TOWARDS A FRAMEWORK FOR THE DISTRIBUTED SIMULATION OF ASYNCHRONOUS HARDWARE

G. THEODOROPOULOS

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
United Kingdom
gkt@cs.bham.ac.uk

D. EDWARDS

Department of Computer Science
University of Manchester
Oxford Road, Manchester, M13 9PL
United Kingdom
doug@cs.man.ac.uk

Abstract: Synchronous VLSI design is approaching a critical point, with clock distribution becoming an increasingly costly and complicated issue and power consumption rapidly emerging as a major concern. The last decade has witnessed a resurgence of interest in asynchronous logic which promises to liberate digital design from the inherent problems of synchronous systems. This activity has revealed a need for modelling and simulation techniques suitable for the asynchronous design style. The concurrent process algebra Communicating Sequential Processes (CSP) is particularly suitable for the specification of asynchronous systems. This paper discusses a framework for the distributed simulation of asynchronous hardware, adopting Balsa, a CSP-like notation, as a hardware description language.

Keywords: Asynchronous hardware, modelling, distributed simulation

1. INTRODUCTION

A digital system is typically designed as a collection of subsystems, each performing a different computation and communicating with its peers to exchange information. Before a communication transaction takes place, the subsystems involved need to synchronise, namely to wait for a common control state to be reached, which guarantees the validity of data exchanged. In synchronous systems, this synchronisation is achieved by means of a global clock whose transitions define the points in time when communication transactions can take place. The operation of a synchronous system proceeds in lockstep, with the different subsystems being activated to perform their computations in a strict, predefined order. Synchronous VLSI design however is approaching a critical point, with clock distribution becoming an increasingly costly and complicated issue and power consumption rapidly emerging as a major concern.

Another digital design philosophy, which promises to alleviate these problems, allows subsystems to communicate only when it is necessary to exchange information. The operation of the system does not proceed in lockstep, but rather is asynchronous; each sub-system operates at its own rate synchronising with its peers only when it needs to exchange information by means of a handshake communication protocol (such as the two-phase bundled data handshake protocol illustrated in Figure 1). The last decade has witnessed a resurgence of interest in asynchronous design techniques [Birtwistle,1995] and a number of asynchronous architectures have been developed [Werner,1997], including a series of asynchronous implementations of the ARM RISC processor (AMULET1, AMULET2e and AMULET3i) developed by the AMULET group at the University of Manchester [AMULET,2001].

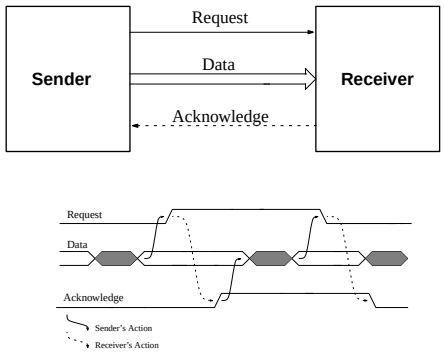

Figure 1: Handshake Protocol

This paper addresses issues related to the modelling and distributed simulation of asynchronous hardware systems.

2. MODELLING ASYNCHRONOUS SYSTEMS

Simulation modelling languages and tools for synchronous logic design have underpinned the development of ever more complex synchronous VLSI circuits. In the case of asynchronous systems, the role of simulation is even more crucial as their concurrent, non-deterministic behaviour makes any attempt to reason about their correctness and performance a very complicated task. This complexity

renders modelling and simulation essential tools in the endeavour to gain an insight and understanding of the behaviour of asynchronous systems. However, although synchronous languages and tools can and indeed have been used for asynchronous hardware too, their application in that context is proving awkward and inefficient. Fundamentally, conventional, sequential, synchronous hardware description languages are not suitable for describing concurrent non-deterministic asynchronous behaviour. Thus, the recent interest in asynchronous design has fuelled an intense research activity aiming to develop techniques appropriate for modelling and simulating asynchronous systems. I-Nets [Molnar,1983], Petri Nets [Cortadella,1997], Signal Transition Graphs [VERSIFY,1998], State Transition Diagrams [Davis,1995], and CCS [Liu,1993] are some of these tools and formalisms that have been employed in asynchronous logic design.

Communicating Sequential Processes (CSP) [Hoare,1985], in particular, the concurrent process algebra developed by Tony Hoare for the specification of parallel systems, has been extensively advocated as a suitable means for describing asynchronous behaviour. Several asynchronous modelling approaches and systems have been developed which use CSP-based notations, including [Martin,1990; Hulgaard,1994; Brunvand,1991; Dill,1989; Josephs,1990; vanBerkel,1991; Gopalakrishnan,1993].

In the context of the AMULET work at the University of Manchester, two different independent activities have used CSP-based notations for the modelling and simulation of asynchronous hardware: *Balsa*/*LARD* and *Occarm*.

3. BALSA AND LARD

Balsa [Bardsley,2000; Bardsley,2000a] is a language for synthesising asynchronous circuits. It builds on the Tangram work of the Philips Research Labs [vanBerkel,1991]. Both Tangram and Balsa use similar CSP like languages to express design descriptions in terms of channel communications and fine grain concurrent and sequential process decomposition. Descriptions of designs are translated into implementations in a syntax directed-fashion with language constructs being mapped into networks of parameterised instances of "handshake components" each of which has a concrete gate level implementation.

A Balsa description of a modulo-10 counter is given in Figure 2. Each procedure in Balsa corresponds to a handshake circuit (comprising basic handshake components). The interface to its environment is by means of the channels in the procedure declaration. In the example above, aclk is a dataless handshake channel, whereas count is an output handshake data-bearing channel (of width 4 bits).

Balsa designs can be simulated by translating (automatically) into the language LARD [Endecott,2001]. LARD provides mixed sequential and parallel commands with channel communication as a basic language feature. LARD resembles a concurrent programming language, being built around a shared memory, time-slice multi-threaded virtual machine.

```
type C_size is 4 bits
constant max_count = 9

procedure count10(sync aclk; output
count: C_size) is
 variable count_reg : C_size
 variable tmp : C_size
 begin
  loop
   sync aclk;
   if count_reg /= max_count then
     tmp := (count_reg + 1 as C_size)
   else
     tmp := 0
   end || count <- count_reg ;
   count_reg := tmp
  end
 end
```

Figure 2: A Balsa Example (Modulo-10 Counter)

Communication primitives are built on top of this machine and function by signalling data validity through an element of a shared memory structure. On top of LARD, an expanded channel library is used to provide pull channels and pull channel enclosure primitives.

4. OCCARM: PARALLEL SIMULATION WITH OCCAM

The simulation of digital systems in general, and computer architectures in particular, has long been categorised among the highly computation intensive applications. In the case of asynchronous systems, the requirement of multiple executions of the simulation model of the system, in order to test the system for deadlocks and evaluate its performance for different sets of delays in the component sub-systems [Furber,1995;Theodoropoulos,2001b] render simulation speed even more crucial. Although the primary initial motive for using CSP is the need to capture and model the concurrent, asynchronous, non-deterministic behaviour of asynchronous hardware, and almost all work undertaken so far in this area has placed emphasis on producing specifications to be used as input to silicon compilers rather than for parallel simulation, the exploitation of the inherent parallelism and the execution of the CSP models on multiprocessor platforms can potentially achieve high simulation performance and can contribute in reducing the duration and cost of the design cycle.

Asynchronous hardware systems are an excellent candidate for distributed simulation. The concurrent operation of the different subsystems of an asynchronous system, the inherent parallelism within each subsystem and the lack of any global synchronisation, are characteristics which support the concurrent execution of events in a simulation model. In his *flashback simulation* approach [Sutherland,1993],

Sutherland attempts to exploit these characteristics of asynchronous systems and allow “out-of-order" processing of events to increase simulation speed; however, his simulation retains its sequential nature, and is intended for execution on conventional von Neumann computers.

To exploit the potential of CSP for distributed simulation, the use of *occam* [Inmos,1988], the executable counterpart of CSP, for the modelling and parallel simulation of complex asynchronous architectures has been investigated in the context of the AMULET work. As part of this project, a generic modelling methodology has been proposed [Theodoropoulos,2000]; *occarm*, an occam simulation model of the AMULET1 asynchronous microprocessor [Woods,1997] has been developed [Theodoropoulos,1995]; and issues related to synchronisation [Theodoropoulos,2001a] and the execution of the occam models on transputer platforms [Theodoropoulos,2001b] have been addressed.

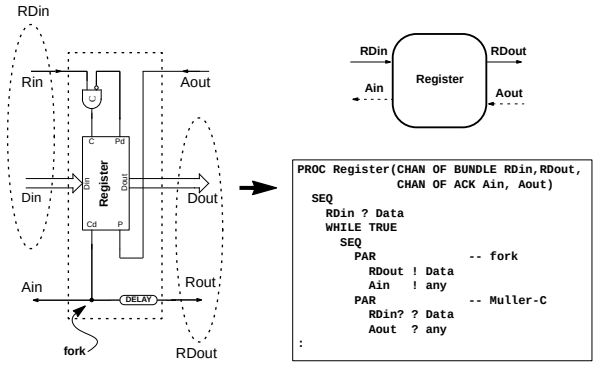

Figure 3: Occarm: The Register Model

Occam is primarily a general purpose programming language, and thus, a specification developed using occam is automatically an executable simulation model of the asynchronous system. No extra simulation engine is required. Within occarm, the system is modelled as a a network of concurrent occam processes, topologically identical to the asynchronous system, with each occam process corresponding to a different functional module of the system, and communicating with its peers via timestamped messages. Each register is modelled as a separate occam process, while the control logic may be implemented as a network of communicating processes, with the occam PAR (parallel execution) and SEQ (sequential execution) commands being used within each process to implement the partial ordering of events of the circuit. Figure 3 illustrates the modelling of a register in occarm (micropipeline).

5. THE NEED FOR AN ALTERNATIVE PARALLEL SIMULATION APPROACH

A number of conclusions were drawn from the work with Occarm which suggest the need for an alternative approach to the distributed simulation of asynchronous hardware. A central element in this argument relates to synchronisation.

In the modelling and simulation of computer systems, simulated time plays a dual role: as a synchronising agent, to drive the simulation engine, and as a quantifier, to provide the means for the performance evaluation of the architecture.

The exploitation of the relationship between CSP/occam semantics and asynchronous hardware systems, assumes and implies a data-driven operation of the processes of the model. The execution of a model consisting of data-driven processes which describe the modelled hardware at the Register Transfer (or higher) level is straightforward. Since at these levels, the correct operation of the asynchronous hardware system does not depend on a global clock, simulated time is not required for the synchronisation of the CSP/occam processes. Processes can be entirely data-driven and self-scheduling, and be synchronised by the protocol employed in the communication semantics of occam, in the same way that the communication protocol employed in the asynchronous system synchronises the different functional modules.

This however is not true for models at lower levels of abstraction. One can certainly use occam to produce textual descriptions of gates and event processing blocks. However, the data-driven operation of an occam model whose concurrent processes model gates and event processing blocks with level sensitive inputs may lead to deadlocks or incorrect results . In this case, the simulated time is needed to act as the synchronisation agent in the model.

Even at the Register Transfer (or higher) level, however, simulated time is still needed as a quantifier, to provide for the performance evaluation of the simulated asynchronous system, but the exploitation of the relationship between CSP/occam and asynchronous hardware systems to perform simulation, trades temporal accuracy for ease of modelling. Although the topological characteristics of the modelled hardware system map naturally onto the model, the temporal characteristics generally do not, a situation that, in the presence of arbiters in the hardware system, may lead to causality errors in the model. These errors compromise the accuracy of any time-based evaluation of the simulated system. A quantitative analysis has shown that the timing error introduced in the model may be quite significant [Theodoropoulos,2001a].

The requirement to test the architecture for potential deadlocks by modifying the delays in the system to achieve different event orderings [Furber,1995; Theodoropoulos,2001b] renders the need for synchronisation even more intense: if the simulated time is not the synchronization agent in the simulator, different event orderings may only be achieved by using occam real time Timers to change the relative scheduling of the occam processes. In this case,

small delays cannot guarantee the intended effects and behaviour in the model, while large delays have a severe effect on the performance of the simulator.

In other words, the exploitation of the CSP/occam semantics facilitates the specification and modelling of asynchronous systems but does not ensure time-accurate distributed simulation. It is generally possible to introduce additional functionality in the occam model to ensure temporal accuracy, but this imposes an extra burden to the modeller and increases the complexity of the model, forcing it to depart from its original philosophy (namely, the exploitation of the CSP semantics).

The complexity of the model is further increased by the need to address monitoring, mapping and load balancing issues, which in the case of occam are all the responsibility of the programmer.

Thus, what is required, is an alternative approach, which would separate modelling and simulation: for the former it would support a CSP-like notation, while for the latter it would utilise a distributed simulation kernel, optimised for the simulation of asynchronous hardware.

As a modelling notation we propose Balsa/LARD. Balsa is demonstrably capable of describing complex hardware strucures such as the DMA controller for the AMULET3i asynchronous microprocessor macrocell [Bardsley,200b]. LARD is a language specifically designed for modelling asynchronous systems with an automatic translation route from the Balsa description.

The underlying simulation kernel will execute the model, taking care of partition, load balancing and synchronisation. The kernel should also provide facilities for monitoring the simulation execution, enabling the debugging and well as the evaluation of the performance of the simulated system.

6. PARTITIONING AND LOAD BALANCING

The envisaged underlying simulation kernel is a decentralised event-driven system based on the “Logical Process” paradigm [Fujimoto,2000]. This approach seeks to divide the simulation model into a network of concurrent *Logical Processes* (*LPs*), each maintaining and processing a disjoint portion of the state space of the system. State changes are modelled as timestamped events in the simulation. From an LP's point of view, two types of events are distinguished; namely internal events which have a causal impact only to the state variables of the LP, and external events which may also have an impact on the states of other LPs. External events are typically modelled as timestamped messages exchanged between the LPs involved.

It is envisioned that once the Balsa specification of the asynchronous system has been produced, it will be translated and partitioned in subsets of state variables which will be assigned to different LPs.

Translating from Balsa into LARD is reasonably straightforward and proceeds from the compiled handshake level description. A library of handshake component behavioural descriptions forms the functional core of each simulation. LARD provides type-polymorphic features which make the description of parameterisable handshake circuits simple, there is no need to create specific instances of the required components for each differing set of parameters. A minor inelegance in the current version arises because LARD's `int` type only allows 32 bit numbers to be represented and so to implement the multi-precision arithmetic of Balsa a different type must be used for simulated channels. The current mechanism for doing this however is somewhat inefficient.

Partitioning should aim to balance the communication and computation load of the simulation system [Norman, 1993]. Finding an optimal partition for a given circuit is an NP-complete problem [Johnson,1979] and various heuristics have been developed to address this problem [Lin,1970; Mattheyses,1982; Newton,1991;Guetaff,1998]. The applicability and suitability of these algorithms for asynchronous hardware should be further examined. In the context of the proposed approach, the partitioning algorithm should take into account the characteristics of the underlying simulation platform, the simulation kernel, and the simulated system (such as the critical path of a simulated microprocessor).

7. SYNCHRONISATION

In decentralised, event-driven distributed simulation, each LP maintains a local clock with the current value of the simulated time, Local Virtual Time (LVT). This value represents the process's local view of the global simulated time and denotes how far in simulated time the corresponding process has progressed. An LP will repeatedly accept and process messages arriving on its input links, advancing its LVT and possibly generating, as a result, a number of messages on its output links. The timestamp of an output message is the LVT of the LP when the message was sent. A fundamental problem in event-driven distributed simulation is to ensure that the LPs always process messages in increasing timestamp order, and hence faithfully and accurately implement the causal dependencies and partial ordering of events dictated by the causality principle in the modelled system.

It has been shown [Lamport,1978] that a distributed system consisting of asynchronous concurrent processes will not violate the causality principle if each *merge* process consumes and processes event messages in non-decreasing timestamp order (the *local causality constraint*; the arrival of an out of order event is referred to as a *preemption*).

At the Register Transfer level, three types of merging are encountered:

- **Synchronous merge**. A functional module has to wait for all input data to become available before it starts its operation. This is the case when a Muller-C element is used for the corresponding request events. In the simulation model, the LP has to wait for all input channels to ``fire" and therefore no preemption occurs as the process is in the position to select the message with the smallest timestamp.
- **Data dependent merge**. The functionality of the system dictates the order in which messages from different source processes should be consumed and processed. This situation is implemented in hardware using a combination of a Select and a Call or Xor. The LP in this case behaves as a single input module, hence causality is not violated.
- **Arbitrated merge**. The order of arrival defines the order of consumption. If events from two subcomponents arrive at the same time, an arbitrary choice is made. This is the case where causality errors may occur if the order in which messages arrive at the arbiter process in the model is not the same as the order of events in the simulated system.

At lower levels, gates and event processing blocks with level sensitive inputs complicate the synchronisation problem introducing more types of merge processes.

Two main approaches have been developed to ensure that the local causality constraint is not violated is asynchronous simulation, namely *conservative* and *optimistic* [Fujimoto,2000]. The good lookahead properties of asynchronous VLSI systems, and the static nature (partition and configuration) of the distributed model suggest that the conservative approach would be suitable and sufficient for the simulation of asynchronous hardware, rendering the substantial extra complexity of optimistic approaches unnecessary.

In the context of Occarm, the concept of *Instruction Lookahead* was introduced and the *Program Driven Synchronization Protocol* (*PDSP*), a conservative, deadlock avoidance framework was developed to address the synchronisation problem at the Register Transfer level [Theodoropoulos,2001a]. PDSP could form the basis for the synchronisation mechanism of the proposed simulation kernel, however, its generality as well as its applicability for models at lower levels of abstraction should be further investigated.

8. MONITORING

Monitoring the runtime behaviour of the simulation model and collecting information regarding the characteristics of the simulated system is one of the main objectives of the simulation process. Monitoring is essential for the testing and performance evaluation of the simulated system as well as for the debugging of both the simulated system and the simulation model. The inherent properties of distributed asynchronous systems make monitoring a difficult and complicated issue for which sequential techniques are insufficient. The fundamental problem is the difficulty to deal with causality and obtain snapshots in a distributed environment [Lamport,1978; Chandy,1985; Babaoglou, 1993]. To determine the system state, all the different local process as well as channel states need to be taken into account; the monitoring system should be able to correlate the histories of the different processes and put them in a global temporal perspective. The main issues which stem out of this fundamental problem and which an ideal distributed monitoring system should address include the multiple threads of control, intrusiveness, non-determinism and the need to cope with (i.e. generate, transport and analyse) a vast amount of monitoring data. For a detailed discussion of these issues the reader is referred to e.g. [Joyce,1987; Riek,1993].

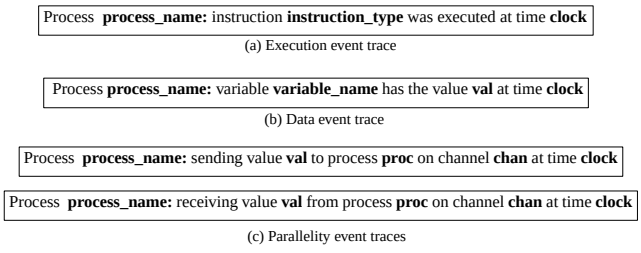

Figure 4: Event Traces

In the case of asynchronous hardware, the simulation has two main objectives, namely the testing and debugging of the system and the evaluation of the system's performance. For the debugging of the architecture (as well as the simulation model) it is necessary to monitor both the flow of control and the flow of data in each of the different LPs in the model; this could be achieved by collecting traces regarding the *execution* and *data events* of the processes respectively (Figure 4a,b). For the detection of deadlocks, it is essential to know the state of the channels in the system when the deadlock occurred. For this purpose, the *parallelity* events, which correspond to communication actions, need to be monitored (Figure 4c). These should appear in pairs, one for the sending and one for the receiving process. The absence of one parallelity event from a pair in the final trace indicates the occurrence of a deadlock.

A deficiency of the current LARD simulation system is that check-pointing is not supported. Since maintenance of state in a distributed environment is essential, adding check-pointing to the simulation environment is essential if the language is to be usable for large designs and it should come as a byproduct of the work envisaged here.

9. HLA COMPLIANCE

Traditionally, hardware systems are evaluated by (meta) executing benchmark suites on the simulation model. However, VLSI circuits such as microprocessors are increasingly used as embedded controllers, incorporated in other devices (e.g. in a car to control the car's brakes). This is particularly true for asynchronous systems as their inherent power-efficiency, electromagnetic compatibility and modularity renders them ideal for embedded applications [Furber,1997]. In such systems, the asynchronous controller interacts with a complex environment and needs to respond rapidly and predictably to events in the outside world.

For this type of applications, the traditional evaluation approach of mere benchmark execution appears insufficient, as in this case what would be desirable is the evaluation of the controller design within the environment that it will operate, by incorporating the simulation model in some hardware-in-loop simulation, or by enabling it to interact with a simulation model of the environment. Such an evaluation approach may be achieved through the High Level Architecture (HLA), the simulator interoperability framework developed by USA DoD/DMSO [HLA,2001]. It is therefore proposed that the simulation kernel should be HLA-compliant.

10. EPILOGUE

Asynchronous logic is being viewed as an increasingly viable alternative digital design approach which promises to liberate VLSI systems from clock skew problems, offer the potential for low power and high performance and encourage a modular design philosophy which makes incremental technological migration a much easier task. Modelling and simulation, being at the heart of the digital design process, may perform a catalytic role in the quest for the realisation of the potential benefitsoffered by asynchronous logic. The concurrent, non-deterministic behaviour of asynchronous digital designs renders simulation speed crucial.

Capitalising on previous experience using two different CSP-like languages, this paper has revisited the problem of distributed simulation of asynchronous hardware, and has argued for the need of an alternative approach, which separates modelling from simulation: for the former it adopts Balsa/LARD, a CSP-like asynchronous hardware description language, while for the latter it envisages the utilisation of a decentralised, event-driven simulation system based on the "Logical Process" paradigm. The paper has discussed issues that need to be addressed for the realisation of this approach.

REFERENCES

AMULET 2001, The AMULET Group, University of Manchester U.K. URL: www.cs.man.ac.uk/amulet/

Babaoglou O. and Marzullo K. 1993, "Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms", *Technical Report UBLCS-93-1*, Laboratory for Computer Science, University of Bologna.

Bardsley A. 2000, "Implementing Balsa Handshake Circuits", *Ph.D. Thesis*, University of Manchester.

Bardsley A. and Edwards D.A. 2000, "The Balsa Asynchronous Circuit Synthesis System", *Proceedings of Third International Forum on Design Languages*, Tubingen, Germany September, pp. 37-44

Bardsley A. and Edwards D. 2000, "Synthesising an Asynchronous DMA Controller with Balsa", *Journal of Systems Architecture*, December, pp. 1310-1319.

Birtwistle G. and Davis A. 1995, "Asynchronous Digital Circuit Design", Springer Verlang,

Brunvand E. and Starkey M. 1991, "An Integrated Environment for the Design and Simulation of Self Timed Systems", *Proceedings of VLSI 1991*, pp. 4a.2.1-4a.3.1.

Chandy K. M. and Lamport L. 1985, "Distributed Snapshots: Determining Global States of Distributed Systems", *ACM Transactions on Computer Systems*, 3(1), pp. 63–75

Cortadella J. et. al., 1997, "Petrify: A Tool for Manipulating Con-current Specifications and Synthesis of Asynchronous Controllers", *IEICE Transactions on Information and Systems*, E80-D(3), pp. 315-325.

Davis S. and Nowick M 1995, "Synthesizing Asynchronous Circuits: Practice and Experience", in [Birtwistle,1995], pp. 104-150.

Dill D. L. 1989, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits", *ACM Distinguished Dissertations*, MIT Press.

Endecott P. B. 2001, The LARD Documentation Home Page: URL: http://www.cs.man.ac.uk/amulet/projects LARD

Fujimoto R. 2000, "Parallel and Distributed Simulation Systems", John Wiley & Sons.

Furber S. B. 1995, "Computing Without Clocks", in [Birtwistle,1995], pp. 211-262.

Furber S.B. et. al., 1997, "Asynchronous Embedded Control", *Journal of Integrated Computer-Aided Engineering*, Wiley, Vol.5 No.1 pp.57-68

Gopalakrishnan G. and Akella V. 1993, "Specification, Simulation, and Synthesis of Self-Timed Circuits", *Proceedings of the 26th Hawaii International Conference on System Sciences*, pp. 399-408.

Guetaff A and Bazargan-Sabet P. 1998, "Using Node Replication to Improve Circuit's Partition in Distributed Logic Simulation", *Proceedings of the 12th European Simulation Multiconference*, SCS, June, Manchester, pp. 235-237

HLA 2001, High Level Architecture, Defense Modeling and Simulation Office, U.S. Department of Defense. URL: http://www.dmso.mil/hla

Hoare C.A.R. 1985, "Communicating Sequential Processes", Prentice Hall International.

Hulgaard H.and Burns S. M. 1994, “Bounded Delay Timing Analysis of a Class of CSP Programs with Choice”, in *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*.

Inmos 1988, “Occam 2 Reference Manual”, Prentice Hall International.

Johnson D.S. and Garey M. 1979, “Computer and Intractability: A Guide to the Theory of NP Completeness”, Freeman.

Josephs M. B. and Udding J. T. 1990, “Delay-Insensitive Circuits: An Algebraic Approach to their Design, in *Lecture Notes in Computer Science*, Vol. 458, pp. 342-366.

Joyce J. et al., 1987, “Monitoring Distributed Systems”, *ACM Transactions on Computer Systems*, 5(2), pp. 121 150.

Lamport L. 1978, “Time, Clocks and the Ordering of Events in Distributed Systems”. *Communications of the ACM*, 21(7), pp. 558–565

Lin S. and Kernigan W., 1970, “An Efficient Heuristic Procedure for Partitioning Graphs”, *Bell System Technical Journal*, Vol 49, pp. 291-307.

Liu Y. at al., 1993, “ Designing Parallel Specifications in CCS”, in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, Vancouver.

Martin A. J. 1990, “Synthesis of Asynchronous VLSI Circuits”, J.Staunstrup, editor, *Formal Methods for VLSI Design*, North Holland.

Mattheyses R. M. and Fiduccia C. M., 1982, “A Linear Time Heuristics For Improving Network Partitions”, *Proceedings of the 19th Design Automation Conference*, pp. 175-181.

Molnar C. E.and Fang T-P. 1983, “Synthesis of Reliable Speed-Independent Circuit Modules: I. General Method for Speci fication of Module-Environment Interaction and Derivation of a Circuit Realisation”, *Technical Report 297*, Computer Systems Laboratory, Institute for Biomedical Computing, Washington University, St. Louis.

Newton R. and Kring C. 1991, “A Cell-Replicating Approach to Mincut-Based Circuit Partitioning”, *1991 International Conference on Computer Aided Design*, pp. 2-5.

Norman M. G. and Thanisch P. 1993, “Models of machines and Computation for Mapping in Multicomputers”. *ACM Computting Surveys*, 25, September, pp. 263–302.

Riek M. Tourancheau B. and Vigouroux X. F. 1993, “Monitoring of Distributed Memory Multicomputer Programs”, *Technical Report UT-CS-93-204*, University of Tennessee, October

Sutherland I. E. 1993, “Flashback Simulation”, *Research Report, SunLab 93:0285*, Sun Microsystems Laboratories, Inc., August.

Theodoropoulos G. 1995, “Strategies for the Modelling and Simulation of Asynchronous Computer Architectures”, *Ph.D Thesis*, Department of Computer Science, University of Manchester, 1995.

Theodoropoulos G. 2000, “Modelling and Distributed Simulation of Asynchronous Hardware”, *Simulation Practice and Theory Journal*, 2000, 7:741-767.

Theodoropoulos G. 2001, “Distributed Simulation of Asynchronous Hardware: The Program Driven Synchronisation Protocol, Journal of Parallel and Distributed Computing, Special Issue on Distributed Simulation, editor C. Tropper, to appear.

Theodoropoulos G. 2001, “Simulating Asynchronous Hardware on Multiprocessor Platforms”, Concurrency Practice and Experience Journal, J. Wiley, to appear.

van Berkel K. 1993, “Handshake Circuits - An Asynchronous Architecture for VLSI Programming”, Cambridge University Press.

VERSIFY Release 2.0, Department d' Arquitectura de Computadors, Universitat Politècnica de Catalunya, Barcelona, Spain, November 1998, URL: http://www.ac.upc.es/vlsi/versify/

Werner T. and Venkatesh A. 1997, “Asynchronous Processor Survey”, *IEEE Computer*, 30(11), pp. 67-76.

Woods J.V et al. 1997, “AMULET1: An Asynchronous ARM Microprocessor”, *IEEE Transactions on Computers*, 46(4), pp. 385-398.

AUTHOR BIOGRAPHIES

Dr Georgios Theodoropoulos received a Diploma degree in Computer Engineering from the University of Patras, Greece in 1989 and MSc and PhD degrees in Computer Science from the University of Manchester, U.K. in 1991 and 1995 respectively. In 1991-1995 he was a member of the AMULET group. Since February 1998 he has been a Lecturer in the School of Computer Science, University of Birmingham, U.K. teaching courses on Hardware Engineering and Computer Networks. His research interests include parallel and distributed systems, computer and network architectures and modelling and distributed simulation.

Dr Doug Edwards is a Senior Lecturer in Computer Science at the University of Manchester. Previous research interests have included the physics of heterojunction structures, high speed optical fibre networks and hardware accelerators for CAD. He is currently a member of the AMULET group researching asynchronous systems.