

A Scalable VLSI Architecture for Binary Prefix Sums *

R. Lin[†] K. Nakano[‡] S. Olariu[§] M. C. Pinotti[¶] J. L. Schwing[‡]
A. Y. Zomaya^{||}

Abstract

The task of computing binary prefix sums (BPS, for short) arises, for example, in expression evaluation, data and storage compaction, and routing. This paper describes a scalable VLSI architecture for the BPS problem. We adopt as the central theme of this effort, the recognition of the fact that the broadcast delay incurred by a signal propagating along a bus is, at best, linear in the distance traversed. Thus, one of our design criteria is to keep buses as short as possible. In this context, our main contribution is to show that we can use short buses in conjunction with shift switching to obtain a scalable VLSI architecture for the BPS problem.

1 Introduction

Recently, in response to the ever increasing demand for speed and flexibility, bus systems whose configuration can be dynamically changed to suit communication needs have been proposed in the literature. Machines featuring a reconfigurable bus system (REBS, for short) include the *reconfigurable mesh* [7], the *polymorphic torus* [2], and the PPA [6], to name just a few.

A typical REBS platform contains two major components: a local programmable switch within each processor and a bus network whose global configuration can be dynamically changed by adjusting the local switches.

The REBS and its variants have proved to be valuable theoretical models [1]. At the same time, the REBS was slow to gain wide acceptance because it ignores important properties of physical architectures. Most of the REBS platforms assume that the time needed to broadcast a signal along a bus is a constant, regardless of the number of

switches through which the signal propagates [6]. Worse yet, the REBS do not scale and, as a consequence, do not support virtual parallelism [6].

Motivated by the goal of developing a realistic and scalable bus-based platform, Lin and Olariu [3] have recently proposed a restricted REBS called the *reconfigurable bus with shift switching* (REBSIS, for short). The idea is to abandon the unrestricted power of the REBS in favor of unidirectional broadcast buses endowed with a new feature called *shift switching*. However, the novelty of their design was to enable switches to cyclically permute an incoming signal on a bus during broadcasting. The resulting architecture, the REBSIS, was shown to lead to superior special-purpose designs ranging, for example, from inner product computation, to sorting, to small adders [4, 5].

We adopt, as the central theme of this effort, the recognition of the fact that the broadcast delay incurred by a signal propagating along a medium is, at best, linear in the distance traversed. Thus, our main design criterion is to keep buses as short as possible. In this context, the main contribution of this work is to show that we can use short buses in conjunction with shift switching to design a scalable VLSI architecture for one of the fundamental problems of parallel processing – computing binary prefix sums (BPS, for short).

2 Shift Switching – an Overview

At the generic level, a REBS consists of an array of processors overlaid with a reconfigurable bus system [4]. The reconfiguration of the platform is supported at the processor level [6] by providing every processor with a number of switches that can be connected according to various rules [2]. Generically, a switch can be seen as an array of w identical switching elements, under the synchronous control of a processor [6]. Several different switch states can be obtained by instructing a switch to set specific line connections indicated by the value stored in a special register, termed *state register*. For simplicity, the switches of a REBS will be referred to as *regular*, to distinguish them from the shift switches introduced below.

A *shift switch* can be informally defined as a switch that can cyclically permute the bit pattern of a passing w -bit signal by 0, 1, or more bus lines. We use the notation $S(w, 1)$ to stand for a shift switch of w switching elements,

*Work supported by NSF grants CCR-9522092 and MIP-9630870, by NASA grant NAS1-19858, by ONR grants N00014-95-1-0779 and N00014-97-1-0562 and by ARC grant 04/15/412/194.

[†]Department of Computer Science, SUNY Geneseo, Geneseo, NY 14454, USA

[‡]Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, JAPAN

[§]Department of Computer Science, Old Dominion University, Norfolk, VA 23529, USA

[¶]I.E.I., C.N.R., Pisa, ITALY

^{||}Parallel Computing Research Lab, Dept. of Electrical and Electronic Eng, University of Western Australia, Perth, Australia

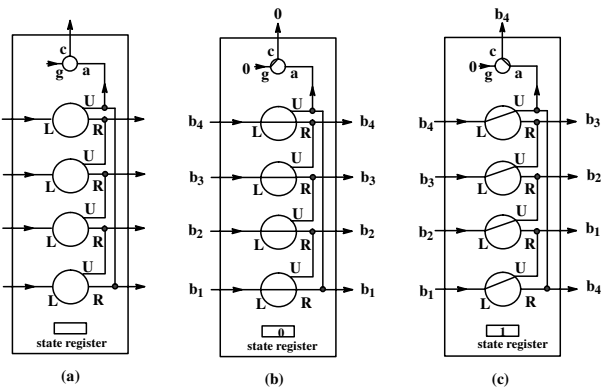


Figure 1. Block diagram of the $S(4, 1)$

with the state setup controlled by a unique bit. A $S(w, 1)$ can have up to 2 switch states.

The block diagram of the $S(4, 1)$ is illustrated in Figure 1(a). Here, each switching element has three contacts: L (left), R (right), and U (up 1 bit). The $S(w, 1)$ features a rotation element with has three contacts, c (rotation bit), a (connecting to switching elements), and g (ground). The $S(w, 1)$ also features w switching elements with the state changes controlled by a 1-bit state register. Equipped with a $S(w, 1)$ switch a processor can cyclically permute the bit pattern of an incoming w -bit signal by 0 or 1 bits, as illustrated in Figure 1 (b)–(c) (All the concepts will be illustrated using an example featuring $w = 4$).

For our purposes, a *processing element* (PE) consists of a $S(w, 1)$, a w -to- $\log w$ encoder and a few more registers. A linear array of processing elements is termed a *linear REBSIS*. The processing elements in left-to-right order are $PE(1), PE(2), \dots, PE(n)$. The input is a bit sequence b_1, b_2, \dots, b_n injected, one bit per processing elements, with $PE(i)$ storing b_i .

Lin and Olariu [4] proved the following result.

Proposition 1 The prefix sums of a sequence of n bits can be computed on a linear REBSIS consisting of n $S(w, 1)$ s in the time of $\left\lceil \frac{\log(n+1)}{\log w} \right\rceil$ broadcasts, each over a bus of length n .

3 The Scalable VLSI Architecture

The main goal of this section is to discuss a simple scalable VLSI architecture for the BPS problem, in which all broadcasts are restricted to buses of length $w^2 - 1$, where w is the assumed width of the bus. Having replaced the long bus by a collection of short buses, one must precompute the shifting signals to be used on each of these buses. Let w denote some small power of 2. We shall use as a basic building block of our design a linear REBSIS consisting of $w^2 - 1$ shift-switches of the type $S(w, 1)$ and a bus of width w . Each basic building block B_i contains w

groups $G_{i,1}, G_{i,2}, \dots, G_{i,w}$, each consisting of w consecutive $S(w, 1)$ s, with the exception of $G_{i,1}$ which consists of $w - 1$ consecutive $S(w, 1)$ s, as illustrated in Figure 2. Each B_i has a special register, the *signal buffer* whose role is to store the signal to be used in the next broadcast through B_i .

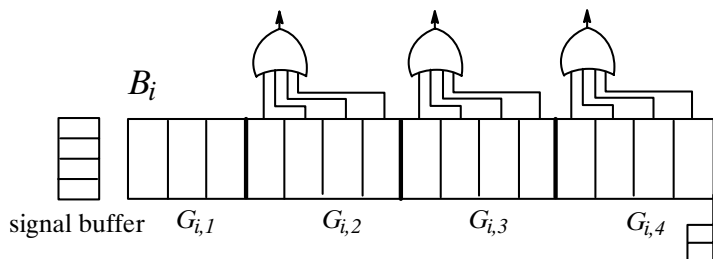


Figure 2. The block diagram of a basic building block

Each group $G_{i,j}$, $2 \leq j \leq w$, is equipped with an OR gate whose role is to combine the rotation bits in the group. It is important to note that, in each group $G_{i,j}$, at most one of the rotation bits of the $S(w, 1)$ s can be a 1. This guarantees that the number of rotation bits of all $S(w, 1)$ s in a basic building block is precisely the number of 1's output by the OR gates associated with the block.

Finally, when the contents x_1 of the signal-buffer is broadcast through block B_i , whose state registers in $S(w, 1)$'s have been loaded with x_2, \dots, x_t , the signal that passes the j -th $S(w, 1)$, ($1 \leq j \leq w^2 - 1$), of B_i represents the sum S of the contents of the signal-buffer and of all the state registers to the left of the j -th $S(w, 1)$'s in B_i . Moreover, there are $\lfloor S/w \rfloor$ 1's amongst the rotations bits.

In essence, our VLSI architecture for solving an instance of size n of the BPS problem is a complete $(w - 1)$ -ary tree of height $h = \left\lceil \log_{(w-1)} n \right\rceil$, with level r , ($r \geq 1$), involving blocks B_i such that $\frac{(w-1)^r - 1}{w-2} + 1 \leq i \leq \frac{(w-1)^{r+1} - 1}{w-2}$. An example with $h = 2$ is depicted in Figure 3. The input sequence is injected one bit per $S(w, 1)$ with bits $b_{(i-1)(w^2-1)+1}, \dots, b_{i(w^2-1)}$ stored in block B_i ¹.

Lemma 1 The i -th digit Ψ_i of the weighted w -ary representation of a generic sum $\Psi = \sum_{k=1}^t x_k$ is:

$$\Psi_i = \left\lfloor \frac{\sum_{k=1}^t x_k}{w^i} \right\rfloor \bmod w = \quad (1)$$

$$\left(\sum_{k=1}^t \left(\left\lfloor \frac{x_k}{w^i} \right\rfloor \bmod w \right) + \left\lfloor \frac{\sum_{k=1}^t (x_k \bmod w^i)}{w^i} \right\rfloor \right) \bmod w.$$

¹Due to stringent page limitations, the proof of the following results are omitted in this extended abstract. They can be found in the journal version of this work.

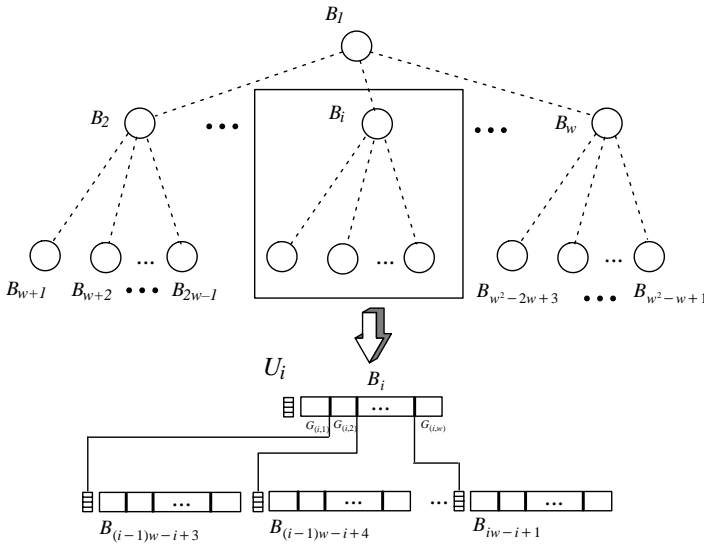


Figure 3. Our 3-level VLSI architecture

Let the number of 1's amongst the rotation bits of $S(w, 1)$'s in B generated while computing Ψ_i according to (1) in Lemma 1 be

$$rbits(\Psi_i) = \left\lfloor \frac{\sum_{k=1}^t \left(\left\lfloor \frac{x_k}{w^i} \right\rfloor \bmod w \right) + \left\lfloor \frac{\sum_{k=1}^t (x_k \bmod w^i)}{w^i} \right\rfloor}{w} \right\rfloor.$$

Lemma 2

$$rbits(\Psi_i) = \left\lfloor \frac{\sum_{k=1}^t (x_k \bmod w^{i+1})}{w^{i+1}} \right\rfloor. \quad (2)$$

Lemmas 1 and 2 imply the following result.

Theorem 1 For every $i \geq 1$,

$$\Psi_i = \left(\sum_{k=1}^t \left(\left\lfloor \frac{x_k}{w^i} \right\rfloor \bmod w \right) + rbits(\Psi_{i-1}) \right),$$

and

$$rbits(\Psi_i) = \left\lfloor \frac{\sum_{k=1}^t \left(\left\lfloor \frac{x_k}{w^i} \right\rfloor \bmod w \right) + rbits(\Psi_{i-1})}{w} \right\rfloor.$$

Observe that Theorem 1 describes how the weighted w -ary representations of x_1, \dots, x_t can be summed to get the weighted w -ary representation of $\Psi = \sum_{k=1}^t x_k$. This is crucial when not all the bits involved in the BPS problem can be summed together directly due to the use of short buses.

In order to explain how our scalable VLSI architecture works, let S_i , ($1 \leq i \leq n$), stand for the number of 1's among the bits in B_i , and let $P_1 = S_1$, $P_2 = S_1 + S_2$, \dots , $P_{n-1} = S_1 + S_2 + \dots + S_{n-1}$ be the prefix sums corresponding to the rightmost bit in each building block. In outline our design performs the following:

- we begin by computing – from the least to the most significant – the digits of the weighted representation of P_1, P_2, \dots, P_n , and then
- for every i , ($2 \leq i \leq n$), we use the weighted digits of P_{i-1} as the (precomputed) shifting signal for computing the digits of the same weight of the prefix sums in block B_i .

First, in Stage 1, the k -th digit of the prefix sums $P_1, P_w, P_{w^2-w+1}, \dots, P_{\frac{(w-1)h-1}{w-2}}$ are computed by evaluating, in a bottom-up traversal of the tree, the k -th digits of the weighted representations of S_j 's, ($1 \leq j \leq n$). In practice, each block, except for the root B_1 , moves its local value to its father, which by a broadcast adds the partial sums available at the rightmost $S(w, 1)$ of its children. In this way, the data are moved one level up at each broadcast, and the sum corresponding to all the bits initially in level i is first partitioned amongst the blocks in level $i-1$, then amongst the blocks in level $i-2$, and so on. Having reached level 2, it is partitioned amongst the $w-1$ blocks at that level. Moving to level 1, the broadcast at B_1 , which adds the local sum to the partial sums of its children, accumulates the sum of all the bits of level i to the sum of all bits stored in levels $1, 2, \dots, i-1$, computing $P_{\frac{(w-1)i-1}{w-2}}$. Subsequently, the tree is traversed top-down and the process is repeated backwards. In this phase of Stage 1, each block supplies its children with the correct signal to compute the local prefix so that the number of the prefix sums available for level i increases by a factor $w-1$ after every broadcast. In practice, from the k -th digit of $P_{\frac{(w-1)i-1}{w-2}}$, the k -th digit, $1 \leq k \leq w-1$, of the prefix sums $P_k^{\frac{(w-1)i-1}{w-2}}$ is determined after the broadcast. In this way, starting at level 1 with $P_{\frac{(w-1)i-1}{w-2}}$ and moving down up to level i , blocks B_j at level i , where $\frac{(w-1)i-1}{w-2} + 1 \leq j \leq \frac{(w-1)i-1}{w-2}$, will learn about the k -th digit of the corresponding prefix sum, P_j . Stage 2 then completes the computation of the prefix sums at each $S(w, 1)$ in B_j .

To address the lower bound, in our model, of the number of broadcasts required in each block, we concentrate on the blocks at level $i \geq 2$. During the top-down phase of Stage 1, due to the fact that the k -th digit of the sums referring to the input bits at level v is computed the first time at level $v-k$, any block at level i of a tree of height h must process to convey to its father the $v-i+1$ less significant digits of the partial sums computed at levels v , with $v \geq i$. Moreover, during the bottom-up phase of Stage 1, since the w -ary representation of the prefix sums corresponding to the blocks at the level v requires $v+1$ digits, any block at level i moves into the signal-buffer of its children $v+1$ digits for the prefix corresponding to the blocks at level v , with $v \geq i+1$. Finally, Stage 2 requires $i+1$ broadcast to compute the $i+1$ digits of the local prefix sums. Hence,

altogether $\sum_{v=i}^h (v-i+1) + \sum_{v=i+1}^h (v+1) + (i+1) = h^2 + 3h + 2 - i(h+2)$ broadcast are required to each block at level $i \geq 2$. The number of broadcasts is maximized for $i = 2$.

Notice that for level $i = 1$, it is clear that $v+1$ broadcasts are required to accumulate the digits at level v , with $v \geq 1$. Therefore, $\frac{(h+1)(h+2)}{2}$ broadcasts are required at level 1.

In conclusion, at least $\max\left(h^2 + h - 2, \frac{h^2 + 3h + 2}{2}\right)$ broadcasts are required to solve an instance of size $n = (w^2 - 1) \frac{(w-1)^{h-1}}{w-2}$ of the BPS problem if the broadcasts are restricted to buses of length $w^2 - 1$.

To give the reader a better idea of how our general architecture works, in the next section we discuss a simple instance of the general design.

4 A 3-level VLSI Architecture

In this section we discuss the details of a 3-level instance of our scalable VLSI architecture for computing optimally the prefix sums of $n = w^4 - w^3 + w - 1$ bits in 10 broadcasts, each over a bus of $w^2 - 1$ switches. The input is a bit sequence $b_1, b_2, \dots, b_{w^4 - w^3 + w - 1}$, injected one bit per $S(w, 1)$, into the BPS unit in Figure 3, with the bits $b_{(i-1)(w^2-1)+1}, \dots, b_{i(w^2-1)}$ stored in the basic building block B_i ($1 \leq i \leq w^2 - w + 1$). In this particular instance of the problem, we observe that each of the prefix sums can be expressed by a weighted w -ary representation of at most 4 digits, termed *low*, *mid1*, *mid2* and *high*, from the least to the most significant. It is noteworthy that the prefix sums at B_1 can be expressed using only the *low* and *mid1* digits (i.e., *mid2* = *high* = 0). Similarly, for the prefix in B_2, \dots, B_w , the most significant digit *high* is 0.

To simplify the exposition, let the j -th $S(w, 1)$ of B_i be provided by the following registers²: let $SR(i, j)$ denote the state-register, $SO(i, j)$ and $c(i, j)$ store, respectively, the signal that passes through the shift switch and the rotation bit during the last broadcast at B_i . Moreover, let $low(i, j)$, $mid1(i, j)$, $mid2(i, j)$ and $high(i, j)$ save the digits of the w -ary representation of the local prefix sum, and let $t0(i, j), t1(i, j), \dots, t6(i, j)$ be a set of temporary registers used to save partial results. Finally, let $SB(i)$ be the signal buffer register of B_i , and let $SB1(i)$ and $SB2(i)$ be temporary registers for signals to be loaded into the signal buffer during the computation.

A description of the algorithm follows. At the beginning of each Step, we assume that the state-registers are cleared.

Step 1.

1. For $1 \leq i \leq w^2 - w + 1, 1 \leq j \leq w^2 - 1, SR(i, j) := b_{(i-1)(w^2-1)+j}$;
2. For $1 \leq i \leq w^2 - w + 1, SB(i) := "000 \dots 01"$;
3. For $1 \leq i \leq w^2 - w + 1$, broadcast $SB(i)$ in $B(i)$;

²All these registers are initialized to 0.

4. For $1 \leq j \leq w^2 - 1, low(1, j) := SO(1, j)$;
5. For $2 \leq i \leq w^2 - w + 1, 1 \leq j \leq w^2 - 1, t0(i, j) := SR(i, j)$;
6. For $2 \leq i \leq w, low(i, w^2 - 1) := SO(i, w^2 - 1)$;
7. For $2 \leq j \leq w, t1(1, j-1) := \bigvee_{k=0}^{w-1} c[1, (j-1)w + k]$;
8. For $1 \leq i \leq w, 2 \leq j, k \leq w, t1[i, (j-1)w + k - 1] := \bigvee_{k=0}^{w-1} c[(i-1)w - i + j + 1, (k-1)w + j]$;
9. For $1 \leq j \leq w^2 - 1, t0(1, j) := c(1, j)$.

Step 2.

1. For $1 \leq k \leq low(1, w^2 - 1), SR(1, k) := 1$;
2. For $1 \leq k \leq low[(i-1)w - i + j + 1, w^2 - 1], 1 \leq i \leq w, 2 \leq j \leq w, SR(i, (j-1)w + k) := 1$;
3. For $1 \leq i \leq w^2 - w + 1, SB(i) := "00 \dots 01"$;
4. For $1 \leq i \leq w^2 - w + 1$, broadcast $SB(i)$ in $B(i)$;
5. For $2 \leq j \leq w, t2(1, (j-1)w) := \bigvee_{k=0}^{w-1} c(1, (j-1)w + k)$;
6. For $2 \leq j \leq w, SB1(j) := SO(1, jw - 1)$.

Step 3.

1. For $1 \leq k \leq SO(1, w^2 - 1), SR(1, k) := 1$;
2. For $2 \leq j \leq w, 1 \leq k \leq SO(i, w^2 - 1), SR(1, (j-1)w + k) := 1$;
3. For $2 \leq i, j, k \leq w, SR(i, (j-1)w + k - 1) := t1(i, (j-1)w + k - 1)$;
4. For $2 \leq i, j \leq w, SR(i, (j-1)w) := \bigvee_{k=0}^{w-1} c(i, (j-1)w + k)$;
5. For $1 \leq i \leq w^2 - w + 1, SB(i) := "00 \dots 01"$;
6. For $1 \leq i \leq w$, broadcast $SB(i)$ through $B(i)$;
7. For $2 \leq j \leq w, t3(1, (j-1)w) := \bigvee_{k=0}^{w-1} c(1, (j-1)w + k)$;
8. For $2 \leq i, j \leq w, t4(1, (i-1)w + j - 1) := \bigvee_{k=0}^{w-1} c(i, (j-1)w + k)$;
9. For $2 \leq i \leq w, mid1(i, w^2 - 1) := SO(i, w^2 - 1)$.

Step 4.

1. For $1 \leq j \leq w, 1 \leq k \leq w - 1, SR(1, (j-1)w + k) := t1(1, (j-1)w + k)$;
2. For $2 \leq j \leq w, SR(1, (j-1)w) := t2(1, (j-1)w)$;
3. For $2 \leq i, j \leq w, 1 \leq k \leq low((i-1)w + i + j + 1, w^2 - 1)$;
4. $SB(1) := "00 \dots 01"$;
5. For $2 \leq i \leq w, SB(i) := SO(1, (i-1)w - 1)$;
6. For $1 \leq i \leq w$, broadcast $SB(i)$ through $B(i)$;
7. For $2 \leq j \leq w, t4(1, j-1) := \bigvee_{k=0}^{w-1} c(1, (j-1)w + k)$;
8. For $1 \leq j \leq w^2 - 1, t5(1, j) := c(1, j)$;
9. For $2 \leq j \leq w, SB2(j) := SO(1, (j-1)w - 1)$;

10. For $2 \leq i, j \leq w$, $t2(i, (j-1)w) := \bigvee_{k=0}^{w-1} c(i, (j-1)w+k)$.

Step 5

1. For $1 \leq k \leq SO(1, w^2-1)$, $SR(1, k) := 1$;
2. For $2 \leq j \leq w$, $1 \leq k \leq mid1(j, w^2-1)$, $SR(1, (j-1)w+k) := 1$;
3. For $2 \leq j \leq w$, $SR(1, (j-1)w) := t3(1, (j-1)w)$;
4. For $2 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $SR(i, j) := t0(i, j)$;
5. $SB(1) := "00 \dots 01"$;
6. For $2 \leq i \leq w$, $SB(i) := SB1(i)$;
7. For $2 \leq i, j \leq w$, $SB((i-1)w+j-i+1) := SO(i, (j-1)w-1)$;
8. For $1 \leq i \leq w^2-w+1$, broadcast $SB(i)$ in $B(i)$;
9. For $2 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $low(i, j) := SO(i, j)$; $t6(i, j) := c(i, j)$.

Step 6.

1. For $2 \leq j \leq w$, $SR(1, (j-1)w) := \bigvee_{k=0}^{w-1} c(1, (j-1)w+k)$;
2. For $1 \leq j \leq w$, $1 \leq k \leq w-1$, $SR(1, (j-1)w+k) := t4(1, (j-1)w+k)$;
3. For $2 \leq i, j \leq w$, $1 \leq k \leq w-1$, $SR(i, (j-1)w+k) := t1(i, (j-1)w+k)$;
4. For $2 \leq i, j \leq w$, $SR(i, (j-1)w) := t2(i, (j-1)w)$;
5. $SB(1) := "00 \dots 01"$;
6. For $2 \leq i \leq w$, $SB(i) := SO(1, (i-1)w-1)$;
7. For $1 \leq i \leq w$, broadcast $SB(i)$ in $B(i)$.

Step 7.

1. For $2 \leq j \leq w$, $SR(1, (j-1)w) := \bigvee_{k=0}^{w-1} c(1, (j-1)w+k)$;
2. For $2 \leq i, j \leq w$, $SR(i, (j-1)w) := \bigvee_{k=0}^{w-1} c(i, (j-1)w+k)$;
3. For $i+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $SR(i, j) := t6(i, j)$;
4. $SB(1) := "00 \dots 01"$;
5. For $2 \leq i \leq w$, $SB(i) := SO(1, (i-1)w-1)$;
6. For $2 \leq i, j \leq w$, $SB((i-1)w-i+j+1) := SO(i, (j-1)w-1)$;
7. For $1 \leq i \leq w^2-w+1$, broadcast $SB(i)$ through B_i ;
8. For $w+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $mid1(i, j) := SO(i, j)$.

Step 8.

1. For $1 \leq j \leq w^2-1$, $SR(1, j) := t0(1, j)$;

2. For $2 \leq i, j \leq w$, $SR(i, (j-1)w) := \bigvee_{k=0}^{w-1} c(i, (j-1)w+k)$;
3. For $w+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $SR(i, j) := c(i, j)$;
4. $SB(1) := "00 \dots 01"$;
5. For $2 \leq i \leq w$, $SB(i) := SO(1, (i-1)w-1)$;
6. For $2 \leq i, j \leq w$, $SB((i-1)w-i+j+1) := SO(i, (j-1)w-1)$;
7. For $1 \leq i \leq w^2-w+1$, broadcast $SB(i)$ in $B(i)$;
8. For $1 \leq j \leq w^2-1$, $mid1(1, j) := SO(1, j)$;
9. For $w+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $mid2(i, j) := SO(i, j)$.

Step 9.

1. For $1 \leq j \leq w^2-1$, $SR(1, j) := t5(1, j)$;
2. For $2 \leq i \leq w$, $1 \leq j \leq w^2-1$, $SR(i, j) := t6(i, j)$;
3. For $w+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $SR(i, j) := c(i, j)$;
4. For $2 \leq i \leq w$, $SB(i) := SB2(i)$;
5. For $2 \leq i, j \leq w$, $SB((i-1)w-i+j+1) := SO(i, (j-1)w-1)$;
6. For $1 \leq i \leq w^2-w+1$, broadcast $SB(i)$ in $B(i)$;
7. For $2 \leq i \leq w$, $1 \leq j \leq w^2-1$, $mid1(i, j) := SO(i, j)$;
8. For $w+1 \leq i \leq w^2-w+1$, $1 \leq j \leq w^2-1$, $high(i, j) := SO(i, j)$.

Step 10.

1. For $2 \leq i \leq w$, $1 \leq j \leq w^2-1$, $SR(i, j) := c(i, j)$;
2. For $2 \leq i \leq w$, $SB(i) := SO(1, (i-1)w-1)$;
3. For $2 \leq i \leq w$, broadcast $SB(i)$ through $B(i)$;
4. For $2 \leq i \leq w$, $1 \leq j \leq w^2-1$, $mid2(i, j) := SO(i, j)$.

References

- [1] Y. Ben-Asher, D. Peleg, R. Ramaswam, and A. Schuster, The power of reconfiguration, *Journal of Parallel and Distributed Computing*, 13 (1991) 139–151.
- [2] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Transactions on Computers*, C-38, (1989), 1345–1351.
- [3] R. Lin, Reconfigurable buses with shift switching – VLSI radix sort, *Proc. of International Conference on Parallel Processing*, August 1992, III, 2–9.
- [4] R. Lin, and S. Olariu, Reconfigurable buses with shift switching – architectures and applications, *IEEE Transactions on Parallel and Distributed Systems*, 6, (1995), 93–102.
- [5] R. Lin and S. Olariu, A practical constant time sorting network, *Proc. ASAP'93*, October 1993, 380–391.
- [6] M. Maresca, Polymorphic processor arrays, *IEEE Transactions on Parallel and Distributed Systems*, 4, (1993), 490–506.
- [7] R. Miller, V. K. P. Kumar, D. Reisis, and Q. F. Stout, Parallel computations on reconfigurable meshes, *IEEE Transactions on Computers*, 42, (1993), 678–692.