Communications Protocols and Mechanisms for Distributed Digital Systems

Anthony C Davies
Visiting Professor, School of Computing and Information Systems,
Kingston University, Penrhyn Road, Kingston-upon-Thames, Surrey, KT1 2EE, England.
tonydavies@ieee.org

Abstract: Synchronous clocking has continued to be the dominant digital design method despite the problems of clock distribution in integrated circuit chips of increasing complexity and speed. The continuing increases will soon force a change to asynchronous design methods, and the communication between large numbers of high-performance processors on a single chip will become a critical issue. An introduction to a classification scheme of mechanisms for interprocess communications via shared memory is described in the paper.

Keywords: Distributed real-time multiprocessors, systems on a chip, asynchronous design, interprocess communications and mechanisms.

1. Introduction

Synchronous methods have dominated digital design for many decades, despite the great increases in complexity and speed of modern integrated circuits. Realistic predictions indicate a fabrication-capability within the next decade of a single-chip with hundreds of processors of ‘Pentium$^{®}$’ complexity of and clock-rates an order of magnitude greater than at present. It is unlikely that a common clock could be distributed over such a chip and so centrally-clocked synchronous designs will no longer be feasible, and some form of asynchronous approach will be essential.
An application area for which a common clock has never been feasible is distributed multiprocessor systems where the parts are in relative motion. Typical of these are military weapons systems, where some processors may be in a static ground-based component, some may be in moving vehicles and others in rapidly moving missiles, all cooperating within a single operating system. Design techniques for such systems (for example the MASCOT method [1]) might contribute concepts useful in future system-on-chip (SOC) designs.

2. Suggested architectures

In meeting the challenges for SOC design, many architectures have been proposed, one of the most developed being ‘Globally-asynchronous, Locally-Synchronous’ (GALS) structures. They comprise synchronous ‘islands’ with asynchronous communication between the ‘islands’. The GALS ‘islands’ can use

pre-designed components (microprocessors, DSP cores, etc) which may be purchased as re-usable IP (intellectual property), either with or without designer-access to the internal structure. The GALS approach enables full use to be made of existing design tools and skills for the individual synchronous processors.
Another proposal is the 'Network on a Chip' approach, for which the processors may communicate by the TCP/IP protocol. The Internet has demonstrated the success of heterogeneous, extendable, multiprocessor systems with no common clock.
Biological systems may offer some ideas, since they use massive parallelism, complex communications paths (e.g. neural systems), and can achieve spectacular performance, including very fast pattern recognition despite being constructed from inherently slow (e.g. electrochemistry-based) components. They clearly have no resemblance to a network of classical von Neumann style processors, and much research is needed before artificial systems of comparable performance can be conceived and implemented.

3. The need for inter-process communication

The semiconductor industry will be able to provide a fabrication capability for at least hundreds of high-speed high-performance processors on a single chip, and the design of the individual processors is a 'solved problem', albeit a complex one requiring advanced digital design automation tools and substantial engineering skills. An essential issue for these future products is to design reliable and effective methods of inter-process communication (and communication across external interfaces, from sensors and to actuators). The processes will typically be running on different processors, but there may also be time-multiplexing of processes by multi-tasking.

3.1 The 'Hardware' context

Transferring data between unsynchronised processors and capturing external data in real-time typically involves clocking the data into some form of latch. To guarantee proper operation, data set up and hold times have to be complied with, but since clock and data are from independently-timed domains, it is inevitable that occasionally these timing requirements will be violated. Such violations can lead to metastability [2,3], a non-linear dynamical phenomenon which can result in occasional excessively-long settling times at gate and flip-flop outputs, during which the output level may remain at a mid-value (neither logic 1 or logic 0).
Arbiters are used to control access by multiple processes to shared resources, and have been shown to be inherently subject to metastability, and Analogue-to-Digital conversion within a fixed conversion time also involves a risk of metastability [4,5].
However, with good design practice, the probability of error due to metastability can be reduced to an acceptably low level.

3.2 The 'Software' context

Multiprocessing software (whether by task switching on a single processor or by separate executions threads on each of many processors, or some combination of these) has been a source of many challenging problems, with a need to avoid deadlock, livelock, and ensure fairness in the allocation of resources to processes. Real-time operating-systems rely on successful solutions to these problems, but as is well known, correctness of design is often not achieved, and ‘system-crashes’, requiring a re-initialisation of the whole system, are a familiar occurrence. While this may be acceptable in a word-processor or computer-game, it is not acceptable in a safety-critical hard-real-time context (for example, in the control of road vehicles, robots and avionics). In a hard-real-time framework, the possibility that one process may have to ‘wait’ for another to be ready (as in the Ada Rendezvous [6]) can be a cause of loss of potential performance, and can even lead to failure of the complete system when only one process fails (if the other processes are all waiting directly or indirectly for this failed process to respond).

3.3 ‘Message passing’ versus ‘Shared Memory’

There are two distinct approaches to communication between processes.
One approach is to have a ‘channel’ along which messages are passed, and which may involve an acknowledgement being returned to the sender. The OCCAM language developed from CSP [7] for the Inmos Transputer represented all interprocess communication by channels, which corresponded with the hardware ‘links’ of the Transputer.
The other approach is to use a shared memory area which can be accessed by each process. The sender places the data in the memory, and the receiver subsequently reads it. This method has a successful track-record in hard-real-time systems.

4. Classification of Communication between processes

The data items to be communicated are assumed to be multi-bit objects (for example an array of several elements or a record of several fields). Correct behaviour requires *coherence* (the reader must always obtain a coherent item, as opposed to one which is partly ‘old’ and partly ‘new’), *sequentiality* (the reader must obtain items in the sequence written) and *freshness* (the reader must obtain the most recently available item) [8, 9].
The original MASCOT method divided communications between a ‘writer’and a ‘reader’ into two classes, achieved by mechanisms called Channels and Pools. The Channel is used when the reader must obtain, in sequence, every item written. The Pool is used when the reader should obtain the most recent (up to date) item written, but does not then need to know the earlier items. In the Channel, the write instruction is non-destructive and the read operation is destructive. In other words, once the reader has read an item from the Channel, it does not need to remain in the Channel and may be discarded, whereas the writer must not overwrite or remove any items which are already in the Channel waiting to be read.

For the Pool, the opposite holds. The reader accesses to the Channel should provide the same item until the writer has produced a new data item, but as soon as this new item is available, the previous data is no longer valid. It follows that it is legitimate (and indeed usual) for many items to be written and never read.

4.1 Algorithmic description of an Asynchronous Communications Mechanism [10]

To introduce the idea of an Asynchronous Communications Mechanisms (ACM), a two-slot Pool mechanism is first described. Writer and reader may be assumed embedded in endless loops, in which the writer prepares or obtains a new data item and writes, and the reader independently reads and uses the new data item. Thus, for the sending entity:

```
loop
   obtain new data item
   writer_process              /* writes the data item */
   carry out other required tasks
endloop
```

and for the receiving entity:

```
loop
   reader_process              /* gets the freshest data item
*/
   use the new data item
   carry out other required tasks
endloop
```

The reader and writer processes might be implemented as follows:

Writer Process:

```
w1:   writepointer := not writepointer;
w2:   SLOT[writepointer] := input;
w3:   LASTWRITTEN := writepointer;
```

Reader Process:

```
r1:   readpointer := LASTWRITTEN;
r2:   output := SLOT[readpointer];
```

Uppercase identifier names denote shared variables. SLOT is a two-item array, addressed by a binary variable. The binary variables *writepointer* and *readpointer* are local to, respectively, writer and reader processes. Successive writes go to alternate slots, and *LASTWRITTEN* is used to inform the reader of the location of the latest write. Fig. 1 illustrates the concept.

Two slots are insufficient for correct operation unless timing-constraints are imposed on the writer and reader or some form of mutual exclusion exists between writer and reader – otherwise, for various timing relationships between writer and reader, it is possible for both to attempt to access the same slot at the same time, so this scheme cannot be used in a fully asynchronous environment. It can be shown that there is no possible fully-correct implementation with two locations, and none have been reported with three locations but various Pool mechanisms with four locations have been invented.

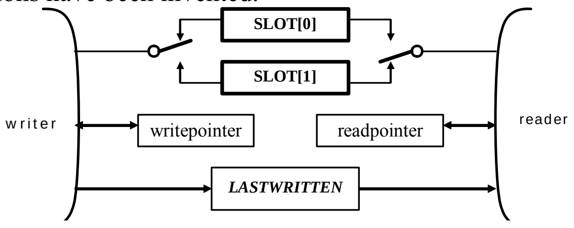

Figure 1 Concept diagram for two-slot mechanism

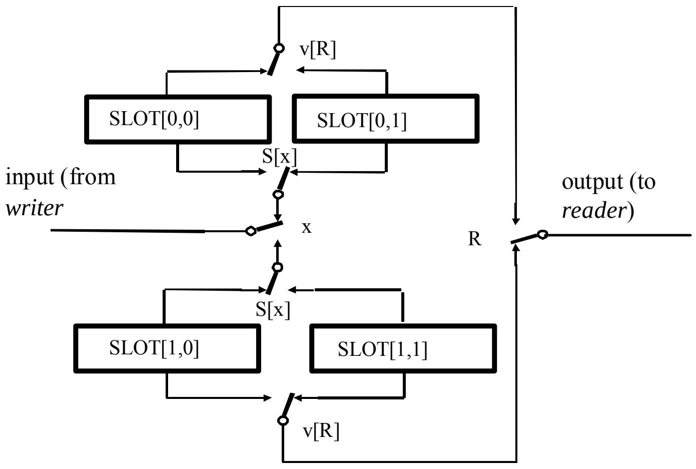

Figure 2 Concept diagram for Simpson four-slot mechanism

The mechanisms typically comprise slots for holding the data items in memory shared by writer and reader, and control variables to ensure that writing and reading is directed to the correct slot. As an example of a Pool ACM which is wait-free and operates correctly for any relative timing of writer and reader, a version of the four-slot scheme devised by Simpson [11] is next described. Fig. 2 illustrates the concept. This seems to have been the first realistic and correct solution proposed, which has been implemented as an ASIC and used in some real-time systems [12].

The four slots are arranged conceptually in two rows and two columns:

SLOT[*row*, *column*]

where *row*, *column* are each binary (0 or 1).

For control, a two-element shared binary valued array *S*[•] and a two-element binary valued array *v*[•] local to the reader are used, together with some other variables (*x*, *R*, *L*). As before, uppercase indentifiers denote the shared variables.

The basic idea may be described as follows:

Write to the opposite column to the previous write and avoid the row of the most recent read activity.
Read from the row, column of the most recently completed write

A compact form of the algorithm is as follows:

```
Writer Process:
w1:  SLOT[$x$, not $S[x]$] := input
w2:  $S[x]$ := not $S[x]$
w3:  $L$, $x$ := $x$, not $R$

Reader Process:
r1:  $R := L$
r2:  $v[0]$, $v[1]$ := $S[0]$, $S[1]$
r3:  output := SLOT[$R$, $v[R]$]
```

4.2 Waiting properties

There are some inherent properties of a Channel concerning the need for the processes to wait. If the channel is empty, the reader must wait until the writer places an item in the Channel. The writer has to wait if the Channel is full.

By contrast, there is no essential requirement for either side to wait with a Pool. The reader can always read and the writer can always write. It is, of course, not possible for them to simultaneously access the same memory location, and it might therefore seem that any implementation would require an arbiter to handle this possibility, with the result that waiting would occasionally still be required. However, at the expense of some added complexity, mechanisms can be designed for which the writer and reader are always able to access a location and the reader always obtains the 'correct' data item, Figure 2 being a specific example. Although shown in the form of mechanical switches, an implementation using conventional digital logic can easily be designed [14]. To generalise, the Signal and Constant were added [14], to cover all possibilities (see Table I). The detailed description, their properties, and further generalisations are given elsewhere [15,16, 17].

	destructive reading	non-destructive reading
destructive writing	***signal***	***pool***
non-destructive writing	***channel***	***constant***

Table I

Conclusions

This paper provides a tutorial introduction to some fundamental concepts which can assist in the design and implementation of multiprocessor real-time systems, and which may be important for the realisation of the complex designs which will inevitable be needed as the complexity of digital integrated circuit chips continues to increase dramatically.

Acknowledgements

Research at University of Newcastle-upon-Tyne and initially at King's College London and then Kingston University has been investigating the design and implementation of communications mechanisms for asynchronous and heterogeneous distributed digital systems, and also the development of design tools for such systems.

The U.K. EPSRC is thanked for financial support (Grant Nos. GR/L92471[†] and GR/R32895[††]). I also thank colleagues at the University of Newcastle-upon-Tyne (Ian Clark, Alex Yakovlev and David Kinniment) and at MBDA UK Ltd. Stevenage *(Matra BAe Dynamics, formerly British Aerospace Dynamics)* (Hugo Simpson and Eric Campbell) for many useful discussions.

[†]see `http://async.org.uk/comfort` [††]see `http://async.org.uk/coherent`

References

[1] Simpson, H.R. 'The Mascot Method' *IEE Software Engineering Journal*, 1986,.1, 103-120

[2] Seitz, C.L. 'System Timing' in Mead C. and Conway, L. *Introduction to VLSI Systems*, Addison Wesley, 1980, 218-262

[3] Davies A.C. 'Metastability in Latches, Arbiters and Data Converters', *Proc. Int. Symp. on Signals, Circuits and Systems (SCS'99)*, 6-7 July 1999, Iaşi, Romania, 1-4

[4] Chaney T.J., Molnar C.E. 'Anomalous Behaviour of Synchroniser and Arbiter Circuits', *IEEE Trans*, 1973, C-22, 421-422

[5] Kinniment D.J., Yakovlev A. and Gao B. 'Metastable behaviour and system performance' *Proc. 2nd UK Forum on Asynchronous Systems*, Department of Computing Science, University of Newcastle upon Tyne, July 1997

[6] Gehani, N.H., Roome, W.D. 'Rendezvous facilities: Concurrent C and the Ada language', *IEEE Trans. on Software Engineering* 1988 14, (11) 1546-1553

[7] Hoare. C.A.R. 'Communicating sequential processes'. *Comms. of the ACM,* 1978, 21, (8). 666-677

[8] Clark I.G., Davies A.C. 'A Comparison of some wait-free Communications Mechanisms', *Proc. Wksp. on Asynchronous Interfaces: Tools, Techniques and Implementations (AINT'2000)*, 19th-20th July 2000, Delft, Netherlands, 23-29.

[9] Simpson, H.R. 'Freshness specification for a class of asynchronous communication mechanisms' *IEE Proc: Computers and Digital Techniques,* 2004, 151 (2) 110-118

[10] Davies, A.C. 'Reasons, Protocols and Mechanisms for Communicating Asynchronously between Digital processes', *Proc. Int. Symp. on Signals, Circuits and Systems (SCS 2001)*, 10-11 July 2001, Iaşi, Romania, 1-10

[11] Simpson H.R. 'Four-slot fully asynchronous communication mechanism', *IEE Proc, Part E, Computers and Digital Techniques*, 1990, 37, 17-30

[12] Campbell E. ‘DIA temporal characteristics and their experimental verification’, British Aerospace Defence Dynamics Ltd., Univ. of York, Admiral Management Services, 1992

[13] Davies A.C., Clark I.G. ‘Asynchronous Communication without waiting: from the concept to the hardware’, *Proc. Int. Conf. on Applied Electronics*, Plzen, 5-6 Sept 2001, 55-59

[14] Simpson H.R. ‘Protocols for process interaction’, *IEE Proc: Computers and Digital Techniques*, 2003, 157-182

[15] Simpson H.R. ‘Correctness analysis for class of asynchronous communication mechanisms’, *IEE Proc, Part E, Computers and Digital Techniques*, 1992, 139, 35-49

[16] Simpson H.R. ‘New algorithms for asynchronous communication’, *IEE Proc, Part E, Computers and Digital Techniques*, 1997, 144, 227-231

[17] Simpson, H.R. 'Multireader and multiwriter asynchronous communication mechanisms', *IEE Proc. Computers and Digital Teqchniques*. 1997, 144, (4), 241-243