A Comparative Power Analysis of an Asynchronous Processor

Aristides Efthymiou, Jim D. Garside, and Steve Temple

Department of Computer Science,University of Manchester
Oxford Road, Manchester, M13 9PL UK
{efthym,jdg,st}@cs.man.ac.uk

Abstract. Distributing a high-speed clock in a large synchronous system is both difficult and power hungry. It has been suggested for some time that asynchronous processors may therefore prove advantageous for low power applications. An analysis of the reasons for this is given, together with a direct comparison showing that the AMULET3 asynchronous ARM processor is at least as energy efficient as its contemporary, synchronous counterpart. Moreover micro-architecture techniques utilising the asynchronous design style that can further improve the power efficiency, are presented.

1 Introduction

Low power consumption has been one of the arguments advanced for the renewed interest in asynchronous design. The chief argument is that an asynchronous (or 'self-timed') system only performs processing 'on demand'. Thus a processor with no work to do can, and will, halt. In a low-leakage CMOS technology halted logic uses very little power.

This argument can be extended to any functional block, so that a part of the processor which is not required will not be operating and wasting energy. An example of this would be multiplication hardware in a general purpose processor, which is infrequently invoked. This argument is sufficiently compelling that many low-power synchronous processors now use clock gating on their subcircuits.

A more subtle influence is that self-timed circuits can be designed to allow for variation in operating speed for extreme circumstances. This avoids the need to introduce specialised hardware to speed up rare, worst-case behaviour to fit into a fixed clock cycle. An arithmetic operation provides a good example where an operation can always take one cycle, but the cycle may be longer for a (rare) multiplication than for a (common) addition or data movement.

AMULET3i [1], an asynchronous ARM-compatible processor system is used here to demonstrate that asynchronous processors are now commercially viable in both performance and power consumption. The energy consumption of the AMULET3 [2] microprocessor (in MIPS/W) is the same as its contemporary synchronous counterpart, ARM9, in the same design rules even when running at full speed. This represents significant progress as ARM is a market leader in 32-bit low power processors.

In addition the asynchronous system can halt (and restart) within a single processor 'cycle', a facility not available in the clocked ARM. When halted AMULET3's power consumption is almost zero.

This paper describes some of the low power design features of the processor, presents a power breakdown of the system and analyses the major power inefficiencies. A comparison with synchronous systems, especially with the equivalent processor, is presented next. Finally, the paper concludes with some suggestions of how asynchronous techniques could further improve energy efficiency in the future.

2 Asynchronous Design for Low-Power

Several features of the current AMULET3 implementation were designed for low power, some of which are applicable to synchronous devices too. Some features which exploit the asynchronous nature of the processor are described below.

AMULET3 implements the ARM version 4T architecture [3], including the 16-bit compressed Thumb instruction set. This compressed subset of the 32-bit instruction set is converted into 'standard' 32-bit operations internally. The primary motivation for Thumb was to compress code; a program compiled into Thumb instructions occupies about 70% of the memory of the same program compiled into ARM instructions. Of course this represents 40% more instructions and is therefore not, in itself, a low power feature in the processor, although the 30% reduction in instruction bandwidth yields a significant power saving in the memory system.

Normally synchronous ARMs proceed at one clock per instruction. This means that, when executing Thumb code, only sixteen bit quantities are fetched so as to maintain the instruction pipeline balance. AMULET3 always fetches 32 bit quantities, halving the number of instruction fetch cycles. If these are to be decoded as Thumb operations they are split at the decoder which produces two output cycles for a single input cycle. The difference in the number of cycles is accommodated by the delaying of subsequent memory cycles and the reduction in bus operations (hence power consumption) is an automatic consequence.

A complementary technique is possible for other operations. For example a comparison, which does not write back to the main register bank, can 'evaporate' from the pipeline once the flags are set; there is no need to propagate a NOP to fill spare pipeline spaces.

Some other features, introduced primarily to increase performance, also have a beneficial effect on the power consumption. One notable example is the branch target buffer[4]. This is a relatively simple branch predictor which nevertheless increases performance by up to 22%, depending on the application. In many processors the branch predictor burns more power than it saves by avoiding erroneous speculative instruction fetches. Here, however this is not the case, primarily because of the small, relatively simple, mechanism used, and the system power consumption is reduced by up to 20%. This is assisted by an asynchronous, two-stage address comparison which exploits the largely sequential nature of the

instruction stream; comparisons are only performed with a few of the least significant bits unless there is a reason for a slow, full address comparison. Another feature is that a predicted branch is retained internally and so does not require an instruction fetch memory cycle, a reduction of about 10% in fetch cycles.

3 Power Analysis

A detailed power analysis of the processor was undertaken (before silicon was available) to provide a basis for further improvement. The analysis is based on simulations because there is no other way to get a power breakdown of the processor at an arbitrary level. The simulator used was Powermill which is a well known tool used widely for estimating power consumption.

The measured results for the processor were somewhat different from the simulation results, the actual processor being slower and consuming less power than expected. Unfortunately no data on the particular fabrication run have been obtained which might explain this. Nevertheless the simulation results are still useful for the relative contribution to the power consumption by the various parts. As all the processor parts are similarly designed, the simulation error should be spread fairly evenly across the subcircuits.

The power consumed by a processor depends on the operations it performs. Unfortunately there is no de-facto benchmark used to measure processor power as there is for performance (e.g. SPEC). Many published results are based on Dhrystone [5], which is a synthetic benchmark that claims to have a dynamic mix of instructions similar to 'typical' application programs. For easy comparison with other processors, the results of using Dhrystone will be most used here. Other programs used in this analysis were DES encryption/decryption and a short term synthesis filter from an implementation of GSM. All the benchmarks are written in the C language and were compiled for speed using the ARM toolkit version 2.51. Simulation time and the small memory size (8 Kbytes) in the AMULET3i were the greatest limiting factors in the selection of the benchmarks. Full transistor level simulation was chosen to obtain the most accurate results possible. The resulting long simulation time limits the number of instructions that can be simulated in reasonable time.

The top-level system power breakdown for Dhrystone is shown in Fig. 1. The core is the major consumer responsible for around 60% of the power for Dhrystone and more for the other benchmarks (up to 73%). The 8 Kbyte RAM consumes from 20% (filter) to 30% (Dhrystone). The remaining portion, which is less than 10%, is consumed mostly by the system bus, as the asynchronous peripherals were not active in the benchmarks.

The power breakdown within the AMULET3 core running Dhrystone is presented in Fig. 2. In this simulation all performance enhancing options were enabled, including the branch prediction hardware. The biggest consumers were found to be the execution and register blocks, followed by the prefetch unit, when branch prediction is enabled, and the decode block.

The percentage used by the prefetch unit drops to 13% when branch prediction is turned off and the core's power consumption is 6% lower. The execution time increases only a little (1%), because the branch prediction method used does not work well for Dhrystone. The situation is reversed when running the GSM filter benchmark, where branch prediction works very well, removing many wrong instruction fetches and increasing performance by 8%.

The block called 'Registers' contains the register file and the reorder buffer and its power consumption is similar to that of the execution unit in all the benchmarks except for the GSM filter, where the heavy use of the multiplier makes the execution unit the major consumer.

3.1 Inefficiencies

The detailed power analysis revealed some inefficiencies in the design which were not identified before manufacture. The most serious problem was in the multiplier circuit. The proportion of energy expended in this unit seemed too high considering that it is used infrequently in the benchmarks. Further study showed that a significant part of the multiplier is always active evaluating whatever data happen to be on the multiplier inputs; these buses are shared with some other functions, notably the data output to memory, and so switch moderately frequently.

This problem, which would be quite easy to fix at a very small cost with input gating, shows the importance of carrying out power analysis as early as possible in the design phase. The improvement in power by fixing this depends heavily on the specific data carried by the buses and the instructions executed. For the DES benchmark, for example, 5.5% of the core's power would be saved.

About 15% of the execution unit power is spent by the ALU output drivers and the result bus drivers. The ALU output is one of the two possible sources of the result bus and it is also connected to the data interface block for calculated addresses (pre-indexed addressing). To make the usual case fast, the result bus is driven by the ALU output by default. A significant proportion of ALU results (around 30% in the set of benchmarks used) are not stored, either because they are memory addresses or because they are the result of comparisons. Moreover the ALU output appears to be quite 'glitchy', especially during subtractions,

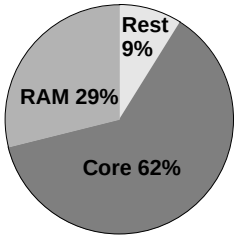

Fig. 1. Processor power breakdown (Dhrystone).

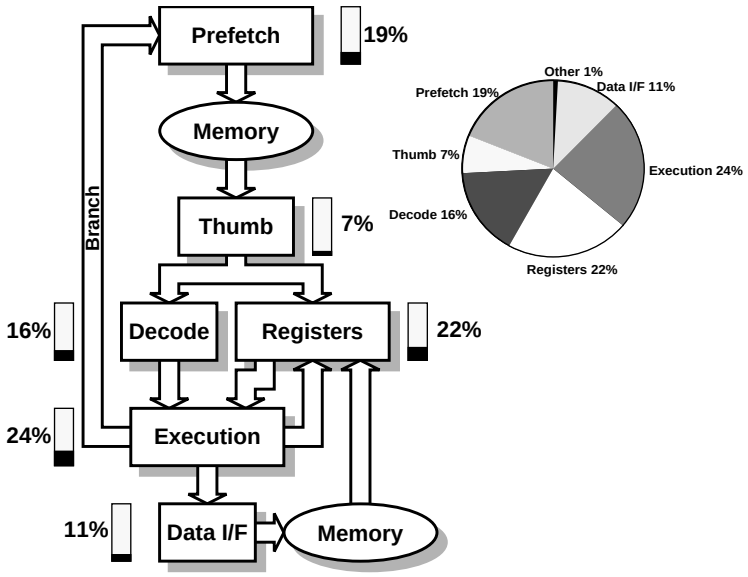

Fig. 2. AMULET3 core power consumption.

because of races in the carry circuitry. These glitches propagate onto the result bus, thus wasting significant energy. All of these problems could be solved by appropriate circuit techniques (gating or latching), but this may increase the power spent in the control part of the processor, so it is not clear yet which is the best solution.

AMULET3 uses a three read port register file so that all of the operands an instruction might require can be fetched simultaneously. Most instructions require fewer than three operands. Analysis on the frequency of the use of each register file port showed that one port is used very frequently, the second is less than half as often and the third even less frequently. Each register read port is a dynamic precharge/discharge bus. Although these buses are not activated when not required, the precharge is a broadcast signal. Because the register ports have significant capacitance the precharge transistors are large and the signal driving them is heavily loaded. Thus using a unified precharge signal is not efficient in terms of power. An estimated 12% of the register file power (1% of core) would be saved simply by using the separate requests for each port as their respective precharge signals.

4 Comparison with Synchronous Systems

In a synchronous processor a large proportion of the power is attributed to the clock. Tiwari et al. [6] state that about 40% of CPU power is spent on the clock, including the generator, drivers, distribution tree and loading. StrongARM [7],

with a similar architecture to AMULET3, is reported to use 26% of its power in the clock, including the PLL. Other processors state similar results.

It would be interesting to estimate the equivalent of this power in AMULET3. As there is no simple way to determine which circuits should be considered equivalent to the clock, the choice is somewhat arbitrary. The closest equivalent is the set of latch controllers of the pipeline latches between the submodules of the core; these include the drivers for the large latch enable loads. In addition some precharge signal drivers which are controlled by handshake signals were also included in this group. The contribution of all those circuits to the power consumption was found to be 10.5% of the core while running Dhrystone 2.1. Comparing this with the proportion of the power taken by a clock shows that asynchronous techniques can significantly reduce power consumption.

Naturally the benefits of the asynchronous design style come at a cost. As the different stages in the pipeline are not synchronised, state information between pipeline stages is difficult to exchange. This leads to duplication of information in several places in the pipeline. For example, in AMULET3 each pipeline stage holds the address of the instruction being processed there because of the difficulty in accessing a central PC. This is not a significant power overhead, because only few of the PC's bits switch each time. In addition to duplicating information, the fine grain control of circuits leads to the existence of more state/sequencing information compared to a synchronous processor. This translates to an increase in control power for the processor, proportionately about 40% of the processor core power in AMULET3. This is quite high compared to other published results, although this includes most of the "clock" power stated above. It has to be noted though that the control circuits are implemented using standard cells, automatically placed and routed by CAD tools, whereas the datapaths are full-custom. Thus the wire capacitances tend to be higher and there is less control over the driving strength of the gates. As synchronous low-power processors are increasingly using extensive clock gating and functional unit guarding techniques their control units tend to be as complicated as their asynchronous counterparts. Moreover, the ARM architecture is quite complex for a RISC machine which makes the control logic inherently more difficult and power consuming. ARM7 [8], which implements the previous version of the ARM architecture, is stated to consume 40% of its power in the control part, although the definition of control circuits may be different.

4.1 Comparison with ARM9

ARM9TDMI is the synchronous implementation closest to AMULET3; both execute the same instruction set and have been implemented on identical technology, occupying the same silicon area (about 4mm^2). On this 0.35 μm process ARM9 [9] operates at up to 120 MHz having a performance of 1.1 MIPS/MHz and consuming 1.8 mW/MHz. This gives an energy per instruction metric of 610 MIPS/W. Unfortunately a power breakdown has not been given.

The measured results from AMULET3i show a power consumption of 221 mW at a performance of 85 Dhrystone MIPS. According to the simulation re-

sults, the core consumes 62% of the power (137 mW). This gives an energy per instruction figure of 620 MIPS/W, effectively the same as that of ARM9.

The performance is slower than anticipated from simulation, which predicted a speed of about 100 MIPS for the system (it is not known how 'typical' the silicon run was.) Furthermore this was limited by the (unoptimised) memory system; the core alone runs at about 130 MIPS in simulation. Thus, it is safe to assume a speed of over 100 MIPS for the processor on silicon, without the limitations of the memory.

5 Power Improvement for Asynchronous Processors

The power consumption of an asynchronous system depends on the same factors as a synchronous one, if the throughput is considered to be the average execution frequency. Thus the greatest benefits come from scaling down the supply voltage. Taking this a step further, dynamically adjusting the voltage level to match the required performance gives excellent results [10], [11].

Although not explicit in AMULET processors this technique is easy to implement in an asynchronous framework and may provide more opportunities for power saving when exploiting data dependencies [12]. Since there is no clock, there is no need to adjust the clock frequency to match the supply voltage. Providing that the matched delays scale to the different voltages, the system will be working at its maximum possible speed for this voltage.

Crusoe [13] is a synchronous processor supporting dynamic voltage scaling. When it switches to a lower supply voltage it stops execution (deep sleep mode) for 20 μs, and restarts at a lower frequency before ramping down the supply voltage. It takes 300 μs to drop from 1.6 to 1.1V. 'lpARM' [14] takes 26 μs to ramp the voltage from 3.3V to 1.1V although this processor can continue operating while the voltage is changing due to careful circuit design. There is an energy penalty for changing the supply voltage which even with a 90-80% efficient DC-DC converter can be up to 4 μJ [15].

For minimum area and power consumption AMULET3i is implemented using bundled data techniques [16]. Because this depends on matched delays it is not guaranteed that the circuits will continue to work at different supply voltages without extensive simulation. However, in practice, almost all the circuits used for delay elements are close analogues of the circuits they model and the processor can work over a wide range of voltage supplies[14]. Other asynchronous techniques, such as delay insensitive circuits, are guaranteed to work under all conditions; the disadvantage of such circuits is that both the circuit size and the switching activity will approximately double, thus rendering them less suitable for low power systems.

Although scaling the supply voltage gives significant power benefits there are cases where either the supply voltage is a fixed design parameter or there is a need for even lower power consumption after the supply voltage has been set to its minimum practical value. Moreover, dynamic voltage scaling requires DC to DC converters which expend some power themselves. For these reasons

micro-architecture techniques utilising the asynchronous design style are being investigated to reduce energy expenditure even further.

5.1 Dynamic Pipeline Occupancy/Depth

The idea here is to eliminate energy spent on speculative operations. One extreme is to remove all pipelining from the processor. An instruction will only be fetched after the current one has executed, so no instructions will be fetched speculatively. In order to save as much power as possible, all blocks that are not going to be used should be turned off. The most important of these are the branch prediction hardware and the reorder buffer.

Rather than removing the pipeline latches, the same effect can be achieved by adjusting the pipeline occupancy. Using appropriate circuits the occupancy can be varied from one (unpipelined) to the maximum (fully pipelined), trading performance for power. A straightforward implementation could use a token FIFO with as many spaces as pipeline stages between prefetching and execution (Fig. 3). Prefetch would remove a token before trying to fetch an instruction and execute would insert one for each executed instruction. The number of tokens in the system will determine the pipeline occupancy at any time. External circuits could be added to insert or remove tokens to change the pipeline occupancy dynamically.

In all but the fully pipelined case some energy will be saved, because fewer speculative operations will be wasted, but at the same time performance will be degraded. The effect is software dependent but in the AMULET3 model it has been shown to be possible to reduce the energy requirement of Dhrystone by 12%, albeit at a 70% performance reduction (16%, 74% respectively when switching branch prediction off). Although the energy-delay product in the less occupied pipeline does not compare favourably to that of the full pipeline there are circumstances where this would be an acceptable trade-off. As the control can be dynamic, the occupancy can always be increased when required.

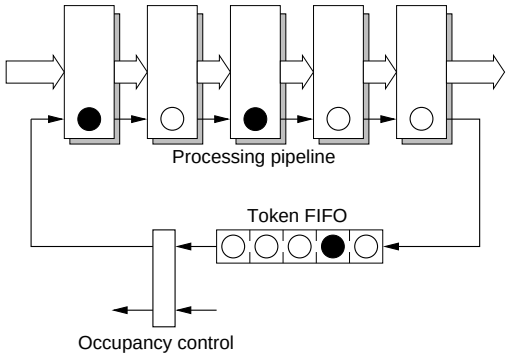

Fig. 3. Simple pipeline occupancy control.

The FIFO control is a simple retro-fit to an existing asynchronous processor. It gains its benefit from reducing speculation but still suffers from the disadvantage that all the pipeline latches are still switching. A similar technique to reduce pipeline occupancy can be implemented by 'collapsing' pipeline stages so that adjacent stages are combined with the latch between them continuously transparent. For example when ARM code is being executed the Thumb stage (see Fig. 2) is merely a FIFO buffer. If this stage is subsumed into the decode stage the branch penalty will be less severe and fewer circuits need to switch on every cycle. As the latch controllers are responsible for 10% of the processor's energy budget this should prove a significant economy. Again it is possible to adjust the controllers to change the asynchronous pipeline structure dynamically.

5.2 Conditional Stalls

All ARM 32-bit instructions can be predicated. While this removes some branches, in order to keep the performance as high as possible, conditional instructions are usually decoded and their operands are fetched before the condition on which they depend is evaluated. Segars [9] states that 10% of the instructions are skipped, thus the energy these instructions consume decoding and reading their operands could be saved.

Depending on the status of a configuration signal (enabling this feature), the decoder, when receiving a conditional instruction, could request the latest flags from the execution unit before continuing decoding. If the instruction currently being executed is going to change the flags the decoding must wait until the flags are set, thus no work will be done speculatively. This can be implemented with a separate handshake channel between the two stages. The increase in decode time is accommodated by the asynchronous pipeline.

6 Conclusions

The recently built AMULET3 processor proves that the asynchronous design style is suitable for low power processors. It equals the leading 32-bit embedded processor in energy efficiency when fabricated in identical 0.35 μm technologies, operating at the same supply voltage and running the same code.

A detailed power analysis showed that the equivalent of clock power is only about 10% in AMULET3, significantly less than synchronous processors. The same analysis also showed that, with simple circuit modifications, the power consumed by AMULET3 could be reduced by another 8%.

Further improvement in power consumption is possible. The dynamic voltage scaling techniques used in synchronous processors can be applied easily because there is no need to adjust the clock frequency. Other microarchitectural techniques based on asynchronous design have also been presented; these can control the speculative operations in the processor by controlling the pipeline occupancy or conditionally stalling the pipeline. Preliminary results on these techniques indicate that up to 12% more can be cut from the energy budget.

AMULET3 has shown that asynchronous microprocessors can have competitive energy efficiency with the best clocked designs. The architectural flexibility of asynchronous design offers the prospect of significant further savings, with a dynamic power/performance trade-off, in future implementations.

7 Acknowledgements

The authors would like to thank the other members of the AMULET group for their contributions, especially Viv Woods who produced some of the data presented here. A. Efthymiou is supported by a scholarship from the Department of Computer Science, University of Manchester. This support is gratefully appreciated.

References

[1] J.D. Garside, et al.: AMULET3i - an Asynchronous System-on-Chip. Proc. Async 2000 Eilat, Israel (Apr. 2000) 162-175

[2] J.D. Garside, S.B. Furber, S-H. Chung: AMULET3 Revealed. Proc. Async'99, Barcelona (Apr. 1999) 51-59

[3] S.B. Furber: ARM System Architecture. Addison Wesley Longman, (1996) ISBN 0-201-40352-8

[4] S.B. Furber, *et. al.*: Power Management in the AMULET Microprocessors. IEEE Design and Test of Computers **18(2)** (Mar/Apr. 2001) 42-52

[5] R. Weicker: DHRYSTONE - A Synthetic Systems Programming Benchmark. Comm. ACM **27(10)** (1984) 1013-1030

[6] V. Tiwari,*et al.*: Reducing Power in High-Performance Microprocessors. Proc. of the Design Automation Conference (1998) 732-737

[7] J. Montanaro, *et al.*: A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. IEEE Journal of Solid-State Circuits **31(11)** (Nov. 1996) 1703-1714

[8] S. Segars: ARM7TDMI Power Consumption. IEEE Micro **17(4)** (Jul/Aug. 1997) 12 - 19

[9] S. Segars: The ARM9 Family - High Performance Microprocessors for Embedded Applications. Proc. ICCD'98, Austin, (Oct. 1998) 230-235

[10] T. Burd and R. Brodersen: Processor Design for Portable Systems. Journal of VLSI Signal Processing **13(2-3)** (Aug. 1996) 203-222

[11] V. Gutnik, A.P. Chandrakasan: Embedded Power Supply for Low-Power DSP. IEEE Trans. on VLSI Systems **5(4)** (Dec. 1997) 425-435

[12] L. S. Nielsen,*et al.*: Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage. IEEE Trans. on VLSI Systems, **2(4)** (Dec. 1994) 391-397

[13] M. Fleishmann: Crusoe Power Management: Cutting x86 Operating Power Through LongRun. Symposium Record Hot Chips 12, Stanford (Aug. 2000)

[14] T. Burd, R. Brodersen: Design Issues for Dynamic Voltage Scaling. Proc. of the 2000 ISLPED (Jul. 2000) 9-14

[15] T. Burd, *et al.*: A Dynamic Voltage Scaled Microprocessor System. Digest of Technical Papers ISSCC'2000 (Feb. 2000) 294-295

[16] S. Hauck: Asynchronous Design Methodologies: An Overview. Proceedings of the IEEE, **83(1)**, (Jan. 1995) 69-93