Analyzing the Timing Error in Distributed Simulations of Asynchronous Computer Architectures

G. Theodoropoulos* and J. V. Woods
Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, England

Abstract

This paper discusses the time modelling problem in RTL distributed simulation models of asynchronous architectures written in occam. The results presented in the paper confirm that even if preemptions are allowed to occur, the timing error is not significant and it is acceptable at this level of simulation.

1. INTRODUCTION

As VLSI technology advances and systems become larger, faster and more complex, timing problems in synchronous systems become increasingly severe and account for more and more of the design and debugging expense. Increased clock speeds make on-chip clock skew significant and interchip skew a major problem. In order to address those problems recently there has been a resurgence of interest in asynchronous design techniques which eliminate the need for global clocking. An asynchronous system may be designed as a set of functional modules each operating at its own rate and cooperating through communication. The synchronization of the functional modules is performed by means of the communication protocol which allows data to be shared between them. There exist many different approaches for designing asynchronous systems [3]. Sutherland's "Micropipelines" [5] use bundled data with an event-signaled handshake protocol for synchronization; the control circuits are implemented by means of a set of *event control blocks*, which include the *Muller-C*, *Select*, *Call*, *Toggle*, *Xor* and the *Arbiter* blocks. Following Sutherland's approach, the AMULET group at the University of Manchester have designed and implemented AMULET1, an asynchronous version of the ARM RISC processor [2].

2. MODELLING ASYNCHRONOUS ARCHITECTURES WITH OCCAM

The recent research activity in the area of asynchronous systems has pointed to the need for suitable techniques for modelling and simulating them. Several notations and techniques have been suggested for this purpose [3]. CSP [4] in particular has attracted the interest of many researchers due to the strong relationship between its semantics (synchronous, unbuffered interprocess communication) and the behaviour of asynchronous systems.

To support the design of AMULET1, a methodology for using occam, a CSP based programming language, to build executable models of asynchronous architectures at the Register Transfer Level has been developed [8]. Using this methodology, an asynchronous architecture is modelled as a set of concurrent communicating occam processes which may be executed on a

*For this work Georgios Theodoropoulos has been supported by the *Mpakalas Foundation*, Athens, Greece, under Grant No. 466/21121992.

Figure 1: Occarm Top Level Process Graph

network of Transputers. The processes are entirely data-driven, and self scheduled. Two channels are used for the synchronization of communicating processes, one for Request/data and one for the Acknowledgement signal. To describe the nondeterministic behaviour of arbiters, the occam ALT construct is used.

2.1. Timing issues

The distributed nature of occam introduces a problem typical in distributed simulations, namely the problem of enforcing and maintaining strict temporal precision so that preemptions are avoided and causality constrains are not violated. In distributed simulations, preemptions occur if merge modules consume and process input messages from different channels in nonincreasing timestamp order. In a micropipelined architecture, micropipelines may be merged in one of the following ways:

- **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 occam process has to wait for all input channels to "fire" and therefore no preemptions occur.

- **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 process 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 micropipelines arrive at the same time, an arbitrary choice is made (arbiters); generally, this order will be different that the order in which the occam channels in the corresponding ALT fire and thus preemptions occur in this case.

Figure 2: The Address Interface

Several techniques have been devised to address the preemption problem in distributed simulations [1]; these techniques include the *Program Driven Synchronization Protocol* which has been developed as part of the work of the AMULET group and is targeted at occam models of asynchronous architectures [7]. These attempts to enforce synchronization in order to maintain absolute timing precision, increase the complexity of the simulation model and reduce its performance. Since the simulated architectures are asynchronous, time is not required for the the correct operation of the occam model; the very presence of an arbiter in the design implies that the order of message consumption may be arbitrary. Time is needed only as a quantifier to provide the means for a performance evaluation of the architecture. In [6] is was argued that even if no attempt is made to enforce timing precision in the arbiter processes, the characteristics of the the simulated systems will not allow the timing error to become significant; the fundamental nature of asynchronous architectures will balance the throughput of the distributed processes preventing thus the local clocks from becoming too skewed. Consequently, the performance results produced by the simulation model will maintain their reliability.

Recent quantitative results, which are presented in the following sections, confirm this claim. The results have been obtained using *occarm*, an occam simulation model of AMULET1 [8].

3. THE OCCARM SIMULATION MODEL

Occarm has been implemented as a hierarchy of occam processes, with each process modelling a different functional module of AMULET1. Its top level process structure graph is depicted in figure 1. *AddInt* and *DatInt* processes model AMULET1's address and data interface to memory respectively. The datapath is modelled by four processes, namely *Decode1*, *Decode2*, *Decode3* and *RegBank*. Decode1 describes the primary decode unit while Decode2 and Decode3 model the execution unit of the processor. RegBank process incorporates the functionality of the register bank. *WrtCtrl* models the operation of AMULET1's write bus control logic. Occarm makes use of three arbiter processes, which are included in AddInt, Decode1 and WrtCtrl. Decode1 arbiter allows the *PCcol signal* to interrupt the flow of instructions from Memory (*Instr* channel) while WrtCtrl arbiter permits data arriving from memory (*Din*) and the execution unit (*DPch*) to access the write bus (see figure 1). AddInt, which is depicted in figure 2, employs arbitration (AddC process) to allow addresses arriving from the datapath on the Wch channel to break the *PC loop* (AddC, MAReg, Incrementer, PC Holding Latches, PC0 register of PC Pipe) and gain

Figure 3: The Arbiter Process

access to the MAReg. The arbiter processes of occarm have been modelled by means of the occam ALT construct. Input messages are consumed and processed as soon as the corresponding input channels fire and no attemt is made to prevent preemptions; the calculation of the timestamps of the output messages is illustrated in figure 3.

4. THE TIMING ERROR IN OCCARM

The accuracy of occarm has been evaluated by comparing the results produced by occarm with those obtained from a different, sequential discrete event simulator of AMULET1 written in Asim, the ARM's in-house simulation language. The results which have been produced by the execution of the Dhrystone benchmark [9] are those that are typically used for the performance evaluation of AMULET1, namely the Dhrystone number, as well as the occupancy and stall periods of the AMULET1 pipelines; since the calculation of these values is based on the simulated time, they are particularly suitable to be used as a means for measuring the degree of timing accuracy of occarm. Like all synthetic benchmarks, Dhrystone tries to match the average behaviour of a large set of real program, therefore the results obtained may be considered representative of the average behaviour of occarm too. The Dhrystone number denotes the number of times that the loop which constitutes the body of the benchmark is executed during the period of one second, thus providing an indication of the performance of the simulated architecture. This number is calculated by sampling the current clock value before (T_{before}) and after (T_{after}) the execution of the loop and employing the formula:

$$Dhrystone.Number = \frac{10^9 * Number.Of.Runs}{T_{after} - T_{before}}$$

Table 1 illustrates the clock values T_{before} and T_{after} of the Dhrystone program as produced by occarm, while table 2 presents the Dhrystone number obtained from the aforementioned values.

As shown in table 2, the timing error with regard to Dhrystone number is 19.72% ; this value may be considered reasonable and indeed acceptable at this level of simulation and at the early stages of the design process. The same applies for the values obtained regarding the pipeline

Table 1
Occarm Timestamp Drift (1 Dhrystone Loop)

T_{before} (ns)			T_{after} (ns)		
Value	Drift	Error (%)	Value	Drift	Error (%)
39036	10313	26.4	114432	28834	25.1

Asim T_{before}= 49349ns
Asim T_{after} = 143266ns

Table 2
Occarm Dhrystone Number (1 Dhrystone Loop)

Dhrystone Number	Error (%)
13263.30	19.72

Asim Dhrystone Number= 10647.69

occupancy and stall periods (which are not included in this paper); the timing error with regard to those values ranges from 0% to about 10% [8] which also is considered acceptable.

In order to examine how preemptions are distributed over time, the number of times that a preemption was detected (the preemption count) and the corresponding accumulated preemption magnitude (in *nanoseconds*) for each arbiter process have been sampled in regular intervals of 10000ns. The results which are presented in figure 4, indicate that preemptions take place at a low frequency, with the corresponding preemption magnitude being relatively small (ranging from less than 25ns to 475ns). Most of the preemptions occur in AddInt process; this may be explained by the fact that the activity of the address interface (i.e number of messages arriving on the input channels of AddC process) is higher, thus increasing the probability of preemptions.

5. CONCLUSIONS

Asynchronous logic promises to provide a solution to global clocking related problems and the means to meet the lower power requirements of VLSI systems. Occam is particularly suitable for rapidly implementing distributed simulation models of asynchronous systems at the Register Transfer Level but its distributed nature introduces time modelling problems. This paper has discussed these problems in connection with the AMULET1 architecture, confirming that they may be ignored since the timing error introduced in the model is insignificant.

REFERENCES

1. Fujimoto, R., "Parallel Discrete Event Simulation", Communications of the ACM, Vol. 33, Number 10, October 1990, pp 31-53.
2. Furber, S., Day, P., Garside, J., Paver, N., Woods, J.V., "AMULET1: A Micropipelined ARM", Invited Paper, COMPCON'94.
3. Hauck, S., "Asynchronous Design Methodologies: An Overview". Technical Report UW-CSE-93-05-07. Department of Computer Science, University of Washington, April 1993.
4. Hoare, C.A.R., "Communicating Sequential Processes", Communications of the ACM, Vol. 21, Number 8, August 1978, pp 666-677.

Figure 4: Preemption Count and Magnitude

5. Sutherland, I., "Micropipelines", Communications of the ACM, Vol. 32, Number 6, June 1989, pp 720-738.
6. Theodoropoulos, G., Woods J.V., "Building Parallel Distributed Models for Asynchronous Computer Architectures", Proceedings of the World Transputer Congress 1994, Como, September 1994, pp 285-301.
7. Theodoropoulos, G., Woods J.V., "Distributed Simulation of Asynchronous Computer Architectures: The Program Driven Conservative Approach", Proceedings of the European Simulation Symposium 1994, Volume 2, Istanbul, Turkey, October 1994, pp 230-234.
8. Theodoropoulos, G., "Modelling and Simulation Techniques for Asynchronous Computer Architectures", Ph.D. Thesis, Department of Computer Science, University of Manchester, to be submitted.
9. Weicker, R. P., "Dhrystone: A Synthetic Systems Programming Benchmark", Communications of the ACM, 27, 10, October 1984, pp 1013-1030.