UV$ where W can be expressed as $W = w_{2n-1}w_{2n-2}\cdots w_0$.

Hence, we can write

$$W = \sum_{i=0}^{n-1} 3^i \sum_{j=0}^{i} u_{i-j}v_j + \sum_{i=n}^{2n-2} 3^i \sum_{j=i-n+1}^{n-1} u_{i-j}v_j \qquad (1)$$

or

$$W = \sum_{i=0}^{n-1} 3^i \sum_{j=0}^{i} a_{i-j,j} + \sum_{i=n}^{2n-2} 3^i \sum_{j=i-n+1}^{n-1} a_{i-j,j}, \qquad (2)$$

where $a_{i,j} = u_iv_j$. Any w_i can be obtained by computing the appropriate summation and adding carry bits that are propagated from the less significant bit positions.

We have implemented a modified Wallace type multiplier that uses full adders in carry-save configuration for the parallel steps of column compression leaving finally only two ternary numbers which should be added by a carry propagate adder. Since the two bits in the balanced ternary system can hold a number in the range $(-4$ to $+4)$, we will use here four-input adders for the column compression

Manuscript received May 7, 1991; revised April 9, 1992.

M. De is with the University Science Instrumentation Centre, University of Kalyani, Kalyani 741 235, India.

B.P. Sinha is with the Electronics Unit, Indian Statistical Institute, Calcutta 700 035, India.

IEEE Log Number 9212763.

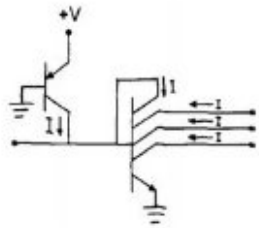

Fig. 1. I^2L current mirror.

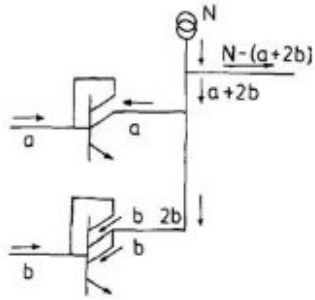

Fig. 2. Weighted sum.

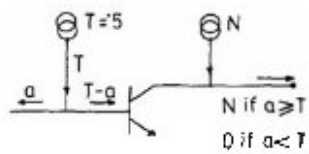

Fig. 3. Threshold detector.

instead of three-input ones as in the binary case. This will eventually decrease the number of parallel steps in column compression. In Section III, we describe a technique for adding the two numbers left after the column compression process by means of a 3-input adder in $\lceil \log_2 n \rceil + 2$ steps.

I^2L *Implementation of Ternary Full Adder*

We have used three basic threshold I^2L logic modules for our different circuit implementations [13], These are 1) input replication: create replicas of an input through current-mirror imaging (Fig. 1); 2) weighted sum: forms the arithmetic sum of several weighted replicas (Fig. 2); and 3) threshold detection: determines if the sum exceeds a predetermined threshold (Fig. 3). For implementation of our proposed multiplication algorithm with the help of above modules, we will associate each logic value with some definite value of input current. For physical realization, the logic values $(-1, 0, 1)$ are mapped onto the values of currents according to

$$-1 \rightarrow L; \quad 0 \rightarrow M; \quad 1 \rightarrow H, \tag{3}$$

where L, M, and H stand for low, medium, and high currents, respectively. As a result, bit-by-bit multiplication (such as a_{i-j}, j) will be mapped onto the current values as shown in the multiplication Table I. An implementation of this table using threshold I^2L logic has been given in [24]. A circuit diagram of four-input ternary adder using the above three threshold logic modules is given in Fig. 4 [24].

III. Precarry Adder

The concept of using precarry adders [26], [27] for generating the carry vector in $O(\log n)$ time in a carry propagate binary adder was as follows: If $a_{i-1} = b_{i-1} = 0$ (binary value) then $c_i = 0$. (c_i is the carry-input to the ith bit position.) If $a_{i-1} = b_{i-1} = 1$ then $c_i = 1$. But if $a_{i-1} = \bar{b}_{i-1}$ then c_i depends on bits previous to $(i-1)$th position. To resolve this situation, a 3-state precarry bit p_i at the ith bit position was introduced. ($p_i = 0$ for $a_{i-1} = b_{i-1} = 0$, $p_i = 1$ for $a_{i-1} = b_{i-1} = 1$ and $p_i = 2$ otherwise.) Using this concept of precarry, two numbers can be added in $\lceil \log_2 n \rceil + 1$ units of time.

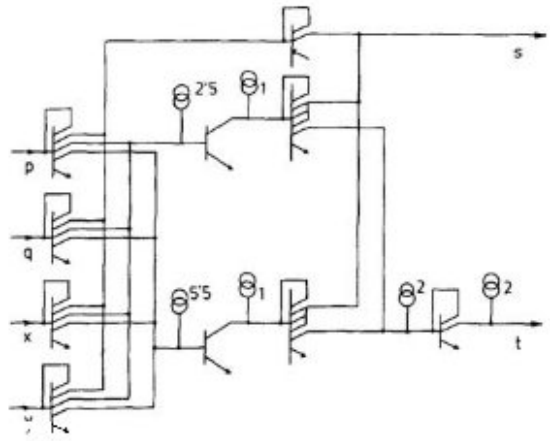

Fig. 4. Mapped balanced ternary four-sum and adder.

TABLE I
Balanced Ternary Product $x.y$ Mapped into Current Values L, M, and H

y\x	−1	0	1
−1	H	M	L
0	M	M	M
1	L	M	H

We extend the above idea to the case of ternary logic. Consider that two ternary numbers $A = a_{n-1} \cdots a_0$ and $B = b_{n-1} \cdots b_0$ are to be added to produce the sum $S = s_n s_{n-1} \cdots s_0$. We need here a 7-state precarry bit p_i for the ith bit position to resolve the carry-input c_i. There are three cases for which we can tell surely what the carries c_i's will be. These are a) ***sure "−1" carry:*** when (digit-sum)$_{-1} = -2$ and (digit-sum)$_{-2}$ is other than 2 or 1, where (digit-sum)$_i = a_i + b_i$ for $i > 0$ and $a_{-1} = b_{-1} = 0$, b) ***sure "0" carry:*** when (digit-sum)$_{-1} = 0$ for any value of (digit-sum)$_{-2}$ and c) ***sure "1" carry:*** when (digit-sum)$_{-1} = 2$ and (digit-sum)$_{-2}$ is other than -2 or -1. Other combinations do not produce any ***sure carry*** but depends on the digit-sum of the previous positions. We resolve the ambiguity of these cases by defining a precarry vector $P = p_n p_{n-1} \cdots p_1$ and an associated fundamental carry operation (fco). We define p_i's $(i > 0)$, depending on the digit-sum of the $(i-1)$th and $(i-2)$th position as shown in Table II.

Define a binary operator fco on x, y according to the truth Table III where x, y, and x fco y each can assume values 0, 1, 2, 3, 4, 5, and 6. Effect of carry propagation from the lower order bits on the value of precarry bit p_i is actually reflected through fco. The operator fco is associative but not commutative.

Now the carry vector C can be generated from the precarry vectors in two stages. The first stage is the repetitive application of fco operation on precany vectors to get $P' = p'_n p'_{n-1} \cdots p'_2 p'_1$, where

$$p'_i = p_i \text{ fco } (p_{i-1} \text{ fco } (p_{i-2} \text{ fco } (\cdots (p_2 \text{ fco } p_1) \cdots))), \qquad 1 < i \leq n$$

and the second stage is a mapping of p'_i's to c_i according to

$$0 \rightarrow -1; \quad 1 \rightarrow -1; \quad 2 \rightarrow 0; \quad 3 \rightarrow 0;$$
$$4 \rightarrow 0; \quad 5 \rightarrow 1; \quad 6 \rightarrow 1. \tag{4}$$

TABLE II
TRUTH TABLE FOR GENERATING PRECARRY DIGIT p_i

p_i	Digit-Sum Value at $(i-1)$th Position	Digit-Sum Value at $(i-2)$th Position
0	−2	−2, −1 or 0
1	−2	1
2	−2	2
2	−1	−2, −1, 0, 1 or 2
3	0	−2, −1, 0, 1 or 2
4	1	−2, −1, 0, 1 or 2
4	2	−2
5	2	−1
6	2	0, 1 or 2

TABLE III
TRUTH TABLE FOR PRECARRY OPERATION fco

v\x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	**0**	**0**	**0**	0	1	**2**	3
2	0	1	2	3	3	3	3
3	3	3	3	3	3	3	3
4	3	3	3	3	4	5	**6**
5	3	4	5	6	6	6	**6**
6	**6**	6	6	6	6	6	6

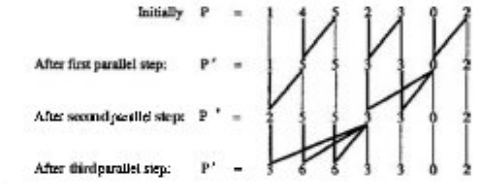

Fig. 5.

This mapping follows immediately from the digit-sum values at $(i-1)$th and $(i-2)$th bit positions corresponding to a definite value of the precarry bit p_i.

We give an illustrative example as follows.

Example: We take two 7-bit ternary numbers A and B, where

$$A = -1 \quad 1 \quad 1 \quad -1 \quad 0 \quad -1 \quad 0$$
$$B = -1 \quad 0 \quad 1 \quad 0 \quad 0 \quad -1 \quad -1.$$

Hence,

$$P = 1 \quad 4 \quad 5 \quad 2 \quad 3 \quad 0 \quad 2.$$

Since fco is an associative operation, the values of precarry bits after successive fco operations can be computed in three parallel steps in Fig. 5 (where the thicker lines represent the data flow for fco operations and the thinner lines stand for unchanged values of p_i at every parallel step).

Finally, the mapping from P' to C is done according to (4) to get

$$C = 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad -1 \quad 0.$$

Now, if we have a sufficient number of hardware modules (precarry generators) that execute the fco operation on two given operands, then the vector P' can be generated from the precarry vector in $\lceil \log_2 n \rceil$ units of time, where in each step the fco operators are executed in parallel. The number $N(n)$ of such modules required to compute the vector P' for $n = 2^k$ is given by $N(n) = k.2^{k-1}$ [7]. An example of interconnecting all these modules to generate the vector P' was also shown in [7]. A circuit can be designed using threshold I^2L technology to implement the precarry logic modules, based on the successive extraction of step functions proposed by Davio and Deschamps [25]. We require two more parallel steps, one for getting C from P' (Fig. 6) and the other for adding C to A and B, resulting in $\lceil \log_2 n \rceil + 2$ time units for adding two numbers.

IV. PARALLEL MULTIPLICATION ALGORITHM

After generating the partial product bits $a'_{i-j,j} = w'_{i-j} v'_j$'s, we are to add n n-bit ternary partial products (with appropriate shifts) to get the final product $W' = U'V'$. To accomplish this, we would use here the column compression technique to reduce the number of bits at each bit position successively by using ternary (4, 2) counters [2]. Ultimately, by repeated application of this column compression, we will be left with two vectors--one sum vector S and a carry vector C to be added to generate the required product. To describe the algorithm formally we need the following notations and explanations. Let x be a positive integer, $x \geq 1$ and x' be another integer defined as, $x' = 4\lceil x/4 \rceil$.

Let T be a function defined on a set of x bits $B = \{b_1, b_2, \ldots, b_x\}$ in the following way where we assume $b_{x+1} = b_{x+2} = b_{x+3} = 0$,

$$T: \{b_1, b_2, \ldots, b_x\} \rightarrow \{S(b_j, b_k, b_l, b_m), C(b_j, b_k, b_l, b_m), Y(B - \{b_j, b_k, b_l, b_m\}) \mid 1 \leq j, k, l, m \leq x'; j, k, l, m \text{ being all different,}\}$$

where $S(b_j, b_k, b_l, b_s)$ and $C(b_j, b_k, b_l, b_s)$ are, respectively, the sum and carry bits in a single bit ternary (four-input) full adder with inputs b_j, b_k, b_l, b_s. The function T operates on x number of bits to generate $\lceil 2x/4 \rceil$, i.e., $\lceil x/2 \rceil$ bits as output. Let m be an integer defined as

$$m = \begin{cases} i & \text{if } i < u \\ n-1 & \text{if } i \geq u \end{cases}$$

and x_i be another integer defined as

$$x_i = \begin{cases} i+1 & \text{if } i < n \\ 2n-i-1 & \text{if } i \geq n. \end{cases}$$

It follows from (2) that w_i can be obtained by adding x_i bits $a'_{i,i-m}, a'_{m-1,i-m+1}, \ldots, a'_{i-m,m}$ together with the final carry bit propagated from the $(i-1)$th bit position. This addition at each bit position i to generate the final sum bit has to be done by repeated application of the T operation; after each T operation, the generated sum bits are to be retained in this bit position, the generated carry bits are to be transmitted to the next higher bit position and the carry bits generated at the preceding bit position are to be collected at this position to be ready for the next T operation. The main idea of our proposed algorithm is as follows.

For each i, the T operation is done on all bits to be added to get w_i, and the generated carry bits $C(b_j, b_k, b_l, b_s)$ are propagated to the next higher order bit position $(i+1)$. However, if $b_k = b_l = b_s = 0$, $C(b_j, 0, 0, 0) = 0$ and this carry bit is not propagated. At the subsequent step, the T operation is performed, at each bit position on the sum bits $S(b_j, b_k, b_l, b_s)$ generated at this bit position and the propagated carry bits $C(b_j, b_k, b_l, b_s)$ from the lower order bit position. This process continues until we get only two bits, one sum bit and one carry bit, at each bit position. All these operations are done in parallel for all i. At this point, the resulting sum vector $S_{2n-2}S_{2n-3}\cdots S_1S_0$ and the carry vector $C_{2n-1}C_{2n-2}\cdots C_1C_0$ are added together by using a 2n−bit precarry adder as before. The algorithm is given in Fig. 7. The function f(n) used in the algorithm is defined as $f(n) = \lceil n/2 \rceil$. If t is the minimum integer such that $f^t(n) = 2$, then after executing T operation t times we get one sum bit and one carry bit. This explains t times execution of loop 2.

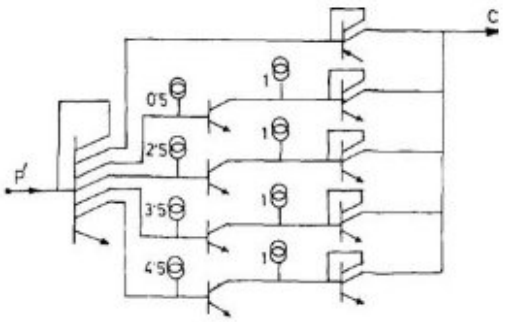

Fig. 6. Circuit for generation of carry C' from P'

```
        begin
        w'_{2n-1} ← 0;
loop1:  for i=1 to (2n-2) do in parallel
          begin
          (b_{i1}, b_{i2}, ..., b_{in}) ← (a'_{n-1,i}, a'_{n-2,i}, ..., a'_{0,i});
          Set y_i ← ⌈ x_i/4 ⌉
          Set t to the smallest integer such that f^t(n) = 2;
loop2:    for k=1 to t do
            begin
            (S_{k1}, ..., S_{ki}, C_{k1}, ..., C_{ki}) ← T(b_{i1}, b_{i2}, ..., b_{in_i});
            (b_{i1}, b_{i2}, ..., b_{in_{i+1}}) ← (S_{k1}, S_{k2}, ..., S_{ki}, C_{k-1,1}, C_{k-1,2}, ..., C_{k-1,y_{i-1}});
            w'_{2i-1} ← w'_{2i-1} + C_{k,2i-1};
            x_i ← y_i + y_{i-1}; y_i ← ⌈ x_i/4 ⌉
            end
          end;
          w'_0 ← a'_0;
loop3:    for i=1 to 2n-2 do in parallel
          p'_{i+1} ← SP( (b_{i1}, b_{i-1,1}, ..., b_{i1}) + (b_{i2}, b_{i-1,2}, ..., b_{i2}) );
          for i=1 to 2n-1 do in parallel
          c'_i ← p'_i
          for i=1 to 2n-2 do in parallel
          w'_i ← (b_{i1}, b_{i2}, c'_i);
          w'_{2n-1} ← w'_{2n-1} + c'_{2n-1}
        end;
```

Fig. 7. Parallel algorithm.

Hence, this integer $t = \lceil \log_{\frac{3}{2}} n \rceil - 1$, for any given n will determine the complexity of the algorithm.

Theorem 1: Time complexity $T(n)$ of our proposed algorithm is

$$T(n) = 2\lceil \log_2 n \rceil + 2.$$

Proof: Loop 2 of the algorithm requires $t = \lceil \log_{\frac{3}{2}} n \rceil - 1$ steps. Loop 3 requires $\lceil \log_2 (2n) \rceil$ steps. Mapping of P' to C requires one step and the final addition to produce w_i''s requires one more step. □

V. An Implementation Example

We now discuss the processor architecture and the associated data flow among the processors implementing a multiplier for 64x64 radix 3 (balanced ternary) digits using our proposed algorithm. To generate the product bit w_i, the partial product bits $a_{j,i-j}$, for $j = 0, 1, \ldots, i$ are added up along with carry bits from $(i-1)$th bit position $(i \geq 1)$ by means of four-input ternary full adders. These full adders for any particular bit position are connected in a way which we call as extended pyramidal structure (EPS) as explained below.

The structure of a pyramidal interconnection [28] with 21 processors is shown in Fig. 8, where the dots represent the processors and the arcs represent the interconnections. In this structure there will not be any interconnection between the processors along the dashed lines. Only upward connection from the processors at level 0 to the processors at level 1 and from processors at level 1 to the processors at level 2 of the structure is required, so that four bits from the preceding level are added in each processor at the succeeding level of the pyramidal structure. Since we have to propagate the carries generated at level 1 and onwards, from one pyramidal structure to

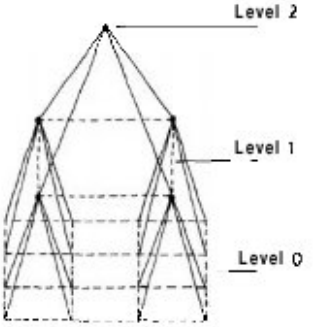

Fig. 8. Pyramidal interconnection with 21 processors

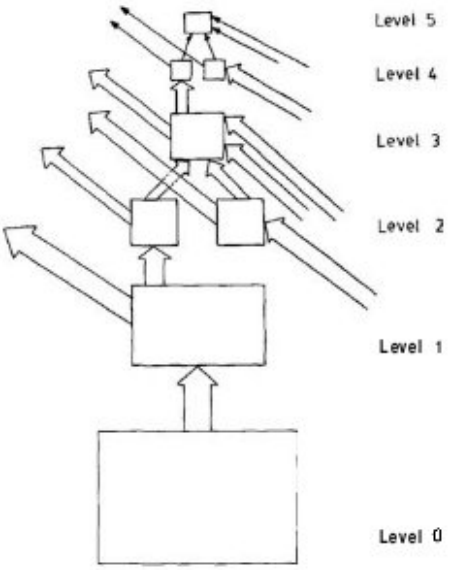

Fig. 9. Processor interconnection and data flow diagram.

the next one (for the next bit position), some extension to the above pyramidal interconnection is necessary. Interconnections between the processors of one level of an extended pyramidal structure (EPS) to the processors of next higher level of the next EPS (corresponding to next higher bit position) is necessary. The sum bits (source bits in case of level 0) are added in higher level of the same EPS, but the carry bits are added in higher level of the next EPS. So, as soon as the carry bits are generated, they are propagated to the next higher level of the next EPS, ready to be added in the next T operation. The EPS for adding 64 bits in a bit position is shown in Fig. 9, where we see that at level 2, two 2 x 2 planes are required instead of one; one plane for adding the sum bits generated at level 1 of the same pyramid and one plane for adding carry bits generated at level 1 of the preceding pyramid. By the same argument, the number of processors required for levels 3, 4, 5 will be, respectively, 4, 2, 1. Note that, at the base of the pyramidal structure (i.e., at level 0), the bits which are to be added are stored initially. So, no adder is required for that plane. The total number of processors required for the 63rd bit-position=16+8+4+2+1 = 31. Let $q = 2^{k_1-1} + 2^{k_2-1} + \ldots + 2^{k_r-1}$.

Then the total number of processors required to add q bits in the $(q-1)$th bit position is given by

$$\lceil q/4 \rceil + \lceil q/8 \rceil + \cdots + 1 = \lfloor q/2 \rfloor - \tau(q) + (k_1 - k_r + \epsilon),$$

where $\epsilon = 0$ for $q = 2^p (p > 0)$ and 1 otherwise, $\tau(q) = r$ = the number of 1's in the binary representation of q.

Assuming that $n = 2^k$, the total number of processors N_p, required to add the bits in all 2n bit positions is given by

$$N_p = 2\sum_{q=1}^{n}[\lfloor q/2 \rfloor - \tau(q) + (k_1 - k_r + \epsilon)] - \lfloor n/2 \rfloor + 1,$$

which can be simplified to $N_p = n^2/2 + n \log_2 n - 9n/2 + 5$.

The AT^2 value of such a realization comes out to be $(n^2/2 + n \log_2 n - 9n/2 + 5)(\log_2 n - 1)^2$, where $n = \log_2 N$, whereas for binary case [7] it is $(n^2/2 + n/2)(\lceil 1.71 \log_2 n \rceil - 1)^2$, where $n = \log_2 N$.

Since the implementation of the algorithm involves regular interconnections between only two types of cells, the algorithm is suitable for single-chip VLSI implementation.

VI. Conclusion

We have proposed a parallel algorithm for multiplication of two n-bit ternary numbers, which requires $\lfloor \log_2 n \rfloor + 2$ units of time and is implementable on an SIMD architecture. The AT^2 measure of the implementation is $O(n^2(\log n)^2)$ where A is measured in terms of the numbers of single-bit full adder units and precarry modules. Since the algorithm involves regular interconnections between only two types of cells, this is very suitable for single chip VLSI implementation. An advantage of using balanced ternary system is that the negative numbers can be obtained only by changing the sign of each digits in its positive number representation and hence the same algorithm will be applicable to the multiplication of two negative numbers.

References

[1] C.S. Wallace, "A suggestion for fast multiplier," ***IEEE Trans. Electron. Comput.***, vol. EC-13, pp. 114–117, Feb. 1964.

[2] L. Dadda, "On parallel digital multipliers," ***Alta Frequency***, vol. 45, pp. 574–580, 1976.

[3] W.J. Stenzel, W.T. Kubitz, and G.H. Garcia, "A compact high speed parallel multiplication scheme," ***IEEE Trans. Comput.***, vol. C-26, pp. 948–957, Aug. 1986.

[4] M.R. Santoro and M.A. Horowitz, "SPIM: A pipelined 64x64-bit iterative multiplier," ***IEEE J. Solid State Circuits***, vol. 24, pp. 487–493, Apr. 1989.

[5] S. Nakamura, "Algorithm for iterative array multiplication," ***IEEE Trans. Comput.***, vol. C-35, pp. 713–719, Aug. 1986.

[6] N. Tagaki, H. Yassura, and S. Yajima, "High speed VLSI multiplication algorithm with a redundant binary addition tree," ***IEEE Trans. Comput.***, vol. C-34 pp. 789–796, Sept. 1985.

[7] B.P. Sinha and P.K. Srimani, "Fast parallel algorithms for binary multiplication and their implementation on systolic architectures," ***IEEE Trans. Comput.***, vol. C-38, pp. 424–431, Mar. 1989.

[8] J.L. Baer, ***Computer Systems Architecture.*** New York: Computer Science Press, 1989.

[9] S.L. Hurst, "Multiple-valued logic—Its status and future," ***IEEE Trans. Comput.***, vol. C-33, pp. 1160–1178, Dec. 1984.

[10] G. Frieder and C. Luk, "Algorithms for binary coded balanced and ordinary ternary operations," ***IEEE Trans. Comput.***, vol. C-24, Feb. 1975.

[11] A. Slob and G.A.A. Bos, "Four valued logic," Nat. Lab. Tech. Note 235/77.

[12] T.T. Dao, L.K. Russell, D.B. Preedy, and E.J. McCluskey, "Multilevel I^2L with threshold gates," ***ISSCC Tech. Dig.***, pp. 110–111, Feb. 1977.

[13] T.T. Dao, "Threshold I^2L and its application to binary symmetric functions and multivalued logic," ***J. Solid State Circuits***, vol. SC-12, pp. 463–472, Oct. 1977.

[14] C. Lee and T. T. Dao, "I^2L logic and arithmetic technology," ***IEEE Dig. COMPCON Spring 77***, pp. 334–337, 1977.

[15] T. T. Dao, E. J. McCluskey, and L. K. Russell, "Multivalued integrated injection logic," ***IEEE Trans. Comput.***, vol. C-26, pp. 1233–1241, Dec. 1977.

[16] A. Druzeta, A.S. Sedra, and Z.G. Vranesic, "Application of multithreshold elements in the realization of many-valued logic networks," ***IEEE Trans. Comput.***, vol. C-23, pp. 1194–1198, 1974.

[17] T.T. Dao, Signetics Corp., Sunnyvale, CA, ***Internal Rep.***, 1979.

[18] F. Capasso and R.A. Kiehl, "Resonant tunneling transistor with quantum well base and high energy injection: A new negative differential resistance device," ***J. Appl. Phys.***, vol. 58, pp. 1366–1368, Aug. 1985.

[19] S. Sen, F. Capasso, A.Y. Cho, and D. Sivco, "Resonant tunneling device with multiple negative differential resistance: Digital and signal processing application with reduced circuit complexity," ***IEEE Trans. Electron Device***, vol. ED-34, pp. 2185–2190, 1987.

[20] N.C. DeTroye, "Integrated injection Logic—Present and future," ***IEEE J. Solid-State Circuits***, vol. SC-9, pp. 206–211, Oct. 1974.

[21] R.A. Allen and K.K. Schuegraff, "Oxide-isolated integrated injection logic," in ***Dig. Tech. Papers, IEEE Int. Solid State Circuits Conf.***, pp. 16–17, Feb. 1974.

[22] C. Mulder and H.E.J. Wulms, "High-speed I^2L," in ***Proc. 1st European Solid State Circuits Conf.***, pp. 28–29, Sept. 1975.

[23] R. Muller and J. Graul, "CHIL and I^2L with passive isolation," in ***Proc. 1st European Solid State Circuits Conf.***, pp. 30–31, Sept. 1975.

[24] T.T. Dao, M. Davio, and C. Gossart, "Complex number arithmetic with odd-valued logic," ***IEEE Trans. Comput.***, vol. C-29, pp. 604–610, July 1980.

[25] M. Davio and J.P. Deschamps, "Synthesis of discrete functions using I^2L technology," ***IEEE Trans. Comput.***, vol. C-30, pp. 653–661, Sept. 1981.

[26] R.P. Brent and H.T. Kung, "A regular layout for parallel adders," ***IEEE Trans. Comput.***, vol. C-31, pp. 260–264, Mar. 1982.

[27] T. Herman, "Linear algorithm that are efficiently parallelized to the time $O(\log n)$," Tech. Rep. TR-85-17, Dept. Comput. Sci., Univ. Texas, Austin, Sept. 1985.

[28] M.J. Quinn, ***Designing Efficient Algorithms for Parallel Computers.*** New York: McGraw-Hill, 1987.