Studying the three-slot asynchronous communication mechanism

F. Xia*
Department of Computing Science, University of Newcastle upon Tyne
I.G. Clark
Department of Electronic Engineering, King's College London

Indexing terms: asynchronous data communications, concurrent systems.

1 Abstract

The 3-slot asynchronous communication mechanism (ACM) has existed for a long time, but has always been regarded as an inferior solution, especially to the 4-slot ACM, primarily because of questions about its behaviour when individual control variable statements are regarded as non-atomic. With the publication of ACM solutions using mutex elements to deal with metastability, however, any statement protected by a mutex can be regarded as atomic. In such ACM designs it may turn out to be advantageous to use a 3-slot algorithm instead of a 4-slot one with potentially significant savings in silicon. Here the 3-slot method is studied in detail, using a particular algorithm as an example, with certain atomic assumptions consistent with mutex protection of statements. The results confirm promising potential for 3-slot ACMs.

2 Introduction

Data communication between processes running in different processors has been extensively studied, especially in distributed and real-time systems. The minimal form of this problem concerns the passing of data between two distributed single-thread processes. One provides the data, which is used by the other. This is schematically shown in Figure 1.

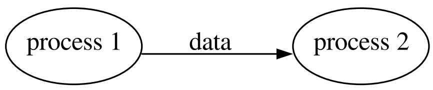

Figure 1 Passing data between two processes.

In distributed systems, processors may not always share a common clock. When the two processes in Figure 1 are not synchronised, some kind of intermediate data repository, usually in the form of shared memory, is often needed between them to facilitate the data passage. This is schematically shown in Figure 2.

Figure 2 Passing data via shared memory.

An asynchronous data communication mechanism (ACM) is a scheme which manages the transfer of data between two processes not necessarily synchronised for the purpose of data transfer. It is assumed that the data being passed consists of a stream of individual items of a given type. It is also assumed that the processes in question are single thread cycles, one providing an item of data during each cycle, the other making use of an item of data during each cycle. The provider of data is known as the "writer" of the ACM and the user of data is known as the "reader" of the ACM.

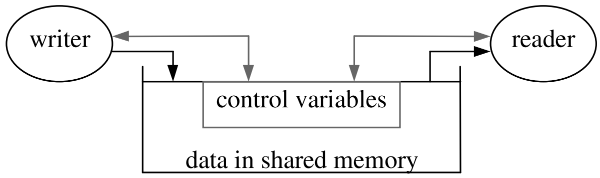

Figure 3 ACM using shared memory and control variables.

Such techniques as the multiple "slot" (or "track") mechanisms described in [1], [2] and [3] realise regular and atomic registers in the data path between asynchronous concurrent processes by employing "safe" bit registers to convey the values of bit-size control variables. They avoid conflicts on data memory

* Contact: fei.xia@ncl.ac.uk

without resorting to explicit critical sections. The general scheme of these data communication mechanisms is shown in Figure 3.

The number of data slots, or shared memory areas for individual items of data, is significant to the most important ACM properties. In order to provide full asynchronism, more than two data slots are needed so that the reader and writer do not access the same slot at the same time. When simultaneous reading and writing occurs on the same slot, it is likely that corrupt data will be passed on. This is known as the *data coherence* problem.

The other important property of such ACMs is data freshness, which describe how up to date any item of data obtained by the reader is, relative to the latest item of data provided by the writer. Data freshness is defined in detail in [4].

It has been shown that various 4-slot ACMs are capable of maintaining data coherence and data freshness in fully asynchronous operation. 3-slot designs, however, have not been as intensively studied, mostly because they have been shown to fail data coherence requirements in subtle ways [1]. Specifically, if certain control variable statements are not regarded as atomic, simultaneous reading and writing of the same slot may occur.

More recently, hardware ACM designs using self-timed circuits have been published [5] where mutex elements are used to deal with the issue of metastable control variables. A side effect of this use of mutex elements is that the relevant control variable statements are protected and may now be regarded as atomic.

A state-space based study of a 3-slot ACM, with atomic assumptions on crucial control variable statements, is presented here. The results of this study confirms that this 3-slot ACM does maintain data coherence and data freshness under these assumptions of atomicity.

3 The 3-slot ACM algorithm

A simple 3-slot ACM is found in [6]. This algorithm is studied here

Writer	Reader
wr: $d[n] := input$	$r0$: $r := l$
$w0$: $l := n$	rd: $output := d[r]$
$w1$: $n := \neg(l \vee r)$	

Figure 4 A 3-slot mechanism.

Figure 4 shows the algorithm of the 3-slot mechanism. Here the writer and reader processes are single thread loops with three and two statements each respectively. The mechanism maintains the storage of three data slots $d[1]$ to $d[3]$ and the control variables n, l, and r, which are ternary signals. This is shown schematically in Figure 5. The statements *wr* and *rd* are the data accesses and the other (control variable) statements are used by the writer and reader to choose slots and indicate such choices to the other side.

Statement $w1$ means that the control variable n is assigned a value different from the current values of both l and r. In practice, the following method can be used to deal with this efficiently:

```
const differ = ((2,3,2),(3,3,1),(2,2,1))
… …
n := differ[l, r]
… …
```

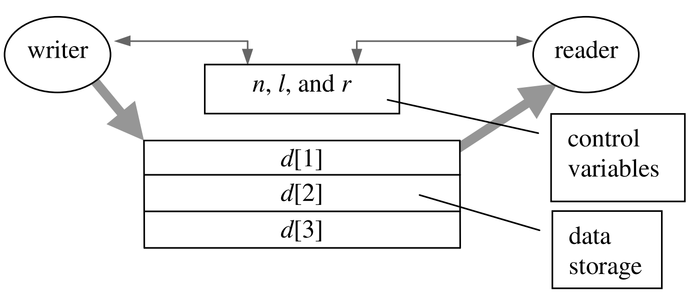

Figure 5 Schematic of the 3-slot mechanism.

4 Modelling and analysis of the 3-slot ACM

We have investigated the 3-slot ACM with the same method employed in our previous studies of the 4-slot ACM [7, 8]. This method involves the establishment of Petri net models of the ACM and reachability analyses based on these models. Properties such as data coherence and data freshness are properly reflected in the models so that the results of state space searches describe system behaviour in terms of these properties.

Petri net models of the 3-slot ACM given in Figure 4 has been established. They are similar to the models of the 4-slot ACM in [7] and [8]. As an illustration, the Petri net model of statement $w1$ is shown in Figure 6.

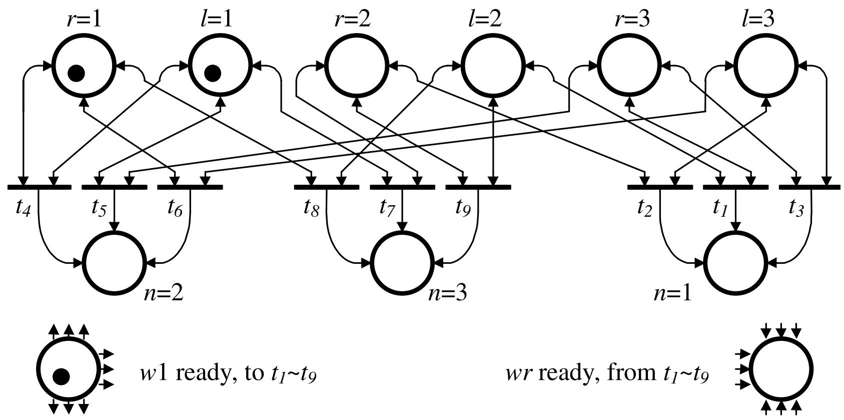

Figure 6 Petri net model of $w1$: $n := \neg(l \vee r)$.

Significantly, the control variable statement $r0$ has been modelled both at atomic and non-atomic. The atomic model is similar to the model in Figure 6, in that any one execution of the statement involves the firing of one transition, which in classical P/T net semantics cannot be interleaved by any other transitions firing. The non-atomic model has two layers of transitions, one representing the start of the statement and the other its finish. Thus one execution of the statement involves the firing of two transitions, in between which other transitions elsewhere in the model may fire so long as they are enabled.

The non-atomic model is thus over cautious in that the state space contains such unlikely situations as the execution of the entire writer cycle any finite number of times when the reader is carrying out the statement $r0$.

Reachability analyses of these models have turned up the following results:

1. The 3-slot ACM fails both data coherence and data freshness when statement $r0$ is considered non-atomic, but only when at least a consecutive pair of statements $w0$ and $w1$ are executed between the start and finish of $r0$.
2. The 3-slot ACM maintains both data coherence and data freshness (both the weaker and stronger definitions of data freshness [4]) when all control variable statements are considered non-atomic.

These results show that if we can assume atomicity for all control variable statements, at least with regard to statements on the other side, the 3-slot ACM maintains both data coherence and data freshness.

5 Comparative robustness of the 3- and 4-slot ACMs

Certain 4-slot ACMs have been shown to maintain both data coherence and data freshness even when a control variable assumes a wrong value once in a single writer or reader cycle [7]. This means that such 4-slot ACMs can effectively utilise the redundancy in data slots to increase robustness with regard to control variable errors.

Similar analyses have been carried out for the 3-slot ACM in this study and the results show that this 3-slot ACM produces a single data coherence violation and a single data freshness violation when a control variable assumes a wrong value once during a single writer or reader cycle.

The 3-slot ACM is therefore less robust with regard to control variable errors than some of the 4-slot ACMs, simply because it does not have enough redundancy in data slots to avoid failures in such circumstances. However, similar to the 4-slot ACMs, this 3-slot ACM is also self-correcting when an error occurs in a control variable, so that data coherence and data freshness failures are non-persistent.

6 Conclusion and acknowledgement

A 3-slot ACM, in a simple form easy to adapt for hardware designs, has been shown to behave reasonably well with regard to data coherence and data freshness, under certain assumptions of atomicity of statements. Since atomicity of statements can be protected by the mutex elements used in self-timed hardware designs to deal with metastability in control variables. The 3-slot ACM might be useful in such implementations where it may help save significant amounts of silicon by have one fewer data slot than the more robust 4-slot solutions.

Intuitively, with one fewer slot, once data coherence is assured, the data freshness behaviour should be quantitatively better. However, this conjecture has not been investigated in this study.

This work is part of the Comfort project at the University of Newcastle supported by the EPSRC grant GR/L93775 and the Comfort project at King's College London supported by the EPSRC grant GR/L92471.

References

1 Simpson, H.R., "Four-slot fully asynchronous communication mechanism", IEE Procs., Vol. 137, Pt. E, No. 1, pp.17-30, January 1990.

2 Tromp, J., "How to construct an atomic variable", Proc. 3rd Int. Workshop on Distributed Algorithms, Nice, LNCS, Springer Verlag, pp.292-302, 1989.

3 Kirousis, L.M., "Atomic multireader register", Proc. 2nd Int. Workshop on Distributed Computing, Amsterdam, LNCS-312, pp.278-296, Springer Verlag, 1987.

4 Xia, F., Yakovlev, A., and Clark, I.G., "Testing the data freshness properties of asynchronous communication mechanisms", elsewhere in this Forum.

5 Xia, F., Yakovlev, A., Shang, D., Bystrov, A, Koelmans A., and Kinniment D., "An Asynchronous communication mechanism using self-timed circuits", Technical Report, TR-686, Department of Computing Science, University of Newcastle upon Tyne, 1999.

6 Simpson, H.R., "Correctness analysis of class of asynchronous communication mechanisms", IEE Proceedings, Vol. 139, Pt. E, No. 1, pp.35-49, January 1992.

7 Xia, F, Clark, I.G. and Davies, A.C., "Petri-net based investigation of synchronisation-free interprocess communication in shared-memory real-time systems", Proc. 2nd UK Asynchronous Forum, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, July 1-2, 1997.

8 Xia, F. and Clark, I.G., "Complementing role models with Petri nets in studying asynchronous data communications", 19th International Conference on Application and Theory of Petri Nets, Hardware Design and Petri Nets Workshop, pp.66-85, June 23, 1998, Lisbon, Portugal.