CS-TR-686, Department of Computing Science, University of Newcastle upon Tyne, 1999.

An asynchronous communication mechanism using self-timed circuits

F. Xia, A. Yakovlev, D. Shang, A. Bystrov, A. Koelmans and D.J.Kinniment

TECHNICAL REPORT SERIES

No. CS-TR-686
October, 1999

Contact:
fei.xia@ncl.ac.uk
http://www.cs.ncl.ac.uk/research/projects/comfort.html

CS-TR-686, Department of Computing Science, University of Newcastle upon Tyne, 1999.

An asynchronous communication mechanism using self-timed circuits

F. Xia[*], A. Yakovlev, D. Shang, A. Bystrov, A. Koelmans and D.J.Kinniment
University of Newcastle upon Tyne

Indexing terms: Asynchronous data communications, concurrent systems, arbiters, metastability, speed independent circuits, Petri nets, STGs.

1 Abstract

An asynchronous data communication mechanism (ACM) using self-timed circuits is presented. Mutual exclusion elements are used to concentrate potential metastability to two discrete points so that it can be resolved entirely within the mechanism itself. Self-timed circuits allow the minimisation of the interface between the reader and writer processes and the mechanism. Initial analysis shows that this VLSI solution is more robust with regard to steering logic metastability, and can potentially run faster than similar solutions under fundamental mode assumptions. It is therefore more suitable for use in on-chip multi-processing systems.

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.

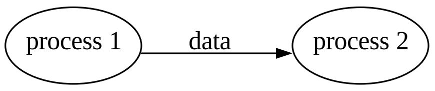

Figure 1 Passing data between two processes.

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.

process 1
data repository
(shared memory)
process 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.

Many ACMs have been proposed in the literature. Since shared memory may have access conflicts when the processes are not synchronised, much work has been done to find techniques whereby such conflicts are avoided and the shared memory is made “regular” and “atomic” [1]. For instance, an obvious way to protect shared memory is to put it into an explicit critical section for each process [2].

Using critical sections, however, may not be acceptable in real-time systems because the unpredictable waiting time makes it impossible to estimate the precise temporal characteristics of a process. It also makes the writer temporally dependent on the reader and/or vice versa. Such dependence may be in conflict with real-time requirements specified for the reader and/or the writer.

For instance, in the Pathfinder mission to Mars, where shared memory used for data communications was managed by a complex arrangement of critical sections, priorities and interrupts, the long critical sections on the shared memory conflicted with the real-time requirements of subsystems, causing occasional system failures [3].

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

In certain applications, reference data needs to be passed from one process to another. Such data may be generated irregularly and used irregularly, with the writer and reader processes mainly preoccupied with other activities. For instance, in the robot simulation system of [4], the robot processes pass their location data to each other and this data is used as reference. At the time of use, only the latest item of such data is of relevance, any previous non-used items having been superseded by it. In other words, when a robot process needs to know where another robot is located, only present location is of interest. On the other hand, for a simulation of multiple robots in a workspace to be realistic, the robot processes must be temporally independent from each other permanently, even when data is being passed from one to another. An ACM between two such processes would need to keep only the latest item of data from the writer, and cannot employ critical sections in the data access.

Such techniques as the multiple "slot" (or "track") mechanisms described in [5], [6] and [7] 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 without resorting to explicit critical sections. These solutions, however, make fundamental mode assumptions on the operations of the control variables. In other words, they shift the problem of synchronisation from the data memory to the control logic, which typically consists of bit-sized shared variables. The general scheme of these data communication mechanisms is shown in Figure 3.

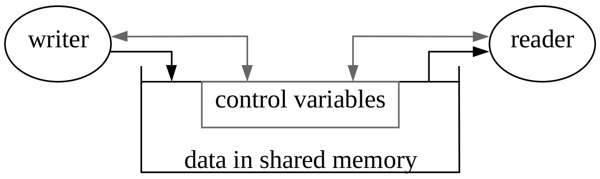

Figure 3 ACM using shared memory and control variables.

Fundamental mode assumptions on control variable operations do not hold when such shared variables become metastable, which is possible in the total absence of synchronisation between the processes.

The non-blocking FIFO described in [8] also avoids critical sections in data access. However, the FIFO arrangement introduces latency which is not suitable for reference data applications.

This paper presents an ACM solution, implemented entirely in VLSI circuits, that provides a flexible choice to potential users between minimal critical sections on bit control variables and full temporal independence between the access processes. Improved handling of metastability using arbiters and the employment of self-timed, speed independent (SI) circuits make this possible. It is envisaged that this solution can become an element in a hardware library which supports data communications between processor or other IP cores implemented on the same chip [9]. It will be especially useful in systems where there are hard real-time requirements.

3 Background studies

A particular ACM solution from the literature has been chosen as a basis of the new design. This is briefly described. The basic properties of ACMs, some of which have been studied in various published work, are collectively introduced. Metastability on control variables is also discussed in some detail.

3.1 The 4-slot ACM algorithm

Of the multiple slot or track ACMs that provide fully asynchronous operations, by far the simplest and easiest to implement in hardware is the Simpson's 4-slot mechanism found in [5]. This mechanism is adapted here to produce our hardware ACM.

Writer	Reader
wr: $d[n, \overline{s[n]}] := input$	$r0$: $r := l$
$w0$: $s[n] := \overline{s[n]}$	$r1$: $v := s$
$w1$: $l := n \parallel n := r$	rd: $output := d[r, v[r]]$

Figure 4 Simpson's 4-slot mechanism.

Figure 4 shows the algorithm of the 4-slot mechanism. Here the writer and reader processes are single thread loops with three statements each. The mechanism maintains the storage of four data slots $d[0, 0]$ to $d[1, 1]$ and the control variables n, l, r, $s[0..1]$, and $v[0..1]$, which are either single bits or vectors of two single bits. This is shown schematically in Figure 5. The statements wr and rd are the data accesses and the other statements are used by the writer and reader to chose slots and indicate such choices to the other side.

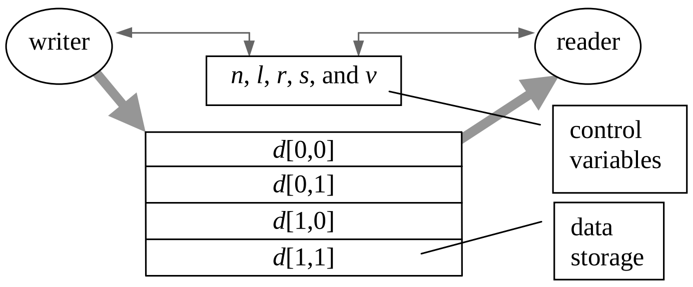

Figure 5 Schematic of the 4-slot mechanism.

3.2 Properties of ACMs

The most significant properties of ACMs are listed below:

1. Asynchronism: An ACM should not require the reader and writer processes to be synchronised to each other permanently. If an ACM provides a complete temporal divide between the reader and writer processes as shown in Figure 6, so that the reader and writer processes are entirely temporally independent from each other, it is said to be *fully asynchronous* or *synchronisation free*.

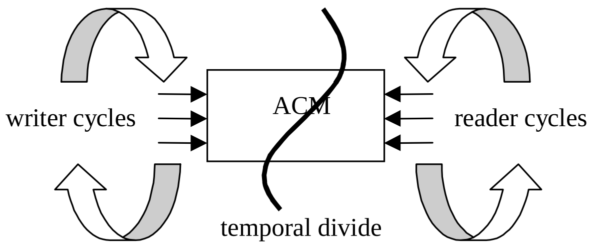

Figure 6 ACM providing temporal independence to processes.

2. Data coherence: Data coherence is violated if at any time both writer and reader access the same data storage location in the shared memory. In the slot-type mechanisms, this means that the writer and the reader should not access the same slot simultaneously. A writer cycle may include a data coherence violation if the writer access conflicts with a reader access at the same slot. The same may be said of a reader cycle. Data coherence can thus be quantitatively described by the rate of cycles not containing violations to total cycles, from either the writer or the reader side. Data coherence is completely maintained if the writer and reader do not access the same slot simultaneously at all, in which case the data coherence rate is 1.

3. Data freshness: Data freshness describes how up to date any item of data that the reader obtains from the ACM is. In slot-type mechanisms, the latest data item is always found in the slot which the writer accessed during its last cycle. Data freshness is normally checked just before a reader access and/or at the beginning of a reader cycle. In this paper, the definition of data freshness by Simpson in [10] is used.

4. Data sequencing: Data sequencing is violated if the reader obtains data items in reverse order to that in which they were written into the ACM by the writer. Data sequencing can be described quantitatively by its violation rate.

5. Data loss: Data loss occurs when some items written into the ACM by the writer are not eventually obtained by the reader. Data loss can be described quantitatively by the rate of its occurrence.

6. Data re-reading: Data re-reading occurs when the reader obtains an item which it has obtained during an earlier cycle. Data re-reading can also be described quantitatively by the rate of its occurrence.

No realisable ACM can completely fulfil all the properties listed above. For instance, if an ACM is fully asynchronous, data loss would be inevitable if the writer is faster than the reader and data re-reading would be inevitable if the reader is faster than the writer. A classical finite capacity buffer (i.e. a *k*-place FIFO) does not provide full asynchronism, because the writer is blocked when the buffer is full. On the other hand, the non-blocking FIFO in [8] achieves full asynchronism by specifying data loss and data re-reading when necessary. In addition, the inherent latency of FIFOs means that data freshness suffers in any FIFO design.

The 4-slot mechanism has been shown to maintain full data coherence and data freshness in [10], [15], [11] and [12] when the ACM is operating in fully asynchronous mode and when the fundamental mode assumptions about the writer and reader statements hold.

3.3 Metastability

The slot-type ACMs avoid critical sections in data slots by synchronising a data slot to an access process during the entire period of access (e.g. statement *wr* or *rd* in the 4-slot ACM). If during this access the other access process wants to start accessing the data area, it is directed to another slot. In effect, critical sections exist in individual slots but not globally. The steering is effected by careful use of control variables. In the 4-slot ACM these are binary variables which are only changed one bit at any time. In addition, for each of these bits either the writer is in charge of changing it and the reader only references in value, or vice versa. Since a bit is the smallest granularity possible in a digital system, it was reasoned that the 4-slot solution is maximising the atomicity of the data transfer.

Ultimately, a bit control variable needs to be implemented in hardware using a bit register in order for the solution to be fully asynchronous in the sense of hard real-time. Such a register may be implemented with D-type or transparent binary latch circuitry. For instance the latch in Figure 7 may be used to carry out the inter-process assignment statement, *cl*: *y* := NOT *x*, where both *x* and *y* are bit variables.

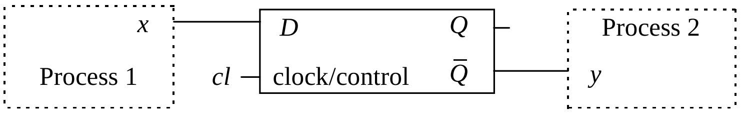

Figure 7 Concurrent processes communicating via a latch.

No fully asynchronous communication scheme can avoid the possibility of metastability [13]. In the slot-type mechanisms, potential metastability happens at the bit registers implementing control variables. For instance, in the case of the reader statement *r*0 in Figure 4, the clock signal for the statement (*r*0) comes from the reader process, while the input signal (*l*) is set by the writer process during statement *w*1. If no synchronisation is permitted between the reader and writer processes, *r*0 may arrive at the same time as or very close to a change of value of *l*, resulting in potential metastability at *r*.

If this metastability has not settled when the value of *r* is used in statement *rd*, more than one data slot may be accessed during *rd*, resulting in a data coherence failure and non-atomic data transfer.

The conventional method to deal with this problem is to make fundamental mode assumptions by specifying that both the writer and reader processes must have enough delay between the acquisition of the value of a control variable, where metastability is possible, and the use of it. This delay ensures that any metastability would have settled with a reasonably high probability by the time it is used.

In the case of Figure 4, the reader process must thus be implemented so that the time span between *r*0 and *rd* is large enough for any metastability at *r* to settle and that between *r*1 and *rd* is large enough for any metastability at *v* to settle. Similar requirements must be put on the writer process. The 4-slot ACM operates correctly if all metastable signals have settled when they are used, regardless of the actual value they have settled to [14, 15]. On the other hand, if metastability has not settled when the control variable is used, data coherence is sacrificed to maintain full temporal independence of the access processes. This is because the timing of the various statements has nothing to do with the actual completion or the lack thereof of the previous statements. There is therefore no option of further waiting and no warning that data coherence may have been violated.

This method puts little extra time/speed burden on the writer and reader processes in practice if the ACM hardware is much faster than the individual statements in the writer and reader processes. In the case of these being statements in software in conventional microprocessors and the ACM control variables being implemented with simple latches built on at least similar VLSI technology to that of the processors, this is usually the case.

Added insurance can be obtained by copying the control variables into the processors where the access processes run before using. Repeated copying between the internal registers of the processors and their cache RAM increases the probability of settling of any metastable value.

In the case of a general hardware library element potentially used to transfer data between IP cores implemented on the same chip, however, these assumptions can no longer be made. With the ACM having essentially the same hardware speed to the access processes it is serving, metastable control variable values would settle with a similar speed whether they are maintained entirely inside the ACM or partly in the client processors. In effect, the time period between the referencing of a control variable value and its use may be the minimal amount of time needed by the hardware implementing the intervening statements. This makes it desirable or even necessary to deal with the issue of metastability within the ACM hardware and not pass the problem to client processors.

4 ACM design and implementation

An ACM based on the 4-slot solution has been designed and implemented in VLSI circuits. SI techniques are employed to simplify the external interfaces connecting to the access processes, and provide the possibility of preserving data coherence even if metastability occurs, when the requirement on temporal independence can be relaxed at the level of control circuits in the ACM. Arbiters are used to concentrate possible metastability to two points in the system, making it easy to implement the SI circuits for the algorithm statements.

4.1 Statement buffering

The algorithm of Figure 4 implies that the ACM interfaces with each access process three times in each cycle. In practice, buffer devices can be employed to manage the three interfaces while the access processes interface with the extended ACM only once per cycle. This idea is shown in Figure 8.

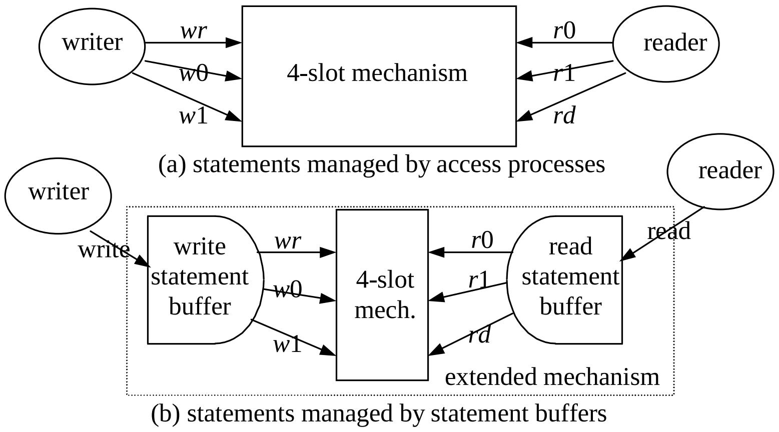

Figure 8 Statement management buffers simplifying the interfaces.

It must be noted that implementations according to Figure 8 (b) manage the time delays to deal with possible metastability in control variables within the statement buffers. This means that these delays are part of the extended ACM rather than the access processes. Assuming fundamental mode operations, these delays are essentially open-loop and fixed. If the buffers are made using the same hardware technology as the basic ACM the time delays become significant and they must be carefully considered when analysing the temporal behaviour of the access processes.

Considering the case of multiple processors on the same chip, both the client processors where the reader and the writer processes execute and the ACM may be implemented on the same chip. This implies that such issues as metastability in the control variables tend to require hardware solutions, and the system response time requirements may not allow fixed delays for each statement just to cope with possible metastability.

4.2 Overall ACM design

The ACM design include statement circuits which are entirely SI, with the sequential arrangements of the statements managed by a series of handshake protocols instead of via fundamental mode assumptions. The overall structure of this ACM is shown in Figure 9.

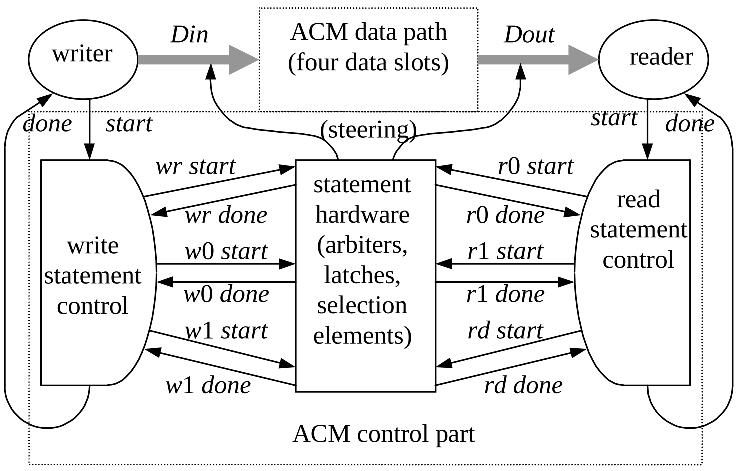

Figure 9 Basic structure of modified 4-slot ACM with SI circuits.

The system includes reader and writer interface control logic, statement circuits (both control variables and data slot selection and indication) and arbiters. Four phase hand shake bundled data protocol is used for *Din*, *start*, and *done* on the writer side and *Dout*, *done*, and *start* on the reader side. The signal sequencing of the write statement control is specified in STG form in Figure 10. This ensures the statement sequencing specified by algorithm of Figure 4. The read statement control has essentially the same STG.

The issue of timing non-interference between the reading and writing sides is more complicated than for a design with fundamental mode assumptions and will be discussed in detail later.

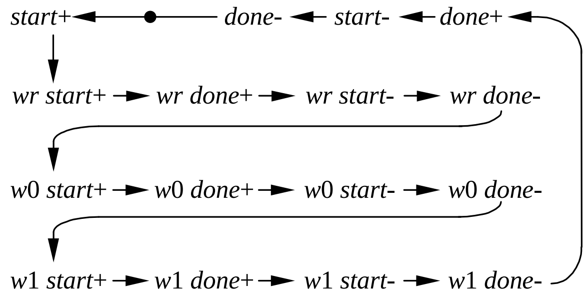

Figure 10 STG specification of write statement control.

4.3 Arbiter circuits used to localise metastability

Implementing fixed delays based on the metastability characteristics of the hardware technology is not efficient as the delays must be present whether metastability (which is a very low probability event) has happened or not. In comparison, it is possible to use SI circuits so that waiting/delay is only invoked when metastability occurs. This would provide, on average, a much higher performance [16].

The capability of finding out whether metastability, if any, has settled, is required in such a self-timed design. This is obtained by using arbiters with metastability detectors, sometimes known as metastability resolvers [17]. In such circuits, the outputs of the metastability detector will not change until any internal metastability has settled. This can indeed prevent metastability from passing on to subsequent circuits.

In the context of the 4-slot mechanism, if all possible metastability is confined within such circuits, a completely self-timed, and potentially SI solution may be obtained where metastable signals are never used. Here we take the definition of SI from [18], i.e. a circuit is considered SI if its responses are not dependent on the relative delays at the outputs of all gates, as long as they are finite.

There are three possible metastability points in the algorithm in Figure 4, owing to shared variable conflicts. One is the setting of the *r* variable mentioned above, another is the setting of the *n* variable in statement *w*1, and the last is the setting of the *v* variable in statement *r*1. The first two points involve possible simultaneous happening of the *w*1 and *r*0 statements and the last the simultaneous happening of the *r*1 and *w*0 statements.

If two statements being carried out simultaneously causes a metastable signal, then by protecting these statements with an arbiter the metastability could be avoided. This method is employed here.

Specifically, from [5], one possible implementation of the statements *r*0 and *w*1 is shown in Figure 11, which is essentially a "shifting register". If the clock pulses *w*1 and *r*0 are generated by an arbiter of the type found in [17] so that they are never near enough in time, there will be no metastability at either *r* or *n*. In this case, any metastability would be moved to the arbiter, and only when it has settled would one of the clock pulses be generated.

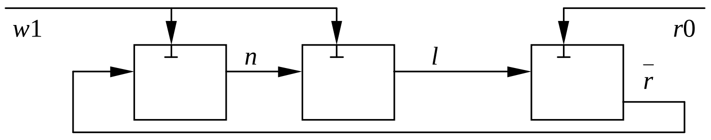

Figure 11 A simple implementation of statements *w*1 and *r*0.

This is schematically shown in Figure 12, where the statement starting signals *w*1 and *r*1 must go through an arbiter before actuating the statement hardware.

Since arbiters require waiting for the side that lost arbitration, the temporal relation between two processes arbitrated by such an element does not conform to full asynchronism. However, any delay is at the bit control variable level, not data slot level.

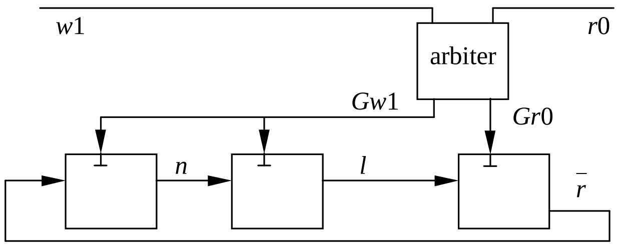

Figure 12 Mutual exclusion between *w*1 and *r*0.

4.4 Self-timed circuits for the statements

In this design, we use SI circuits in the read and write statement buffers so that the statement sequences are secured by control elements to obtain functional equality with the fundamental mode assumptions. This means that any statement may start only when the preceding one has finished. It is therefore important that each statement be implemented with hardware providing a *start*/*done* handshake interface to its environment, as specified in Figure 10.

The control variable assignment statements *w*0, *w*1, *r*0 and *r*1 are implemented by SI latch circuits within the ACM which contain completion signals. These latch circuits are reported in [19]. They fully support the start/done handshake interface protocol. The circuit implementation of statement *w*1 is given in Figure 13 as an example here.

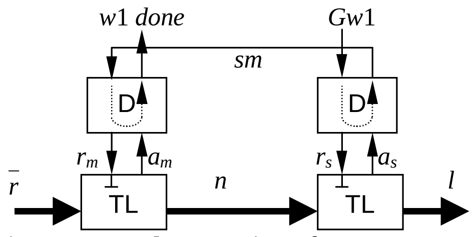

Figure 13 Implementation of statement *w*1.

In Figure 13, the control variable assignments are done through a couple of SI transparent latches (Figure 14). Each of the latches is controlled by a handshake decoupling element (the D element found in [20]) which is also SI. Consequently, the entire statement is now purely sequential with every event triggered by the previous event.

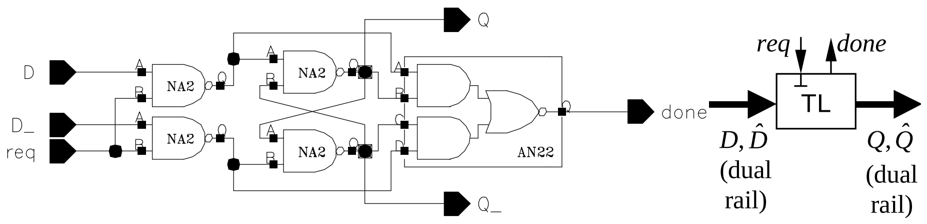

Figure 14 SI transparent latch circuit used in control variable statements.

The data slot access statements *wr* and *rd* are implemented with SI circuits which also contain completion signals. The overall design of the hardware implementation of *wr* is shown in Figure 15. That of *rd* is similar.

In this implementation, we have incorporated circuits for the data slots in order to demonstrate the operations of the design. Here it is assumed that the data item being transmitted is a single byte and the slot accessing statements take one clock cycle to accomplish. In real applications, the data path is usually the client's province and of flexible size.

The slot steering logic consists of simple selection elements implemented in purely combinational logic. There are no hazards or SI violations because when the signal *wr* comes, the values of *n* and *s* are entirely stable, having been set during the previous *w*0 and *w*1, which are guaranteed to have been completed by now according to Figure 10.

The completion of *wr* is drawn from the data path in this implementation, which is conservative in terms of SI considerations. If the data items being transmitted are large in size, however, such completion may turn out to be overly complex and performance inhibiting. In that case the external protocol can be modified and this completion signal specified to be the responsibility of the writer. Then it will be simple to implement using normal processor to memory communications assumptions.

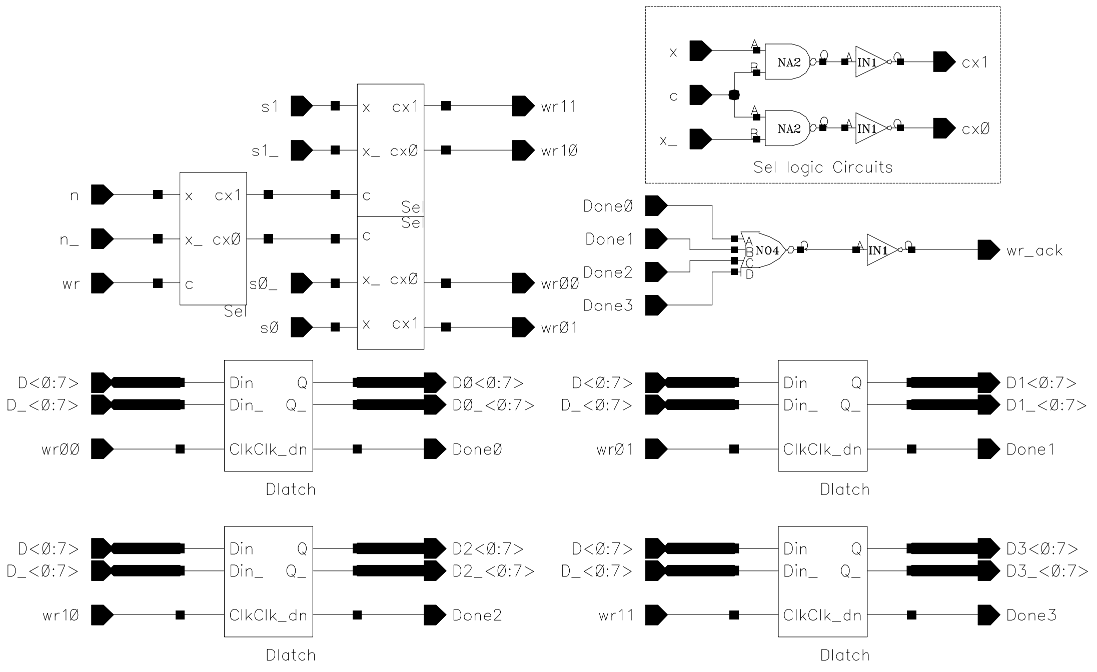

Figure 15 Hardware for statement *wr.*

4.5 Statement control elements

The write statement control element is specified by the STG in Figure 10. In order to retain an element of regularity and extendibility, a circuit known as David's element or David's cell [20], is chosen as the building block with which to assemble this circuit.

The David's cell consists of a flip-flop and a NOR (or NAND depending on the implementation of the flip-flop) gate and is completely SI. A control circuit managing four consecutive handshakes needs four David's cells connected in series. By organising the initial condition so that only one of the cells has an output of (0,1) and the others have (1,0), the circuit shown in Figure 16 produces an STG shown in Figure 17.

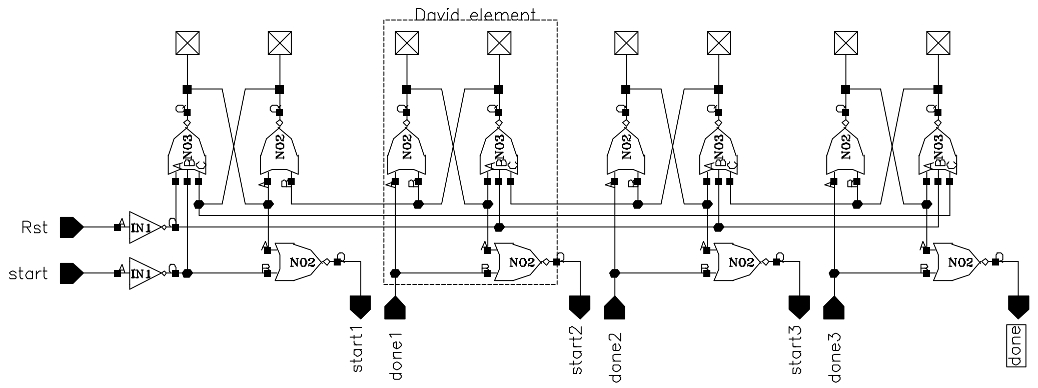

Figure 16 Write statement control logic using David's cells.

$$\begin{tikzcd}
start+ \arrow[d] & done- \arrow[l, "\bullet" marking] & Q3+ \arrow[l] & \hat{Q}3- \arrow[l] & \hat{Q}0+ \arrow[l] & Q0- \arrow[l] & start- \arrow[l] & done+ \arrow[l] \\
start1+ \arrow[r] & done1+ \arrow[r] & Q1- \arrow[r] & \hat{Q}1+ \arrow[r] & \hat{Q}0- \arrow[r] & Q0+ \arrow[r] & start1- \arrow[r] & done1- \arrow[dlllllll] \\
start2+ \arrow[r] & done2+ \arrow[r] & Q2- \arrow[r] & \hat{Q}2+ \arrow[r] & \hat{Q}1- \arrow[r] & Q1+ \arrow[r] & start2- \arrow[r] & done2- \arrow[dlllllll] \\
start3+ \arrow[r] & done3+ \arrow[r] & Q3- \arrow[r] & \hat{Q}3+ \arrow[r] & \hat{Q}2- \arrow[r] & \hat{Q}2+ \arrow[r] & start3- \arrow[r] & done3- \arrow[uuu, bend right]
\end{tikzcd}$$

Figure 17 STG of the circuit in Figure 16.

In Figure 17, the Q's and $\hat{Q}$'s are the output signals of the flip-flops within the David's cells and are not directly made use of by the control logic. They serve the same purpose as the CSC signals from a `Petrify` solution. From the STG, it is clear that this circuit can be used for both the write and read statement control logic blocks by using the *start*n and *done*n signals (n ∈ {1, 2, 3}) for the appropriate statement handshakes.

5 Differences in temporal relations between fundamental mode and SI solutions

The fundamental mode 4-slot solutions proposed in [5] provide full asynchronism for the reader and writer. They are however dependent on the fundamental mode assumption, that the switching processes in the hardware implementation settle between adjacent statements.

In the SI solution, there is certainly not an absolute temporal division between the reading and writing sides *within* the ACM, because of the waiting required by the arbiters. It is worth noting, however, that such waiting only happens during control variable setting statements and the data slot access statements *wr* and *rd* are not affected directly. In other words, by retaining the 4-slot ACM algorithm, the broad idea of realising atomic data transfer by using safe bit registers is retained. In effect, critical sections are moved from data slots to bit variables.

Temporal independence, when required, is required between the reader and writer processes and not between the internal read and write sides of the ACM. From Figure 9, it is clear that there are two pairs of handshakes where such temporal divisions can be maintained in the new design. These are the global read and write *start*/*done* interfaces. For instance, rather than the more rigid protocol normally associated with the handshake, the writer can be specified to follow the more flexible protocol outlined below:

- Initiate writing by issuing *start* to the write side of the ACM;
- Wait for *done* from the ACM;
- In the absence of *done*, wait for a predetermined maximum time period;
- Continue its own cycle, knowing whether *done* or the expiration of the maximum time period has happened.

This allows the writer client to decide whether to operate in fundamental mode or SI fashion. While still realising the potential of speeding up the response provided by the SI solution, it also allows an upper bound for the complete ACM write cycle to be specified, therefore effectively decoupling temporally the writer process from the reader one. A similar arrangement can be employed at the reader side.

Such a maximum waiting period can be easily obtained by finding the normal time expenditure of all statements and then assuming that metastability happens at one of the points of arbitration (It is trivial to show that in a single cycle of operation only one of the points of arbitration could be activated, assuming that both the read and write sides of the ACM are implemented using the same hardware technology and built on the same chip.) and then the own side loses the arbitration. The statement timing can be obtained through simulations, since the entire ACM is designed in hardware "in house". The metastability settle time can be estimated by the method outlined in [21], where it is demonstrated that 5ns is sufficient time for all metastability to have settled firmly in modern CMOS technology with "practically" probability 1.

6 Circuit implementation and analysis

The overall ACM design has been put through the VLSI design flow using Cadence tools. Top-level simulations, both analogue and digital, have been carried out. The simulations show that the functional behaviour of the circuit is as expected. Analogue simulations have established that metastability does not propagate throughout the system, but is contained within the arbiters. Digital simulations have revealed the important properties listed in 3.2. Stochastic Petri net analysis has been used to study the circuit's response characteristics.

6.1 Analogue simulation results

Analogue simulations have been run with the Spectre simulator from within the Cadence toolkit. Apart from studying the entire circuit under a number of possible operating conditions, effort has been concentrated on the behaviour of arbiters and the entire system when the statements *r*1 and *w*0 occur simultaneously. The result of this study is given below. Similar close scrutiny was given to the case when the statements *r*0 and *w*1 occur simultaneously, with similar conclusions.

Figure 18 shows the transient response of the handshake signals associated with the statements *r*1 and *w*0 when metastability has been generated within the arbiter between these statements, because the requests are close in time. The metastable response within the arbiter (at signals *net*7 and *net*9, between 4 and 5 ns) only delays the response of the rest of the system and is never propagated out of the arbiter. The arbiter is also shown to have successfully created mutual exclusion between the two statements.

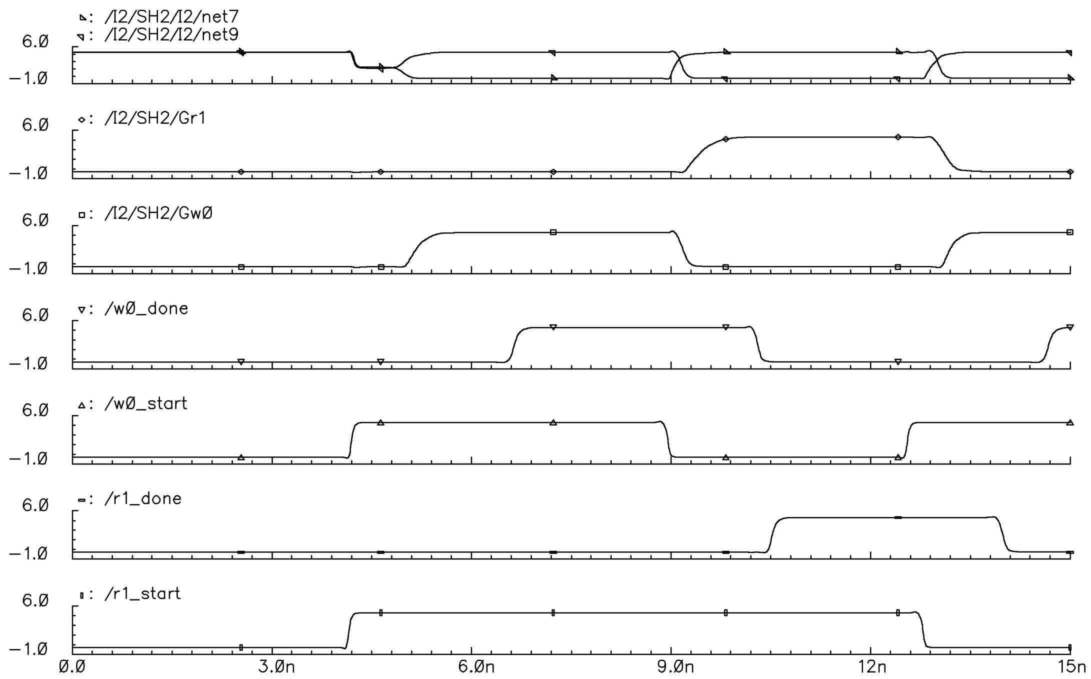

Figure 18 Analogue simulation waveforms with metastability within arbiter.

Figure 19 shows the general handshake operations on the writer side. Similar results have been obtained for the reader side. This conforms with the specifications given in Figure 10.

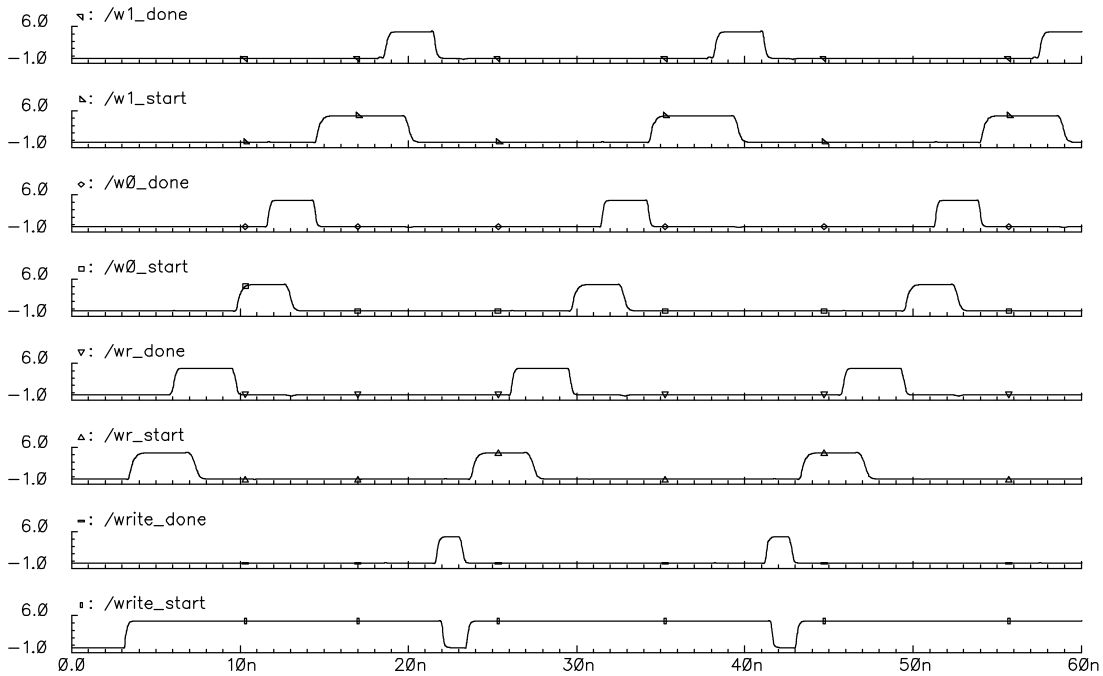

Figure 19 Analogue simulation waveforms showing general handshake operations.

	min time (ns)	max time (ns)
*w*0	3.67	7.27
*w*1	5.97	9.15
*r*0	2.18	9.14
*r*1	2.59	7.38

Table 1 Time taken by reader and writer control variable statements.

From analogue simulations, the time taken by each statement in the ACM design has also been found. These are given in Table 1. It can be seen from Table 1 that the control variable statements for the writer side take less than 10 ns to execute if there is no arbitration conflict with those of the reader. The longest possible time needed for these two writer statements, if there is no metastability, is 13.24 ns, when *w*0 loses arbitration to *r*1. The reader control variable statements take between less than 5 ns and 11.73 ns to execute.

The statements *wr* and *rd* depend on the way in which the data path is organised. The following timing information has been obtained from simulations based on our single byte data path implementation. The entire cycle of the writer statement buffer, incorporating statements *wr*, *w*0 and *w*1, takes from 20.31 ns to 23.24 ns to complete. The cycle of the reader statement buffer, with statements *r*0, *r*1 and *rd*, takes 11.76 ns to 18.73 ns to complete. The writer cycle is slower because the double assignment in statement *w*0 takes two clock cycles to complete.

6.2 Studying the time response with stochastic Petri net techniques

Using the timing information obtained from the analogue simulations, we established stochastic Petri net models for the ACM, and ran these models through the tool `PEI` v1.0 [22]. We assumed exponential distribution for the length of time taken by the reader and writer outside the ACM access in each cycle (called "reader extra-ACM delay" and "writer extra-ACM delay" in this paper) and investigated the behaviour of the mechanism both with a faster reader and slower writer and vice versa. Many data points were collected, and the resulting mean times taken by the ACM statement cycles were used to generate the diagram shown in Figure 20.

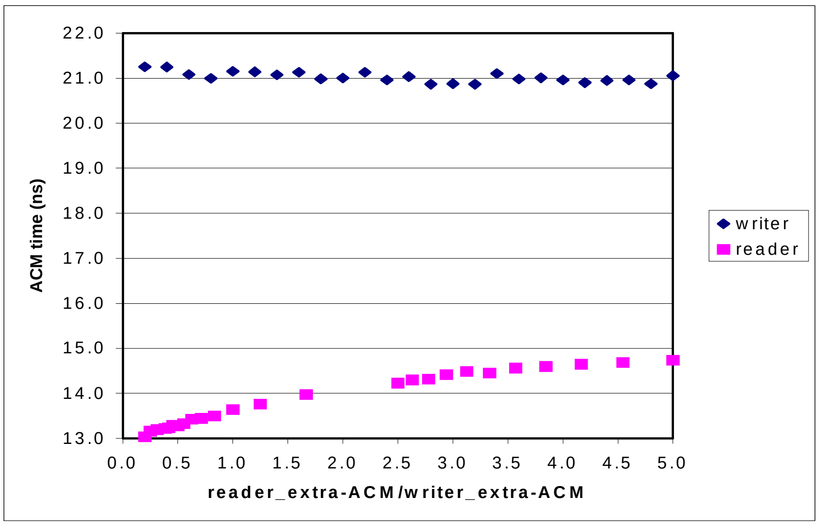

Figure 20 ACM cycle times

The probability of either the reader ACM cycle or the writer ACM cycle encountering arbitration in one of its statements changes with the relative frequency of the reader and writer cycles. This relative frequency depends mostly on the extra-ACM delays of the two sides. In order to obtain a more complete picture, we have changed the mean time taken by the reader and writer processes outside their ACM access statements and used the ratio between these times as the horizontal axis in generating Figure 20. It can be seen from the results that the ratio does not have a very large influence on the ACM cycle times which are nearer to their minimum than maximum possible values. This is to be expected as the probability of encountering arbitration is not high and the consequence of losing arbitration is not quantitatively large.

6.3 Digital simulation results

Digital simulations have been run from the Cadence toolkit on the circuit. In order to maximally reveal the properties listed in 3.2, a writer process was created in VERILOG code which sends byte type data for 255 cycles, with the data increasing in value from 1 to 255. The data received at the reader end is then collected for analysis. The writer and reader processes are programmed so that their extra-ACM delays take exponentially distributed time lengths with mean values varying from 10 to 500 ns.

From these simulations, no data coherence and freshness violations have been observed. This is true even when, owing to the stochastic nature of the reader and writer extra-ACM delays, one side goes many cycles with the other side stuck. This is to be expected because this ACM design is a faithful implementation of the 4-slot mechanism which has been verified analytically to maintain these properties if the fundamental mode assumptions hold. This ACM design, by dealing with the issue of metastability explicitly and using SI circuits, makes sure that no statement gets started without its preceding one having completed. From the state-transition system point of view this is functionally equivalent to the fundamental mode assumptions holding in the original 4-slot mechanism.

In addition, no data sequencing violation has been revealed from the simulations.

Data loss and data re-reading rates have been shown to be dependent on the relative speed of the writer and reader processes. The results are shown in Figure 21 and Figure 22.

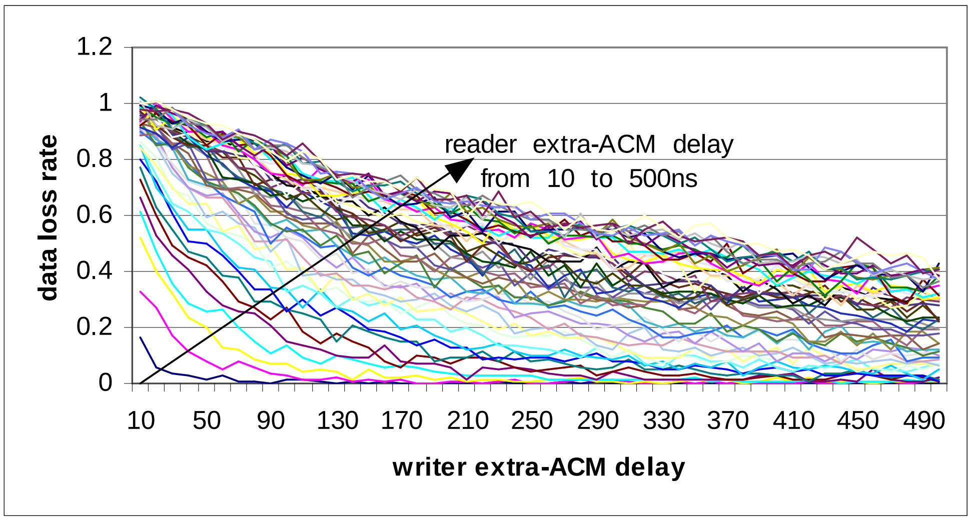

Figure 21 Data loss rates in relation to the mean values of writer and reader extra-ACM delays.

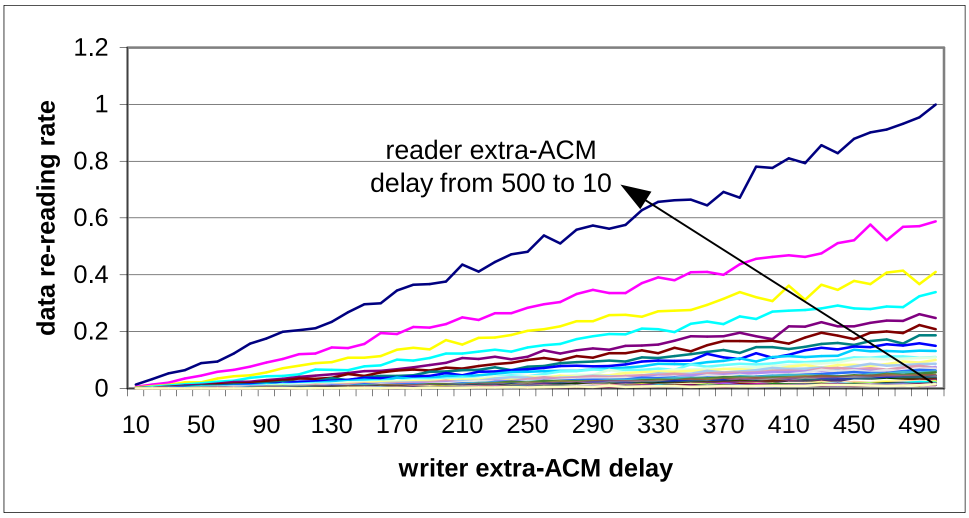

Figure 22 Data re-reading rates in relation to the mean writer and reader extra-ACM delays.

From these diagrams, it is clear that data loss increases if the writer becomes faster and data re-reading increases when the reader becomes faster. Since each simulation run consisted of 255 writer cycles, when the reader is faster it may include many thousands of reader cycles and when the reader is slower it may include only a few reader cycles. This is the main reason that the diagrams are not shaped similarly to each other. The qualitative results are however consistent.

The data value sequences for *Din* and *Dout* from a simulation run is selectively shown in Figure 23 and Figure 24. Data coherence, freshness, sequencing, loss and re-reading properties can all be obtained by observing such sequences. For instance, the loss of data items 22 and 23 can be observed in Figure 24, and the re-reading of data items 01 and 04 can be observed in Figure 23. The second reading of data value 04, while data value 05 has clearly been available for some time, does not violate data freshness according to the definition found in [10]. According to this definition, when the statements *r*0 and *r*1 overlap with the statements *w*0 and *w*1, the reader is allowed to obtain the latest but one item of data in the ACM. This is because the location of the slot where the latest data item resides is indicated by the writer through *w*0 and *w*1 and obtained by the reader during *r*0 and *r*1. When these statements overlap in time the ACM should not be expected to always pass the latest data item.

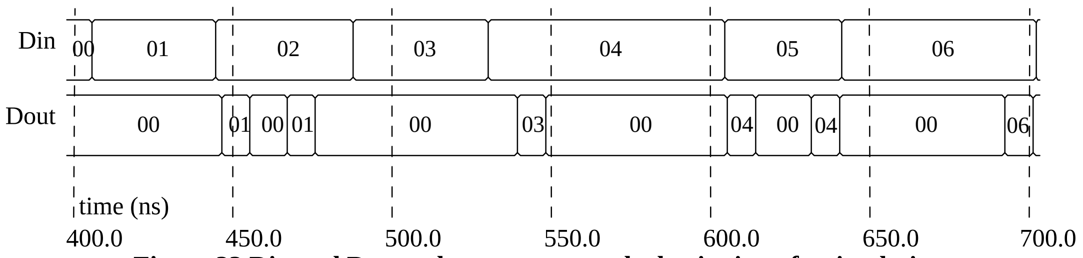

Figure 23 Din and Dout value sequence at the beginning of a simulation.

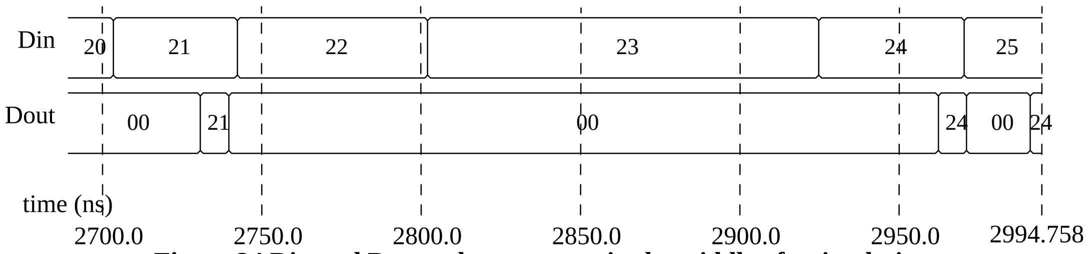

Figure 24 Din and Dout value sequence in the middle of a simulation.

7 Current work, conclusion and discussion

Layout detailing is being carried out and further analogue simulations with the help of parameters extracted from layout information will be the next step. The target of this work is the eventual fabrication of testable ACM circuits for further, hardware-based, testing and prototyping.

This version of the 4-slot ACM, within its local boundary, is not fully asynchronous by virtue of the unpredictable waiting introduced by the arbiters. However, unlike the original fundamental mode 4-slot system proposed in [5], which may forcibly ensure full asynchronism by relaxing the requirement on data coherence when metastability occurs, this implementation gives the client (the designer of the writer and reader processes) the choice of either sacrificing timing independence or data coherence by defining the overall protocol between the reader and the read statement buffer and the writer and the write statement buffer (Figure 9).

If a choice is made to give temporal independence priority over data coherence, then the new ACM would perform similarly to the original design in terms of data coherence violation rates. This is because the statistical profile of metastability is unchanged, and the arguments of settling metastability in repeated copying inside processors do not apply when both ACM and the client processors are on the same chip.

In addition, the client designer may choose not to lose data coherence when the *done* signal is not forthcoming but still be able to preserve timing integrity for the access process. Since at this point the access process has the information that metastability has occurred within the ACM, it may be specified to not carry out the access during the present cycle but either throw out the current item of data (for the writer) or use the item of data acquired during the last cycle (for the reader). In this case *data freshness* [5] is sacrificed. This is not a real sacrifice because when data coherence is not maintained data freshness becomes automatically meaningless.

The new option of letting the ACM run as fast as it can should produce significant speed gains simply because metastability is such a rare event.

In essence, this design eliminates critical sections on the data slots by using the 4-slot ACM as the basis and shifts critical sections to bit control variables by arbiters and SI statement circuits, and gives clients the choice of whether to make full use of these minimised critical sections.

In order to be more confident of our design, further work is planned to quantitatively analyse the behaviour of the two ACM designs to see how they compare in time response, temporal independence, and other areas.

8 Acknowledgements

The authors would like to thank A.A.Madalinski for running the PEI tool, A.Xie and P.A.Beerel for kindly making PEI available and providing advisory support about it, and E.Campbell, I.G.Clark, A.C.Davies, and D.A.Fraser for valuable discussions. This work is part of the Comfort project at the University of Newcastle supported by the EPSRC grant GR/L93775.

1 Lamport L., “On interprocess communication”, Parts I and II, Distributed Computing, Vol.1986, No.1, pp.77-101, Springer-Verlag, 1986.

2 Lynch, N.A., “Distributed Algorithms”, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

3 Wilner, D., “Vx-Files: What Really Happened on Mars?”, Keynote Speech, The 18th IEEE Real-Time Systems Symposium San Francisco, California, December 2-5, 1997.

4 Xia, F., Velastin, S.A., and Davies, A.C., “A parallel simulation of multiple mobile robots using the DORIS design method”, Proceedings, 1994 IEEE international conference on robotics and automation. San Diego, CA, USA, 2482-2487, 1994.

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

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

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

8 Yakovlev, A., Kinniment, D.J., Xia, F. and Koelmans, A.M., “A FIFO buffer with non-blocking interface”, IEEE Computer Society TCVLSI Technical Bulletin, pp. 11-14, Fall 1998.

9 Craft, D., “Improved CMOS core interconnect approach for advanced SoC applications”, IP'99 Europe, Edinburgh UK, November 1999.

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

11 Semenov, A. and Yakovlev, A., “Contextual Net Unfolding and Asynchronous System Verification”, Technical Report, TR572, Department of Computing Science, University of Newcastle upon Tyne, 1997.

12 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, Lisbon, Portugal, June 23, 1998.

13 Marino L.R. “General theory of metastable operation”, IEEE Trans. Comput., C-30(2):107-115, February 1981.

14 Simpson, H.R., “Correctness analysis for class of asynchronous communication mechanisms”, IEE Procs., Vol. 139, Pt. E, No. 1, pp.35-49, January 1992.

15 Xia F., Clark I.G., and Davies A.C., “Petri-net based investigation of synchronisation free interprocess communication in shared-memory real-time systems”, *Proceedings*, Second UK Asynchronous Forum, Newcastle upon Tyne, UK, July 1-2, 1997.

16 Kinniment D.J., Gao B., Yakovlev A.V., and Xia F., “Towards asynchronous A-D conversion”, *Proceedings*, Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'98), March-April 1998, San Diego, CA, IEEE Computer Society Press.

17 Seitz, Ch., “Ideas about arbiters”, Lambda, vol.1 pp 10-14, First Quarter 1980

18 Muller, D.E. and Bartky, W.C., "A theory of asynchronous circuits", Annals of Comput. Lab., Harvard University, pp. 204-243, 1959.

19 Bystrov, A., Shang, D., Xia, F. and Yakovlev, A., “Self-timed and speed independent latch circuits”, 6th UK Asynchronous Forum, The University of Manchester, Manchester, UK, July 12-13, 1999.

20 Varshavsky, V. et al, Self-Timed Control of Concurrent Processes, Kluwer Academic Publishers, P.O. Box 17,3300 AA Dordrecht, The Netherlands, 1990 (Russian Edition: Nauka, Moscow,1986).

21 Kinniment, D.J., “Measurements on a high speed arbiter”, Technical Report Series, TR677, Department of Computing Science, University of Newcastle, 1999.

22 Xie, A. and Beerel, P.A., “Performance analysis of asynchronous circuits and systems using stochastic timed Petri nets” (invited paper), Proc. of 2nd Workshop on Hardware Design and Petri Nets, pp. 35-62, June 1999.