A Result Forwarding Unit for a Synthesisable Asynchronous Processor

Luis Tarazona, Doug Edwards
Advanced Processor Technologies Group, School of Computer Science
The University of Manchester, Manchester, M13 9PL, United Kingdom
Email: {tarazonl,doug}@cs.man.ac.uk

Abstract

For deep pipelined processors, forwarding of results from the production points to the consuming units is an key operation to deliver performance. The implementation of an efficient result forwarding unit for asynchronous processors faces the problem of the inherent lack of synchronisation between result producer and consumer units. An efficient solution to this problem has been proposed and implemented before (in the AMULET3 asynchronous processor) but with the disadvantage of being a full-custom design, with the consequent limitations on design-space exploration and technology portability. A synthesisable description is attractive in terms of rapid development, technology mapping transparency and design space exploration. This paper presents the description of a synthesisable result forwarding unit for a synthesisable asynchronous microprocessor using the Balsa Synthesis system. The description of such a system serves as an evaluation of the capabilities and limitations of the Balsa synthesis system for the description of performance-demanding asynchronous systems.

1. Introduction

Result forwarding [1] is a method used in pipelined microprocessors to reduce the penalty caused by inter-instruction data dependencies. Although modern compilers can optimise the compiled code to reduce these dependencies using different methods, they cannot completely solve this problem. As presented in [2], the forwarding mechanism can also be used to allow partial overtaking of (normally slow) memory operations, but making sure that the instructions complete in the same order as they appear in the program. The latter is particularly important if the memory operation cannot complete, so the processor state does not change incorrectly, allowing the restart of the aborted memory operation. Figure 1 depicts some potential performance benefits of the result forwarding mechanism.

In an asynchronous environment, the problem of implementing a result forwarding mechanism is more complicated due to the lack of synchronisation between units producing and consuming results. In this case, one cannot rely on a control signal that indicates which

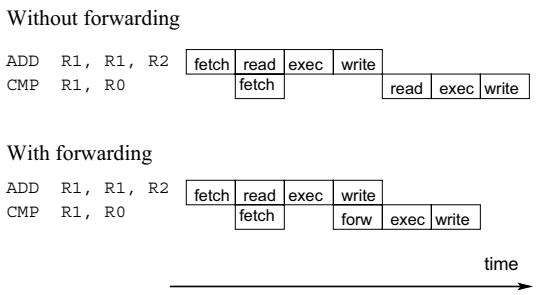

Figure 1: Potential performance benefits of result forwarding in a 4-stage pipeline.

cycle an instruction is in. This would require a lockstep operation of the pipeline that would heavily penalise the performance. Earlier asynchronous techniques for resolving dependencies include: register locking [3], register locking plus "last result" register [4], the counterflow pipeline architecture [5] and the asynchronous reorder buffer (AROB) [2] for the AMULET3 processor [4].

1.1. The target processor: nanoSpa

The forwarding unit described in this paper was designed to be used in the nanoSpa processor [6]. NanoSpa is an experimental, new specification of the SPA processor [7], a fully synthesised asynchronous implementation of the 32-bit ARM v5T ISA. NanoSpa is currently under development and shares the same architecture organisation: an ARM-style 3-stage Fetch-Decode-Execute pipeline with a Harvard-style memory interface. Both processors were described using the Balsa synthesis system [8]. To date, nanoSpa has the following functional differences with respect to SPA: it does not have support for Thumb instructions, interrupts, memory aborts or coprocessors. Figure 2 shows a simplified version of the nanoSpa pipeline.

1.2. The Balsa synthesis system

Balsa [8] is an open-source synthesis system that generates purely macromodular asynchronous circuits. Balsa closely follows the style and philosophy of the Philips Haste/Tangram system [9]. These use a process

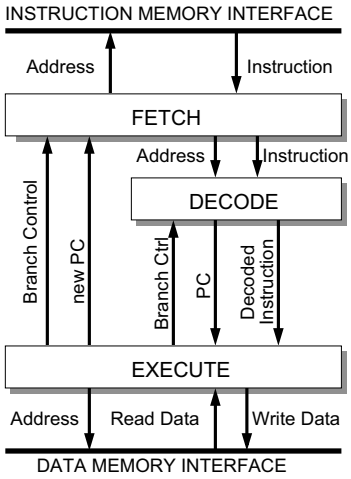

Figure 2: The 3-stage nanoSpa pipeline.

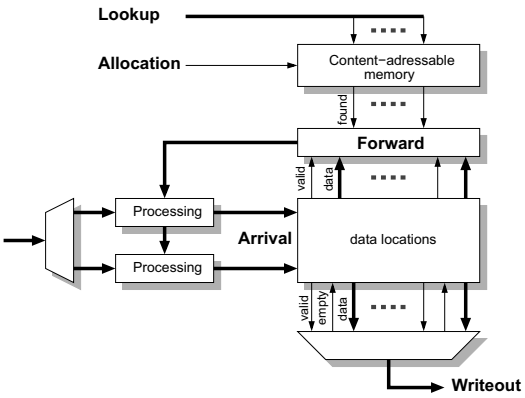

Figure 3: AROB process model.

called syntax-directed compilation to map behavioural descriptions into a network of pre-designed modules called handshake components. This approach allows the designer to make changes in the circuit description to optimise it in terms of area, power or performance. These changes at the description level will result in predictable changes at circuit level. Balsa has been successfully used to synthesise complex asynchronous systems such as the SPA processor.

1.3. Paper organisation

This paper is organised as follows: Section 2 presents the related work, making emphasis in the process model and architecture presented in [2]. Section 3 describes the architecture of the proposed forwarding unit, discusses the similarities and differences with respect to the referenced architecture [2] and the challenges faced describing such architecture in Balsa. Section 4 presents the implementation of the unit and discusses some issues related to it. Simulation resultsand discussion are given in Section 5. Conclusion and future work are presented in Section 6.

2. Related work

The AMULET3 processor features an asynchronous reorder buffer (AROB) [2] that also acts as a forwarding unit. As nanoSpa is also an ARM core, the process model of the asynchronous queue in the AROB was used as the reference model for the nanoSpa forwarding unit (nFU). The AROB features a circular "queue" FIFO [10] which can be read in parallel to provide forwarding capabilities. Figure 3 shows a diagram of the AROB process model. The queue operation consist of 5 distinct processes: *Lookup*, *Allocation*, *Forward*, *Arrival* and *Writeout*.

In order to improve the speed of the lookup process, the AMULET AROB uses a small CAM to hold the information about the registers written in the buffer. Speculative read of the default value from the register bank is also performed in case the source operand is not present in the buffer. The AROB has a centralised control and features three read ports for forwarding and two write ports for arrival. The AMULET3 and the AROB were implemented using single-rail data encoding.

In a more recent work, a distributed control architecture has been proposed for an AROB for an asynchronous on-chip bus [11]. In that work, because the target application has simpler requirements, the process model is simpler and the number of required ports is reduced.

3. Architecture of the nanoForward Unit

The initial purpose of the nFU in nanoSpa is to allow the decoupling of register bank reads and writes. Given that the architecture of the nanoSpa pipeline is simpler than that of the AMULET3 (memory operations are not decoupled from execution) the nFU does not act as a reorder buffer as this would require a large modification of the pipeline. The nFU has been designed around the five-processes model of the AMULET3 AROB described in section 2 and it also has the same number of read ports (3) and write ports (2). Figure 4 shows the architecture of the nFU. and its location in the new nanoSpa pipeline.

Although the five-processes model allows some concurrency, correct operation of the nFU still requires some synchronisation between processes: forwarding requires data from lookup, allocation must wait for lookup to finish and arrival must wait for forwarding, otherwise, wrong values could be forwarded and, in the case of dual-rail implementations, this could also lead to deadlocks. For similar reasons, the dequeue process must be locked until forwarding finishes. These implementation issues will be discussed in next section.

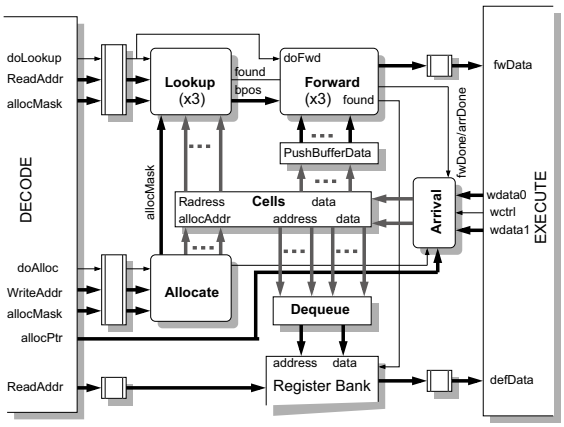

Figure 4: The nanoForward Unit architecture

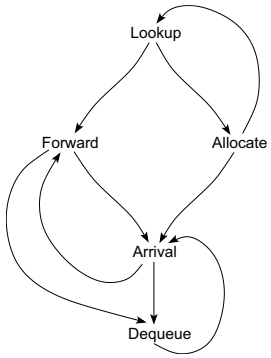

Figure 5: Inter-process dependencies in the nFU.

4. nanoFU implementation

The nFU implementation targeted a dual-rail encoding implementation as this is a robust data encoding scheme potentially immune to process variability that affects current fabrication technologies. This advantage comes at the cost of higher area, energy consumption and in some cases, restricted solution alternatives. The queue size is parameterised and the description allows for sizes of 4, 5, 6 and 8. The style used for the Balsa description was data-driven, with performance as the main goal.

4.1. Implementation issues

4.1.1. Synchronisation between processes. To guarantee correct operation, on each instruction the nFU must perform sequentially some operations as shown in figure 5. To allow synchronisation among handshake modules, the Balsa language provides special zero-data channels called *sync* channels. In an initial description, the use of sync channels to synchronise the processes generated a large performance penalty, so alternatives were looked for. A solution that reduced dramatically this penalty was to perform synchronisation using data instead of sync tokens: To decouple forwarding from arrival, the buffer contents are read speculatively and sent through data channels to the forwarding process. Lookup and allocation were decoupled using an *allocation mask* that blocks the reading of the buffer locations that will be modified by the allocation/arrival process during the current instruction. This masking reduces the effective length of the queue in 1 or 2 locations when an instruction reads and writes, so the minimum usable queue size is 5 for this architecture for a decoupled writeback. These solutions obviously dissipate more power and require larger area.

4.1.2. CAM implementation. Balsa does not provide a way to describe a CAM and generate an efficient circuit structure. The Balsa synthesised circuit consists of a number of logic comparators that, despite being relatively simple, do not perform as well as an optimised CAM.

4.1.3. Register bank operation. Because AMULET3 was built using single-rail data encoding, speculative register read in parallel with register writeback operations was safe as the AROB can always provide the correct data if the read register is being written concurrently. In dual-rail encoding, this can potentially lead to incorrect operation and deadlocks. As this is the case of the nFU implementation, the register read operation cannot be made speculatively and it must always wait for information from the lookup process. The number of data token present in the pipeline guarantees that the writeout process never conflicts with a read operation. Another approach, that has been avoided in this design but that will be explored in further work, is the use of arbitration to resolve possible conflicts.

5. Results

The nanoSpa with the nFU was synthesised in 180nm technology. After a series pre-layout, transistor level simulations it was found that, for this architecture, the optimum queue sizes are 4 (but register bank cannot be decoupled) and 5 (with fully decoupled register bank). The processors were tested running Dhrystone. Table 1 shows that performance increases were similar for both queue sizes and close to 6%, with an area penalty of 15.2% in spite of the limitations faced with the Balsa description. Results also show how the techniques used for desynchronising the processes achieved close to 50% increase in performance relative to the use of sync channels. Unfortunately it is not possible to make a relative comparison of the performance gain with respect to the AMULET3 AROB, because there are no published figures with and without the AROB. Pre-implementation, VHDL-based performance figures in [12] suggest that the use of the AROB in AMULET3 would increase its performance by 22.5% when running the Dhrystone

benchmark. Notice also that the AMULET3 pipeline has a decoupled memory stage and this feature is not present in nanoSpa.

The nFU makes extensive use of arrayed variables and arrayed channels for storing and broadcasting data. At the moment, the authors are looking into the generated structures to find ways of implementing those as optimised handshake modules that could be describe with new constructs. From the ongoing analysis of the nFU some peephole optimisations such as 4-phase broad semi-decoupled transferrers (different to those presented in [9]), semi-decoupled encoders and the use o a new control component: the conditional parallel/sequencer has been proposed by the authors. At the time of writing this paper, preliminary results suggests combined performance increases of more than 10% These ideas and results will be presented in future papers.

device	DMIPS	perf. gain %	area ov. %
nanoSpa no nFU	73.5	0.00	0.00
nanoSpa + 4-place nFU using sync signals	52.4	-28.8	5.2
nanoSpa + 4-place nFU	77.8	5.88	11.5
nanoSpa + 5-place nFU	77.5	5.36	15.2

Table 1: Simulation results for nanoSpa using the nFU

6. Conclusions and future work

The work presented in this paper demonstrates the feasibility of describing a synthesisable result forwarding unit in Balsa and obtain some performance increase. Compared to the AMULET3 AROB, the nFU only achieves 1/4 of the expected performance increase. The work also highlights some of the performance issues that arise from the use of a synthesisable forwarding unit in Balsa, namely the lack of efficient ways of describing and synthesising associative arrays (CAM) and the problem of deadlock-safe concurrent writes and reads in dual-rail variables to perform speculative reading. These problems are currently being analysed together with some peephole optimisations that the nFU design has highlighted, including semi-decoupled transferrers and a new conditional parallel/sequencer control component. Future work will include extending the pipeline depth of nanoSpa to decouple the memory stage and explore the effects of the suggested optimisations and components.

Acknowledgment

The authors would like to thank Luis Plana, Charles Brej, Andrew Bardsley and William Toms, all of them members of the Advanced Processor Technologies group, for all their cooperation, suggestions and comments during the development of this work.

References

[1] J.L. Hennessy and D.A. Patterson. *Computer Architecture: a Quantitative Approach (2nd edition)*. Morgan Kaufmann, 1996.

[2] D. A. Gilbert and J. D. Garside. A result forwarding mechanism for asynchronous pipelined systems. In *Proc. International Symposium on Asynchronous Circuits and Systems*, pages 2–11. IEEE Computer Society Press, April 1997.

[3] N. C. Paver, P. Day, S. B. Furber, J. D. Garside, and J. V. Woods. Register locking in an asynchronous microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, pages 351–355. IEEE Computer Society Press, October 1992.

[4] Stephen B. Furber, James D. Garside, and David A. Gilbert. AMULET3: A high-performance self-timed ARM microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, October 1998.

[5] Robert F. Sproull, Ivan E. Sutherland, and Charles E. Molnar. The counterflow pipeline processor architecture. *IEEE Design & Test of Computers*, 11(3):48–59, Fall 1994.

[6] L.A. Plana, D. Edwards, S. Taylor, L. Tarazona, and A. Bardsley. Performance-driven syntax directed synthesis of asynchronous processors. In *Proc. International Conference on Compiles, Architecture & Synthesis for Embedded Systems*, pages 43–47, September 2007.

[7] L. A. Plana, P. A. Riocreux, W.J. Bainbridge, A. Bardsley, J. D. Garside, and S. Temple. SPA – a synthesisable Amulet core for smartcard applications. In *Proc. International Symposium on Asynchronous Circuits and Systems*, pages 201–210. IEEE Computer Society Press, April 2002.

[8] A. Bardsley. *Implementing Balsa Handshake Circuits*. PhD thesis, Department of Computer Science, University of Manchester, 2000.

[9] A. Peeters and K. van Berkel. Single-rail handshake circuits. In *Proc. Working Conf. on Asynchronous Design Methodologies*, pages 53–62, May 1995.

[10] Kees van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*. Cambridge University Press, 1993.

[11] E.G. Jung, J.G. Lee, H. Fraz, K.S. Jhang, J.A. Lee, and D.S. Har. Implementation of asynchronous reorder buffer for asynchronous on-chip bus. In *Proc. International Symposium on Signals, Circuits and Systems (ISSCS)*, volume 2, pages 773–776, July 2005.

[12] David Alan Gilbert. *Dependency and Exception Handling in an Asynchronous Microprocessor*. PhD thesis, Department of Computer Science, University of Manchester, 1997.