A Low-Power Processor Architecture Optimized for Wireless Devices

Aristides Efthymiou
School of Informatics,
University of Edinburgh,
Edinburgh, UK
aefthymi@inf.ed.ac.uk

Jim D. Garside
School of Computer Science,
University of Manchester,
Manchester, UK
jdg@cs.man.ac.uk

Ioannis Papaefstathiou
Institute of Computer Science,
Foundation of Research
and Technology - Hellas,
Heraklio, Crete, Greece
ygp@ics.forth.gr

Abstract

The advantages of power-aware processors are well known. This paper presents an innovative processor architecture optimized for wireless environments. The presented architecture incorporates a certain power-aware microarchitectural technique, called pipeline depth adaptation and it is tailored to self-timed processors. With this technique a processor is able to alter its pipeline depth, while in operation, trading speed and energy use.

The pipeline depth is changed by making selected pipeline registers transparent. A shallow pipeline has lower energy consumption for two reasons: the capacitance driven by the load signal of the 'collapsed' pipeline registers is not switched and the reduction in branch latency and data-dependent stalls reduce the cycles per instruction (CPI) of the processor.

An analysis of the advantages of using pipeline depth adaptation in an asynchronous processor is given, supported by simulation results based on a real asynchronous processor and on applications that are frequently executed on a wireless environment. Finally a method of dynamically adapting the pipeline depth is described and evaluated which only reduces the pipeline depth when a branch instruction is expected. The presented architecture has a relatively lower power consumption than a conventional similar architecture, therefore it can be useful in wireless environments.

Keywords: Low power, Pipeline depth, configurable pipeline, power-adaptive processors, asynchronous circuits.

1 Introduction

Probably the most crucial factor of a processor employed in wireless devices is its power consumption. Therefore, several techniques have been examined that can reduce the energy consumption of a certain CPU, very often at the cost of reduced performance. One such method is to adapt the processor speed according to the requirements of the executing task, by mainly altering the supply voltage. Due to the quadratic relationship of the supply voltage to the dynamic power consumption, supply voltage scaling (DVS) has been employed in a number of research projects and commercial products as an energy-saving technique [1][2].

Recently, *microarchitectural* techniques for adapting the speed and energy consumption of a processor have also emerged [3][4][5][6]. As such techniques essentially adapt the effective switched capacitance of the processor, in principle, they cannot offer as large energy savings as DVS. Nevertheless, such techniques are useful since they can be combined with DVS for further energy savings, while they do not suffer from the DVS disadvantages: long transition times between operating modes and increased circuit complexity leading to higher verification effort.

Although DVS appears to have more potential for saving energy than microarchitectural techniques, in future technologies this potential will be reduced. As transistor feature sizes shrink, so does the nominal supply voltage, but the transistor threshold voltages are not scaled accordingly because that would cause a large increase in leakage current. As a result, the minimum safe operating voltage will not drop as quickly as the nominal supply voltage, causing the operating range of DVS to be quite small. Within that voltage range, the energy savings will be relatively low.

This paper presents a low power processor architecture incorporating *pipeline-depth adaptation* (PDA), a power-adaptive, microarchitectural technique which enables a conventional processor to alter its pipeline depth [7] [8] [9]. Energy is saved by the reduction of speculatively executed instructions and the reduction of stall cycles, at the expense of an increase in the cycle time. Similar techniques, called Dynamic Pipeline Scaling [10] [11] and Pipeline Stage Unification [12] have been proposed independently for standard synchronous processors. As explained later, a typical, syn-

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

chronous processor has significant restrictions to the way it can implement PDA. On the contrary an asynchronous processor has enough flexibility that allows a wider choice of variations.

Section 2 introduces pipeline-depth adaptation and reviews the related work. Section 3 shows simulation results for the asynchronous processor where PDA is applied statically, while section 4 shows an example of dynamically adapting the pipeline depth while a program is executed. Finally, we conclude in section 5.

2 Pipeline-Depth Adaptation

A processor with pipeline-depth adaptation has a number of operating modes depending on the possible configurations of its pipeline. When the highest performance is required, the pipeline should be as deep as possible so that the instruction throughput is the highest. When energy is to be conserved, the pipeline can be made shallower, with a corresponding decrease in the performance as the processing throughput drops.

By decreasing the pipeline depth, energy is saved for two main reasons: First, the drivers for the pipeline registers that are *collapsed*, i.e. made transparent, are gated so a fraction of the capacitance of the clock network is not switched in every cycle. Second, a shallower pipeline has a lower 'cycles per instruction' (CPI) metric, (or higher Instructions Per Cycle, IPC). This is due to the lower branch latency and the reduction in cycles lost to stalls caused by data dependencies between instructions. A lower CPI leads to a lower number of cycles required to execute a particular task. Assuming that the energy expended in a cycle is the same regardless of its duration, a lower CPI leads to lower energy consumption.

Decreasing the pipeline depth has certainly a negative impact on the performance. For example, halving the pipeline depth also drops the processing throughput by half. As the energy savings, by using PDA in such a case, are lower than 50%, as it will be demonstrated, in terms of energy-delay product, or, even worse, of $ET^n, (n \geq 1)$ [13] the proposed technique loses out. However, this technique is very useful, in wireless devices like Mobile Phones or low-end Personal Digital Assistants, when, very often, there is ample time to complete a task, but not enough energy to be consumed. In those cases the energy savings are greater than what could be achieved only by slowing down the clock, while keeping the pipeline depth unchanged.

2.1 Implementation Issues

Implementing PDA in a processor requires solving two main issues: how to collapse the pipeline registers and how the pipeline interlock and forwarding mechanisms are affected.

Collapsing the pipeline registers is relatively easy. If level sensitive latches are used, collapsing involves keeping them transparent, regardless of the clock transitions. For edge-triggered registers, multiplexors can be used that select either the register input or the output, depending on whether the pipeline register is collapsed or not. Figure 1 shows how a local signal (*collapse*) modifies the operation of both types of pipeline registers.

The implementation of pipeline interlock and forwarding logic that can operate with variable pipeline depths is a hard problem in a synchronous processor.

In one of the two available synchronous PDA proposals, [11], a top-down approach was followed. They start with a relatively shallow pipeline, show how to split the key pipeline stages into two and, finally, add extra hardware to handle interlocking when these two stages are unified.

In [12] a bottom-up approach is presented; the starting point is a very deeply pipelined processor and a method of unifying pairs of pipeline stages and handling pipeline interlocks is presented. Each stage produces a local hazard signal; these are ORed together so that each stage stalls when it or a stage downstream detects a hazard. Multiplexors then select the appropriate stall signals, depending on which pipeline register is collapsed.

When a pipeline register is collapsed, loops in combinational logic could be formed, in case a signal from the downstream stage is used as an input to the upstream stage. Such loops could be formed in the result forwarding paths, for example. Since the stages are unified there should be no need to use these forwarding paths and the control logic should disconnect them by setting the appropriate multiplexors, which are present in these paths already. An alternative solution is not to collapse the specific bits of the pipeline registers that will form loops, which is the choice preferred in [12]. Unfortunately this solution presumes that those bits are still clocked by the original, fast clock signal. Thus two clock signals are needed which means that a considerable proportion of the energy savings, from not switching all of the clock network, is lost.

From the above discussion it is clear that PDA probably seems 'unnatural' for a synchronous design. The de-

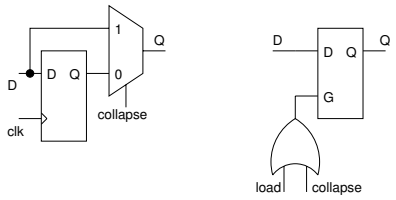


Figure 1. Collapsing pipeline registers.

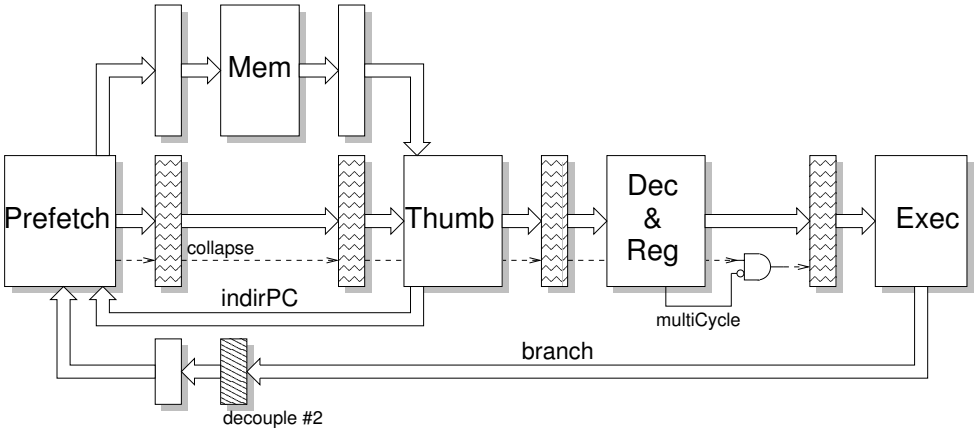


Figure 2. Block diagram of AMULET3 with collapsible pipeline latches.

signer must have a clear plan about which combinations of pipeline registers can be collapsible, so that the appropriate clock signals can be routed to the right registers and the pipeline interlock mechanism can be configured accordingly. For these reasons the existing synchronous implementations offer only 2 ([10]) or 3 ([12]) possible pipeline depths: the original depth d, $d/2$, where every other register is collapsed, and $d/4$, where only one in four consecutive pipeline latches is left uncollapsed.

2.2 PDA In Asynchronous Processors

In the remainder of this paper we show the options available for PDA using an asynchronous processor. Since the global clock is replaced with a number of local 'clocks' produced by handshake signals, there is considerably more flexibility in the timing of an asynchronous processor. Moreover, the asynchronous processors, under certain circumstances, consume less power than the corresponding synchronous ones ([8]), therefore they are more suitable for devices that should consume as less energy as possible. The explicit communication/synchronization among the different stages, in an asynchronous processor, has the additional advantage of simplifying the interlock mechanism used in the pipeline.

The details for the circuit and microarchitecture modifications implemented to AMULET3 [14] (an asynchronous ARM processor) in order to support PDA are described in [9]. A standard latch controller is implemented with approximately 10 logic gates, while the collapsible controller requires about double the number of gates. The area overhead of this technique is minimal, since only the processor's 5 latch controllers need to be modified, a tiny proportion of the total processor area.

Another worthwhile feature of PDA in an asynchronous environment is that changes to the configuration of a pipeline latch can be made dynamically while the processor is operating. For example, in AMULET3 some instructions (e.g. long multiplications) require multiple execution cycles to complete, so when the pipeline stage between *decode* and *execute* is collapsed (fig. 2), these instructions cannot be executed. A good solution is to temporarily clear the collapse signal for the decode-execute pipeline latch. This re-instates the pipeline stage between decode and execute for the duration of the multi-cycle instruction, so that multiple execution cycles can be performed as required. Section 4 shows another example of dynamic PDA which involves more pipeline latches than in the simple case above.

3 Experimental Results for Static Pipeline Adaptation

The proposed architecture has been evaluated by simulating a mixed behavioural/structural Verilog model of it. The proposed processor is essentially an AMULET3 with the latch controllers being replaced with collapsible ones and some other minor modifications mainly for distributing the appropriate control signals. One of the reasons for using AMULET3 as the basis of our design is that it supports the instruction set of the ARM processor, which is a processor very widely used on mobile devices. Therefore, since the presented architecture's sophisticated features are transparent to the software, the presented CPU can easily be incorporated in such devices, since no alterations is software would be needed.

A block diagram of the newly introduced parts of the processor is shown in figure 2. All the stages shown in the figure are the comprehensive pipeline ones, with the possible exception of 'Thumb' which is an optionally active instruction pre-decoder, decoding a number of 16-bit instruc-

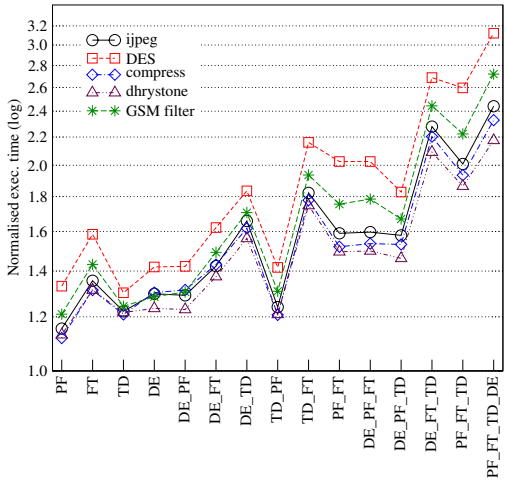


Figure 3. Effect on execution delay.

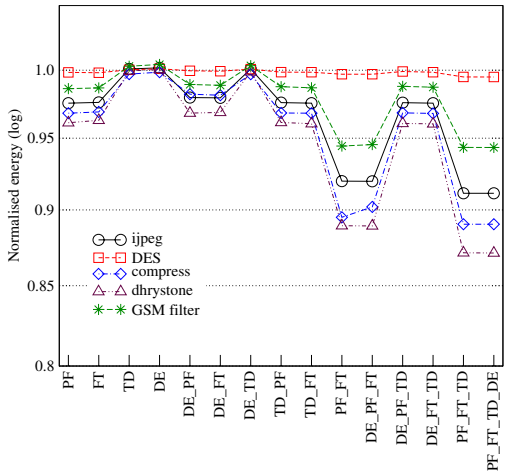


Figure 4. Effect on energy consumption.

tions supported by AMULET3. The processor is not deeply pipelined, thus the energy savings of PDA are expected to be limited. A processor with a larger number of pipeline stages would achieve considerably greater energy savings by adapting its pipeline depth.

The energy consumption of the processor was estimated by counting the number of transitions of the processor's nodes and multiplying by the corresponding capacitances (incl. interconnect) which were extracted from the post-layout AMULET3 netlist, designed in a 0.25 μm technology.

The benchmarks were chosen so as to represent typical applications for mobile devices. Therefore, we have used (a) Dhrystone a string handling benchmark representing the performance of the similar applications running on mobile phones (e.g. handling the phone book), (b) a filter selected from an implementation of GSM encoding, (c) a DES encryption program very frequently used in any wireless devices, and two SPECInt95 programs used in wireless environments where both storage and network bandwidth are limited : (d) compress and (e) ijpeg. All are written in C and compiled with speed optimizations enabled using the compiler provided with the ARMtools 2.51. The input size for all benchmarks was relatively small, so that the simulation can complete in reasonable time. For the same reason ijpeg was only allowed to do one cycle of compression and decompression.

Of these benchmarks, DES encryption differs as it comprises almost entirely of sequential code; the only branches that occur are a few subroutine calls. Thus there is almost no energy wasted in erroneous prefetch.

Figures 3 and 4 show the effect in delay and energy of collapsing every combination of pipeline latches in the processor. Table 1 contains the key to the pipeline latch names. So the rightmost points in the graphs show the results when all the pipeline stages are collapsed, whereas the leftmost points correspond to the collapse of only a single pipeline stage. The reported values are normalized with respect to the fully-pipelined machine. When all the pipeline stages are unified the execution delay increases by as much as 3.12 times (DES), while the energy can be reduced by up to 13% (Dhrystone).

It has to be noted that the actual average pipeline occupancy observed in the fully-pipelined processor is slightly over three simultaneous instructions. This is because the instruction memory is significantly slower than the processor and the Thumb stage which decodes 16-bit instructions is almost never occupied as the benchmarks used here are all using the 32-bit instruction set.

4 Dynamically Adapting the Pipeline Depth

Using the collapsible latch controllers to set a constant processor pipeline depth for the duration of a program execution, as shown above, is the most conservative way to use PDA. As an example of *dynamically* (i.e. while the program is executed) adapting the pipeline depth we present a scheme where the whole pipeline is collapsed when there is an indication that a branch may be imminent. This saves the energy wasted in prefetching instructions beyond the

Table 1. Pipeline latch names.

PF	Prefetch - Fetch
FT	Fetch - Thumb
TD	Thumb - Decode
DE	Decode - Execute

branch, while reducing the pipeline depth for only a fraction of the execution time.

Since we are not interested in finding the branch target, a simpler and more energy efficient technique than standard branch prediction methods can be used. It is reported that over 80% of the branches are conditional [15]. Thus an instruction that sets a branch condition could be used as a hint that a branch is approaching. As mentioned in the last section, AMULET3 implements the ARM architecture which uses condition codes to specify the branch condition. For processors that do not use condition codes, comparison-type instructions could be detected instead. Once such an instruction has been detected, the processor could be configured to a pipeline depth of 1, so that when the branch arrives no following instruction will be fetched until its target has been computed. When the first instruction from the branch target is fetched, the processor will resume its normal operating mode. We call this scheme condition code detection (CC-detection).

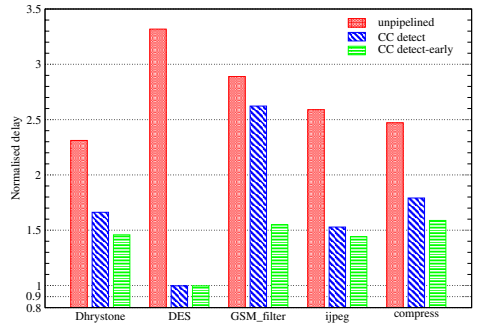
Detecting instructions that change the condition code is simple in the ARM instruction set architecture as there is a specific bit in the data-processing instruction format which controls this. It is important that the detection occurs as early as possible in the pipeline; the number of stages between prefetch and the detecting stage determine how many instructions following the one which sets the condition code have already been fetched and thus how much energy might be wasted. In our modified AMULET3 the detection can be done at the first decode stage (Thumb in figure 2), so only one instruction may be already fetched before the detection happens. This is very useful as over 50% of branches in the benchmarks used here, immediately follow the instruction (usually compare - CMP) that sets the condition codes.

In the ARM instruction set architecture all instructions can be conditional. Thus if the condition is set for a non-branch instruction, the processor will stay in the single-stepping mode until a branch happens to be taken. To test the impact of this in the execution delay, another variation of the CC detection scheme was also designed and evaluated. This scheme detects instructions that are conditional but not branches and forces the processor to return to the fully-pipelined mode (CC detect-early), when it realises that there is no branch to be taken. This method is expected to improve the execution delay without significantly compromising the energy savings achieved.

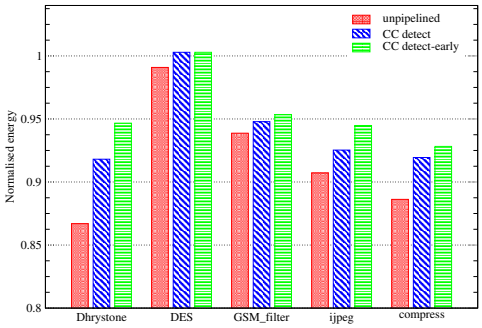
4.1 Evaluation

The two condition-code setting detection techniques require sufficiently small changes to the processor. In order to measure their efficiency two Verilog models were produced and simulated following the methodology described earlier.

The execution delay of each benchmark, for the two variations described above and the fully unpipelined version, are shown in figure 5(a), normalized relative to the fully-pipelined version. Figure 5(b) shows the normalized energy consumption for the same programs and PDA techniques.



(a) Delay



(b) Energy

Figure 5. Evaluation of condition code setting.

DES encode has almost no speculative instruction fetches, so there are insignificant energy savings to be gained by using a shallower pipeline. Thus the interest is in the execution delay overhead of the method tested. In this case both condition-setting detection variations managed to keep the execution delay to the levels of the fully-pipelined version. In comparison the fully unpipelined version is over three times slower.

The difference in the two variations can be seen in a benchmark like *GSM filter*. The execution delay of *CC-detect* is very close to that of the unpipelined processor, while the execution delay of *CC detect-early* is almost half that of the former. As the *GSM filter* code has many conditional data-processing instructions, it clearly benefits from restoring the fully-pipelined mode early.

Generally, both variations managed to reduce the execution delay, compared to the fully unpipelined processor, for an increase in energy consumption. *CC detect-early* is consistently faster than *CC-detect*, but it also consumes more energy. It is very useful for benchmarks that have a large number of data-processing conditional instructions, such as *GSM filter*. This may be less significant in other processors which do not have conditional data operations. By using those techniques we managed to reduce the delay overhead to just 50% of the fully pipelined version, while still reducing the overall power consumption by up to 12%.

Unfortunately, the energy delay product of both variations of this technique is lower than that of the fully pipelined processor. Thus they are useful in systems where the energy consumption is more important than the performance, and this is very often the case in the wireless devices.

5 Conclusions

This paper presented a low power processor asynchronous architecture, tailored to the wireless environments. The proposed architecture incorporates "pipeline depth adaptation", a microarchitecture-level, power-aware technique. Using this technique the presented processor is able to alter its pipeline depth, while in operation, trading off speed for energy reduction.

The pipeline depth is changed by making selected pipeline registers transparent. In order to employ PDA, the required circuit modifications are minimal and the resulting processor can change its pipeline depth from 1 to its maximum of 5. As the basis CPU -even though it is very widely used in embedded systems- is not a deep pipelined machine and the actual maximum pipeline occupancy is 3, the energy benefits of collapsing the pipeline, in a static manner, are not as profound as those expected for a significantly deeper pipeline: up to 13% less energy is consumed when the pipeline depth is 1, while the execution delay can increase by up to 3 times. In order to further increase the performance of the presented architecture, while maintaining its low power consumption, a number of methods of adapting the pipeline depth in real-time have also been incorporated and evaluated. Those methods lower the pipeline depth when a branch instruction is expected, and increase it after the branch is executed. As the experiment results demonstrate, those schemes increase the performance (compared to the static scheme) of a number of representative applications while they still reduce the power consumption compared to the fully-pipelined non-collapsible underlying architecture.

References

[1] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Dig. of Technical Papers International Solid-State Circuits Conference*, pages 294–295, February 2000.

[2] L. Clark, E. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. Velarde, and M. Yarch. An embedded 32-b microprocessor core for low-power and high-performance applications. *IEEE Journal of Solid-State Circuits*, 36(11):1599–1608, November 2001.

[3] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the International Symposium on Computer Architecture*, pages 132–141. ACM Press, June 1998.

[4] Tejas Karkhanis, James E. Smith, and Pradip Bose. Saving energy with just in time instruction delivery. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 178–183. ACM Press, August 2002.

[5] David H. Albonesi. Dynamic IPC/clock rate optimization. In *Proceedings of the 25th Annual International Symposium on computer Architecture*, pages 282–292, 1998.

[6] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November-December 2000.

[7] A. Efthymiou and J. D. Garside. Adaptive pipeline depth control for processor power-management. In *Proceedings of International Conference on Computer Design*, pages 454–457. IEEE Computer Society Press, September 2002.

[8] A. Efthymiou, J. D. Garside, and S. Temple. A comparative power analysis of an asynchronous processor. In *Workshop on Power And Timing Modelling Optimization Simulation*, September 2001.

[9] A. Efthymiou and J. D. Garside. Adaptive pipeline structures for speculation control. In *Proc. of the 9th Intl. Symposium on Asynchronous Circuits and Systems*, pages 46–55, May 2003.

[10] Jinson Koppanalil, Prakash Ramrakhyani, Sameer Desai, Anu Vaidyanathan, and Eric Rotenberg. A case for dynamic pipeline scaling. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–8. ACM Press, 2002.

[11] Prakash Ramrakhyani. Dynamic pipeline scaling. Master's thesis, Department of Computer Engineering, North Carolina State University, 2003.

[12] Hajime Shimada, Hideki Ando, and Toshio Shimada. Pipeline stage unification: a low-energy consumption technique for future mobile processors. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 326–329. ACM Press, 2003.

[13] Victor Zyuban and Philip Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 166–171. ACM Press, August 2002.

[14] Stephen B. Furber, Douglas A. Edwards, and James D. Garside. AMULET3: A 100MIPS asynchronous embedded microprocessor. In *Proceedings of International Conference on Computer Design*, pages 329–334. IEEE Computer Society Press, September 2000.

[15] John Hennessy and David Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, second edition, 1996.