On Seitz' Arbiter

Alain J Martin

Computer Science Department
California Institute of Technology

5212:TR:86

On Seitz's Arbiter

Alain J. Martin
Department of Computer Science
California Institute of Technology
Pasadena CA 91125

1. Introduction

In [5], C.L. Seitz proposes a self-timed circuit, called an "arbiter", with the following specification. Two independent processes compete for mutually exclusive access to a shared resource—e.g., a store. The arbiter selects a request from one of the two processes and, before granting the resource to the selected process, activates—i.e. communicates by a four-phase handshaking protocol with—a so-called "transfer module" associated with the process. The transfer module prepares parameters for the shared resource which is activated after the transfer module activity has terminated.

All communications—i.e. communications between the arbiter on the one hand and the processes, or the transfer modules, or the shared resource on the other hand— are implemented by four-phase handshaking protocols. The arbiter is self-timed, i.e. no assumptions are made about the propagation time in wires and the gate delays (see [4]). However we will assume that some wires inside the arbiter are equipotential (isochronic), i.e. the wires are short enough to assume that propagation is instantaneous along those wires.

In [1], Gregor Bochmann shows that the original solution proposed by Seitz is wrong, and proposes a modified version. In [2], D.L.Dill and E.M.Clarke apply their circuit verification method to Bochmann's circuit and discover an error in it. They also propose a modified version. Due to the successive modifications to the original circuit, their solution is unnecessarily complicated. Applying the synthesis method described in [3], we have been able to derive a simple and correct solution. The solution is different from, and simpler than, all correct solutions the author knows of.

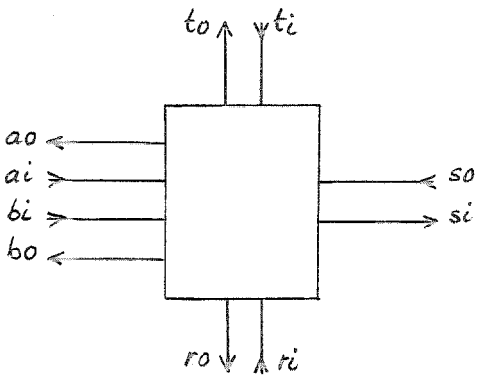
The purpose of this small note is to present this new solution directly and informally. It is left as an exercise to the interested reader to derive it rigorously applying the method of [3].

2. The Solution

The arbiter communicates with processes A and B, transfer modules T and R (associated with A and B respectively), and shared resource S. Each communication uses two directed

wires. A directed wire is an operator with a boolean input and a boolean output: the wire $(x \text{ w } y)$ has input x and output y.

For $X \in (A, B, T, R, S)$, the arbiter communicates with X by the input xi of a wire and the output xo of another wire (see Fig. 1).



–Figure 1–

Further, for boolean variable x, $x\uparrow$ means $x := \textbf{true}$ and $x\downarrow$ means $x := \textbf{false}$.

From the specification of the problem, if process A is granted the resource the following sequence of actions occurs:

$$\begin{aligned} &ai\uparrow; to\uparrow; ti\uparrow; so\uparrow; si\uparrow; ao\uparrow; \\ &ai\downarrow; to\downarrow; ti\downarrow; so\downarrow; si\downarrow; ao\downarrow. \end{aligned} \tag{1}$$

And similarly if process B is granted the resource:

$$\begin{aligned} &bi\uparrow; ro\uparrow; ri\uparrow; so\uparrow; si\uparrow; bo\uparrow; \\ &bi\downarrow; ro\downarrow; ri\downarrow; so\downarrow; si\downarrow; bo\downarrow. \end{aligned} \tag{2}$$

Sequence (1) is enforced by the arbiter if it implements the set of commands:

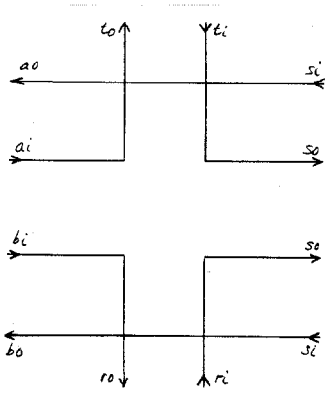
$$\begin{aligned} ai &\rightarrow to\uparrow \\ ti &\rightarrow so\uparrow \\ si &\rightarrow ao\uparrow \\ \neg ai &\rightarrow to\downarrow \\ \neg ti &\rightarrow so\downarrow \\ \neg si &\rightarrow ao\downarrow. \end{aligned} \tag{3}$$

(A command of the form $C \rightarrow D$ means that when C holds, D is executed. "The arbiter implements the set of commands (3)" means that the activity of the arbiter consists of repeatedly executing the commands of (3).) Observe that the commands of (3) are not ordered. The ordering corresponding to (1) is enforced by the oredering of actions in the four-phase handshaking protocol used by A, T, and S.

The six commands of (3) are obviously implemented by the three wires $(ai \text{ w } to)$, $(ti \text{ w } so)$, and $(si \text{ w } ao)$. Similarly, sequence (2) is enforced by the arbiter if it implements the set of commands:

$$
\begin{aligned}
bi &\rightarrow ro\uparrow \\
ri &\rightarrow so\uparrow \\
si &\rightarrow bo\uparrow \\
\neg bi &\rightarrow ro\downarrow \\
\neg ri &\rightarrow so\downarrow \\
\neg si &\rightarrow bo\downarrow .
\end{aligned}
\tag{4}
$$

The commands of (4) are implemented by the three wires (bi **w** ro), (ri **w** so), and (si **w** bo). (See Fig.2.)

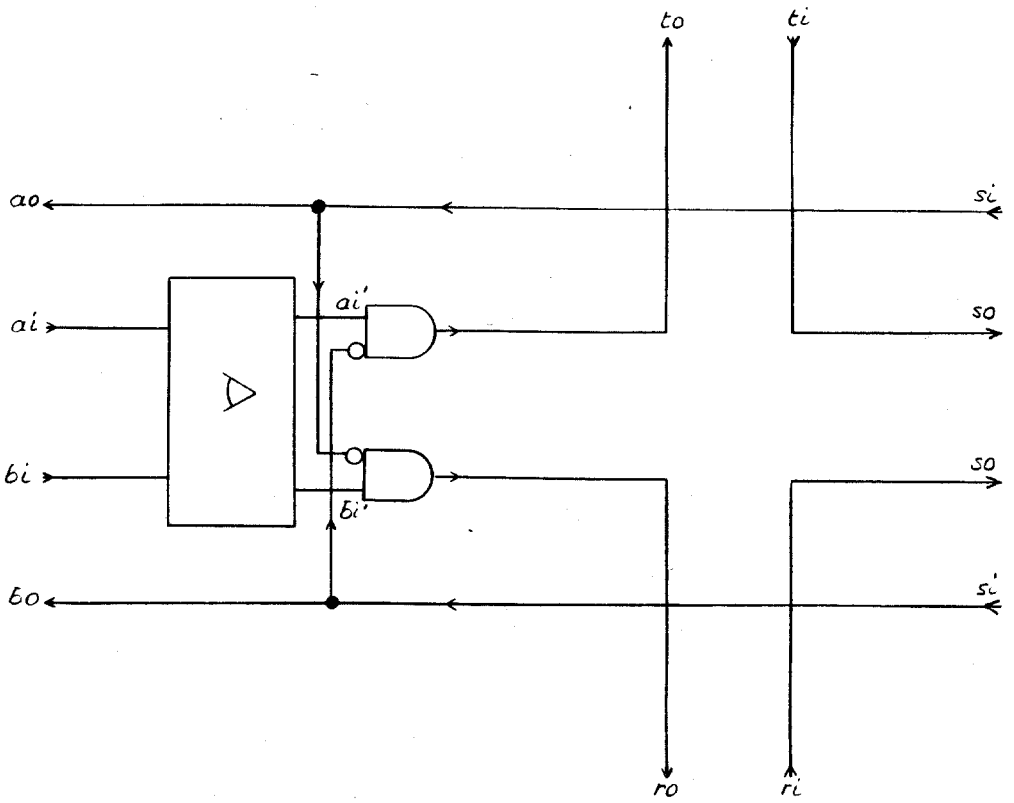


–Figure 2–

Now, the composition of (3) and (4) inside the arbiter has to enforce mutual exclusion between the two sequences. As in all other solutions, we introduce the mutual exclusion element ((ai, bi) **me** (ai', bi')) where ai and bi are inputs and the new variables ai' and bi' are outputs. The specification of **me** is the set of commands:

$$
\begin{aligned}
ai \wedge \neg bi' &\rightarrow ai'\uparrow \\
bi \wedge \neg ai' &\rightarrow bi'\uparrow \\
ai' \wedge \neg ai &\rightarrow ai'\downarrow \\
bi' \wedge \neg bi &\rightarrow bi'\downarrow .
\end{aligned}
\tag{5}
$$

We replace ai by ai' in (3), and bi by bi' in (4). The resulting sets of commands are called (3') and (4') respectively. But observe that as soon as $\neg ai'$ holds in (3'), the mutual exclusion element can execute the second command of (5), which could cause the sequence (2) to start before the sequence (1) is completed. (And the same holds for $\neg bi'$ in (4').) These possible interferences are avoided if we replace ai' by $ai' \wedge \neg bo$ and $\neg ai'$ by $\neg ai' \wedge bo$ in the first and fourth command of (3') respectively. And similarly for (4'). The verification that the new (3') and (4') still enforce the sequences (1) and (2) is left to the reader. The implementation of the arbiter so far is shown on Fig. 3.



–Figure 3–

Further, since so and si are duplicated in (3') and (4'), we replace so by so' in (3') and by so'' in (4'), and si by si' in (3') and si'' in (4'). And we add the two commands

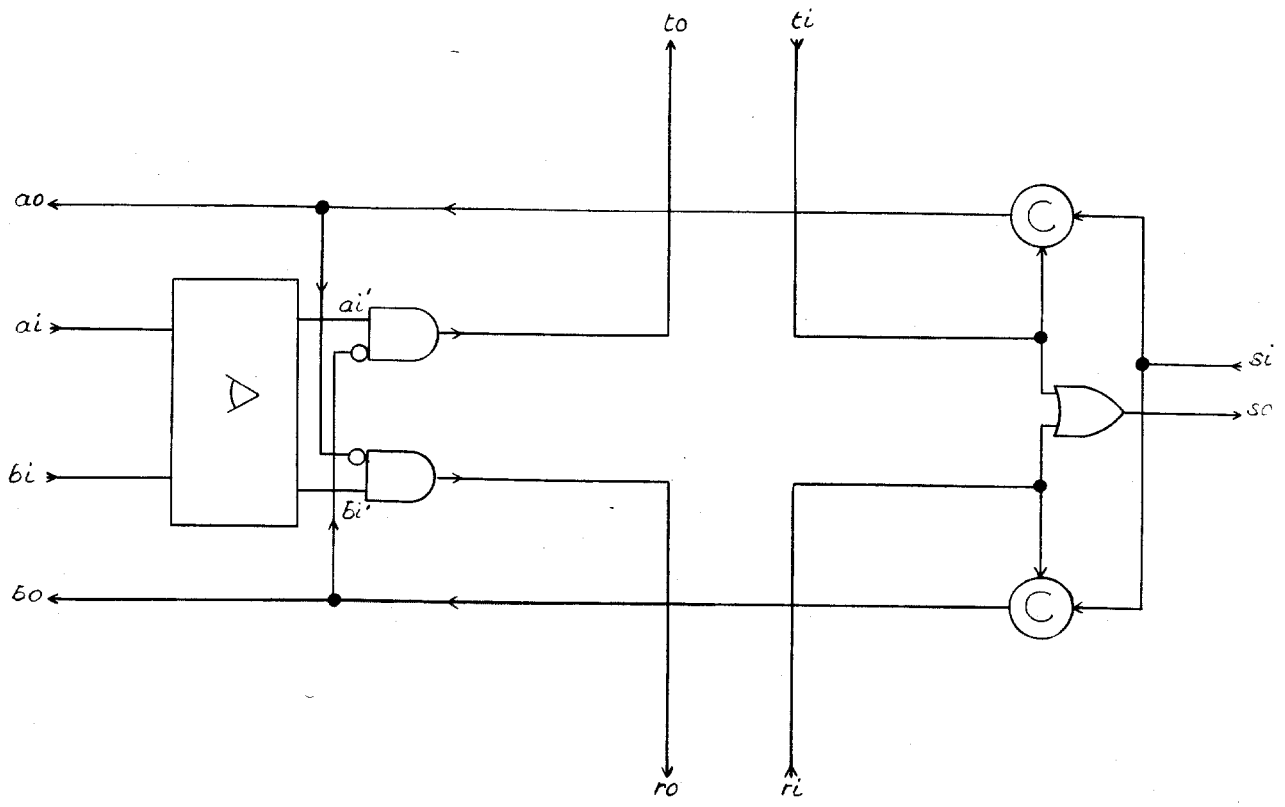
$$so' \vee so'' \rightarrow so\uparrow$$
$$\neg so' \wedge \neg so'' \rightarrow so\downarrow,$$

which are implemented by the *or*-operator $((so', so'')$ **or** $so)$.

The transition $si\uparrow$ should cause only one of the two transitions $si'\uparrow$ or $si''\uparrow$ depending on whether (1) or (2) is being executed, which depends on whether so' or so'' holds. Hence we add the two C-elements $((si, so')$ **C** $si')$ and $((si, so'')$ **C** $si'')$. (The C-element $((x, y)$ **C** $z)$ is specified by the two transitions:

$$x \wedge y \rightarrow z\uparrow$$
$$\neg x \wedge \neg y \rightarrow z\downarrow.$$

Which gives the final circuit of Fig. 4.



–Figure 4–

3. References

[1] G. V. Bochmann, "Hardware Specification with Temporal Logic: An Example", *IEEE Trans. Comput.*, vol. C-31, March 1982.

[2] D. L. Dill and E. M. Clarke, "Automatic Verification of Asynchronous Circuits Using Temporal Logic", in *Proc. 1985 Chapel Hill Conf. VLSI.* Ed. Henry Fuchs, pp. 127-143, March 1985.

[3] A. J. Martin, "The Design of a Self-timed Circuit for Distributed Mutual Exclusion", in *Proc. 1985 Chapel Hill Conf. VLSI.* Ed. Henry Fuchs, pp. 247-260, March 1985.

[4] C. L. Seitz, "System Timing", Chapter 7 in Mead & Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA (1980).

[5] C. L. Seitz, "Ideas About Arbiters", *LAMBDA*, First Quarter, 1980.

June 1985.