# Occam: An Asynchronous Hardware Description Language?

G. K. Theodoropoulos*
G. K. Tsakogiannis†
J. V. Woods

Department of Computer Science, University of Manchester
Oxford Road, Manchester, M13 9PL, United Kingdom

**Abstract**

*Recently, there has been a resurgence of interest in asynchronous hardware due to the potential of asynchronous logic for higher performance, power efficiency and immunity from clock-related timing problems. This activity has revealed the current lack of suitable languages and notations for the description of asynchronous hardware systems and has fueled an intense research effort in this area. Communicating Sequential Processes (CSP) in particular, has attracted the interest of many researchers as a potential means for the modelling of asynchronous designs. Contributing to this effort, this paper examines whether Occam, a CSP-based language may provide a solution to this endeavour.*

## 1 Introduction

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. Asynchronous digital design styles promise 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. The desire to exploit the potential advantages offered by asynchronous logic has recently fueled a revival of interest in asynchronous systems [3] [2]. 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.

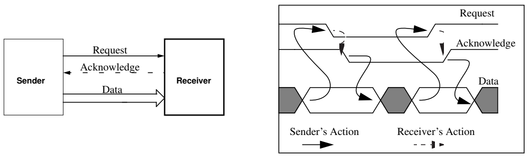


Figure 1: The Bundled Data Interface Protocol

## 2 Micropipelines

In his influential 1989 Turing Award lecture, Ivan Sutherland presented "Micropipelines", an asynchronous design framework whereby a system is designed as a set of elastic pipelines, whose stages operate asynchronously and exchange data via a two-phase bundled data synchronization protocol (figure 1) [15].

Sutherland also proposed a set of event control blocks for the design of control circuits in micropipelined systems as well as event controlled storage elements to be used in such systems.

The event control blocks include the *Muller-C*, *Select*, *Call*, *Toggle*, *Xor* and the *Arbiter* (figure 2).

An event controlled storage element is the *Capture-Pass* latch, depicted in figure 3.

The latch is controlled by two control signals, namely *Capture* ($C$) and *Pass* ($P$). Initially the latch is in its *transparent* state, where the input is connected through to the output (i.e. $Din = Dout$). When an event is issued on the Capture wire ($C$) the input-output connection is interrupted, the data is "latched", and an event is issued on the $Cd$ signal (Capture done) to indicate the change of state in the latch (i.e from transparent to *opaque*); the latched data does not change with subsequent data input changes. When an event arrives on the Pass wire,

*Dr Georgios Theodoropoulos is currently with the Computer Technology Institute, 70 G. Papandreou Str., Zografou, Athens 157 73, Greece. He may be reached via email at georget@cti.gr

†Dr Georgios Tsakogiannis is currently Director of the Informatics Development Agency, Ministry of Internal Affairs and Public Administration, Athens, Greece.

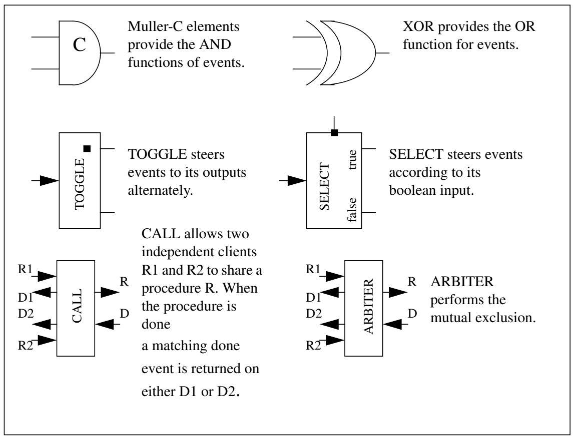


Figure 2: Event processing blocks

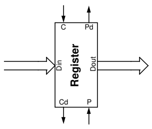


Figure 3: The Capture-Pass Storage Element

the input is connected back through to the output, thus making the latch transparent again; this change is indicated by an event on the $Pd$ (Pass done) signal. The Capture-Pass may repeat, with events arriving alternately on the $C$ and $P$ wires respectively.

The simplest micropipeline is a series of Capture-Pass registers connected together to form a FIFO structure as depicted in figure 4. A micropipeline may perform processing on the data, by interposing the necessary logic between adjacent register stages (figure 5).

## 3 AMULET1

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 [7] [8]. Figure 6 illustrates the physical layout of the 1.2 micron implementation of the processor. AMULET1 comprises five major units, namely the address interface, the data interface, the execution unit, the register bank and the primary decode. The execution unit consists of two major stages, namely Decode2 (Dec2-Ctrl2) which controls the operation of the shifter and multiplier units of the processor, and Decode3 (Dec3-Ctrl3) which controls the ALU.

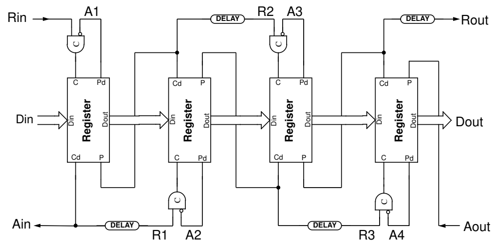


Figure 4: Micropipeline Without Processing

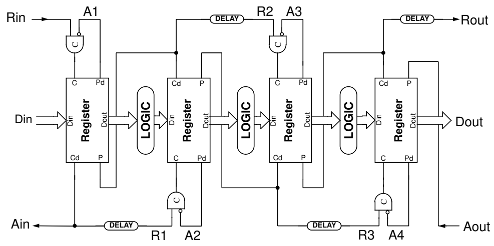


Figure 5: Micropipeline With Processing

## 4 Modelling Asynchronous Systems

Modelling, being at the heart of digital system design, may perform a catalytic role in the quest for the realization of the potentials offered by asynchronous logic. Hence, the recurrence of interest in asynchronous design has been accompanied by an intense research activity aiming at investigating and developing languages and notations appropriate to describe and model asynchronous systems. I-Nets, Petri-Nets, Signal Transition Graphs and State Transition Diagrams are some of the notations employed for this purpose.

Communicating Sequential Processes (CSP) [9] in particular, has attracted the interest of many researchers (e.g [12] [11] [21] [4]) as a potential means for the modelling of asynchronous designs due to the strong relationship between its semantics and the behaviour and structure of asynchronous systems:

- CSP supports a concurrent, process-based, asynchronous, non-deterministic model of computation which exactly matches the behaviour of asynchronous hardware.
- In CSP, the communication between different modules is point-to-point, synchronous and unbuffered. This behaviour directly reflects the interaction between subsystems in asynchronous hardware, where a sender and a receiver ren-

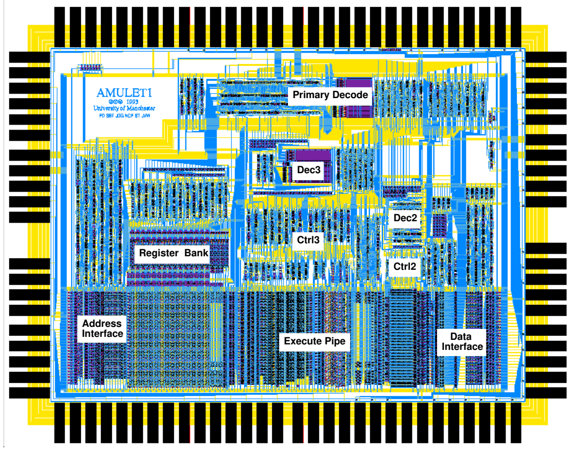


Figure 6: The AMULET1 Processor Physical Layout

dezvous before they physically exchange data via wires, which are memoryless media.

Contributing to the quest for modelling techniques and description languages suitable for asynchronous design, and motivated by the increasing advocacy of the potential use of CSP for this purpose, the research presented in this paper has investigated the suitability of occam [10], a CSP-based programming language, for the modelling and simulation of complex asynchronous designs.

# 5 Why Occam

There are several factors that advocate the candidacy of occam for the construction of models of asynchronous systems:

- Occam forms a practical realization of CSP, and, consequently, it maintains the strong relationship with regard to communication and computation between CSP and asynchronous systems.

- Occam allows explicit description of parallel as well as sequential computation. This explicit control of concurrency which extends down to the command level, along with its simple but powerful syntax and "send" and "receive" commands, makes occam ideal for describing digital systems; indeed, occam has been employed for modelling digital systems at various levels (e.g [22]).

- Occam is primarily a general purpose programming language which may be executed on a computer (transputer). Thus, a specification developed using occam is automatically an executable simulation model of the asynchronous system.

- Occam is a parallel programming language and thus may be used to perform distributed simulation (and it has indeed been employed for this purpose, e.g. [1]). A simulation model written in occam may be distributed on a transputer network and execute concurrently to achieve high performance. Asynchronous hardware systems are an excellent candidate for distributed simulation. The concurrent operation of the different subsystems of the asynchronous system, the inherent parallelism within each subsystem and the lack of any global synchronization, are characteristics which support the concurrent execution of events in a simulation model. In his *flashback simulation* approach [16], Sutherland attempts to exploit these characteristics of asynchronous systems and allow the "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.

# 6 The Modelling Philosophy

The main objective of the modelling philosophy proposed in this paper is to exploit:

1. The strong relationship between CSP (and occam) and asynchronous hardware, in order to achieve easy and rapid construction of models, and

2. The inherent parallelism of the hardware, in order to achieve high simulation performance.

The latter may be exploited at any level of abstraction that the system is modelled. The former, however, may be exploited only at the Register Transfer, or higher, level of a Micropipelined system.

Assuming a correct implementation of the communication protocol, at the Register Transfer Level, a Micropipelined system may be viewed as a network of concurrent modules communicating via synchronous, unbuffered communication. The modules are data-driven; each module will start computation as soon as data is available on its input wires, and will signal when the result has been computed. At this level, the system may be modelled as a network of concurrent, communicating occam processes, topologically identical to the asynchronous system, with each occam process corresponding to a different functional module of the system. This approach is similar to the "Logical Process Paradigm", typically employed in distributed simulation modelling [6].

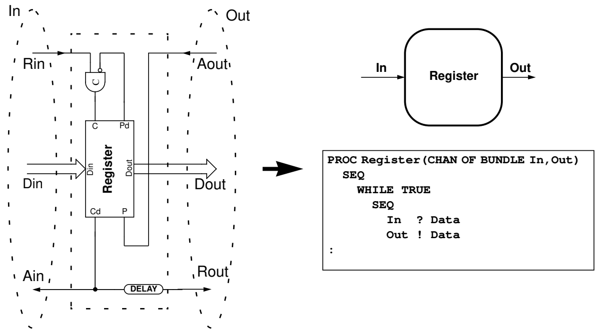


Figure 7: Micropipeline Without Processing: The Register Model

Since the correct operation of the asynchronous system does not depend on a global clock, simulated time is not required for the synchronization of the occam processes of the model. Processes are entirely data-driven and self-scheduling, and are synchronised by the protocol employed in the communication semantics of occam, in the same way that the communication protocol employed in the asynchronous system synchronizes the different functional modules. Each process will always consume event messages as soon as they become available, and it will always wait for subsequent messages if the messages it has generated have been successfully forwarded.

This methodology provides a natural way for modelling an asynchronous system based on the similarities between the system's behaviour and the semantics of occam. This basis, however, is not available for modelling at lower levels of abstraction. In this case, no assumptions should be made regarding the correctness of the communication protocol in the system; instead, explicit modelling of the protocol to verify that it adheres to the bundled data delay constraint is required. Occam may still be used for the description of low level circuit elements (event control elements and gates), however simulated time is essential for the synchronization and the correct operation of the simulation model.

### 6.1 Modelling a Pipeline Without Processing

Following the proposed modelling philosophy, a register in a Micropipeline without processing may be modelled as depicted in figure 7.

The request and acknowledge signals in the circuit are used to synchronize the register with its neighbouring registers in the pipeline. In the model, the synchronization between occam processes is performed by the communication protocol specified by the occam channel. Thus, no extra channels are required for the request and acknowledge signals. The register model makes use of two channels, for input and output respectively. The register process repeatedly reads data from its input channel and forwards it to the next process in the pipeline before it reads the next input value, thus manifesting a behaviour similar to that of a Micropipeline stage. A multi-stage Micropipeline may be modelled by means of a parallel replication (occam PAR construct) of the register process.

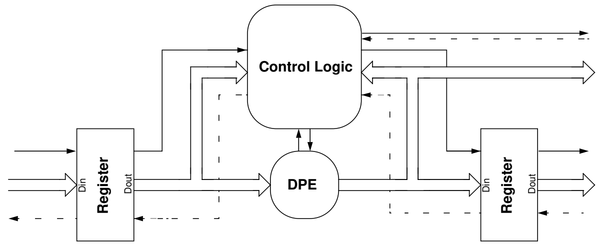


Figure 8: Micropipeline With Processing: A High Level View

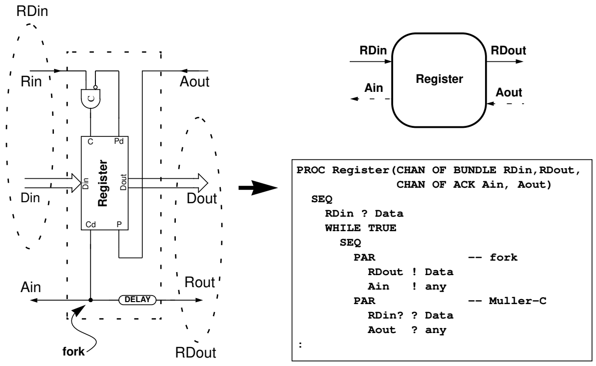


Figure 9: Micropipeline With Processing: The Register Model

### 6.2 Modelling a Pipeline With Processing

At the Register Transfer Level, a general Micropipeline with processing may be viewed as depicted in figure 8. The sending register outputs its contents, consisting of data and control bits, onto the data bus and produces a request event (request wires are indicated in the figure by solid lines, while acknowledge wires are denoted by dotted lines. The control bits, are used by the control logic to direct the request event to its correct destination, activating if necessary the data path elements (DPEs, e.g. ALUs, multipliers, shifters etc.) of the circuit. Data passes through the DPEs and propagates to the next stage.

This general Micropipeline may be modelled by three occam processes, two for the registers and one for the control/data processing logic; the control logic and the DPE may be modeled as one process, with the DPE being a procedure called by the control process.

However, the simple register model described in the previous section, is not suitable for modelling the behaviour of a stage in a Micropipeline with processing. Indeed, using this simple register model would force the control process to act as a buffer, decoupling the register processes; the sending process would be free to read the next value from its input channel, without first ensuring that the previous value has been received by the destination register process. Thus, the control logic process would introduce an extra pipeline stage in the model, a stage that does not exist in the physical system. To avoid this situation, the register processes must be kept tightly coupled and synchronized. This may be achieved by using two channels for a communication transaction between two register processes, one for the request/data and one for the acknowledge event.

Figure 9 illustrates the generic register occam model. The model makes use of two PAR statements, one to model the Muller-C element and one to model the fork on the Ain/Rout wire.

### 6.3 Modelling Control Logic

The control logic is inherently concurrent; different parts of the circuit operate concurrently while, within each part, events take place in a deterministic sequential order, i.e. the control logic implements a partial ordering of events. The simulation model should have the same degree of concurrency as the physical circuit. The control logic may be implemented as a network of communicating processes, with the occam PAR and SEQ commands being used within each process to implement the partial ordering of events of the circuit. The number of these processes depends on the degree of modularity and fidelity required in the simulation model.

Adopting a data driven approach to model asynchronous systems, it is essential to have a mechanism for modelling the functionality and the non-deterministic behaviour of arbiters. The occam ALT construct provides for the non-deterministic choice of messages from different channels and therefore may effectively model the behaviour of an arbiter.

## 7 Modelling AMULET1

In order to investigate the suitability and applicability of the approach described in the previous chapter for modelling large and complex asyn-

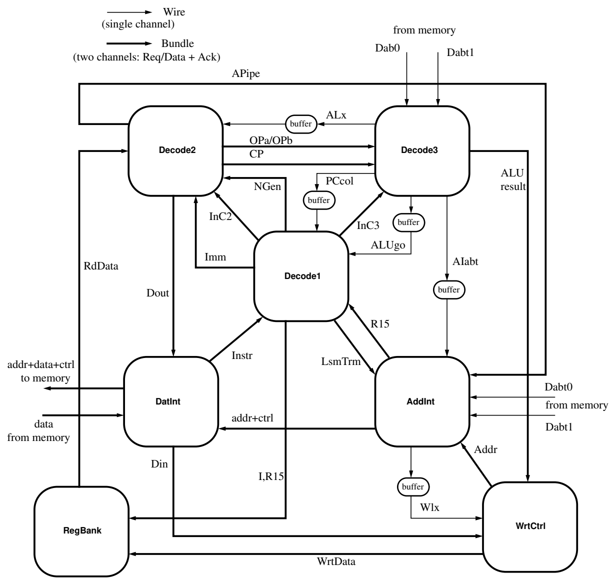


Figure 10: Occarm Top Level Process Graph

chronous hardware systems, *occarm*[1], an occam simulation model of the AMULET1 processor has been developed.

Occarm consists of more than fifteen thousand lines of occam code and describes AMULET1 at the Register Transfer Level. It executes ARM6 machine code produced by a standard ARM compiler. Instructions enter the simulator as 32-bit integer numbers in hexadecimal format. Instruction decoding is performed by means of PLA models which are implemented as two dimensional arrays of boolean values; the model makes use of a library of occam functions which has been developed to allow instructions to be treated both as integer values and as one dimensional boolean arrays.

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 10.

*AddInt* and *DatInt* processes model AMULET1's address and data interface units 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 two major components of 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.

[1]The name of the model is derived from the combination of the words **occ**am and A**RM.**

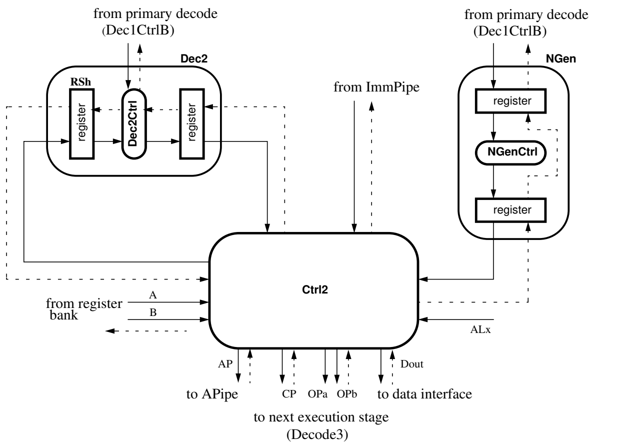


Figure 11: The First Execution Stage Model (Decode2)

All the registers of AMULET1, have been modelled using the generic register model described in section 6.2, with interprocess communication being performed using pairs of request/data and acknowledgement channels.

Indicatively, figure 11 depicts the internal structure of *Decode3*, while figures 12 and 13 illustrate the description of one of the control modules of Decode3 in occam (register models are depicted as rectangles while processes which model control logic are shown as squares with rounded edges). For a detailed description of the structure and operation of occarm, the reader is referred to [20].

### 7.1 Non-Bundled Signals

AMULET1 makes use of a number of control signals which are not part of a bundled communication and do not obey the protocol specified by the Micropipelines framework. These signals are transmitted via simple wires. The modelling of these signals by simple occam channels may lead to deadlocks in the simulation model. This is due to:

1. The communication semantics of occam, and
2. The direction of the signals.

In AMULET1 the simple wires are used to send information to previous stages of the pipeline, thus forming closed paths (loops). The synchronous communication supported by occam, forces the processes in the loop to block waiting for each other, a situation that is susceptible to deadlock.

To overcome this problem, in occarm, these signal wires have been modelled as buffer processes which

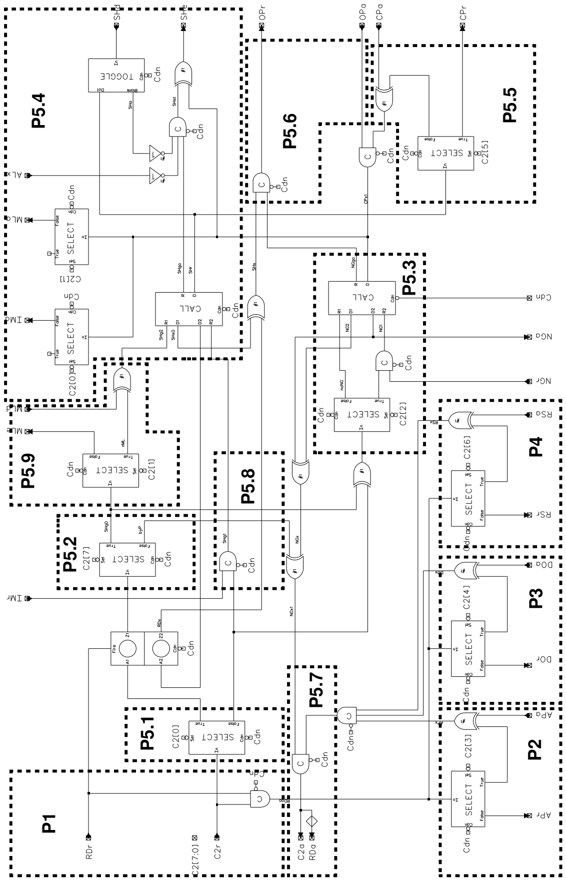


Figure 12: Ctrl2 Control Circuit

hold the value of the signal at any particular moment. The buffer decouples the processes involved in the wire communication, thus eliminating the deadlock susceptible behaviour. The size of the buffer for each signal is one; this is adequate since the value on the corresponding wire will change only once for each interaction of the processes. The buffer process is shown in figure 14.

## 8 Simulation Issues

The order in which the occam channels fire in the model does not adhere strictly to the order in which the corresponding events occur in the asynchronous system. This may cause causality errors in the case of arbiters where non-deterministic processing of events is allowed. These errors do not affect the correct functionality of the model but introduce an error in the simulated time and consequently in any evaluation of the system based on the concept of simulated time. Experiments undertaken with occarm have indicated that the inaccuracy introduced is not significant and

```
SEQ
  P1
  PAR
    P2
    P3
    P4
    SEQ --P5
      P5.1 --IF
        TRUE
          P5.2 --IF
            TRUE
              SEQ
                PAR
                  P5.3
                  SEQ
                    P5.9  --Multiplier
                    P5.4  --Shifter
                PAR
                  P5.5
                  P5.6
            FALSE
              SKIP
        FALSE
          SEQ
            PAR
              P5.3
              SEQ
                P5.8
                P5.4  --Shifter
            PAR
              P5.5
              P5.6
  P5.7
:
```

Figure 13: The Ctrl2 process

thus may be ignored [18] [20]. A synchronization protocol has also been developed for the elimination of causality problems in this context [17] [19] [20].

A multi-transputer configuration of occarm has yielded a speedup of 2.3 on a seven transputer network [20].

## 9 Assessment of Occam

The static, process-based model of computation of occam provides a natural and convenient means for the description of the behaviour and structure of asynchronous computer systems. Its support for explicit control of concurrency even at the command level, and its simple "send" and "receive" commands makes occam particularly suitable for describing digital systems.

Occam can describe asynchronous control circuits at a level which is very close to their implementation; consequently it may provide guidance for the realization of the design (e.g. an IF statement will correspond to a Select block, a PAR of input commands will be implemented using a Muller-C block etc). This characteristic may also be exploited for the automatic derivation of circuits from occam specifications as suggested in the next section.

```
PROC buffer()
  SEQ
    WHILE TRUE
      SEQ
        in ? data
        out! data
:
```

Figure 14: The Buffer Process

Furthermore, the parallel, distributed nature of occam forces the designer to think in "asynchronous terms" and to perceive its design as a distributed, asynchronous structure where a global state does not exist. The work with occarm has suggested that this may be the most important advantage of using occam for asynchronous architectural modelling. Indeed, the construction of occarm exposed behaviour patterns of AMULET1, thitherto unknown, whose operation was time-dependent rather than asynchronous.

Occam is a parallel language which may be executed on multiprocessor systems and thus has the potential for high performance.

On the negative side, occam lacks several data structures (and protocols for that matter) that would be extremely useful in a hardware description and simulation endeavour (e.g. records). Furthermore, for the manipulation of hardware circuit models, efficient and easy means for treating numbers as booleans (and the reverse) are extremely important, an area where occam is weak.

Another disadvantage of occam, as suggested by the programming effort of the research presented in this paper, is its rigid and verbose layout format and the semantic significance of indentation which makes both, the development and debugging of programs time consuming and frustrating tasks.

Occam2.1 and the new development system in support of the T9000 transputer alleviate some of the aforementioned deficiencies (e.g. records).

## 10 Conclusions

The concurrent, asynchronous, process-based model of computation of CSP, with the support for non-deterministic behaviour, and the point-to-point, synchronous and unbuffered inter-process communication are particularly suitable for describing the concurrent, non-deterministic behaviour of asynchronous hardware systems and provide a natural and convenient means for the description of asynchronous hardware and the rapid construction of asynchronous hardware models. However, if the benefits of using CSP for describing asynchronous systems are to be exploited and taken advantage of, it is essential to use a standard description language, that would be easily and widely available.

Occam may well serve this purpose: it is based on CSP, it is an executable programming language with well defined syntax and semantics, it is widely used and commercially supported, and is expected to be supported by a wide range of hardware platforms [13]. This paper has attempted to contribute to the realization of this potential by introducing a framework for modelling asynchronous hardware systems using occam as a hardware description language.

The simulation of digital systems in general, and computer architectures in particular, has long been categorized among the highly computation intensive applications. The same is true for the simulation of asynchronous digital systems. For the testing and evaluation of the AMULET1 design, for instance, more than 4 million instruction cycles were simulated [14], a number which corresponds to many hours of simulation. Hence, a parallel approach to simulation, such as the one described in this paper, could contribute significantly in reducing the duration and cost of the design cycle.

## Acknowledgements

For this work Dr Georgios Theodoropoulos has been supported by a University of Manchester Research Studentship and by the *Mpakalas Foundation*, Athens, Greece, under Grant No. 466/21121992.

## References

[1] Alonso, J. M., et al., "Conservative Parallel Discrete Event Simulation in a Transputer-Based Multicomputer", Proceedings of the World Transputer Congress 1993, Aachen, September 1993, pp. 636-650.

[2] "Asynchronous Digital Circuit Design", Editors Birtwistle, G., Davis, A, Springer Verlang, 1995.

[3] Brozowski, J. A., Seger, C., J., H., "Asynchronous Circuits", Springer Verlang, 1995.

[4] Brunvand, E., "Translating Concurrent Communicating Programs into Asynchronous Circuits", Ph.D Thesis, Carnegie Mellon University, 1991.

[5] Davis, A., Nowick, S. M., "Asynchronous Circuit Design: Motivation, Background and Methods", In [2], pp. 1-49.

[6] Fujimoto R., "Parallel Discrete Event Simulation", Communications of the ACM, 33, 10, October 1990, pp. 31-53.

[7] Furber, S. B., et al., "A Micropipelined ARM", Proceedings of VLSI '93 (Best Paper Award), September 1993, pp. 5.4.1-5.5.8.

[8] Furber, S. B., "AMULET1: A Micropipelined ARM", IEEE CompCon '94, Invited Paper, March 1994.

[9] Hoare, C.A.R., "Communicating Sequential Processes", Communications of the ACM, 21, 8, August 1978, pp. 666-677.

[10] "Occam 2 Reference Manual", Prentice Hall International, 1988.

[11] Josephs, M. B., Udding, J. T., "Delay-Insensitive Circuits: An Algebraic Approach to their Design", Lecture Notes in Computer Science, 458, 1990, pp. 342-366.

[12] Martin, A. J., "Compiling Communicating Processes into Delay-Insensitive VLSI Circuits" Distributed Computing, 1, 4, April 1986, pp. 226-234.

[13] "Occam For All, A Case for Support", available at URL: http://www.hensa.ac.uk/parallel.

[14] Paver, N. C., "The Design and Implementation of an Asynchronous Microprocessor", Ph.D Thesis, Department of Computer Science, University of Manchester, 1994.

[15] Sutherland I. E., "Micropipelines", Communications of the ACM, 32, 1, January 1989, pp. 720-738.

[16] Sutherland I. E., "Flashback Simulation", Research Report SunLab 93:0285, Sun Microsystems Laboratories, Inc., August 1993.

[17] 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.

[18] Theodoropoulos, G., Woods J.V., "Analysing the Timing Error in Distributed Simulations of Asynchronous Computer Architectures", Proceedings of the Eurosim Congress '95, Vienna, Austria, September 1995.

[19] Theodoropoulos, G., Woods J.V., "Dealing with Time Modelling Problems in Parallel Models of Asynchronous Computer Architectures", Proceedings of the World Transputer Congress 1995, Harrogate, England, September 1995.

[20] Theodoropoulos, G., "Strategies for the Modelling and Simulation of Asynchronous Computer Architectures", Ph.D Thesis, Department of Computer Science, University of Manchester, December 1995.

[21] Van Berkel, C. H., Rem, M., Saeijs, R. W. J. J., "Compilation of Communicating Processes into Delay-Insensitive Circuits", Proceedings of ICCD, 1988, pp. 157-162.

[22] Welch, P. H., "Emulating Digital Logic Using Transputer Networks (Very High Parallelism = Simplicity = Performance)", Lecture Notes in Computer Science, 258, (PARLE'87), 1987 pp. 357-373.