2002

Fast Inner Product Computation on Short Buses

R. Lin

S. Olariu
Old Dominion University

Original Publication Citation

Lin, R., & Olariu, S. (2002). Fast inner product computation on short buses. *VLSI Design, 14*(4), 337-347. doi: 10.1080/10655140290011140

VLSI Design, 2002 Vol. 14 (4), pp. 337–347

Taylor & Francis
Taylor & Francis Group

Fast Inner Product Computation on Short Buses

R. LIN[a] and S. OLARIU[b,*]

[a]*Department of Computer Science, SUNY at Geneseo, Geneseo, NY 14454, USA;* [b]*Department of Computer Science, Old Dominion University, Norfolk, VA 23529, USA*

(Received 3 December 2000; Revised 12 April 2001)

We propose a VLSI inner product processor architecture involving broadcasting only over short buses (containing less than 64 switches). The architecture leads to an efficient algorithm for the inner product computation. Specifically, it takes 13 broadcasts, each over less than 64 switches, plus 2 carry-save additions (t_{csa}) and 2 carry-lookahead additions (t_{cla}) to compute the inner product of two arrays of $N = 2^9$ elements, each consisting of $m = 64$ bits. Using the same order of VLSI area, our algorithm runs faster than the best known fast inner product algorithm of Smith and Torng ["Design of a fast inner product processor," *Proceedings of IEEE 7th Symposium on Computer Arithmetic* (1985)], which takes about 28 $t_{csa} + t_{cla}$ for the computation.

Keywords: Application specific architectures; Computer arithmetic; Inner product processor; Reconfigurable bus system; Shift switching

INTRODUCTION

Processor arrays with buses have become the focus of much interest due to recent advances in VLSI and fiber optics[3]. Architectures featuring a reconfigurable bus system (REBS) including the reconfigurable mesh [13], and the polymorphic-torus [5] allow the configuration of the corresponding bus system to be changed dynamically under program control, to suit communication needs. These architectures have been extensively investigated and many efficient algorithms have been proposed. Examples include several fundamental algorithms on sorting, tree search, image processing, computational geometry, vision, and graph theory [1,4–6,10–13,15,16,18,20,23].

Recently the authors have proposed a new way of looking at bus systems. Our idea applies to both static and REBSs and involves enhancing traditional buses by the addition of a new feature that we call shift switching [7–9]. Just as in the reconfigurable architectures, our shift switching mechanism features local switches within each processing element (PE). However, the novelty of our idea is that we adopt a new class of switch states, which are manipulated by each processor. Specifically, we enable switches to rotate connections between lines (or tracks) of a bus. We show that this is a simple and powerful approach to improve the flexibility of a bus system.

The reconfigurable bus model did not gain wide acceptance because of its basic assumption: the time needed to transmit a signal along any bus is constant, regardless of the number of switches that the signal propagates through. According to traditional semiconductor technology, it is true that the transmission rate of a single switch has a lower bound, however, recent VLSI implementations have demonstrated that the rate is indeed quite small in terms of machine instruction cycles [17,21,22]. For example, broadcasting on a 1024 processor YUPPIE chip [11] requires only 16 instruction cycles (or 1 cycle for 64 processors). It takes even shorter delay on another chip called GCN, which adopts pre-charged circuits [14,19]. This confirms the feasibility and potential benefits of the models. What makes the models particularly attractive is the combination of (1) the lack of diameter concern due to the use of bus structures, (2) the multiple interconnection schemes due to the use of program control switches, (3) the high possibility for partial-optical or future all-optical implementations [1], thus, eventually achieving the $O(1)$ time broadcast in general.

The purpose of this paper is to propose a VLSI inner product processor involving only broadcasting over short (to be defined later) buses. The architecture leads to an efficient algorithm for the inner product computation. Specifically, it takes 12 broadcasts, each over 64 switches, plus 2 carray-save additions (t_{csa}) and 2 carry-lookahead additions (t_{cla}) to compute the inner product of two arrays of $N = 2^9$ elements, each consisting of $m = 64$ bits. Using the same order of VLSI area, our algorithm runs likely

*Corresponding author.

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/10655140290011140

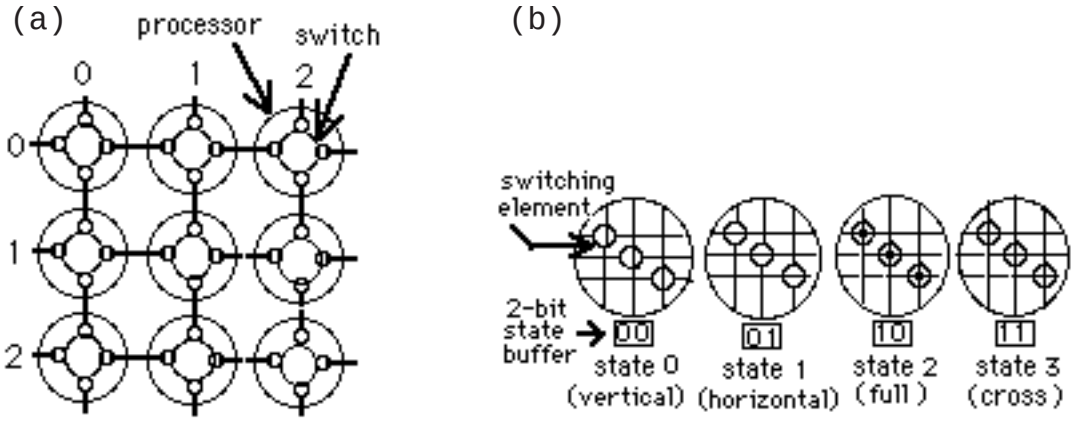


FIGURE 1 Reconfigurable bus system. (a) 3 × 3 reconfigurable mesh, (b) a switch and its four states $m = 3$.

faster than the best known fast inner product algorithm of Smith and Torng [18], which takes about $28t_{csa} + t_{cla}$ for the computation.

The paper is organized as follows: th second section gives a brief review on the concept of REBS with shift switching, which has been introduced in [8–10]. The third, fourth and fifth sections present the shift switching multiplier, shift switching counter, and the inner product processor architecture, respectively. The sixth section concludes the paper.

SHIFT SWITCHES

To make the paper self-contained, we shall review the basic features of a REBS, and the concept of shift switching which has been introduced in Ref. [5] For illustration purposes, consider the case of a reconfigurable mesh and refer to Fig. 1(a). A reconfigurable mesh consists of an $N \times N$ VLSI array of processors overlaid with a REBS. Every processor features four ports denoted by N, S, E and W. Local connections between these ports can be established under program control creating a powerful bus system that changes dynamically to accommodate various computational needs. We assume a single instruction stream: in each time unit, the same instruction is broadcast to all processors, which execute it and wait for the next instruction. Each instruction can consist of setting local connections (we refer to these as *switches*), performing an arithmetic or Boolean operation, broadcasting a value on a bus, or receiving a value from a specified bus. The regular structure of the reconfigurable mesh makes it suitable for VLSI implementation. In accord with other workers [1,4,8,9,13], we assume that broadcast along a bus of N switches takes $\delta(N)$ time. Recent experiments with the YUPPIE system [8] seem to indicate that $\delta(N) = O(1)$ is a reasonable working hypothesis. In particular, experimental results seem to indicate that wherever the number of switches (or processors) involved is less than 10^6, the broadcasting delay is a small constant, or $O(1)$. For our purposes, a switch (see Fig. 1(b)) can be seen as an array of m identical switching elements, which are under synchronous control of a processor. Every switching element involves a number of lines of buses. Several different *switch states* can be obtained by instructing a switch to set line contacts in different ways. For simplicity, the switches of a REBS are referred to as simple switches as opposed to shift switches introduced below.

A new type of switch (see Fig. 2), which we call a *shift switch* can be constructed from simple switches with changes only in internal wiring which guarantees that shift (or rotation) connections between the incoming and outgoing bus lines can be dynamically constructed. The notation $S_{m:d}$ stands for a switch featuring m switching elements, with the state changes controlled by d bits. Equipped with an $S_{m:d}$ switch a processor can shift one (or zero) bit of an incoming m-bit signal. We also assume the following:

(1) Each switch has a d-bit buffer called *state buffer*: if the contents of this buffer is k, then the processor can trigger its switch to state k;
(2) A switch has a special element, called a *rotation element*, to output the *rotation bit*;
(3) A processor can read the rotation bit and write the state-buffer.

Figure 2 illustrates an $S_{3:2}$ switch involving three switching elements and 4 states denoted by state 0, or I_0 (shift 0), state 1, or I_1 (shift 1), state 2, or H (horizontal), and state 3, or V (vertical). For example, when a processor sets its switch to state

- I_0, the following contacts are established: w (west) and e (east) in every switching element, as well as c (rotate-bit) and g (ground, which provides a 0 signal);
- I_1, the following contacts are established: w and i in every switching element, as well as a, b and c in the rotation element (it will soon be clear that c is not the same as but similar to a carry bit of an addition).

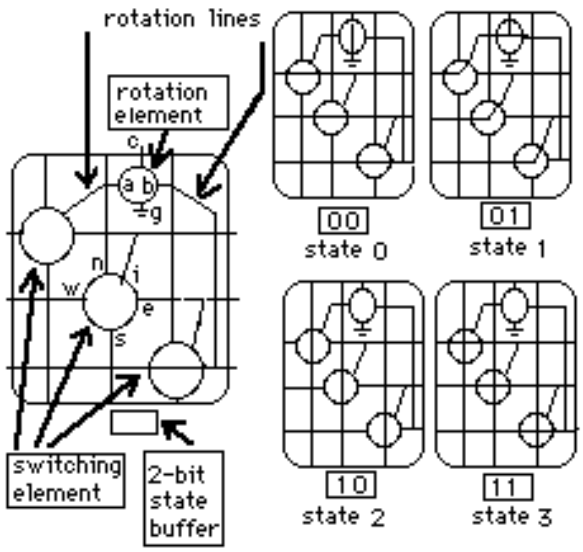


FIGURE 2 Shift switch $S_{3:2}$ and its four states.

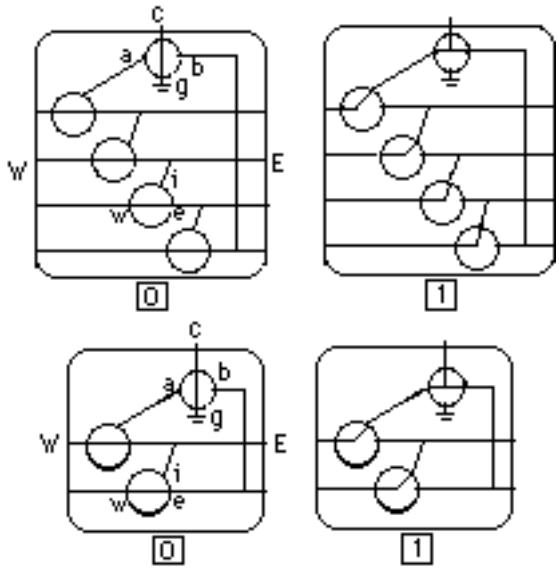


FIGURE 3 Shift switches $S_{m:1}$ ($m = 2$ and 4) with two ports.

It is easy to confirm that in state I_0 (I_1) the incoming signal is shifted 0 (1) lines. For the reader's convenience, two additional switches, $S_{4:1}$ and $S_{2:1}$ (also called a *basic shift switch*) are featured in Fig. 3: both have only two ports W and E and two states I_0 or I_1

The algorithm `Sum_N_Bits` which computes the sum of N binary numbers on a bus of N $S_{m:1}$ switches, which is also referred to as *a bus with cycle of N*, can be simply described as:

Assuming the state buffer of each switch has been loaded with the corresponding binary number, we iterate $k = [\log N/\log m]$ steps for the following five operations (see Fig. 4):

(1) Set up switches according to the values in their state buffers,
(2) Broadcast an m-bit signal $\overbrace{00\cdots01\mathrm{M}}^{m}$, that we call shifting signal, from the west end of the bus,
(3) Encode the output signal in the east end of the bus, and
(4) Shift the result register $\log m$ bits to the right, save the Encoded value to its first $\log m$ bits,
(5) Move the notation bit of each switch to its state buffer.

The sum of the N input bits, is in the result register.

It is easy to verify the correctness of the operation and time complexity which are given below

Theorem 1 of Ref. [8]): *On a linear array of N shift switches* (*or PEs*) *with bus width m*, *in* ($\log N/\log m$) *steps*, *we can compute the sum of N bits*, *using an encoder*, *a shift register and no adder*.

This result implies two important improvements for a REBS: First, the time for the fundamental parallel operation, sum of N bits, is reduced by a factor of ($\log m$). For many applications m is at least $\log N$, thus, $\mathrm{Log}\, N/\log\log N$ time is enough for the computation. When $m = N^{1/k}$, the approach achieves a constant time (k steps) summation of N bits. In particularly, some small k, for example, 1 to 4 are interested in shift switching bus designs. Second, no adder is now required within each PE, and no significant amount of additional hardware is needed for the construction of such a PE (switch) array.

Note that Based on YUPPIE chip [11] and GCN chip (which adopts pre-charged circuits) experiments, it is reasonable to say that when N is smaller enough (say 64 or less) each broadcast can be done in one or little more than one instruction cycle (say 30–60 ns.). In "SS counter and SAS unit" section, we introduce an efficient design of shift switching buses for summation of $N(N = 2^9)$ bits (called SS counter).

SHIFT SWITCHING MULTIPLIER

In this section, we introduce the architecture of a novel multiplier, that we call *SS multiplier*, based on shift switching mechanism. It is composed of m specific switch units, denoted as $U(m, 2)$, and an accumulator (or a CSA plus a CLA, i.e. carry save adder and carry lookahead adder). A $U(m, 2)$ is a union of m basic switches of $S_{2:1}$ (Fig. 5). The configuration of m $U(m, 2)$ units (Fig. 6) ensures the multiplier can receive the bit-product matrix (including sign bits) from $2(m + 1)$ input lines for two m-bit sign-magnitude numbers a and b. We assume that the binary representations for a and b are $a(m)a(m-1)\ldots a(0)$, and $b(m)b(m-1)\ldots b(0)$, respectively, and $a(m)$ and $b(m)$ are sign bits, $s = a(m) \oplus b(m)$ where $\oplus$ denotes modulo 2 addition, $a(k)$ and $b(k)$ (for $0 \le k \le m-1$) are binary digits. Figure 6 indicates that the j-th 1-bit state-buffer of k-th $U(m, 2)$ receives $a(j)\cdot b(k)$,

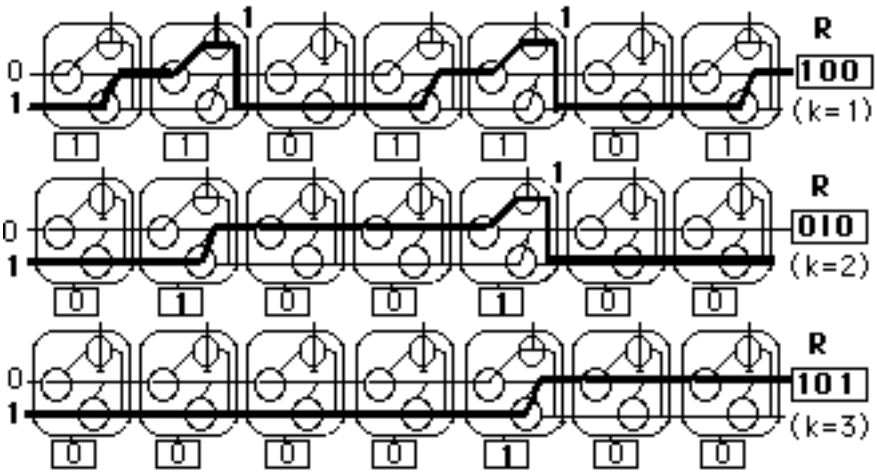


FIGURE 4 Summing $B = (1, 1, 0, 1, 1, 0, 1)$ (the ends of operation 4 for $k = 1$, 2 and 3 are shown).

$(0 \le j, k \le m - 1)$, thus the data in the state buffers of the SS multiplier is the bit-product matrix of a and b. Each of $2m - 1$ vertical 2-line buses (the maximum bus cycle is m) sums bits of the same magnitude (by applying theorem 1) in $\log m/\log 2 = \log m$ steps of broadcast. At the beginning of each step, all switches turn to new states simultaneously as dictated by the values in the state-buffers. Each rotation bit has a connection to the state-buffer of the same switch, thus the bit can be loaded into the state-buffer in the following clock cycle. j-th bus generates a single bit output to the j-th bit of the shifter in each step, the shifter (shifting 1 bit) and the accumulator (CSA), correctly concatenates and accumulates the 1-bit outputs of all vertical 2-line buses respectively, the final two numbers are added by a CLA. The sign bit s is generated in parallel. The product: $d = |a \cdot b|$ and the sign bit s are obtained in $\log m/\log 2 = \log m$ broadcasts plus a CLA addition (the CSA additions and broadcasts are executed in parallel). Compared with well-known add-and-shift multiplication scheme, and other types of multiplier, SS multiplier is competitive, because it requires only $\log m$ short bus (with a cycle of m) broadcasts (plus a CSA addition and a CLA addition), using m 2 basic switches plus a CSA adder and a CLA adder of $2m$ bits. However, the proposed SS multiplier is not a critical hardware component for our inner product processor. Clearly, any better multipliers can be used to replace the SS multipliers for the computation. The purpose of introducing SS multiplier here is for the further illustration that a fast inner product processor can be constructed solely using short shift switching buses.

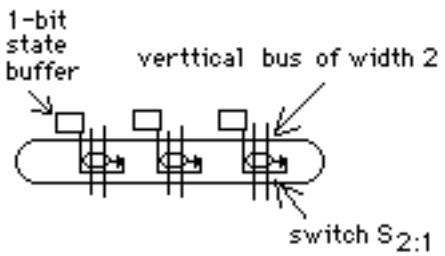


FIGURE 5 Linear switch unit: $U(m, 2)$ for $m = 3$.

SS COUNTER AND SAS UNIT

To sum N bits (or to count number of 1's in N bits) many parallel counter are available. However, they are either too expensive (requiring large amount of adder bits) or are too slow to cooperate with our processor. For our purpose, we can also use a shift switching bus of width $N^{1/2}$ and cycle N, and apply algorithm `Sum_N_Bits` on the bus to obtain the result in $\log N/(\log N^{1/2}) = 2$ broadcasts. However, for the computation, each broadcast signal must propagate through N switches. For large N (for example, $N = 2^9$), this may take unacceptable many instruction cycles (say, each of 20 ns) under the current VLSI technology, thus is not practical. Now we introduce a new efficient shift switching counter, that we call *SS counter*. An SS counter inputs N bits and generates four results: R_i^1, R_i^2, Q_i^1 and Q_i^2 in 4 steps of short broadcast, with the count of the N bits equal to $R_i^1 + R_i^2 * W + Q_i^1 * W + Q_i^2 * W^2$. However, these results (in successive three steps: Step 2, 3 and 4) are not weighted and added to obtain the sum, instead, they are directly loaded into another device, called *short array summation unit* (SAS unit for short), which is capable of summing all these results in parallel with SS counter's broadcasting, thus greatly improving the time performance of the whole inner product computation. We leave the detail description of an SAS unit in the next section. For simplicity, we restrict our discussion of SS counter for input $N = 2^9$, in general, for $N \ge 2^9$, the technique is likely to result in the same significant gain in broadcasting time and hardware cost. To explain the idea we first illustrate a simplified example below.

Let $N = 17$, in stead of using a single bus of width, $W = 5 = \sqrt{17}$, and cycle 17, we use three levels of short buses of width, $W = 4$. Level 1 consists of 4 buses of

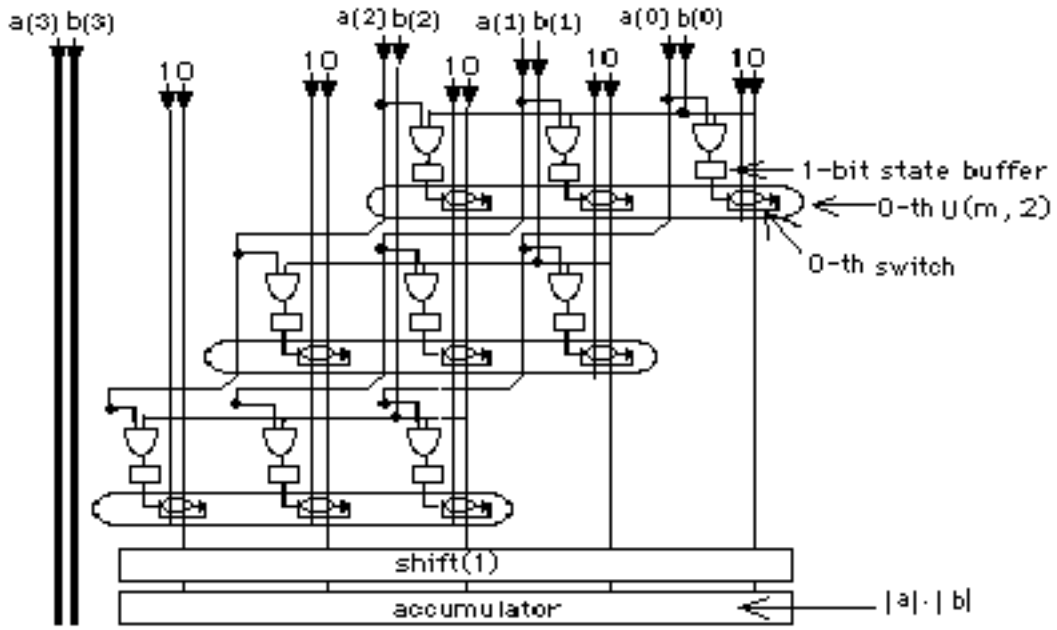


FIGURE 6 The shift switching multiplier with sign-magnitude input a and b of $m = 3$ bits. (a) Step 1, (b) Step 2, (c) Step 3, (d) Step 4.

cycles, 5, 4, 4, 4, respectively. The state buffers of level 1 buses receive 17 input bits. Both level 2 and 3 consist of a single short bus, with cycle 63 and cycle 57, respectively. Also a limited number (8, to be precise) of OR gates are used to connect the first two level buses. To compute the sum of 17 input bits, we use four steps as follows.

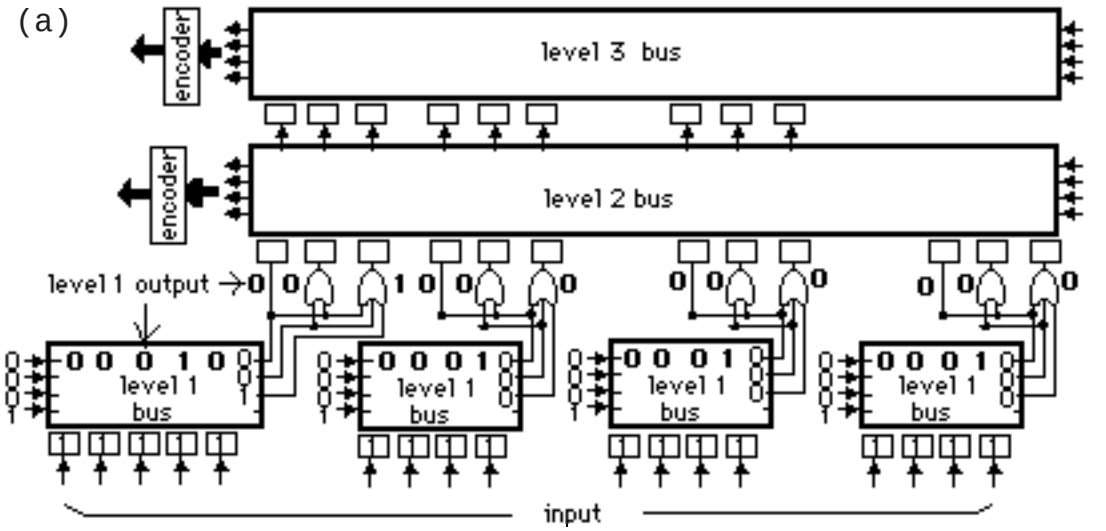
In Step 1 (Fig. 7(a)), each bus of level 1 broadcasts a 4-bit shifting signal 0001. Three output signal bits (except the 0-th bit) from the east end of each bus go to the state buffers of level 2. Each rotation bit goes back to the state buffer of its own switch; In Step 2 (Fig. 7(b)), buses of levels 1 and 2 trigger the switches and then broadcast shifting signals. Three output signal bits from the east end of each bus in level 1 go to the state buffers of level 2. The output signal bits of level 2 is encoded into a binary number, denoted by R_i^1 (to be consistent with the notations used in next section, we add subscript i here to mean the computation is on i-th SS counter). Each rotation bit of level 2 goes to the corresponding state buffer of level 3. In Step 3 (Fig. 7(c)), both buses of level 2 and level 3 trigger switches and broadcast the shifting signal. The output signal bits of level 2 is encoded into a binary number, denoted by R_i^2, the output signal bits of level 3 is encoded into a binary number, denoted by Q_i^1. Each rotation bit of level 2 again goes to the corresponding state buffer of level 3. In Step 4, only level 3 triggers switches and broadcasts the shifting signal. The output signal bits are encoded, denoted by Q_i^2. It is easy to verify that

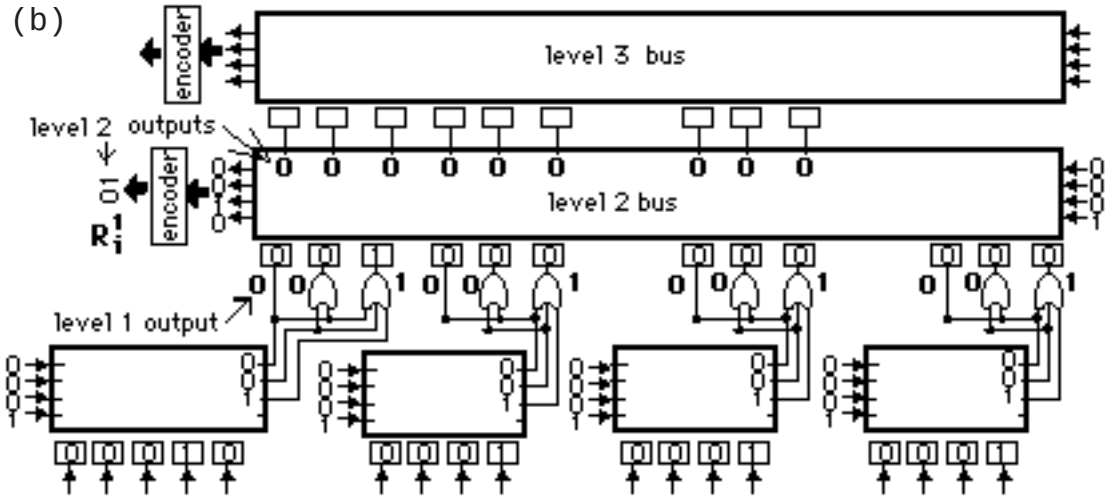
$$\text{Count of } N \text{ input bits} = R_i^1 + R_i^2 \times W + Q_i^1 \times W + Q_i^2 \times W^2. \quad \text{(A)}$$

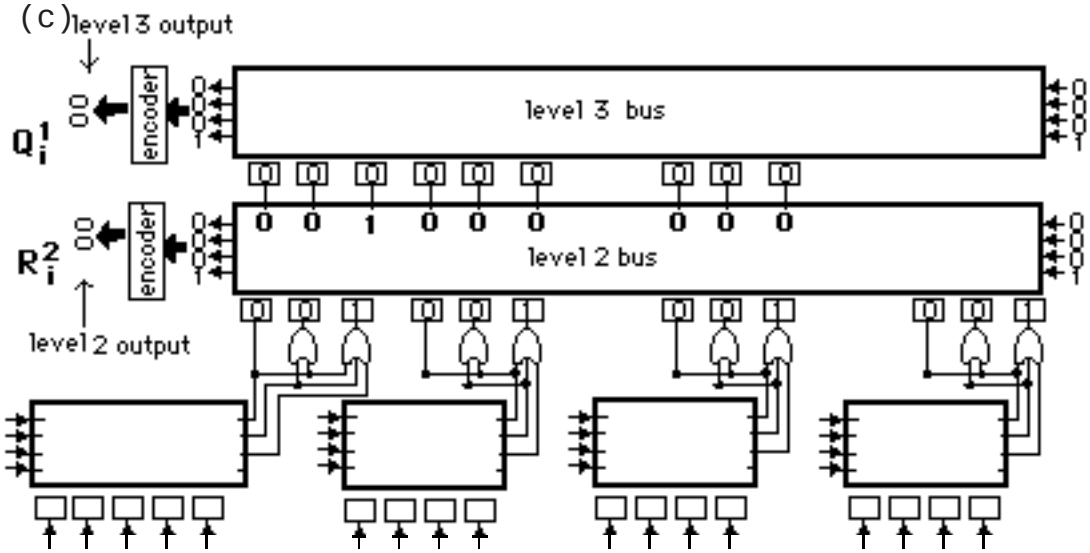
Since the size of the SS counter shown in the above example is too small, it does not reduce the time for the computation. However, if we apply the approach, to construct an SS counter for $N = 2^9$, we can have significant reductions on both running time and hardware cost in contrast with the design of using a bus of width $N^{1/2}$ and cycle N. The SS counter for $N = 2^9$ is the same as shown in the example in structure, but has different component sizes as described below. In level 1 of the SS counter, we use 9 bus segments of width 8, all having a cycle of 57 except one which has cycle 56, The facts, $57 \times 8 + 56 = 512$ and $57 < 8^2$ ensures that level 1 needs only two broadcasts. In level 2, we use a bus of width 8 and cycle 63. In level 3, we use a bus of width 8 and cycle 56. Similarly, both level 2 and 3 need only two broadcasts, thus the total number of broadcasts is 4 (due to the overlaps of the operations as shown in the example). It is easy to verify the following summary:

(1) It requires four steps of broadcasts, over 57, 63, 63, 56 switches, respectively (or broadcasting over total of 239 switching elements).
(2) It requires $2^9 \times 9 + 63 \times 9 + 56 \times 9 = 5664$ switching elements (including rotation elements), plus total 54 OR gates and two 8-to-3 encoder, which means that SS counter requires 11 switching elements and about 0.1 OR gate per bit.
(3) Level 2 and 3 have only $8(63 + 57)$ switching elements in total, it turns out that a VLSI layout of 4×512 will fit these two levels, thus the VLSI area of a SS counter is $(8 + 4)N \times a^2$, for $N = 2^9$, assume a switching element has area a^2.

In contrast with the shift switching bus of width $N^{1/2}$ and cycle N, SS counter requires significantly less amount of time, by a factor of $(2 \times 2^9)/239 = 4.3$, and less VSLI area (by a factor of 2) and less switching elements


(a)
encoder
level 3 bus
encoder
level 2 bus
level 1 output
level 1 bus
level 1 bus
level 1 bus
level 1 bus
input


(b)
encoder
level 3 bus
level 2 outputs
encoder
level 2 bus
level 1 output


(c)
level 3 output
encoder
level 3 bus
encoder
level 2 bus
level 2 output

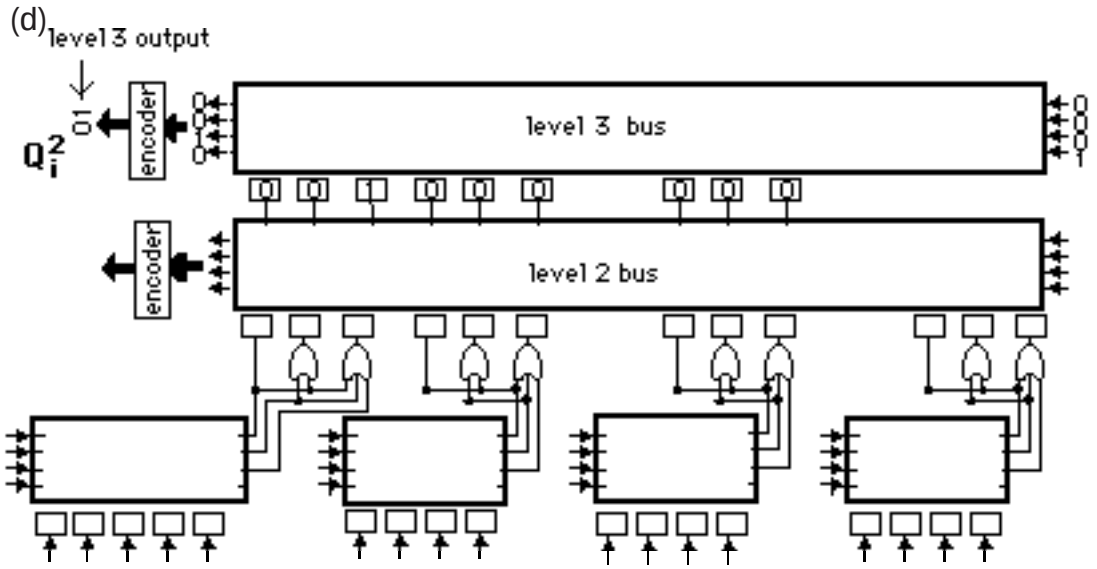


FIGURE 7 Summation of 16 bits (all 1 s) on a SS counter (SS counter $i = 0$ is shown). The outputs of each step are shown in bold. The counter's outputs are $R_i^1 = 01$ (step 2); $R_i^2 = 00$, $Q_i^1 = 00$ (step 3); $Q_i^2 = 01$ (step 4). The sum $= R_i^1 + R_i^2 \times 4 + Q_i^1 \times 4 + Q_i^2 \times 4^2 = 1 + 0 \times 4 + 0 \times 4 + 1 \times 4^2 = 17$.

$(N^{1/2} + 1 = 25$ switching elements per bit vs. 11 switching elements per bit).

THE PROCESSOR ARCHITECTURE AND INNER PRODUCT COMPUTATION

The overall inner product processor architecture consists of N SS multipliers, $2m$ SS counters (each with N input bits) and an SAS unit. In the following, we describe how the inner product can be computed on our proposed architecture.

Let input arrays: $A = (a_{N-1} \ldots a_j \ldots a_0)$, $B = (b_{N-1} \ldots b_j \ldots b_0)$, and $A \cdot B = \sum_{j=0}^{N-1} d_j$, here $d_j = |a_j \cdot b_j|$.

We compute each d_j and sign bit s_j (for $0 \le j \le N-1$) using an SS multiplier. The products of all SS multiplier are divided into two groups, the positive and the negative, with $\sum_{j=0,s_j=0}^{j=N-1} d_j$ representing the positive, and $\sum_{j=0,s_j=0}^{j=N-1} d_j$ representing the negative, thus

$$A \cdot B = \sum_{j=0,s_j=0}^{j=N-1} d_j + \sum_{j=0,s_j=0}^{j=N-1} d_{\mathrm{j}}$$

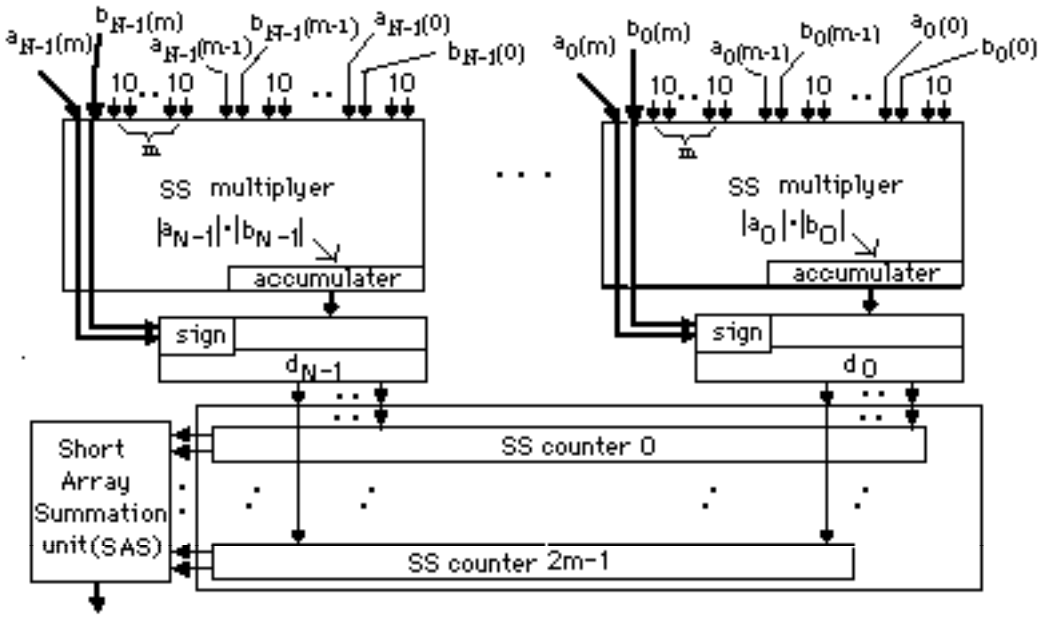


FIGURE 8 The inner product processor for the computation of $A \cdot B$. (a) The block diagram of SAS unit. (b) SAS bus 7.

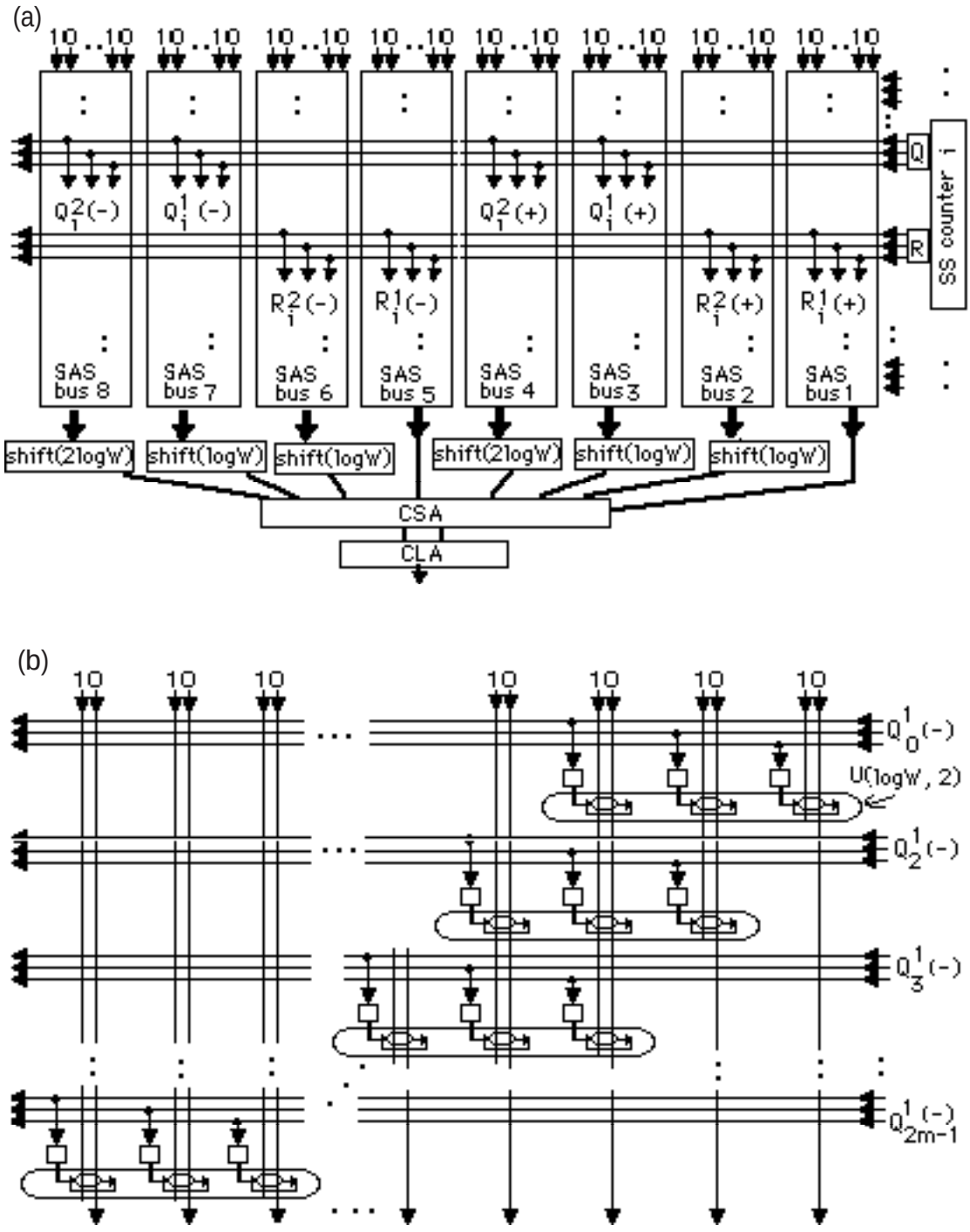


FIGURE 9 SAS (Short Array Summation) unit (for $N = 2^9$, $W = 8$), where each SAS bus line (vertical bus) contains no more than 3 switching elements.

Replacing d_j by its binary representation, we have

$$A \cdot B = \sum_{j=0,s_j=0}^{j=N-1} \sum_{i=0}^{2m-1} 2^i d_j(i) + \sum_{j=0,s_j=1}^{j=N-1} \sum_{i=0}^{2m-1} 2^i d_j(i)$$

$$= \sum_{i=0}^{2m-1} 2^i \sum_{j=0,s_j=0}^{j=N-1} d_j(i) + \sum_{i=0}^{2m-1} 2^i \sum_{j=0,s_j=0}^{j=N-1} d_j(i)$$

The positive and negative products are output from SS multipliers separately. The negative numbers are output 2 steps after the positive numbers are output to the SS counters. The i-th bit ($0 \le i \le 2m$) of j-th product (in j-th multiplier) goes to the j-th input (state buffer) of i-th SS counter. That is, i-th SS counter counts i-th bits of all N products. It takes 6 steps for each of $2m$ SS counters to complete the computation with eight results (R_i^1, R_i^2, Q_i^1, and Q_i^2, twice each) output to the SAS unit. We denote 4 results for positive products by $R_i^1(+)$, $R_i^2(+)$, $Q_i^1(+)$ and $Q_i^2(+)$, and denote the others for negative products by $R_i^1(-)$ $R_i^2(-)$

and $Q_i^1(-)$ $Q_i^2(-)$. We spell out the 6 steps as follows: (refer to the example), in Step 1 level 1 receives all positive products, and only level 1 buses broadcast; in Step 2, level 1 and 2 broadcast, only level 2 outputs $R_i^1(+)$; in Step 3, level 1 receives all negative products, and all three levels broadcast, while level 2 outputs $R_i^2(+)$, level 3 outputs $Q_i^1(+)$; in Step 4, all three levels broadcast, while level 2 outputs $R_i^1(-)$ and level 3 outputs $Q_i^2(+)$; in Step 5 level 2 and 3 broadcast, while level 2 output $R_i^2(-)$, level 3 output $Q_i^1(-)$; in Step 6, only level 3 broadcasts and outputs $Q_i^2(+)$ (Fig. 8).

Now by (A), we have

$$\sum_{j=0,s_j=0}^{j=N-1} d_j(i) = R_i^1(+) + R_i^2(+)\times W + Q_i^1(+)\times W + Q_i^2(+)\times W^2$$

and

$$\sum_{j=0,s_j=0}^{j=N-1} d_j(i) = R_i^1(-) + R_i^2(-)\times W + Q_i^1(-)\times W + Q_i^2(-)\times W^2$$

The outputs from SS counters are directly loaded into (the state buffers of) SAS unit. Noticed that each of these eight numbers has $\log W = 3$ ($W = 8$ for $N = 29$) bits, the configuration of SAS unit ensures the output of i-th SS counter is shifted to the magnitude of 2^i ($0 \le I \le 2m-1$). The SAS unit (Fig. 9) has eight short shift switching buses (SAS buses), four of them receive positive outputs, the other four receive negative outputs each having $2m$ $U(\log W = 3, 2)$ switch units. It is clear that each SAS bus has cycle 3, i.e. each vertical bus line contains no more than 3 switching elements. By theorem 1, it takes 2 broadcasts, each over 3 switches, to finish the summation of $2m$ array elements. The result from of each SAS bus are shifted to the corresponding magnitudes as follows: $R_i^1(+)$ and $R_i^1(-)$ are not shifted; $R_i^2(+)$, $R_i^2(-)$, $Q_i^1(+)$ and $Q_i^1(-)$, are shifted to the magnitude of W (i.e. $\log W = 3$ 0 s are added); $Q_i^2(+)$ and $Q_i^2(-)$ are shifted to the magnitude of W^2 (i.e. $2\log W = 6$ 0 s are added). These eight weighted numbers are output successively and are accumulated in a CSA (three inputs two outputs). Since every two very short broadcasts of SAS unit takes less time than one broadcast of SS counters over about 63 switches, after $Q_i^2(-)$ is output, it takes only two more very short broadcasts plus one CSA addition and one CLA addition we can obtain the final result, $A{\cdot}B$, thus

$$A{\cdot}B = \sum_{i=0}^{2m-1} 2^i R_i^1(+) + \sum_{i=0}^{2m-1} 2^i R_i^2(+)\times W + \sum_{i=0}^{2m-1} 2^i Q_i^1(+) \times W + \sum_{i=0}^{2m-1} 2^i R_i^2(+)\times W^2 - \sum_{i=0}^{2m-1} 2^i R_i^1(-) - \sum_{i=0}^{2m-1} 2^i R_i^2(-)\times W - \sum_{i=0}^{2m-1} 2^i Q_i^1(-)\times W - \sum_{i=0}^{2m-1} 2^i R_i^2(-)\times W^2$$

is computed.

We summarize the proposed inner product processor of input size, $N = 2^9$, as follows:

Time: $(\log m)t_b(m) + t_{csa} + t_{cla}$ {by SS multipliers) + $4t_b(63) + 2t_b(57)$ {by SS counters) + $2t_b(3)$ {by SAS unit, total 8 steps, but 6 steps are executed in parallel to the 6 steps of SS counter} + t_{csa} + t_{cla} {the final additions}

Here $t_b(x)$ means broadcast time on a bus of cycle x; t_{csa} means the time for one carry-save addition, t_{cla} means the time for one carry-lookahead addition.

If $4t_b(63) + 2t_b(57) + 2t_b(3)$ is counted as $7t_b(64)$, the total time is

$$(\log m)t_b(m) + 7t_b(64) + 2(t_{csa} + t_{cla}).$$

For 64-bit data ($m = 64$), the time is $13t_b(64) + 2(t_{csa} + t_{cla})$. For 16-bit data, the time is $7t_b(64) + 4t_b(16) + 2(t_{csa} + t_{cla})$.

Our processor likely works faster than the well-known fast inner product processor of Smith and Torng [18], which has computation time of $2(\log N + \log m - 1)t_{csa} + t_{cla}$. For $N = 2^9$, $m = 64$, it is $28t_{csa} + t_{cla}$. For $N = 2^9$, $m = 16$, it is $24t_{csa} + t_{cla}$.

Hardware (for $N = 29$)

number of switching elements:

$3Nm^2$ {N SS multipliers of m^2 basic shift switches, each having 3 switching elements} + $2m$ $(N(8+1) + 63\times(8+1) + 57\times(8+1))$ {$2m$ SS counters of 3 levels, each having 9 switching elements including rotation bits} + $2m$ $(3\times 3)\times 8$ {SASunit}ė$3Nm^2 + 22mN + 144m$

number of adder bits:

$N\times 2m\times 2${NSS multipliers} + $8\times(2m+3)\times 2 = 4\mathrm{N}m + 16m + 24$

Our processor likely has a less hardware elements than that of Smith and Torng's processor, which uses $Nm^2 + 2Nm + 2m\log N + m\log m - \log N - 3m$ carry-propagate adder bits. This means roughly that we use every 4 switching elements, they use one full adder bit, which likely costs more.

VLSI area: Assume that a switching element has area of a^2, an adder bit has an area of b^2 and a wire has width of l. By result in the summary (3) of "SS counter and SAS

unit" section, each SS counter has a area of length of $N \times a$ and width of $12 \times a$, now the total $2m$ SS counter require an area (vertical $2m$ data lines for each of N input are included):

$$A1 = Nm \times 2m\{N \text{ SS multipliers}\} + 2m \times 12a \times (N \times a + N \times 2m \times l) = 24Nma^2 + 48Nm^2 \times a \times l$$

It is reasonable to have the following rough estimate: $l = 1$, $a = 5$, $b = 25$, then (for $N = 2^9$)

$$A1 = N \times (24 \times 25m + 48 \times 5 \times m^2) = 120(5 + 2m)Nm.$$

The area of Smith and Torng's processor [18] is

$$A2 = N \times m \times (\log N + \log m)b2 = 625(9 + \log m)Nm.$$

For $m = 16$

$$A1 = 4440, \quad A2 = 8125m, \quad \text{i.e.}\, A2 = 1.8A1$$

For $m = 64$, $A1 = 15,840m$, $A2 = 8125$, i.e. $A1 = 1.9A2$.

Thus $A1$ and $A2$ are of the same order of magnitude.

CONCLUDING REMARK

Recently the authors have proposed a new way of looking at bus systems. Our idea applies to both static and REBSs and involves enhancing traditional buses by the addition of a new feature that we call shift switching [7–9]. It turns out that this is a simple and powerful approach to improve the efficiency and flexibility of a bus system. In this paper, we adopt and modify the switching mechanism to obtain a novel VLSI inner product processor architecture involving broadcasting only over short buses (containing no more than 64 switches). The architecture leads to an efficient algorithm for the inner product computation. Specifically, it takes 13 broadcasts, each over 64 switches, plus 2 carry-save additions (t_{csa}) and 2 carry-lookahead additions (t_{cla}) to compute the inner product of two arrays of $N = 2^9$ elements, each consisting of $m = 64$ bits. And it takes only 11 broadcasts, with 7 of them over 64 switches, and 4 over 16 switches, plus $2(t_{\text{csa}} + t_{\text{cla}})$ to compute the inner product for 16-bit data. Using the same order of VLSI area, our algorithm runs faster than the best-known fast inner product algorithm of Smith and Torng [18], which takes about $28t_{\text{csa}} + t_{\text{cla}}$ and $24t_{\text{csa}} + t_{\text{cla}}$ to compute the corresponding inner products for 64-bit and 16-bit input data, respectively.

Acknowledgements

The work was supported, in part, by National Science Foundation under grants MIP-9307664, CCR-9522093, and MIP 9630870, and by Grant N00014-91-1-0526.

References

[1] Bondalapati, K. and Prasanna, V.K. (1997) "Reconfigurable meshes: theory and practice", Proceedings of Reconfigurable Architecture Workshop: International. Parallel Processing Symposium (IEEE Computer Society Press).
[2] Hwang, Kai (1979) Computer Arithmetic (Wiley, New York).
[3] Kung, H.T. and Leiserson, C.E. (1980) "Algorithms for VLSI Processor Arrays", In: Mead, C. and Conway, L., eds, Introduction to VLSI Systems (Addison-Wesley, Reading, MA).
[4] Leighton, F.T. (1992) Parallel Algorithms and Architectures: Arrays Trees Hypercubes (Morgan Kaufmann Publishers, California), p 0.
[5] Li, H. and Maresca, M. (1989) "Polymorphic-torus network", *IEEE Transactions on Computers* **38**(9), 1345–1351.
[6] Lin, R. (1991) "Fast algorithms for the lowest common ancestor problem on a processor array with reconfigurable buses", *Information Processing Letters* **40**, 223–230.
[7] Lin, R. (1992) "Reconfigurable buses with shift switching—VLSI Radix sort", Proceedings of International Conference on Parallel Processing (ICPP), Chicago, IL **Vol. III**, pp 2–9.
[8] Lin, R. and Olariu, S. (1995) "Reconfigurable buses with shift switching—concepts and applications", *IEEE Transactions on Parallel and Distributed systems* **6**(1), 93–102.
[9] Lin, R. and Olariu, S. (1999) "Efficient VLSI architecture for Columnsort", *IEEE TVLSI* **7**(1), 135–139.
[10] Lin, R., Nakano, K., Olariu, S., Pintoti, M.C., Schwing, J.L. and Zomaya, A.Y. (2000) "Scalable hardware-algorithms for binary prefix sums", *IEEE Transactions on Parallel and Distributed Systems* **March**.
[11] Lin, R., Olariu, S., Schwing, J. and Zhang, J. (1992) "Sorting in $O(1)$ time on an $n \times n$ reconfigurable mesh" *Proceedings of the 9th European Workshop on Parallel Computing*, Barcelona, Spain, March pp. 16–27.
[12] Lin, R., Olariu, S., Schwing, J.L. and Wang, B.-F. (1999) "The mesh with hybrid buses: an efficient parallel architecture for digital geometry", *IEEE TPDS* **10**(3), 266–280.
[13] Miller, R., Kumar, V.K.P., Reisis, D. and Stout, Q.F. (1993) "Parallel computations on reconfigurable meshes", *IEEE Transactions on Computers* **42**, 678–-692.
[14] Olariu, S., Schwing, J.L. and Zhang, J. (1991) "Fundamental algorithms on reconfigurable meshes", Proceedings of the 29-th Annual Allerton Conference on Communications, Control, and Computing, pp 811–820.
[15] Olariu, S., Schwing, J.L. and Zhang, J. (1992) "Fast computer vision algorithms on reconfigurable meshes" *Proceedings of the Sixth International Parallel Processing*, Beverly Hill, CA, pp. 252–256.
[16] Rothstein, J. (1988) "Bus automata, brains, and mental models", *IEEE Transactions on Systems Man Cybernetics*, 18.
[17] Schaeffer, J. and Makarenko, D. (1985) "Systolic polynomial evaluation and matrix multplication with multiple precision", Proceedings of the IEEE 7th Symposium on Computer Arithmetic.
[18] Smith, S.P. and Torng, H.C. (1985) "Design of a fast inner product processor", Proceedings of IEEE 7th Symposium on Computer Arithmetic.
[19] Shu, D.B., Nash, J.G., *et al.* (1988) "The gated interconnection network for dynamic programming", In: Tewsburg, S.K., ed, Concurrent Computations (Plenum Publishing Corp).
[20] Shu, D.B., Chow, L.W. and Nash, J.G. (1988) "A constant addressable, bit serial associate processor," *Proceedings of the IEEE workshop on VLSI signal processing*, Montery, CA.
[21] Swartzlander, Jr., E.E., Gilbert, Barry K. and Reed, Irving S. (1978) "Inner Product Computers", *IEEE Transactions on Computers* **C-27**(1), 21–31.
[22] Swartzlander, Jr, E.E. (1990) Computer Arithmetic (IEEE CSP, CA) **Vol. 1**.
[23] Wang, B.F., Lu, C.J. and Chen, G.H. (1990) "Constant time algorithms for the transitive closure problem and some related graph problems on processor array with reconfigurable bus system", *IEEE Transactions on Parallel and Distributed Systems* **1**(4), 500–507.

Authors' Biographies

Stephan Olariu received MSc and PhD degrees in computer science from McGill University, Montreal in 1983 and 1986, respectively. In 1986, he joined the Computer Science Department at Old Dominion University where he is now a professor. Dr Olariu has published extensively in various journals, book chapters, and conference proceedings. His research interests include wireless networks and mobile computing, parallel and distributed systems, performance evaluation, and medical image processing. Dr Olariu serves on the editorial board of several archival Journals including "IEEE Transactions on Parallel and Distributed Systems," "Journal of Parallel and Distributed Computing," "International Journal of Foundations of Computer Science," "Journal of Supercomputing," "International Journal of Computer Mathematics," "VLSI Design," and "Parallel Algorithms and Applications."

Rong Lin received the BS degree in mathematics from Peking University, Beijing, China, the MS degree in computer science from Beijing Polytechnic University, Beijing, China, and PhD degree in computer science from Old Dominion University, Norfolk, Virginia, in 1989, where he now is a professor and the chair of the computer science department. Dr Lin's current research interests include parallel architectures, VLSI arithmetic circuits, run-time-reconfigurable digital circuits, and parallel algorithm designs.