# Petri-net based investigation of synchronisation-free interprocess communication in shared-memory real-time systems

Fei Xia* Ian G. Clark † Anthony C. Davies

Division of Engineering
Kings College London, University of London
Strand, London, WC2R 2LS, England.

## 1 Introduction

For real-time multi-processor systems, the use of shared memory for interprocess communication is often a preferred choice. In many cases the communication requires process-synchronisation at the moment of data-transfer. This is, for example, implied in the ADA rendezvous in which the 'entry' and 'accept' statements enforce a wait by the faster of the two processes needing to communicate. In safety-critical real-time control-systems, such timing constraints are not always acceptable, since failures or delays in one part of a system can hold up or stall other processes - in a manner which does not arise in analogue continuous-time systems.

For synchronisation-free communication we are restricted to a 'destructive write non-destructive read' form of communication. This has the inevitable side effect that the data being passed may be duplicated or lost as a result of writing and reading process temporal independence. It is also due to this temporal independence that process overlap may occur, possibly resulting in corrupted data. Therefore to avoid this loss of data coherence some form of mutual exclusion must be implemented, even with dual-port shared memory. If mutual exclusion is performed while using a single shared memory data location (called a 'slot' [1, 2]), process waiting must be introduced. However, process waiting violates the synchronisation free paradigm.

By duplicating or triplicating the shared memory locations (slots) and adding appropriate steering logic (via process internal and shared control variables) to prevent overlapped writing to and reading from the same slot, while at the same time ensuring that the reader always obtains the freshest data, it is possible to reduce the probability that the writing and reading processes will attempt to access the same slot simultaneously. However, Simpson [1, 2] has shown that only with four such slots is it possible to provide complete temporal freedom to both writer and reader.

The analysis of such mechanisms to ensure that they perform as required under all relative timing conditions/circumstances is complex because of the very large number of states and possible state-transitions. Although a thorough manual analysis supported by computer simulations and enumeration of alternative conditions has shown the correctness of the four-slot mechanism [1], it continues to present some analytical difficulties and an extension to more general cases (e.g. multiple writers and/or mulitple readers) is restricted by this complexity.

This paper presents the results of the analysis of the synchronisation-free interprocess-communications mechanisms using Petri-net based methods [3]. This provides an automated and more straightforward alternative verification of the validity of the previous analyses by Simpson, offers an easier foundation for the investigation of the more general cases, and extensions to investigate the effects of metastability on the mechanism.

This paper is intended to be read in conjunction with [1] and [2] which provide more detailed background information on the mechanisms including discussion of their practical and theoretical significance.

## 2 Modelling of the mechanisms

### 2.1 Modelling considerations

Data coherence and freshness are essential issues which any model must reflect clearly and unambiguously. Data coherence as defined in [1] can be paraphrased as follows: The reader is obtaining correct data so long as the writing and reading operations do not overlap in time on a single shared memory location. If such an overlap occurs data coherence is lost. Data freshness as defined in [1] is more complicated when access processes with multiple steps are concerned. The reader is considered to be obtaining fresh data so long as it is reading from a shared memory location containing data which was either the freshest or the previous freshest at the beginning of the reading sequence, or else newer data, which has arrived after the beginning of the reading sequence. This definition excludes re-reading of data older than the last read.

Both properties require careful modelling of each step of the reading and writing actions including unambiguous representations of the beginning and end of both the direct writing and reading access to the slots.

*Now with Department of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, England.
†contact author, email: ian.clark@kcl.ac.uk

### 2.2 The one-slot mechanism

The one-slot mechanism [1] defines a single shared memory location or slot (*data*) for data in transit between the writing and reading sides, together with two access processes (*read* and *write*). The initialisation of slot *data* to the value *null* is to ensure data coherence (reader always obtains valid data) should the reader start reading before the first write.

Because of the concerns listed in Section 2.1 the model of the writer must incorporate a cyclic action of "*writing*, end writing, $\overline{writing}$, start writing" (the notation $\overline{writing}$ is used to represent the state of 'not writing'), while that of the reader must incorporate a cyclic action of "*reading*, end reading, $\overline{reading}$, start reading". The appropriate approach must take into account that the actions of *writing*, *reading*, $\overline{writing}$ and $\overline{reading}$ take finite but undefined time while the starting and ending points of these actions can be viewed as atomic. If the states (*writing*, $\overline{writing}$, *reading*, $\overline{reading}$) are modelled with Petri net places and the transitions from one state to another (the beginning and end of reading and writing) are modelled with Petri net transitions a marking will show the current state of the system and by studying the markings correct and incorrect operating modes can be discovered. With this objective a Petri net model for the one-slot mechanism was developed (Figure 1). The Petri-net is cyclic, so all possible states are reachable from any legal initial condition. The initial condition was arbitrarily set as $\overline{writing}$ and $\overline{reading}$. Once the net starts operating it is clear that the reading side and the writing side are temporally independent of each other as the transitions on one side of the net may fire in any possible interleaving sequence with respect to the transitions of the other side, reflecting the synchronisation free nature of the mechanism. An extension to the one slot is the two slot and the three slot mechanisms [1], these introduce control/shared variables to steer the writer/reader to the desired slot.

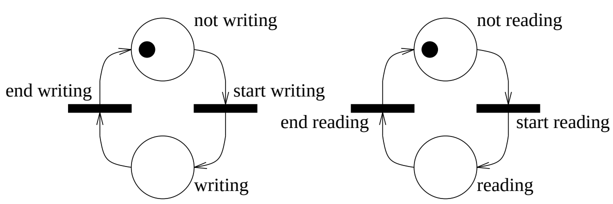


Figure 1: One-slot Petri net model.

### 2.3 The four-slot mechanism

The basic algorithm of the four-slot mechanism may be given in the same format as those of the other mechanisms [1]. A version for hardware implementation given in [1] is shown in Figure 2. As they are algorithmically equivalent [1] and the effect of metastability, which relates only to a hardware implementation, is of particular interest, the modelling and analysis is restricted to the latter in this paper. A full model of the former and its analysis can be found in [4].

Petri-net places are used to represent the state of the mechanism and its variables. The places are used in pairs

```
ENTITY four_slot IS
  PORT(input_data: IN integer; output_data: OUT integer;
       write_clock, read_clock: IN BIT);
END four_slot;

ARCHITECTURE behavioural OF four_slot IS
  TYPE four_slot_data IS ARRAY (0 TO 1, 0 TO 1) OF INTEGER;
  SIGNAL d: four_slot_data := ((0,0),(0,0));
  SIGNAL s: BIT_VECTOR(0 TO 1);
  SIGNAL l, r: BIT;

BEGIN
  write_process: PROCESS(write_clock)
  VARIABLE n: BIT := '0';
  BEGIN
    IF n = '0' AND s(0) = '1' THEN d(0,0) <= input_data; END IF;
    IF n = '0' AND s(0) = '0' THEN d(0,1) <= input_data; END IF;
    IF n = '1' AND s(1) = '1' THEN d(1,0) <= input_data; END IF;
    IF n = '1' AND s(1) = '0' THEN d(1,1) <= input_data; END IF;
    s(0) <= s(0) XOR (NOT n); s(1) <= s(1) XOR n;
    l <= n;
    n := NOT r;
  END PROCESS;

  read_process: PROCESS(read_clock)
  VARIABLE r: BIT;
  VARIABLE v: BIT_VECTOR(0 TO 1);
  BEGIN
    r := l;
    v := s;
    IF r = '0' AND v(0) = '0' THEN output_data <= d(0,0); END IF;
    IF r = '0' AND v(0) = '1' THEN output_data <= d(0,1); END IF;
    IF r = '1' AND v(1) = '0' THEN output_data <= d(1,0); END IF;
    IF r = '1' AND v(1) = '1' THEN output_data <= d(1,1); END IF;
  END PROCESS;
END behavioural;
```

Figure 2: VHDL description of Four-slot algorithm.

as the four-slot states and variables are binary in nature, e.g. two paired states could be used to represent *writing* and $\overline{writing}$ in the model. These two places must always be marked with only one token between them (i.e. the state of *writing* and $\overline{writing}$ can never happen simultaneously). As before changes of state are represented by Petri net transitions.

The Petri net model for the writing cycle: start writing $\rightarrow$ *writing* $\rightarrow$ end writing $\rightarrow$ $\overline{writing}$ to slot $d(0,0)$ (part of the writing step $wr$) is shown in Figure 3, with its initial marking. Here 'start write' can fire (writing can start on $d(0,0)$) only when $n = 0$ and $s(0) = 1$ and this action does not modify the values of $n$ and $s$. In this step, only one of the slots is written to and the whole step takes two transitions to finish. Data transfers to and from all four slots are represented in this way throughout the model.

The Petri net model of the second writing step ($w0$) is shown in Figure 4. This step takes one transition to finish and only one of the four transitions fires according to the values of $n$ and $s$. Following conventional Petri net notation, double-arrowed arcs indicate that a place is

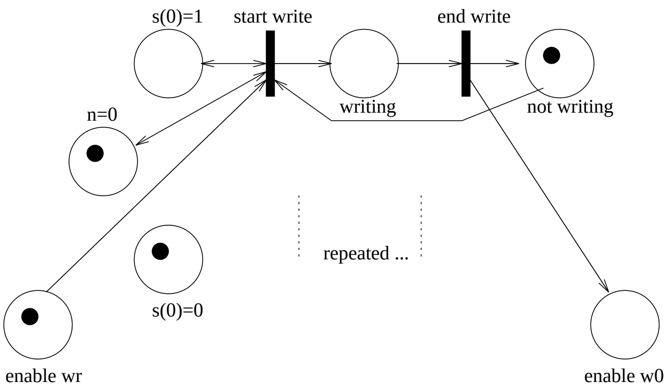


Figure 3: Four-slot 'wr' Petri net model.

both an input and output place for a transition. Therefore this step references the value of $n$ without modifying it. Control variable value cycles are represented in this way throughout the model.

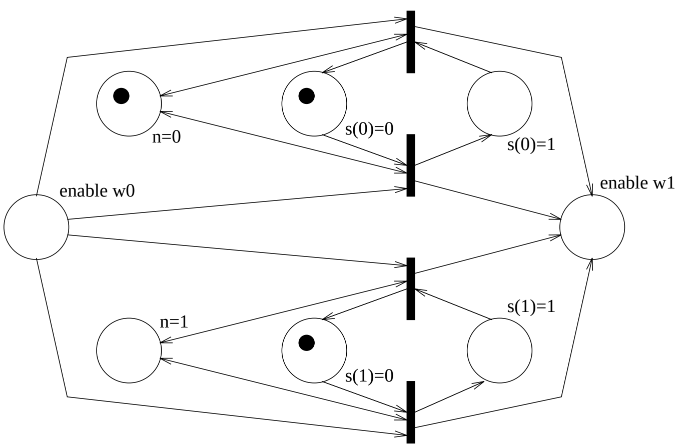


Figure 4: Four-slot 'w0' Petri net model.

Ref [4] contains the complete Petri net model of the four-slot mechanism. All of the control variable setting steps take one transition firing to complete and only one transition can fire in each of the writer or reader model at any time. This complies with the convention adopted in the two- and three-slot mechanism models and although not the only way to model such actions, does provide a clear view of the operation. The reading and writing processes are only connected through the sharing of control variable places. There is no direct relation between the starting places of each reading and writing step, as required for modelling a synchronisation free design.

# 3 Analysis of data coherence

Data coherence would be lost if a writing and a reading action were to overlap on the same slot. To analyse whether such overlaps in time could occur on any slot requires a reachability search through the state space of the Petri net model. If results included simultaneous markings of writing to and reading from the same slot, this would show the mechanism to be incapable of maintaining data coherence. If no such marking is found anywhere in the state space, this demonstrates the correctness of the mechanism with respect to data coherence. In the discussions below, states (markings) found by reachability searches are called *reachable states*, of which the ones causing data coherence or freshness problems are called *faulty states*.

## 3.1 The one-slot mechanism

A reachability search on the one-slot mechanism model can be done manually. The reachable states are: $[\overline{writing}, \overline{reading}]$, $[\overline{writing}, reading]$, $[writing, \overline{reading}]$, and $[writing, reading]$ which is faulty as data coherence is lost and can be reached from the initial state in many ways. The two shortest are (1) 'start reading', then 'start writing' and (2) 'start writing', then 'start reading'. In other words, the one-slot mechanism does not maintain data coherence. The two and three slot mechanisms also fail to maintain data coherence, the analysis can be found in [4].

## 3.2 The four-slot mechanism

The reachability tree of the complete Petri net model of the four-slot confirms that the mechanism has no faulty states in the sense of any slot being concurrently read from and written to. This confirms the conclusions in [1] and [2] that the four-slot mechanism maintains data coherence.

## 3.3 Modelling and analysis of data freshness

Freshness analysis of the one to three-slot mechanisms is unnecessary since they fail the data coherence test. The data freshness property of the four-slot mechanism can be investigated by extending its Petri net model followed by reachability search analysis. To test for data freshness as defined in [2] a record of which slot was the last written ($freshest$) and which slot was the previous last written (*previous_freshest*) has to be kept. Adopting a modified 'role analysis' similar to that used in [2] requires extra *role* places to indicate $freshest$, $\overline{freshest}$, *previous_freshest*, $\overline{previous\_freshest}$, *valid* and $\overline{valid}$ states ($F$, $\overline{F}$, $P$, $\overline{P}$, $V$ and $\overline{V}$). A set of places representing these binary states is needed for each slot in the model (24 places in total for the four-slot mechanism model). They are initialised to $\overline{V}$ at the start of the analysis. The $F$, $P$, and $V$ roles can be maintained and verified using an additional monitoring Petri net which is an extension to the existing model. The roles need to be updated according to requirements shown in Figure 5 [2]. If all data read is valid, (i.e. there is a token in the $V$ place for the slot from which data is being read for all reading states) data freshness is maintained.

The implementation of the monitoring net can be carried out in various ways [4]. The monitoring net must ensure that the relevant freshness role places are always and only updated at the correct time with respect to state changes in the main net and this updating does not in any way affect the main net's behaviour in a reachability

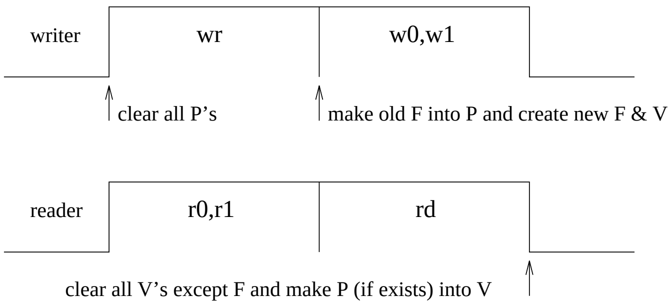


Figure 5: Freshness checking.

search. The simplest monitoring net which accomplishes this employs an *enable* place similar in function to the step starting places in the main model - (these control the firing order of the model so that it follows the algorithms' sequential nature). The enable state (a token in the enable place) is removed when a role update occurs thus stopping the main net until the role update is finished. For instance, when a start writing transition (beginning of $wr$) fires, the *enable* token is removed and put back only when the freshness state update has been completed (in this case having moved tokens from $P$ to $\overline{P}$ for all slots, once again taking one transition firing to accomplish). This does not interfere with the operation of the main net in a reachability search. The role updating part of the monitoring net is constructed in the same way as the control variable setting/resetting parts of the main net as both are safe nets.

The complete model including the freshness monitoring net was analysed and none of the reachable states were faulty. This indicates that the four slot mechanism maintains data freshness under normal operating conditions, and so verifies the results reported in [2].

# 4 Metastability and shared variables in the 4-slot mechanism

## 4.1 Modelling

Variable data in digital hardware is usually stored in some form of flip flop, consisting, in the simplest case, of a pair of cross coupled logic gates. Of many implementations the most common is the clocked flip flop, which normally accepts data from one or two inputs, with control inputs (typically clock, pre-set and clear) and two outputs $Q$ and $\overline{Q}$ representing the stored data and its complement. Modern flip flops are both fast (typically capable of speeds upto 200 MHz for sub-micron processes) and reliable.

However such devices have operating requirements that must be met for reliable operation, e.g. sufficient set up and hold time of the data before the clock edge occurs and a minimum clock pulse period. Compliance with these constraints can be guaranteed in synchronous systems but not in fully asynchronous systems, because the timing of data and clock changes is independent. This can cause metastability in the shared variables of the four-slot mechanism hardware implementation.

A metastable state [5, 6] can be the result of a timing violation between the data and the clock signal; this typically takes the form of data being changed during the set-up or hold time. In the four-slot mechanism hardware the control variables are influenced by both the reading and writing sides, which are temporally independent of each other. Therefore the transfer of control variables is via devices that have their input value supplied from a source of independent timing. The probability of violating these conditions is extremely small in a well-designed system but nevertheless non-zero. In [2] it is claimed that the four-slot mechanism will not fail in the event of metastability and that it will self correct during the next cycle of operation; this does not explain what happens to the device under these conditions. By extending models from earlier sections of this paper the effect of metastability can be modelled and its effect on the transfer of data analysed. This analysis leads to a better understanding of the four-slot mechanism which is essential for any safety critical application.

Metastability has several forms, the most common of which is the output level being halfway between '0' and '1', where it remains before settling for an undefined period (which can typically be many times the typical propagation delay of the device). The final settled state depends on factors such as random component noise in the device. The normal way of modelling binary systems in petri nets is to use a two level system representing '0' and '1'. This does not allow us to model metastability, except in an extended propagation delay form. However if we do model a system using this two level representation we can analyse the algorithms behaviour to wrongly valued variables, which is still valuable. The initial work was based on this two level representation, an extended three level representation is under development and results are still preliminary. This two level representation is valid for modelling the four-slot mechanism being used between two synchronous processes, in which the algorithm instructions can be interleaved in any order. However if we are interested in true asynchronous operation, then we need to be able to model a third state which is used to represent a metastable state of any form.

If the mechanism is implemented using flip-flops in the manner shown in Figure 6 as proposed in [1]), metastable states are possible for the variables $n$, $r$, $v(0)$ and $v(1)$ due to the asynchronism between the clock and input signals of the associated flip-flops. If the assumption is made that the metastability would have settled before the variables are used, the two level representation is sufficient to model the possibility that they may have settled on either value. Therefore this model supports the analysis on the algorithm's resistance to wrongly valued control variables.

## 4.2 Analysis results

The control variable $n$ takes its input from the reader variable $\overline{r}$ and is clocked by $w1$. We assume that the mechanism is externally supplied with correct control pulses and anomalous behaviour due to mis-clocking is ignored. Metastability of $n$ could affect the clock steering devices controlled by $w0$ and $wr$, and the value of the $l$ variable. As we assume that the metastable state will have changed to a definite logic level by the time of the next control pulse we can model this as $n$ having possibly as-

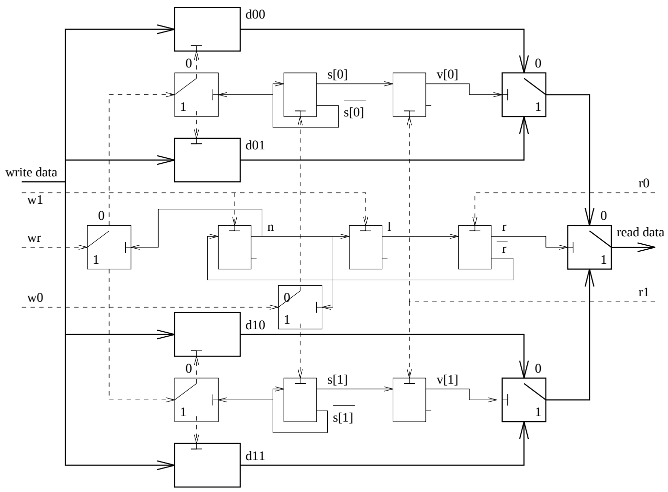


Figure 6: The four-slot hardware.

sumed a wrong value. By visual inspection of figure 6 we see that this could cause the writer to write to the wrong pair of slots, or cause an invalid toggle of the slot control variables $s(0)$ or $s(1)$. The mechanism should self correct for this wrong value of $n$ by using its value in $l$ for the next cycle, thus moving the reader and writer to different pairs of slots.

The control variable $r$ takes its input from the writer variable $l$ and is clocked by $r0$. Again we assume that the device receives correct control pulses. Again by visual inspection of figure 6 we see that metastability of $r$ could lead to the reader reading data from the wrong pair of slots, which means that the data freshness property of the mechanism may be lost. The complement of $r$, $\overline{r}$ is transferred to the $n$ variable in the next cycle, so again the mechanism should self correct in the next cycle, and move the reader and writer into different pairs of slots.

The last two control variables are considered together as they perform similar functions to each of the pairs of data slots individually. The control variables $v(0)$ and $v(1)$ take their inputs from $s(0)$ and $s(1)$ and are both clocked together by $r1$. Again by visual inspection of figure 6 we see that metastability on $v(0)$ or $v(1)$ has a very similar effect to metastability on $r$, in that it may affect the slot from which the data is being read.

Reachability search results confirming the above assumptions were not ready by the printing deadline, please see the presentation for our findings.

In this analysis we model metastability as a single occurance of a wrong value on one variable in any state space search. Two or more variables becoming metastable together in the same cycle (e.g. metastability together on $r$ and $v(0)$ or $v(1)$) will be investigated in the future.

# 5 Concluding remarks

Petri net models have been developed for an asynchronous communication mechanism and computer analysis based on the models carried out. The results provide further verification that the Simpson four-slot mechanism algorithm maintains data coherence and freshness. The models are designed so that a reachability search reveals all possible operation modes showing all possible data coherence and freshness problems. The model of the four-slot mechanism has been extended to assist investigations of the consequences on system behaviour of metastable states in the control variables $l$, $n$, $r$, $s$, and $v$. Such consequences were suggested but not extensively explored in [2].

The Petri net method presented in the paper facilitates state-space investigations amenable to systematic analysis by computer, overcoming limitations of manual reasoning techniques, offers repeatability and extensibility, and limitations of run-time simulations (i.e. completeness). The techniques can also be used for analysing properties other than data coherence and freshness, if a set of specific states can be isolated as representing the property to be investigated.

# 6 Acknowledgements

The authors acknowledge helpful discussions about asynchronous real-time systems with Dr Hugo Simpson of British Aerospace.

# References

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

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

[3] Peterson, J.L. Petri net theory and the modeling of systems, Prentice-Hall, 1981.

[4] Xia, F. and Clark, I.G. Petri net models of a class of asynchronous communication mechanisms, Research Report No. 116/SCS/95, ISBN-1-898-783-07-1, Dept. of Electronic and Electrical Engineering, Kings College London, 1995.

[5] Horstmann, J.U., Eichel, H.W. and Coates, R.L. Metastability behaviour of CMOS ASIC flip flops in theory and test, IEEE Journal of Solid State Circuits, Vol. 24, No. 1, pp.146-157, February 1989.

[6] Marino L.R. General theory of metastable operation, IEEE Trans. Comput., Vol. 30, No. 2, pp.107-115, 1981.