An asynchronous communication mechanism using self-timed circuits

F. Xia*, D. Shang, A. Yakovlev and A. Koelmans
Department of Computing Science, University of Newcastle

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

1 Abstract

A modified 4-slot asynchronous communication mechanism (ACM) using entirely self-timed circuits to implement the algorithm is presented here. Mutual exclusion elements are used to concentrate potential metastability to a couple of discrete points so that it can be resolved entirely within the mechanism itself, while the self-timed circuits allow the interface between the reader and writer processes and the mechanism to be minimised. Initial analyses show that this solution is more robust with regard to steering logic metastability, and can potentially run faster, than the original 4-slot solution.

2 Introduction

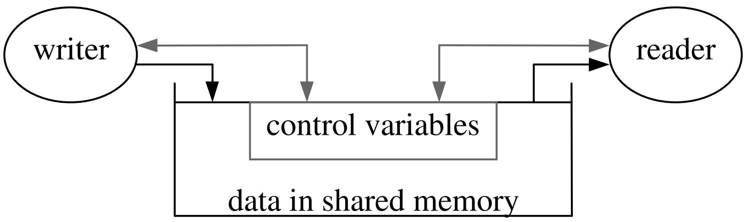


Figure 1 Asynchronous data communication mechanisms using shared memory and control variables.

Data communication between concurrent processes often employ shared memory which may have access conflicts when the processes are not synchronised. The most obvious way to protect shared memory is to put it into a critical section for each process. Such techniques as the "slot" mechanisms described in [1] realise "regular" and "atomic" registers in the data path between asynchronous concurrent processes by employing "safe" bit registers [2] to convey the values of bit-size control variables. They avoid conflicts on data memory *without resorting to explicit critical sections*. These solutions, however, depend on the control logic to function correctly under all conditions. 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, which may go metastable in the total absence of synchronisation between the processes. The general scheme of these data communication mechanisms is shown in Figure 1.

---

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

The safe bit registers may be implemented with D-type or transparent binary latch circuits, such as the one shown in Figure 2, which carries out the inter-process assignment statement, *cl*: *y* := NOT *x*.

Figure 2 Concurrent processes communicating via a latch.

Figure 3 shows the algorithm of Simpson's 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 4.

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

Figure 3 Simpson's 4-slot mechanism.

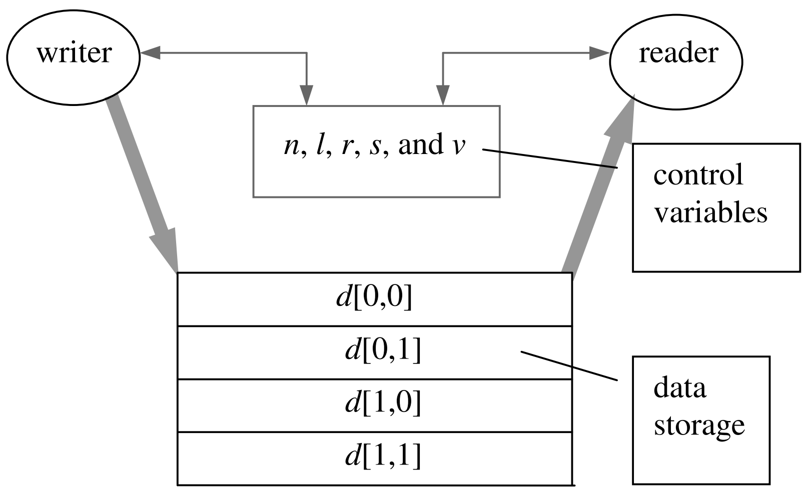


Figure 4 Schematic of the four-slot mechanism.

No fully asynchronous communication scheme can avoid the possibility of metastability [3]. In the slot mechanisms, potential metastability happens at the safe registers. For instance, in the case of the reader statement *r*0 in Figure 3, 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 or very near to a change of value of *l*, resulting in potential metastability at *r*.

If this metastability is not settled when the value of r is used in statement rd, more than one data slot may be accessed during rd, resulting in data corruption. This is a type of *data coherence* problem as defined in [1].

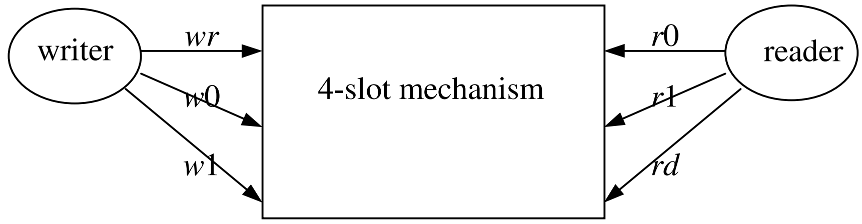
The conventional method of dealing with this problem is to specify that both the writer and reader processes must have enough delay between the acquisition of the value of a control variable, where metastability is possible, and the use of it. This delay ensures that any metastability would have settled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

In the case of Figure 3, the reader process must thus be implemented so that the time span between $r0$ and rd is large enough for any metastability at r to settle and that between $r1$ and rd is large enough for any metastability at v to settle. Similar requirements must be put on the writer process. The Simpson 4-slot mechanism operates correctly if all metastable signals have settled when they are used, regardless of the actual value they have settled to [4, 5]. On the other hand, if the low probability event of metastability not having settled when the control variable is used happens, data coherence is sacrificed to maintain full temporal independence of the access processes. This is because the system is assumed to operate in *fundamental mode*, with the timing of the various statements having 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 corrupt data may have been written/read.

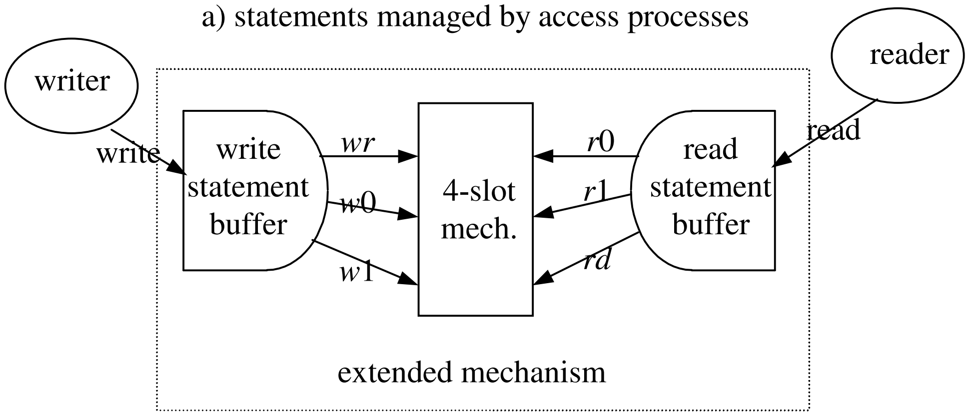
This method puts little extra time/speed burden on the writer and reader processes in practice if the mechanism hardware is much faster than the individual statements in the writer and reader processes. In the case of these being statements in software in conventional microprocessors and the communication mechanism 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 even their private RAM increases the probability of settling of any metastable value.

The algorithm implies that the mechanism interfaces with each access process three times in each cycle. In practice buffer devices can be employed to manage the three interface points while the access processes, possibly running in remote processors, make only one interface with the extended mechanism once per cycle. These ideas are shown in Figure 5.



a) statements managed by access processes



b) statements managed by statement buffers

Figure 5 Statement management buffers simplifying the interfaces.

It must be noted that implementations according to Figure 5 b) necessarily 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 mechanism 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 mechanism the time delays become significant and they must be carefully considered when analysing the temporal behaviour of the access processes.

Recently the need for building multiple processors on the same chip has become apparent in many different applications. In this situation both 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 be more hardware oriented problems, and the system response time requirements may not allow fixed delays for each statement just to cope with possible metastability.

3 A modified 4-slot mechanism with full hardware implementation employing self-timed circuits

An ACM design using self-timed circuits has been carried out. Based on the 4-slot mechanism and respecting the algorithm of Figure 3, it employs mutex elements to protect the circuits implementing the statements and concentrate metastability on two points. The statement circuits are entirely self timed and 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 6.

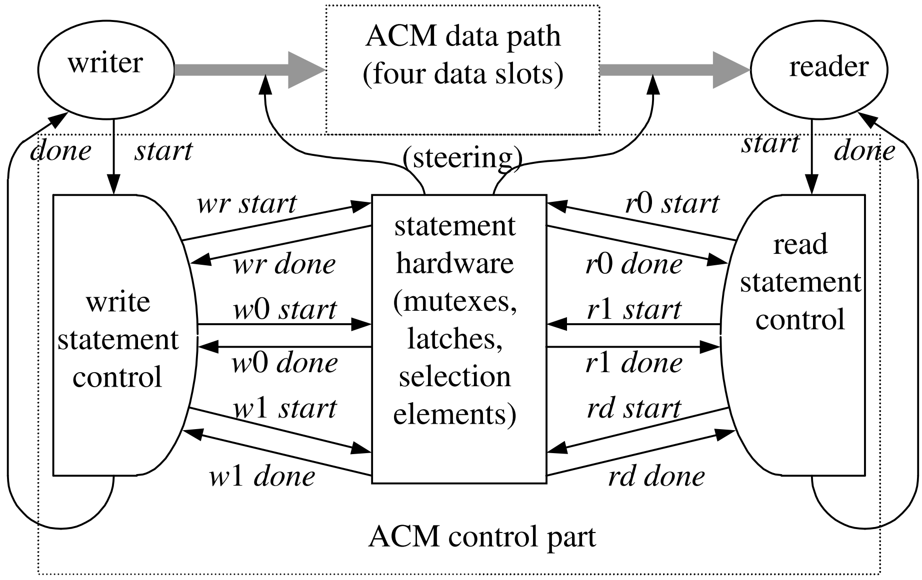


Figure 6 Basic structure of modified four-slot ACM with self timed circuits.

The system includes reader and writer interface control logic, statement circuits (both control variables and data slot selection/data path steering) and mutex elements. The signal sequencing of the write statement control is specified in STG form in Figure 7, assuming four-phase protocol which is adopted in this design. This ensures the statement sequencing specified by algorithm of Figure 3. The read statement control has essentially the same STG with the appropriate reader statements substituted in.

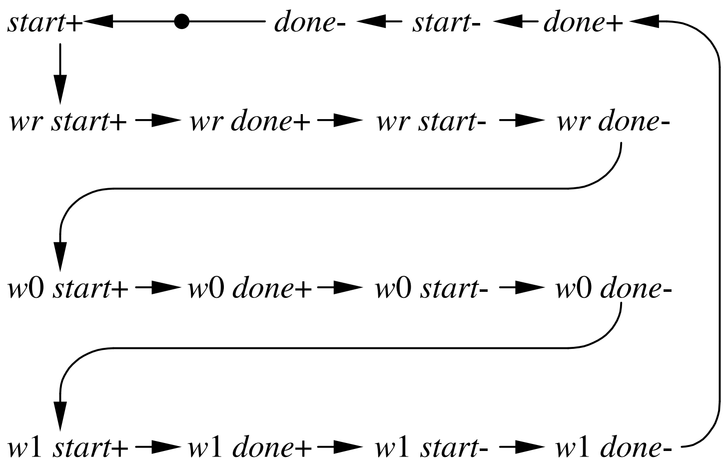


Figure 7 STG specification of write statement control.

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.

3.1 Mutex/arbiter circuits used to localise metastability

Implementing fixed delays based on the metastability characteristics of the hardware technology is not an efficient way of doing things 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 make use of self-timed, speed independent circuits so that waiting/delay is only invoked when metastability occurs. This would provide, on the average, a much faster performance [6].

The capability of finding out whether metastability, if any such should occur, has settled, is of utmost importance in such a self-timed design. This is easy to obtain by using arbiters with metastability detectors (sometimes known as metastability resolvers) [7]. One simple circuit which may serve this purpose is shown in Figure 8.

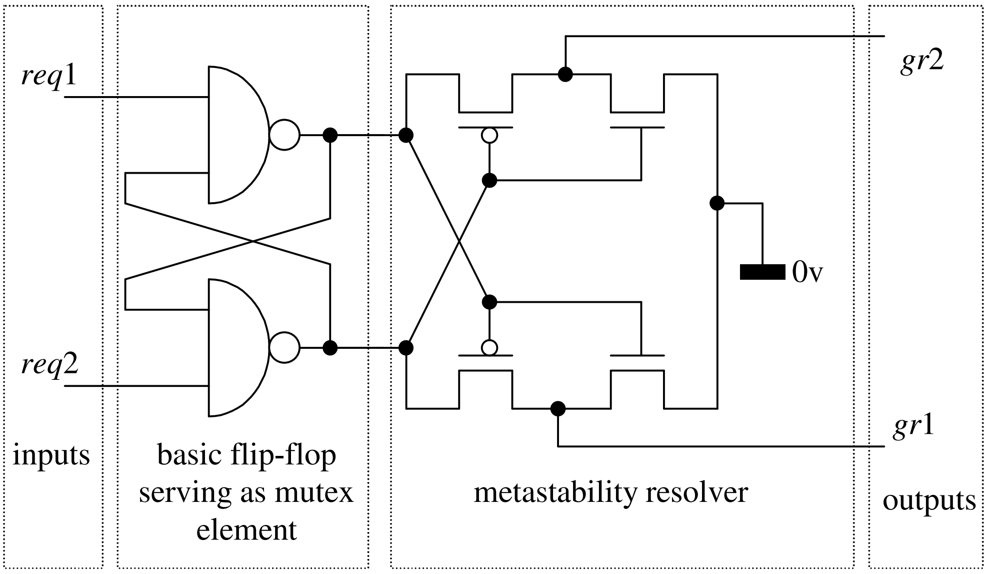


Figure 8 Arbiter with metastability resolver.

For the circuit in Figure 8, the intermediate outputs of the flip-flop can indeed become metastable if the inputs arrive at approximately the same time. The outputs of the metastability resolver, however, will not change until this metastability has settled. If properly reset and arranged with "spacer" states an arbiter constructed from this or based on a similar concept 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 speed independent (SI) solution may be obtained where metastable signals are never made use of.

There are three possible metastability points in the algorithm in Figure 3 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 $r1$. The first two points involve possible simultaneous happening of the $w1$ and $r0$ statements and the last the simultaneous happening of the $r1$ and $w0$ statements.

If two statements being carried out simultaneously is the cause of a metastable signal then by protecting these statements with a mutex/arbiter the metastability could be avoided. This is the concept used in this paper.

Specifically, in [1], one possible implementation of the statements $r0$ and $w1$ is shown in Figure 9, which is essentially a "shifting register". If the clock pulses $w1$ and $r0$ are generated by a mutex element in the form of Figure 8 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 mutex element, only when it has settled would one of the clock pulses be generated.

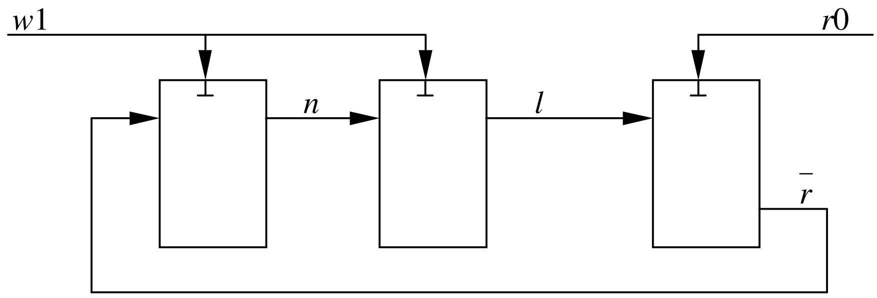


Figure 9 A simple implementation of statements $w1$ and $r0$.

This is schematically shown in Figure 10, where the statement starting signals $w1$ and $r1$ must go through a mutex element before actuating the statement hardware.

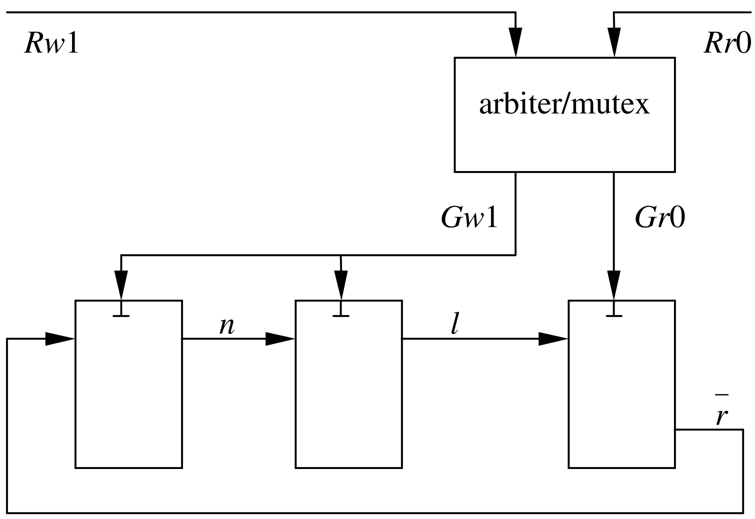


Figure 10 Mutual exclusion between $w1$ and $r0$.

Since mutex elements mandate waiting for the side that lost arbitration, the temporal relation between two processes arbitrated by such an element does not conform to the "full asynchronism" described in [1].

3.2 Self-timed circuits for the statements

In this design, we take the approach given in Figure 5 b) with self-timed circuits employed in the write and read statement buffers. This requires that a sequential arrangement of two statements be secured by control elements so that the second statement starts only when the first 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 7. This is shown in Figure 11.

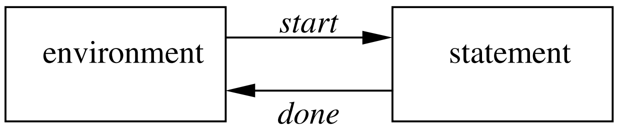


Figure 11 Handshake interface between a statement and its environment.

The control variable assignment statements $w0$, $w1$, $r0$ and $r1$ are implemented by SI latch circuits within the ACM which contain completion signals. These latch circuits are reported in [8]. They fully support the handshake interface protocol of Figure 11.

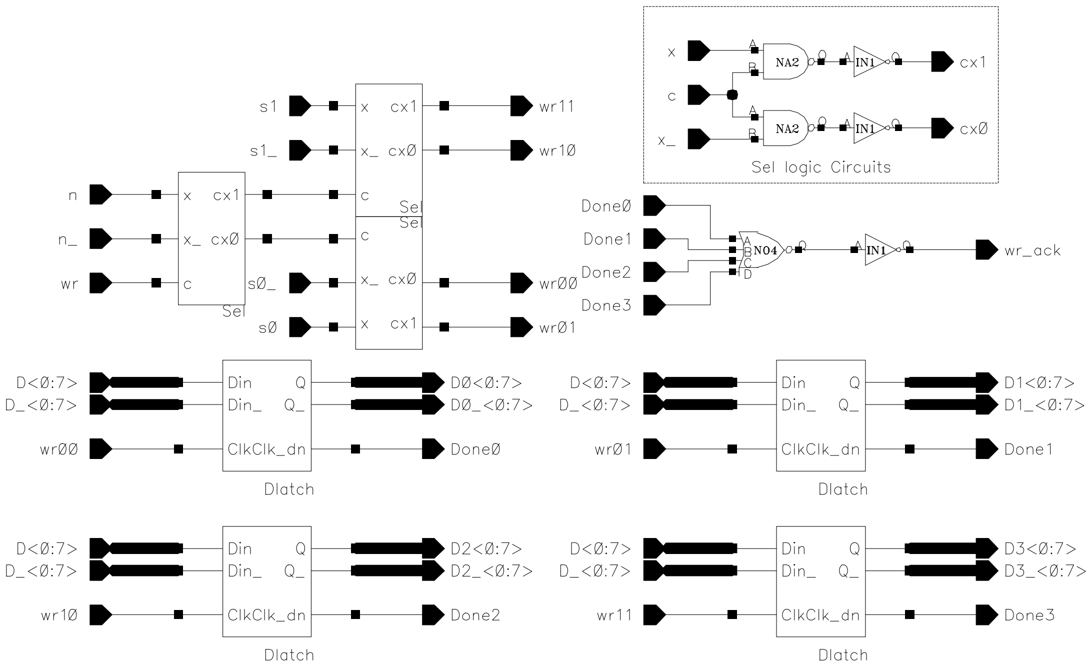


Figure 12 Hardware for statement *wr*.

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 12. That of *rd* is similar.

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 7.

The completion of *wr* is drawn from the data path in this implementation, which is conservative in terms of self timed 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.

3.3 Statement control elements

The write statement control element is specified by the STG in Figure 7. In order to retain an element of regularity and extendibility, a circuit known as David's element or David's cell [9], is chose as the building block with which to assemble this circuit.

The David's cell consists of a flip-flop and a NAND gate and is completely speed independent. 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 valid output (0,1 in the dual rail wires) and the others have "spacers" (0,0 in the wires), the circuit shown in Figure 13 would produce an STG shown in Figure 14.

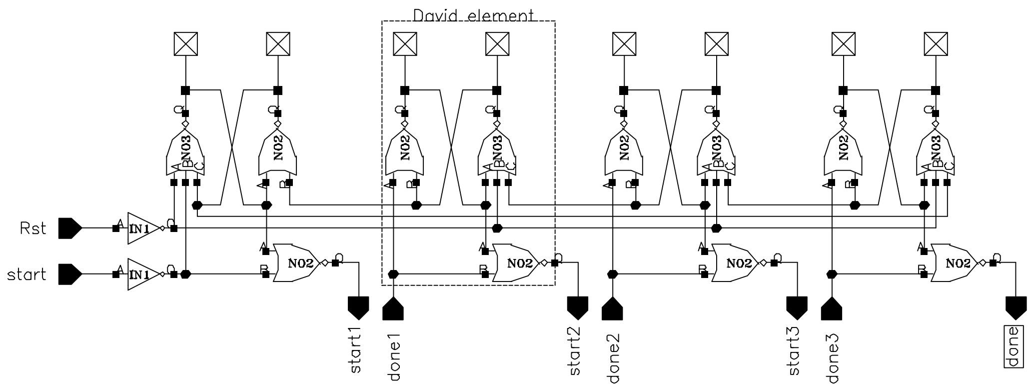


Figure 13 Write statement control logic using David's cells.

In Figure 14, 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.

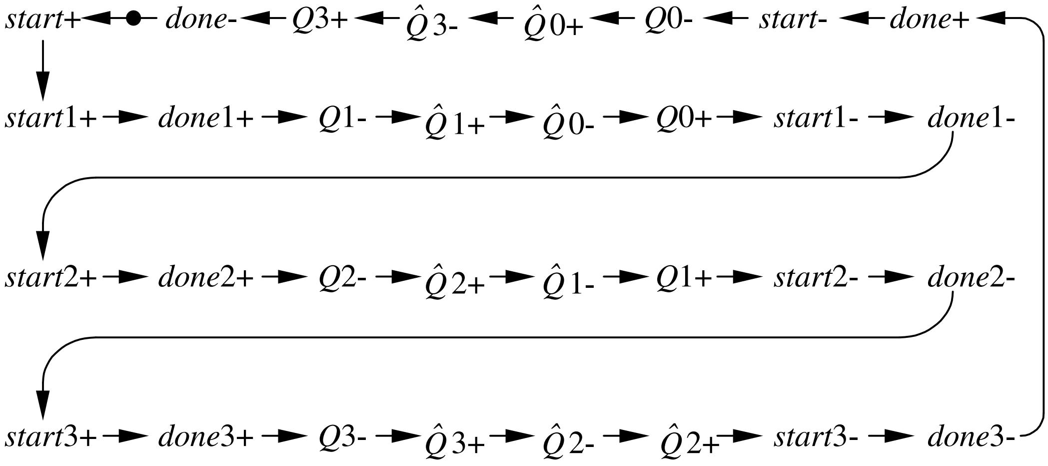


Figure 14 STG of the circuit in Figure 13.

4 Differences in temporal relations between FM and self timed solutions

The fundamental mode four-slot solutions proposed in [1] provide full asynchronism for the reader and writer in the sense that the temporal characteristics of each side are fully determined within itself and is not related in any way to the operations of the other side. This is conceptually shown in Figure 15.

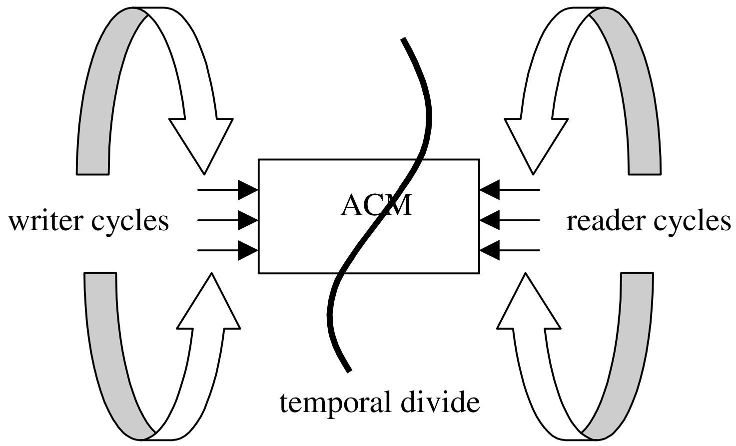


Figure 15 ACM providing temporal independence to processes.

In the self timed solution, there is certainly not an absolute temporal division between the reading and writing sides inside the ACM, because of the waiting required by the mutex/arbiter elements. 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 four-slot mechanism algorithm, the broad idea of realising atomic data transfer by using safe bit registers is retained.

On the other hand, if temporal independence is paramount, it should be between the reader and writer processes and not between the internal read and write sides of the

ACM. In other words, it is not necessary to provide this temporal independence down the middle of the ACM itself, so to speak, as the FM solution does. So long as it is possible for the reader and the writer processes to maintain independent timing with regard to the other side, the temporal division can be located anywhere between them.

From Figure 6, it is clear that there are two pairs of handshakes where such temporal divisions can be maintained. 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 self timed fashion. Still realising the potential of speeding up the response provided by the self timed 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 can be trivially shown that in a single cycle of operation only one of the points of arbitration would 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 [10], where it is demonstrated that 5ns is sufficient time for all metastability to have settled firmly in modern CMOS technology with "practically" probability 1.

5 Circuit implementation, simulation results, and further discussion

The overall ACM design has been put through the VSLI design flow using Cadence tools, and 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 especially have established that metastability does not propagate throughout the system but is contained within the mutex elements. Layout detailing is being carried out and further analogue simulation with the help of parameters extracted from layout information will be the next stage. The target of this work is the eventual fabrication of testable ACM circuits for further, hardware-based, testing and prototyping.

This version of the four-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 four-slot system proposed in [1], which may forcibly ensure timing independence 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 control and the writer and the write statement control.

Intuitively, 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* [1] is sacrificed. This is not a real sacrifice because when data coherence is not maintained data freshness becomes automatically meaningless.

Also intuitively, 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.

However, 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.

This work is supported by the EPSRC Comfort project at Newcastle University. The authors wish to thank Alex Bystrov for extensive discussions and practical help with Cadence tools.

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

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

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

5 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.

6 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.

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

8 Bystrov, A., Shang, D., Xia, F. and Yakovlev, A., “Self-timed and speed independent latch circuits”, elsewhere in this forum.

9 V. Varshavsky 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).

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