Some Unusual Micropipeline Circuits[1]

Ganesh Gopalakrishnan

UUCS-93-015

Department of Computer Science
University of Utah
Salt Lake City, UT 84112, USA

Last Updated : December 11th, 1993.

Abstract

We present a few unusual Micropipelines (Sutherland, CACM, September 1989) that employ the Muller C-ELEMENT or an extension of the C-ELEMENT called LOCKC (Liebchen and Gopalakrishnan, ICCD, 1992). We first describe two variations of the two-dimensional Micropipeline structure realized using ordinary C-ELEMENTS. These micropipelines can be used to control *wavefront arrays* (S.-Y.Kung *et.al.*, IEEE Computer, 1987). Next, we present a ring style arbiter realized using a LOCKC-based one-dimensional micropipeline. Finally, we present a solution to the symmetric crossbar arbitration problem posed by Tamir and Chi (IEEE Trans. Parallel and Dist Systems, Jan '93) using a circuit that employs the two-dimensional micropipeline as well as the LOCKC. We present various circuits to solve the symmetric crossbar arbitration problem, including ones that consume very little power when idling.

[1]This work was supported in part by NSF Award MIP-9102558

Some Unusual Micropipeline Circuits

GANESH GOPALAKRISHNAN* (*ganesh@cs.utah.edu*)
University of Utah
Dept. of Computer Science
Salt Lake City, Utah 84112, USA

Keywords: Self-timed/Asynchronous Circuits, Micropipelines, Wavefront array processors, Arbiters, Lockable C-ELEMENT

Abstract. We present a few unusual Micropipelines (Sutherland, CACM, September 1989) that employ the Muller C-ELEMENT or an extension of the C-ELEMENT called LOCKC (Liebchen and Gopalakrishnan, ICCD, 1992). We first describe two variations of the two-dimensional Micropipeline structure realized using ordinary C-ELEMENTs. These micropipelines can be used to control *wavefront arrays* (S.-Y.Kung *et.al.*, IEEE Computer, 1987). Next, we present a ring style arbiter realized using a LOCKC-based one-dimensional micropipeline. Finally, we present a solution to the symmetric crossbar arbitration problem posed by Tamir and Chi (IEEE Trans. Parallel and Dist Systems, Jan '93) using a circuit that employs the two-dimensional micropipeline as well as the LOCKC. We present various circuits to solve the symmetric crossbar arbitration problem, including ones that consume very little power when idling.

1 Introduction

The micropipeline is a well known asynchronous control structure proposed by Sutherland in his Turing award lecture [1]. In this paper, we present an extension of micropipelines to higher (spatial) dimensions; specifically, we propose a few organizations of *two dimensional micropipelines.* In [1], Sutherland presents an analogy between one dimensional micropipelines and elastic media: both support the propagation of *elastic waves.* Likewise, two-dimensional micropipelines propagate waves that travel parallel to the diagonal.

Two-dimensional micropipelines are interesting for several reasons. In the past, several researchers have proposed rectangular circuit structures that support two-dimensional wave propagation. In [2, 3], S.-Y.Kung *et.al.* present several wavefront array processors. In [4], Tamir and Chi present several applications of wavefront propagation in rectangular array structures for performing *symmetric cross-bar arbitration* (detailed in Section 5.1). In this paper, we consider applications similar to those studied by S.-Y.Kung *et.al.*, and Tamir and Chi, except that we employ Sutherland's micropipeline structure (albeit with the two-dimensional extension) and a new circuit component (the LOCKC [5]) to implement our circuits.

Another novelty of our approach, compared to the approaches taken by S.-Y.Kung *et.al.*, and

*Supported in part by NSF under award MIP-9102558. The author wishes to thank Armin Liebchen for his help in developing LOCKC, Madhu Penugonda for discussions and his work in simulating a two-dimensional micropipeline, Ratan Nalumasu for verifying a few wavefront arbiters using the SMV system, and the ASIC seminar group for feedback.

Tamir and Chi, is that we employ asynchronous circuits in our proposed designs. As recent work shows [6, 7, 1, 8, 9, 10], asynchronous circuits often possess many advantages over synchronous circuits, including greater modularity, incremental expandability, absence of global clock distribution problems, and naturalness for applications involving phenomena such as pipelining, variable-rate computations, and arbitration. The last mentioned point is worth emphasizing. Implementing synchronously clocked arbiters that deal with purely asynchronous request signals is a difficult problem owing to the propensity of inviting failures due to metastability [11]. An additional feature of our work is that we employ *new asynchronous circuit design paradigms* such as two dimensional wavefront propagation and asynchronous locking of enabled C-ELEMENTS, which seem to lead to rather elegant and efficient circuits.

The remainder of this paper is organized as follows. In Section 2, we describe the two two-dimensional micropipeline architectures that we have identified, including some of their applications. In Section 3, we describe the design of the lockable C-ELEMENT. We also briefly describe the context in which LOCKC was originally conceived. In Section 4, we describe the design of an arbiter based on a (one dimensional) micropipeline that uses LOCKC in place of the standard Muller C-ELEMENT (on which most micropipeline designs are based). A way to avoid power wastage when idling is also discussed in this section. The idea behind this arbiter extends naturally to two dimensions, resulting in a symmetric crossbar arbiter based on a two dimensional micropipeline that uses LOCKC in place of the C-ELEMENT. Called the *wavefront arbiter* by Tamir and Chi [4], we describe this design in Section 5. Finally, in Section 6, we present another wavefront arbiter design called the *wrapped diagonal wavefront arbiter* by Tamir and Chi. In Section 7, we present a symmetric crossbar arbiter that achieves the maximum possible number of connections. Concluding remarks are provided in Section 8.

2 Two-dimensional Micropipelines

Figures 1 and 2 show the proposed two-dimensional micropipeline topologies. The former takes lesser area, but can pack only a small number of wavefronts. More specifically, assuming that all the C-ELEMENTs have the same delay, the wavefronts are separated by 3 positions, as the following analysis shows (a C-ELEMENT with index i, j is said to occupy position $i + j$):

- When the **Start** transition is applied, wavefront #0 occupying position 0 is created at time 0. This wavefront also travels through the MERGE element to apply another transition on the C-ELEMENT at position 0. However, since this C-ELEMENT has not been re-enabled through its inverting input yet, it doesn't fire as yet.

- Wavefront #0 at position 0 propagates to position 1 at time 1, and then to position 2 at time 2.

- When at position 2, wavefront #0 re-enables the C-ELEMENT at position 0, which fires at time 3. This creates wavefront #1 occupying position 0. In addition, wavefront #0 now occupies position 3. This sequence repeats[2].

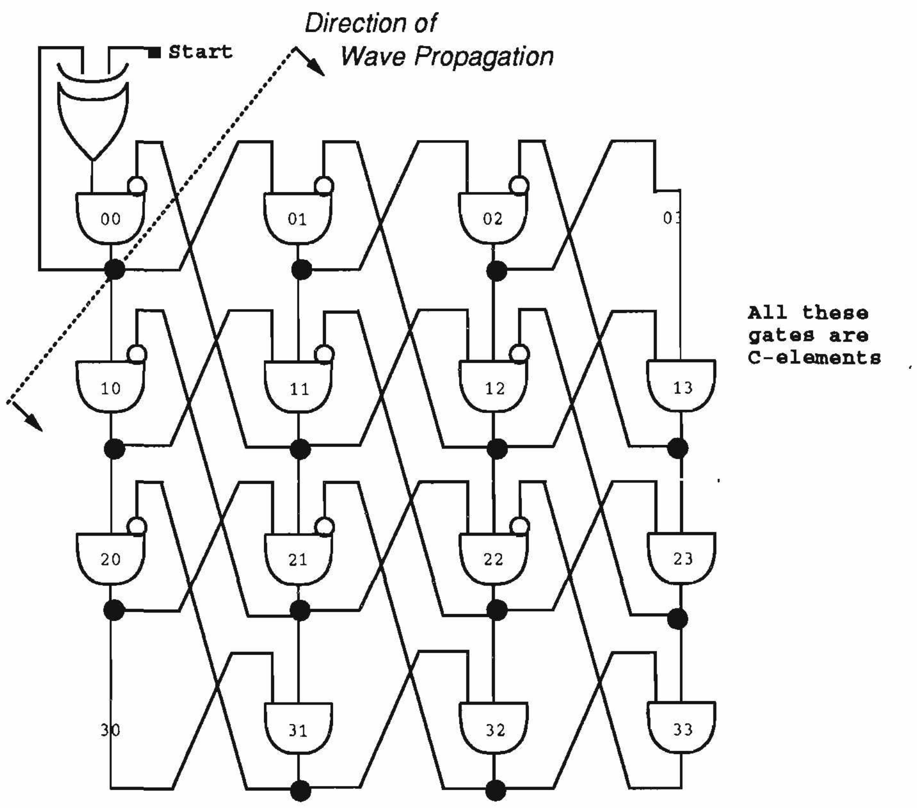


Figure 1: A Two-dimensional Micropipeline Organization

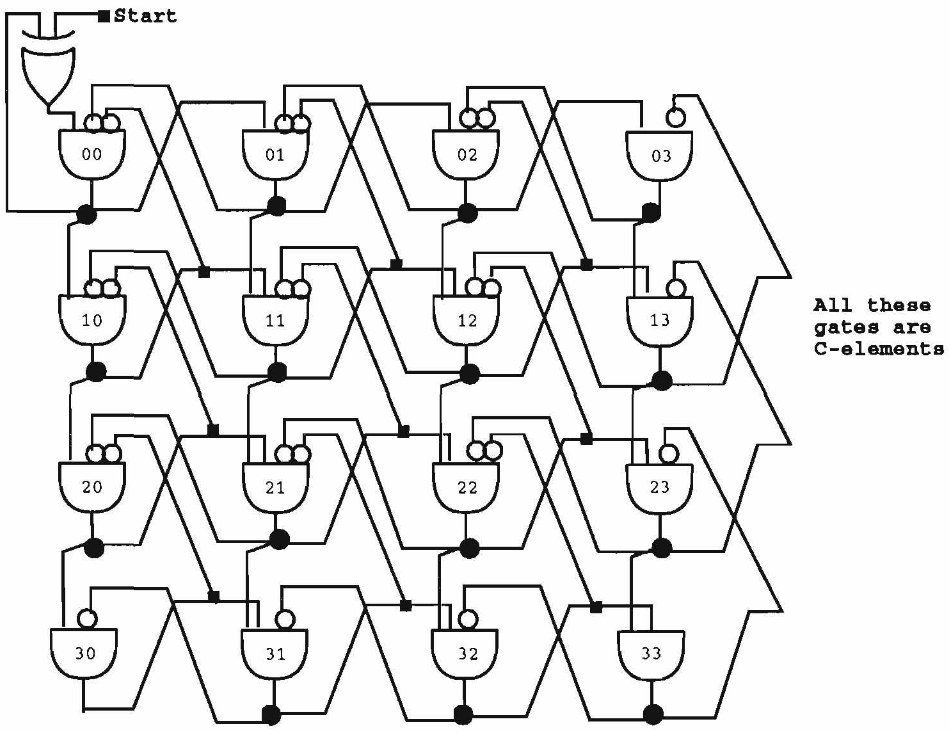


Figure 2: The Second Two-dimensional Micropipeline Organization

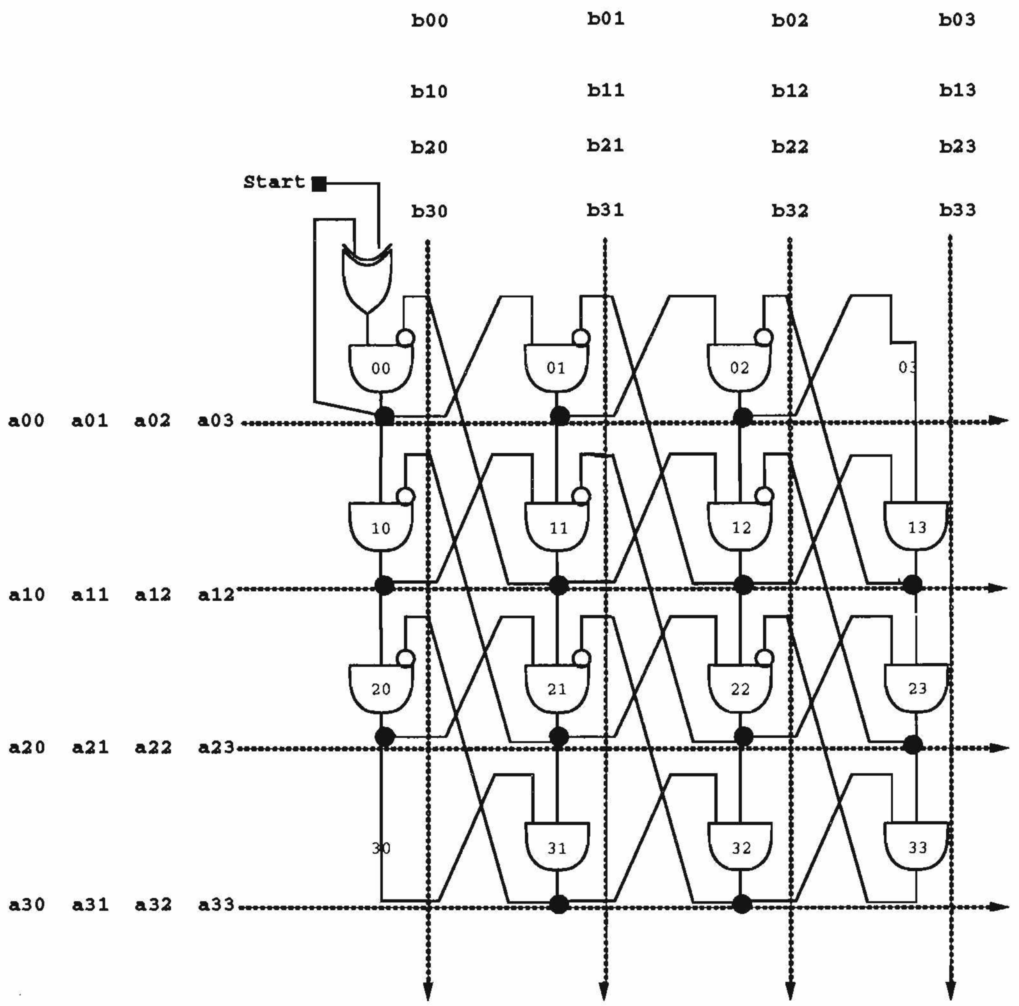


Figure 3: The Use of Wavefront Array Processors to Multiply Matrices

Figure 2 shows another two-dimensional micropipeline organization. This circuit takes more area because of the higher connectivity and higher in-degree of the C-ELEMENTS, compared to the circuit in Figure 1. However, this circuit can pack more wavefronts per C-ELEMENT; more specifically, assuming the same delay for the C-ELEMENTS, the wavefronts will be two positions apart. A noteworthy feature of this array is that examined row-wise or column-wise, this micropipeline resembles a standard (single dimensional) micropipeline.

Two-dimensional micropipelines propagate waves parallel to their diagonal, exactly as the wavefront arrays proposed by S.-Y.Kung *et.al.* do. Also, it is easy to see that the wavefronts do not intersect. Therefore, two-dimensional micropipelines can be used to control wavefront array processors. As an example, consider one of the most widely quoted of wavefront array algorithms: matrix multiplication. Let 4x4 matrices A and B be multiplied to yield matrix C, as illustrated in Figure 3. (Note: This figure uses the micropipeline organization of Figure 1 although that of Figure 2 will also work correctly.) In this figure, it is assumed that there is one datapath cell ij associated with each C-ELEMENT ij (which we do not draw, to avoid clutter).

The algorithm proposed by S.-Y.Kung *et.al.* works as follows. When a wavefront occupies position k, matrix elements situated nearest to cells 0k and k0 are brought into the datapath cells

[2]As S.-Y.Kung *et.al.* point out in [2], wave propagation in this array resembles light-wave propagation as described by Huygen's principle of light-wave propagation.

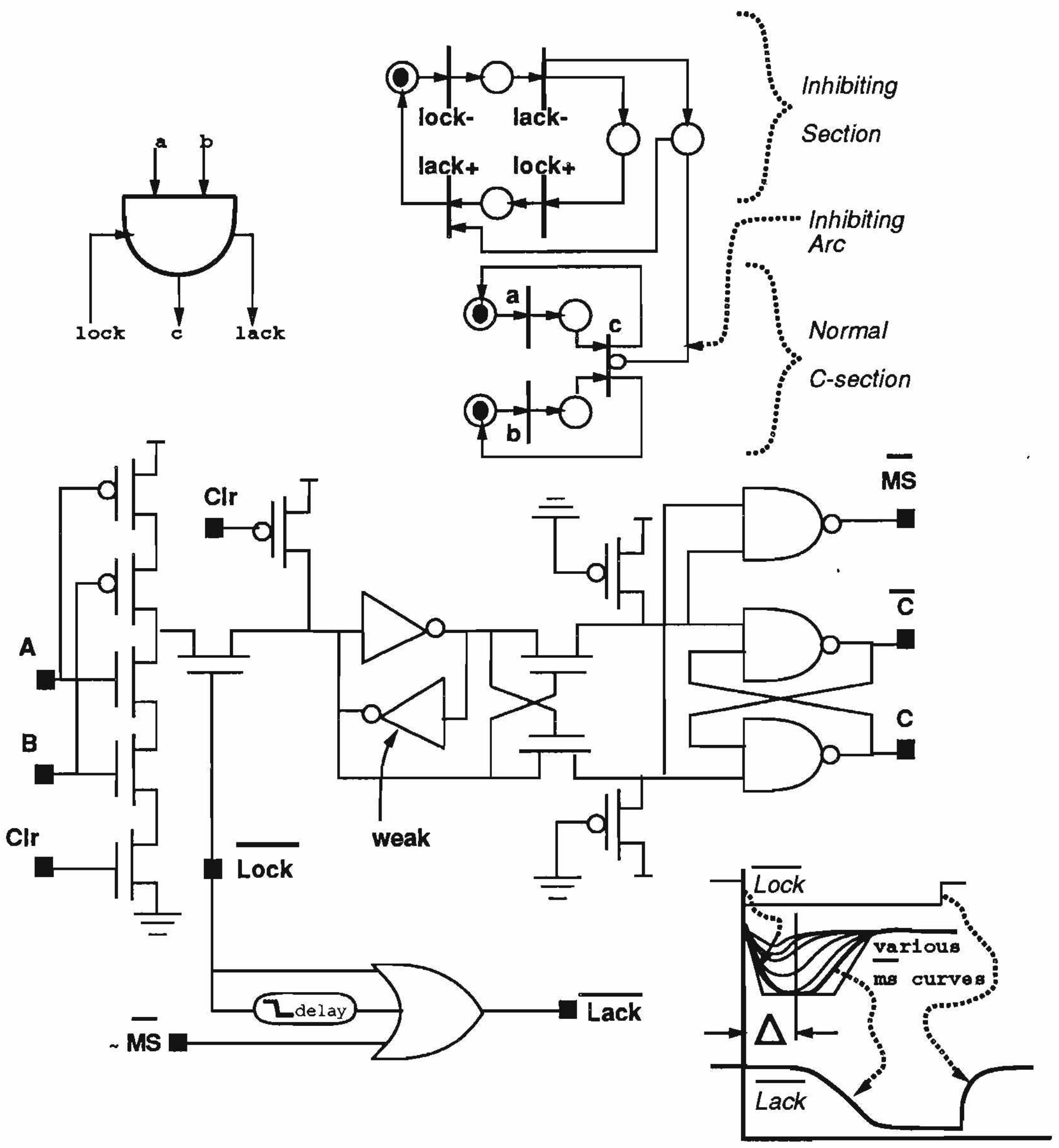


Figure 4: A Lockable C-Element, LockC

0k and k0. For example, in the beginning, when a wavefront occupies position 0, the elements b30 and a03 are brought into datapath cell 00. They are multiplied and the result is accumulated (with the accumulator starting at value 0). The next wavefront to occupy position 0 brings in b20 and a02 which are multiplied and added to the previous value accumulated at datapath cell 00. Due to the diagonal movement of wavefronts, when a wavefront advances, it carries all the a elements that are in the array rightwards one step; it also carries all the b elements that are in the array downwards one step. This ensures that elements are correctly brought together to be multiplied and accumulated during each step of the algorithm. The C matrix forms within the datapath cells, and can be brought out at the end.

3 LockC: A Lockable C-Element

In [5], Liebchen and I have described a new device that we have fabricated and tested: an extension of the Muller C-ELEMENT, called LOCKC. LOCKC works as follows. A *lock* command can be issued to it at an arbitrary point in time. This command has the effect of preventing the c output of LOCKC from firing, the "next time" LOCKC is enabled. In other words, if *lock* is asserted well before LOCKC gets enabled[3], locking succeeds. In this case, LOCKC remains enabled until *lock* is de-asserted, when it continues to behave like a normal C-ELEMENT. On the other hand, if *lock* is applied close to when LOCKC gets enabled, then (non-deterministically) LOCKC can "fire" (making $a = b = c$) or not fire, and remain enabled. In the former case (*i.e.*, when LOCKC actually fired, apparently ignoring the *lock* request), the *lock* request will be honored the next time LOCKC gets enabled.

As described in [5], LOCKC was invented to build *reordering micropipelines.* A reordering micropipeline buffers data in a FIFO fashion, and its contents can be periodically examined and subject to peephole optimization. For example, in one application, the reordering micropipeline is used to buffer instructions targeted to a processor. In this application, the flow of instructions through the micropipeline can be periodically halted, two arbitrary adjacent instructions can be examined, and the instructions can be reordered before letting the flow of instructions resume. In [5], we show how to build a Sutherland micropipeline using LOCKCs to realize a reordering micropipeline. In this paper, we explore newer applications of LOCKC. For the sake of completeness, we will now thoroughly describe the external as well as internal behavior of LOCKC. In Section 4, and later, we will describe various applications of LOCKC.

Figure 4 describes the operation of LOCKC. The pinout of LOCKC is shown on the top left-hand side. The Petri-net describes LOCKC's external behavior. This Petri-net has two sections, namely the "Normal C-section" and the "Inhibiting Section". We interpret this Petri net as follows. Each transition of the Petri net is labeled with a *signal-transition* of the form $x+$, $y-$, or z. The labels $x+$ and $y-$ indicate a rise of x or a fall of y respectively, and a label of the form z connotes a *change* of z's Boolean value. When a transition has tokens in all its enabling input places, and has no tokens in its inhibiting input places, the transition *fires.* The firing of a transition does the following things: it instantaneously removes one token from each of its enabling input places; then, after a bounded (but arbitrary) duration of time, it causes the associated signal-transition to occur, and simultaneously inserts one token into each of its output places.

The Normal C-section consists of two transitions a and b, whose firing eventually enables transition c. After c fires, transitions a and b are eventually re-enabled. The Inhibiting Section receives *lock*$-$ from the user. After *lack*$-$ is generated, it is guaranteed that transition c cannot occur; it can occur only after *lack*$+$ is generated.

The circuit realization of LOCKC shown here is different from that described in [5], and is, therefore, explained now. The input section consists of the normal pull-up and pull-down transistors of a two-input C-ELEMENT. A single N-type transistor connected to $\overline{Lock}$ disconnects the cross-coupled inverter pair from the input section. (We employ a single N-type transistor, as opposed to a transmission gate, to ensure sharper cut-off; ratioing is done to mitigate the effects of the threshold drop.) The inverter pair stores the internal state of LOCKC. When the inverter pair goes metastable[4], the interlock transistors cutoff; the output state is, meanwhile, held steady by the

[3]A C-ELEMENT is enabled when $a = b = \neg c$.

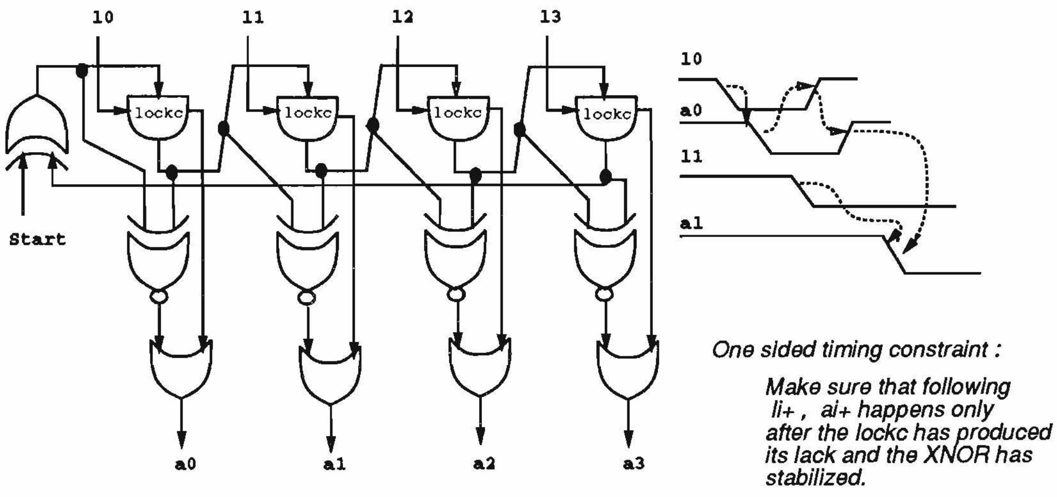

10
11
12
13
lockc
lockc
lockc
lockc
Start
a0
a1
a2
a3
10
a0
11
a1
One sided timing constraint :
Make sure that following li+ , ai+ happens only after the lockc has produced its lack and the XNOR has stabilized.

Figure 5: An Arbiter Realized Using LOCKC

$\overline{R}\,\overline{S}$ flip-flop. We derive signal $\overline{lack}$ from $\overline{ms}$. However, as signal $\overline{ms}$ can exhibit a non-monotonic change (as shown by the traces), we employ an OR-gate and a delay element (whose value is greater than Δ, the maximum duration for which $\overline{ms}$ could ever have negative slope) to mask-off the down-going portion of $\overline{ms}$. When LOCKC is unlocked by de-asserting *lock*, the OR-gate forces *lack* to be de-asserted after the OR-gate delay. Since DELAY is asymmetric (effective only for down-going transitions), it also resets immediately. Thus, LOCKC is available for use again, very soon after *lack* is de-asserted.

4 An Arbiter Based on LOCKC

Figure 5 shows how an arbiter can be built using a (special case of a one-dimensional) Sutherland micropipeline that employs one-input LOCKC elements. After power-up, the output state of all the LOCKC elements is reset to zero, and their *lock* inputs are held de-asserted (high). The Start transition is then applied. This causes a *single* token to circulate in the loop. Since we only need *one* token to circulate, we should not re-enable C-ELEMENT $i-1$, as soon as C-ELEMENT i fires (as in a standard micropipeline). Therefore, we should have only one input to our LOCKCs. (Note: One input LOCKCs are the same as Q-LATCHES, described in [12].)

Without loss of generality, consider a situation in which station 0 wants to access the shared resource guarded by the arbiter. It first asserts *l0*, and waits for *a0* to be asserted (to go low). Signal *a0* is lowered when LOCKC asserts (lowers) the *lack* signal, and the parity between the input and the output of the LOCKC gate is odd. The attainment of odd parity between the LOCKC input and output indicates that LOCKC has successfully trapped the circulating token. Since the output of LOCKC cannot change after it asserts *lack*, the odd parity will be maintained (*i.e.*, it cannot be a transient "glitch"). Therefore, as soon as station 0 sees *a0* getting asserted, it can access the shared resource. Once the access of the shared resource is over, *lock* is de-asserted, causing *lack* also to be de-asserted.

Now consider the case when there is even-parity between the LOCKC input and output even after *lack* has been asserted. This situation can arise only when the token is at a station numbered greater than 0. In this case, after all the stations with number higher than 0 have released the token, the token is guaranteed to come to station 0. When this happens, odd parity between the

[4]This can happen if, for example, when *lock* is asserted when the inverter pairs are half-way along in their state change.

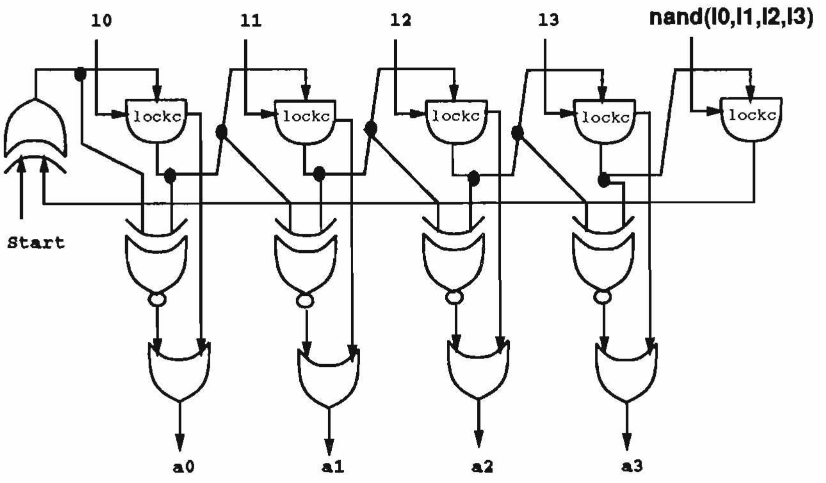


Figure 6: A Low Power Version of the One Dimensional Ring-Style Arbiter

LockC input and output will be attained, causing *a0* to be asserted. Thus, each station follows a four-phase protocol with respect to *l0* and *a0* (with *l0* and *a0* being active low). An example usage of this arbiter appears in Figure 5.

Note that there is a one-sided timing constraint on raising l_i, for $i \in 0 \ldots 3$: after raising l_i, we must ensure that the *lack* signal of the corresponding LockC has been generated and the XNOR gate has stabilized before a_i is raised. This is because as soon as a_i is raised, l_i can be asserted (lowered) once again.

A low-power version of the one-dimensional arbiter given in Figure 5 is presented in Figure 6. This version prevents the token from idling around the loop when none of the lock inputs are asserted. In this sense, this circuit is similar to Martin's token ring arbiter [13], although there are many differences between these circuits in terms of implementation. Since the token is prevented from idling around the loop, power consumption is reduced. This idea can be suitably incorporated into other arbiters discussed in this paper. The extra LockC acts as a place-holder for the token when no lock requests are active.

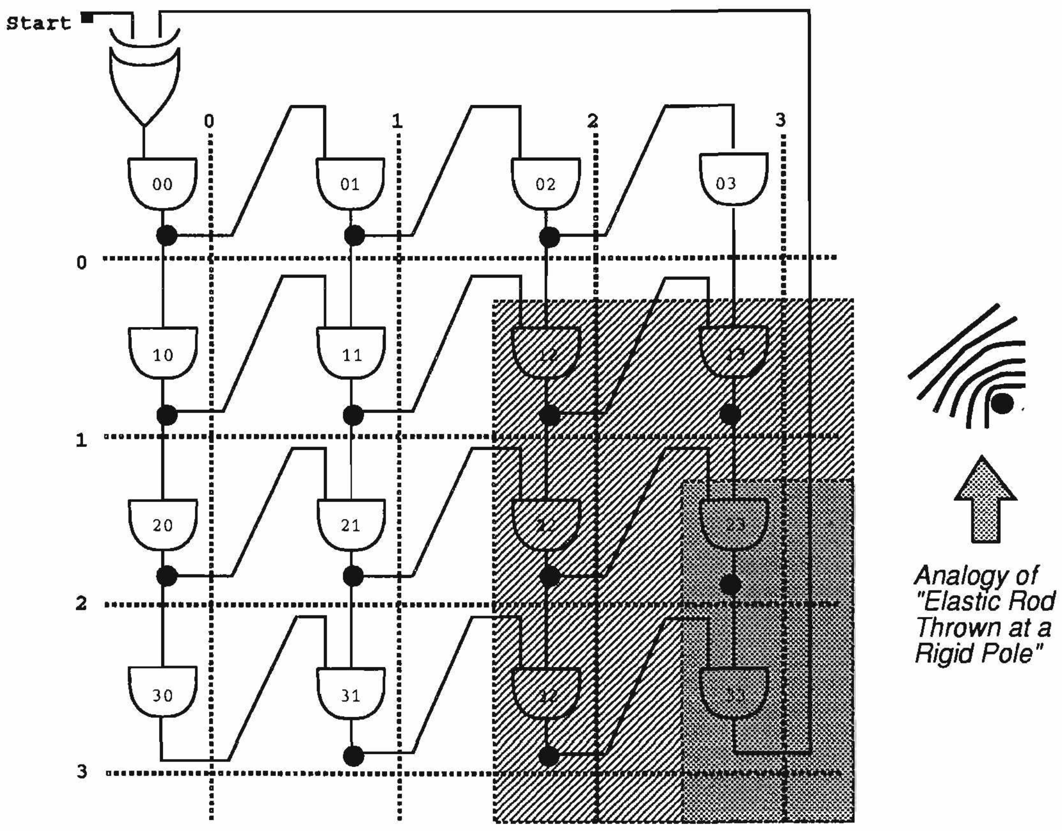
5 The Wavefront Arbiter

Figure 7 shows a square array of LockC elements that implements a solution for the *symmetric crossbar arbitration* problem. This is a direct extension of the arbiter in Figure 5 to two dimensions. (We do not show the XOR-gate and the OR-gate associated with each LockC to avoid clutter.) In Section 5.1, we specify the symmetric crossbar arbitration problem. In Section 5.2, we explain the workings of the wavefront arbiter.

5.1 Symmetric Crossbar Arbitration: Problem Definition

Consider a crossbar switch that can connect row wires numbered 0 through 3 to column wires numbered 0 to 3, where the row and column wires are illustrated in Figure 7. Each row wire may be connected to at most one column wire. Similarly, each column wire may be connected to at most one row wire.

At any instant in time, each row-wire may simultaneously request to be connected to several column wires; the symmetric crossbar arbiter has to decide *which* column wire to connect the row



All gates labeled "xy" above are lockc elements

▨ Shadow cast by locking element 12

▒ This circuit cannot make connections in this region while element 12 is locked. However the crisscrossing 1-d arbiters circuit given later can make such connections.

Figure 7: The Wavefront Arbiter

wire to. Since all the different rows may be making similar requests concurrently, it must also be guaranteed that if a row wire is connected to a certain column, then none of the other rows are connected to the same column. For example, if the set of requests made by row wire 0 is the set of column wires {0,2}, that of row wire 1 is the set of column wires {0,1}, that of row wire 2 is {1}, and that of row wire 3 is {2,3}, then one set of legal connections consists of row wire 0 to column 0; row wire 1 to column 1; row wire 2 to *no* column wire (as column 1 is already taken); and row wire 3 to column wire 3. A better (in terms of utilization) set of connections is: row wire 0 to column 2; row wire 1 to column 0; row wire 2 to column 1; and row wire 3 to column wire 3. Since there are many switches arbitrating for exclusive access to row- and column-wires, we must avoid possible deadlocks due to "hold and wait".

A formal definition of the symmetric crossbar arbitration problem for an $N \times N$ switch matrix is as follows (with our solutions as well as that of Tamir and Chi only trying to approximately solve the problem). Let $i_k, k \in N$ denote the requests on row wire k. (Numbers are viewed as sets; e.g. $4 = \{0,1,2,3\}$.) Each i_k is of the form $0^{j_{k,0}}1^{j_{k,1}}2^{j_{k,2}}\ldots(N-1)^{j_{k,N-1}}$ for $k \in N$ and $j_{k,l} \geq 0$ for

$l \in N$. For example, $0^2 1^3 2^0 3^1$ indicates 2 requests for column-wire 0, 3 for column-wire 1, none for column-wire 2, and 1 for column-wire 3. The problem is to find k_x, $x \in N$, $k_x \in N \cup \{-1\}$, where (k_x is the column wire assigned to row x—with -1 indicating "none assigned") such that

$$k_x \neq -1 \Rightarrow j_{x,k_x} > 0, \; for \; x \in N \tag{1}$$
$$(x \neq y) \wedge (k_x = k_y) \Rightarrow (k_x = k_y = -1) \tag{2}$$
$$\sum_{x \in N} Count(k_x) \; is \; maximal. \tag{3}$$

Equation 1 says that we assign a column wire only if that column is requested. Equation 2 says that we do not allocate the same column wire to more than one row wire. Equation 3 uses a suitable definition of $Count(x)$ (*for example if* $x \geq 0$ *then* 1 *else* x), and tries to maximize the number of connections made.

5.2 Operation of the Wavefront Arbiter

The solutions proposed by Tamir and Chi rely on the classic idea of using a total ordering on resource numbers. We can think of the *switches* as resources and the wires as processes contenting for the switches. Then, the switch *positions* can be used for the ordering. Using this idea, we can examine the switches in the order in which they are visited by diagonal wavefronts flowing from the top-left corner to the bottom-right corner. This leads us to the design of *wavefront arbiters.*

The operation of the wavefront arbiter presented in Figure 7 is as follows. Assuming that all the LockCs are initially unlocked, a `Start` transition ripples through the array, initially creating a wavefront at position 0, which then moves on to position 1, 2, and so on. When it reaches position 6, the LockC at that position repeats the above sequence of wave propagation. Thus, at any time, there is exactly one wavefront combing through the array diagonally.

Now, assume that station 12 wants to arbitrate (get access to row-wire 1, and column wire 2). It asserts the *lock* input of LockC number 12, and awaits the assertion of *lack*, as well as the attainment of a state in which the *a* and *b* inputs are different from the *c* output. When this happens, we can be sure that the wavefront has been successfully blocked by LockC number 12. When the OR-gate associated with this LockC asserts its *l12* signal, crossbar switch 12 can connect row-wire 1, and column wire 2. After the message has been routed, station 12 will de-assert *lock*, causing the de-assertion of *l12* as well. The wavefront can now proceed beyond station 12, triggering the C-ELEMENTs 13 and 22 on its course. Consider the situation in which the wavefront is at position 3, and is blocked only by one node—namely 12—at position 3. The wavefront can still continue to move forwards. However, it cannot enter the "shadow" cast by element 12 (the shadow includes all switches with row numbers ≥ 1 and column numbers ≥ 2) [5].

The wavefront arbiter prevents a row or a column from being connected multiple times, for the following reasons. Assume that the wavefront is successfully blocked by LockC xy. Then, it cannot be passing through switches xa for $a < y$, and by for $b < x$, for, it has already departed from these

[5]To take a physical analogy, the wavefront behaves like "an elastic rod thrown lengthwise at a rigid pole", as shown in Figure 7.

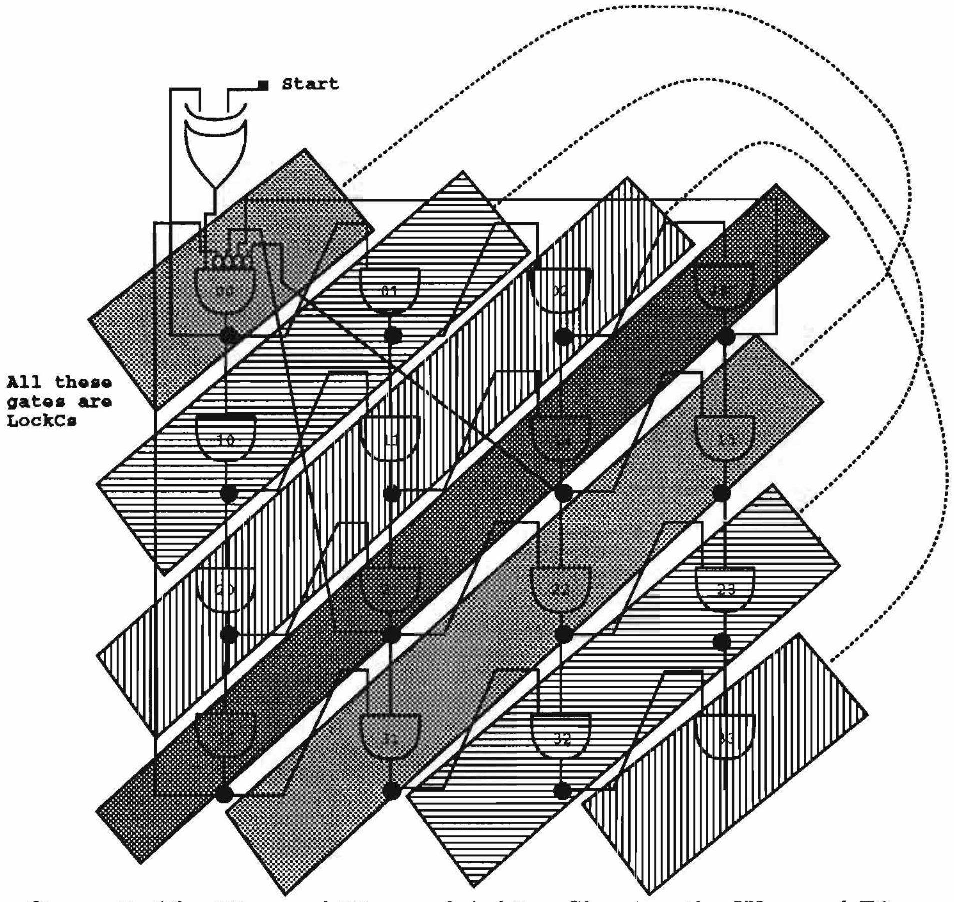


Figure 8: The Wrapped Diagonal Arbiter Showing the Wrapped Diagonals

locations before coming to location xy. It cannot be passing through the switches in the shadow cast by xy (all switches with row number $\geq x$ and column number $\geq y$), for those LockC elements can fire only after the LockC at xy has fired. Thus, row-wire x and the column-wire y cannot be having other connections made on them. If a LockC other than xy is also locked, the situation can only get better in terms of avoiding deadlocks as well as conflicting wire usages. The scheme avoids deadlocks because a row and a column wire passing through a switch are allocated together, and never piece-meal.

Actually the above solution is an overkill: it is perfectly safe to allow connections to be made in the *umbra* of the shadow cast by xy (switches whose row numbers are $> x$ and column numbers are $> y$—see Figure 7), because the corresponding rows and columns are free. In Section 7, we present a circuit that achieves this.

6 The Wrapped-diagonal Arbiter

The scheme in Figure 7 is inefficient because only one diagonal (wavefront) is supported. As Tamir and Chi show, it is possible to have two diagonals, with a given minimum spacing constraint, such that arbitration decisions are allowed to be made on points situated on both the diagonals. Figure 8 shows the various "wrapped diagonals". Initially, there is a wrapped diagonal occupying position 0. In the next time step, this wrapped diagonal moves forwards to occupy position 1. At the next time step, it occupies position 2, and then position 3. When this wavefront leaves position 3, it moves to position 4, and also enables a wavefront to get formed at position 0. Thus,

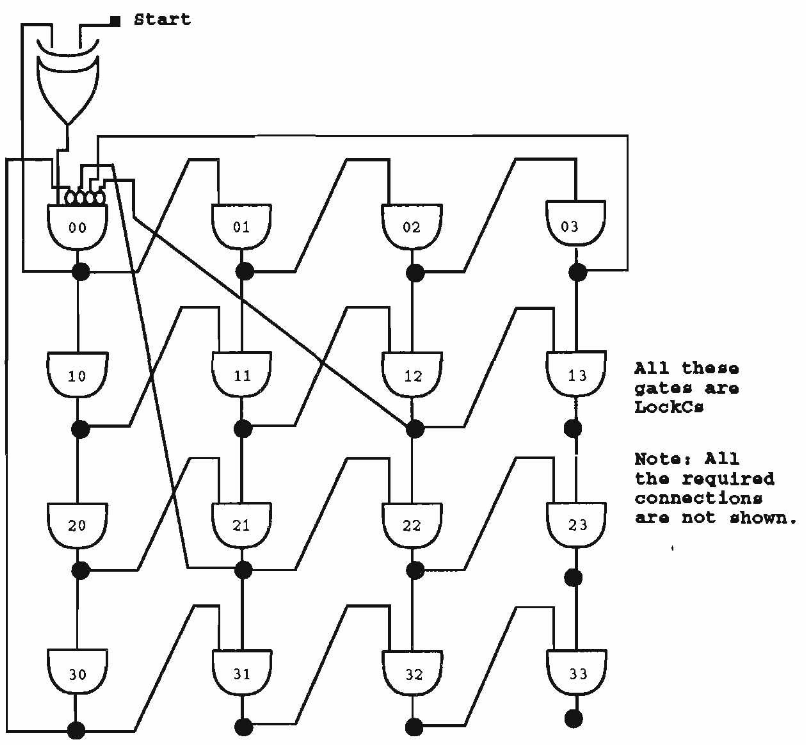

Start
00
01
02
03
10
11
12
13
20
21
22
23
30
31
32
33
All these gates are LockCs
Note: All the required connections are not shown.

Figure 9: Wrapped Diagonal Arbiter: Achieving Spacing Between Diagonals

a wrapped diagonal has formed, with its ends occupying positions 0 and 4. Moving along, the ends of the wrapped diagonal occupies positions 1 and 5, and then 2 and 6. The next wrapped diagonal occupies only position 3. After it leaves position 3, the above sequence repeats, with the ends of a wrapped diagonal at positions 0 and 4, and so on. Each wrapped diagonal passes through the largest possible number of switches that do not have any row- or column wires in common [6]. Thus, they capture the largest number of conflict-free concurrent connections possible.

The required spacing between the wrapped diagonals is achieved by arranging the re-enabling input for the LockCs to come from those LockCs that are "sufficiently far away", as shown in Figure 9. (Not all connections are shown, to avoid clutter.) The connections shown in this figure imply that the LockC at 00 is prevented from firing until after all the LockCs at position 3 have fired. This prevents the next wavefront at position 0 from being formed until the wavefront at position 3 is about to move on to position 4. This allows positions 0 and 4 to get one wavefront, each, which they can choose to "trap" using the *locks* of the associated LockCs. Similar connections are necessary from position 4's outputs going to position 1's inputs, and position 5's outputs to position 2's inputs.

Actually, it is not necessary to connect every output of position 3 to an input of position 0, and so on. Rather, a small, regular set of such connections suffice, as shown in Figure 10. The connections in this figure can be derived by the following line of reasoning. Clearly, the wavefront at position 3 must have vacated positions 30 and 03 before a wavefront is permitted at position 00 (to avoid conflicting on column 0 or row 0, respectively). However, positions 21 and 12 need not be vacated before allowing a wavefront onto position 00. Before letting a wavefront into positions 10 and 01, however, the wavefront must have vacated positions 21, 31, 12, and 13. It actually suffices

[6]By the "pigeon-hole principle".

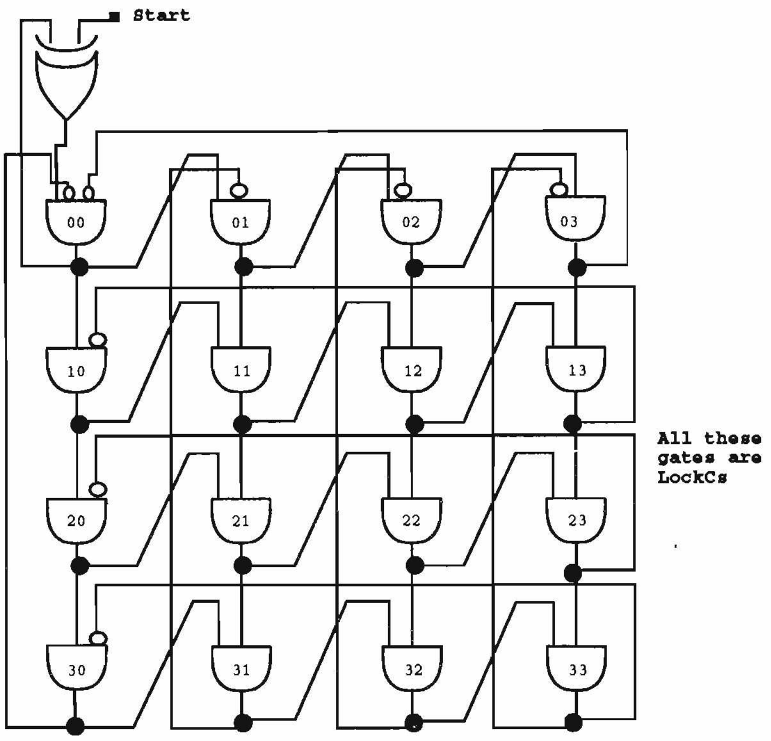


Figure 10: Wrapped Diagonal Arbiter: Least Constraint Version

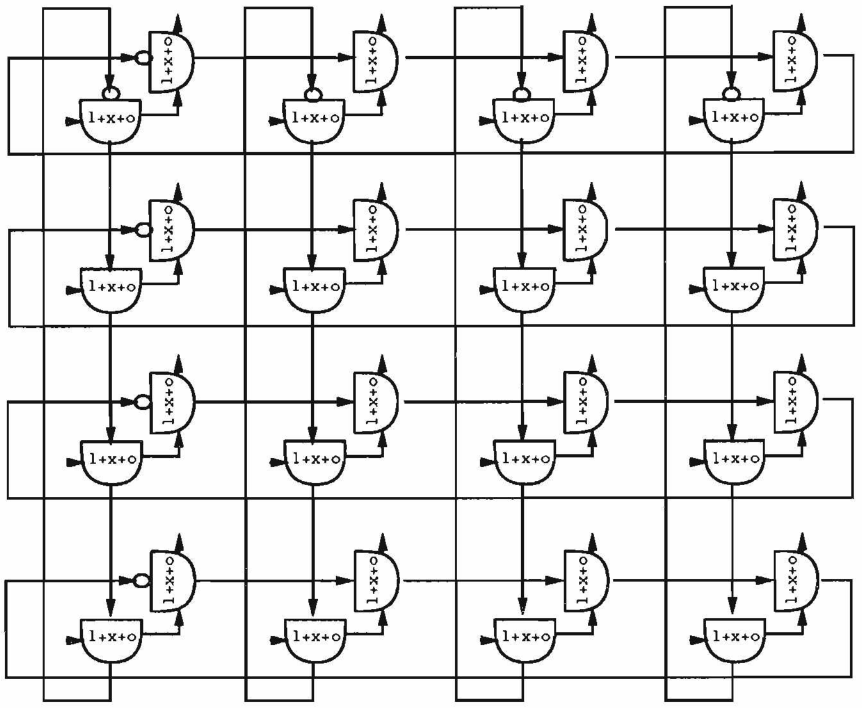
to check whether the wavefront has vacated positions 31 and 13 alone, because the firing of the cells 31 and 13 is an indirect acknowledgement of the fact that the cells 21 and 12, respectively, have also fired.

By avoiding the barrier synchronizations implicit in the wrapped diagonal scheme suggested by Figure 9, the scheme in Figure 10 can achieve higher performance.

7 Crisscrossing One Dimensional Arbiters

Figure 11 presents a symmetric crossbar arbiter that can achieve the maximum number of non-conflicting connections as defined by Equations 1 through 3 of Section 5.1. This arbiter essentially consists of N one-dimensional arbiters of the kind described in Figure 5 arranged row-wise and the same number of one-dimensional arbiters arranged column-wise in a crisscrossing fashion (thus it takes more area than the circuit in Figure 10). The basic building block, called L+X+O, consists of a LockC element, an XNOR gate and an OR gate, similar to a stage of Figure 5. Connecting the two L+X+O cells at any coordinate position in the manner shown in Figure 11 makes sure that at any coordinate position, the column-token is trapped before the row-token is trapped. This is to prevent deadlocks.

The crisscrossing one-dimensional arbiter array works as follows. When a lock is requested at location (i, j), the ith row first traps the single token circulating around the ith row at location (i, j). Thereafter, the jth column traps the single token circulating around the jth column at location (i, j). When both these steps are over, the external acknowledge signal is generated.



l+x+o consists of a one-input lockc, the XNOR gate that detects that a token has been trapped, and the OR gate that generates the acknowledge.

The input to l+x+o is the lock signal and the output is the ack signal.

Figure 11: Crisscrossing One Dimensional Arbiters

8 Concluding Remarks

A new architectural idea (two-dimensional micropipelines) as well as new applications for a recently proposed circuit element (LockC) were presented. The circuits described in this paper are unusual in several ways. The idea of two-dimensional micropipelines is believed to be new. The computational paradigm underlying our arbiter circuits—locking moving wavefronts in a two-dimensional micropipeline—is also believed to be new.

The circuits proposed by S.-Y.Kung *et.al.* in [3] are synchronous in nature. They are conceptually mismatched with the *computational paradigm* put forwards by S.-Y.Kung *et.al.* (which is that of asynchronous dataflow computation!). Also, their circuits require the use of synchronizers that allow sufficient time for metastable states to exit, entailing a loss of performance. In contrast, our proposal to use two-dimensional micropipelines to control wavefront arrays appears conceptually simpler, as well as elegantly matches the computational paradigm followed by wavefront arrays. Although two-dimensional micropipelines seem to have a larger spread between two successive wavefronts than necessary, they could have shorter cycle-times, as the critical path includes only a few LockCs.

Compared to Tamir and Chi's circuits, our circuits do not need an extra shift-register stage or a clock distribution network. In Tamir and Chi's scheme, there is a tacit assumption that outstanding requests for switch connections are examined at the beginning of every clock cycle. Since these requests can come truly asynchronously with the clock in a message routing network (where these arbiters find application), designers are required to adopt suitable metastability handling techniques—an issue that Tamir and Chi do not address. The use of synchronizers that allow sufficient time for the metastable states to exit entails a loss of performance. In contrast, our circuits operate truly asynchronously, and metastability handling is inherent in the design of the

LockC elements. Since LockC elements delay the generation of *lack* only so long as their internal metastability lasts, only rarely will *lack* be noticeably delayed, as the probability of the arrival of the wavefront coinciding with the assertion of *lock* is low. Our circuits also guarantee fairness: every request will eventually be honored, as the wavefronts, by their nature, take off from where they were trapped due to locking.

As discussed in Section 4, if the arbiter discussed in this paper are equipped with an extra stage of LockCs, the idling token(s) can be trapped there when the arbiter is not making connections, thus saving power.

At present, we have a working MOSIS chip containing LockC. We also have simulated the two-dimensional micropipeline array successfully. The wrapped diagonal arbiter, least constraint version (Figure 10) has been modeled in Computational Tree Logic using the SMV language [14] and verified for its correctness. We plan to build test-chips for the remaining circuits shortly.

References

1. Ivan Sutherland. Micropipelines. *Communications of the ACM*, June 1989. *The 1988 ACM Turing Award Lecture.*
2. S.Y. Kung, R.J. Gal-Ezer, and K.S.Arun. Wavefront array processor: Architecture, language, and applications. In *Proceedings, Conference on Advanced Research in VLSI, MIT*, pages 4–19. Artech House Inc., 1982.
3. S.Y. Kung, S.C. Lo, S.N. Jean, and J.N.Hwang. Wavefront array processor—concept to implementation. *IEEE Computer*, 20(7):18–35, July 1987.
4. Yuval Tamir and Hsin-Chou Chi. Symmetric crossbar arbitration. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):13–27, January 1993.
5. Armin Liebchen and Ganesh Gopalakrishnan. Dynamic reordering of high latency transactions in time-warp simulation using a modified micropipeline. In *International Conference on Computer Design (ICCD)*, pages 336–340, 1992.
6. John Brzozowski and Carl-Johan Seger. Advances in Asynchronous Circuit Theory: Part I: Gate and Unbounded Intertial Delay Models; and Part II: Bounded Intertial Delay Models, MOS Circuits, Design Techniques. Technical report, University of Waterloo, 1990.
7. Ganesh Gopalakrishnan and Prabhat Jain. Some recent asynchronous system design methodologies. Technical Report UUCS-TR-90-016, Dept. of Computer Science, University of Utah, Salt Lake City, UT 84112, 1990.
8. C. A. Mead and L. Conway. *An Introduction to VLSI Systems.* Addison Wesley, 1980. *Chapter 7, entitled "System Timing".*
9. *Articles in the Minitrack "Asynchronous and Self-Timed Circuits and Systems," of the Computer Architecture Track of the 26th Hawaiian International Conference on System Sciences, January, 1993. (Minitrack Organizers: Ganesh Gopalakrishnan and Erik Brunvand.).*
10. *Workshop on "Asynchronous and Self-Timed Systems," University of Manchester, March 1993. (Organizer: Steve Furber).*

11. T.J.Chaney and C.E.Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, April 1973.

12. Erik Brunvand. Parts-r-us. *a chip aparts(s)....* Technical Report CMU-CS-87-119, Carnegie Mellon University, May 1987.

13. A. Martin. Distributed Mutual Exclusion on a Ring of Processes. *Science of Computer Programming*, 5:265–276, 1985.

14. Kenneth L. McMillan. *Symbolic Model Checking.* Kluwer Academic Press, 1993.