# Power Management in the Amulet Microprocessors

**Steve B. Furber**

**Aristides Efthymiou**

**Jim D. Garside**

**David W. Lloyd**

**Mike J.G. Lewis**

**Steve Temple**
University of Manchester

Amulet microprocessors are asynchronous (clockless) implementations of the ARM 32-bit RISC architecture. Their asynchronous control framework has positive benefits for low-power applications because it reduces activity to the minimum required to perform a task, whereas a clock inevitably incurs wasteful activity.

**THE FIRST 50 YEARS** in the history of computing have seen spectacular improvements in the capabilities of machines. The first computer to run a program stored in its own memory, the Manchester University's 1948 *Baby* machine, as shown in Figure 1, occupied a medium-sized room, used 3.5 kW of electrical power, and executed 700 instructions per second.

The latest machine designed at the University of Manchester, the Amulet3i asynchronous subsystem on the Draco chip, shown in Figure 2, occupies 25 $mm^2$ of silicon, consumes 132 mW, and executes up to 100 million instructions per second. The improvement in the power-efficiency of machines over this 50 year period may be measured by comparing the energy per instruction from Baby to Draco, and the result is a staggering factor of over 3 billion. Few technologies have seen improvements of this order over half a century, and the technology continues to improve.

## Asynchronous design

The use of clock signals dominates modern microelectronic design. Clock-based design methodologies have enabled great progress in design tools and designer productivity. However, for the reasons we will outline later, clocks are disadvantageous for power-efficient design.

Amulet microprocessors do not use clocks. Instead they are based upon asynchronous (self-timed) design techniques where local handshakes control the data flow. There is no global synchronization of activity on the chip, and there is no activity at all unless there is useful work to be carried out.

An alternative to fully asynchronous design is globally asynchronous, locally synchronous (GALS) design, where a system is composed from several clocked modules that communicate using asynchronous interconnections. This enables a more conventional design flow and gives some of the advantages of fully asynchronous design. Here, though, we are concerned with fully asynchronous operation.

## Power-efficient design

Dynamic power dominates the power characteristics of a well-designed CMOS circuit, and it is proportional to the switching frequency, switched capacitance, and the square of the supply voltage. The switching frequency is usually viewed as the product of the clock fre-



0740-7475/01/$10.00 © 2001 IEEE

quency and a measure of *circuit activity*—the proportion of the switched capacitance that changes state in each clock cycle.

In many applications, system power is not the most important issue; rather, the total energy required to perform a particular function is what's important. With this in mind, we look at various strategies aimed at improving energy-efficiency.

- Reducing the supply voltage. This is the most effective means of improving the energy-efficiency of a digital CMOS IC; the dynamic energy per transition is (roughly) proportional to the square of the voltage. The down side is that propagation times increase as the voltage falls, so throughput suffers.

  To address this problem, some designers—notably Transmeta with their Long Run technology[1]—use dynamic voltage scaling to provide just enough performance at any time. This technique is even easier to apply to asynchronous circuits. There is no need to sequence changes in the clock frequency and supply voltage; voltage changes are accommodated automatically.
- Reducing the capacitance. By far the greatest improvements in power efficiency have resulted from changes in the underlying process technology, mainly the reduction in switched capacitance due to smaller feature sizes.
- Reducing the clock frequency. Reducing the rate at which instructions are processed will reduce the power consumption commensurately. However, this will not change the energy per instruction unless the design exploits the reduced performance to use a lower supply voltage.
- Reducing activity. The principal approach to power reduction that is under the circuit designer's control is the suppression of unnecessary circuit activity.

Clocked circuits suffer from constant activity in the clock distribution network and connected circuits. At maximum throughput little is wasted, but most circuits have variable loads. Energy-efficient clocked circuits employ clock gating to reduce the waste in inactive subsystems. Such circuits use clock scaling when the whole system is lightly loaded. However, both mechanisms have an energy cost and require sophisticated management.

Asynchronous circuits, on the other hand, only generate the minimum activity required. They require no sophisticated power management or clock-gating strategy. They do incur a control overhead that has a power cost: Local handshake control circuits are more complex than clock buffers. However, the power cost of these circuits is much lower than that of a clock distribution network.

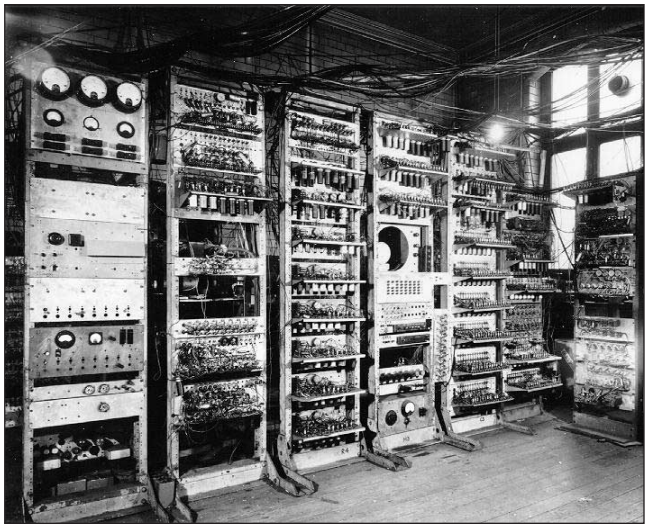

**Figure 1. Manchester University's 1948 Baby machine.**

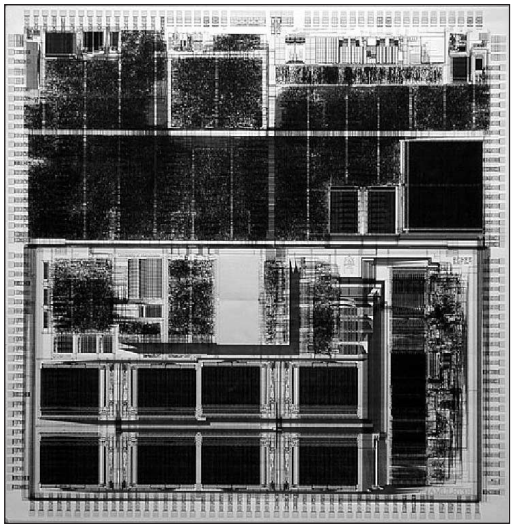

**Figure 2. Draco chip. Manufactured in 2000, this chip uses an Amulet3i subsystem and achieves 3 billion times the power efficiency of Baby.**

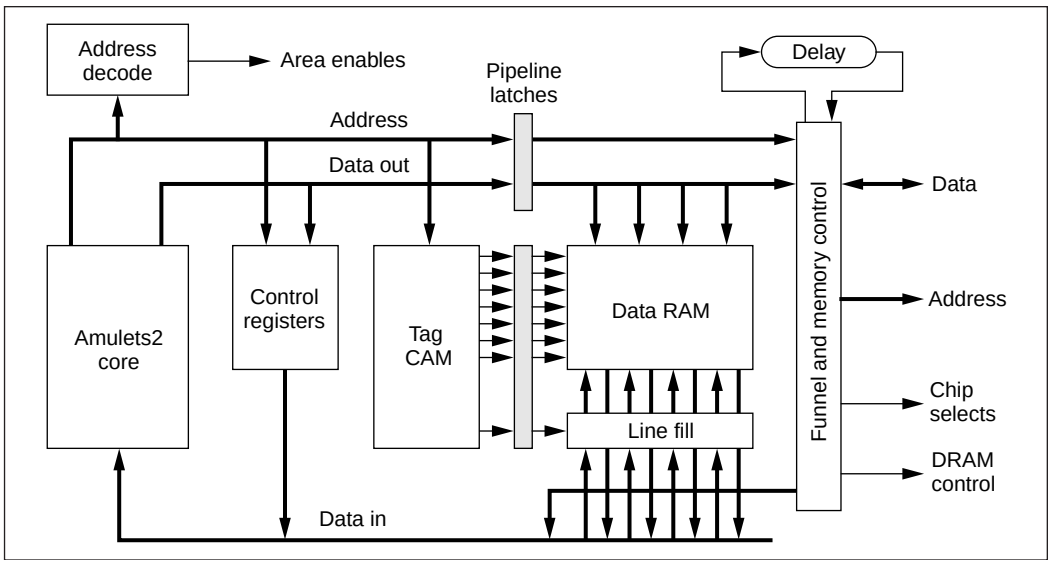


**Figure 3. AMULET2e internal organization. CAM = content addressable memory. DRAM = dynamic RAM. RAM = random access memory.**

### The Amulet microprocessors

The Amulet group at the University of Manchester has spent a decade investigating how to exploit the benefits of asynchronous design in practice through the development of asynchronous implementations of the ARM 32-bit RISC (reduced instruction set computer) architecture.[2]

## Amulet2e

Amulet2e is a self-timed controller designed for power-sensitive embedded applications. Amulet2e's key features are a self-timed ARM-compatible microprocessor core (Amulet2), 4-Kbyte on-chip memory, flexible memory interface, and several control functions. Its organization is illustrated in Figure 3.

### Memory subsystem

The memory system has 4 Kbytes of RAM that can be memory mapped or used as a cache.[3] It is a composition of four identical 1-Kbyte blocks, each having an associated 64-entry tag content addressable memory (CAM). Line fetch is performed with the addressed word first and is nonblocking. This scheme allows hit-under-miss, because the line fetch is a separate, asynchronous task.[4]

### Cache power-efficiency

The cache is a highly associative CAM-RAM structure with a pipeline stage between the two memories. A high degree of associativity is beneficial in a small cache, but unfortunately CAM is power hungry.

To reduce power consumption, CAM activation is suppressed for sequential accesses that lie in the same cache line as their predecessors. This confers the added advantage of reducing the CAM stage cycle time for some cycles. The true advantage in an asynchronous environment is that these variable-length cycles can be exploited.

Speeding up only one pipeline stage doesn't benefit throughput. To exploit sequential qualities in the AMULET2e cache, the RAM look-up is also optimized by suppressing the RAM precharge unless the subsequent cycle is not sequential. With the RAM still in read mode the data for a sequential access is available more rapidly, which matches the faster CAM bypass cycle. An unnecessary precharge/discharge cycle is also averted, which also saves considerable power in the RAM block.

### Branch prediction

Clearly, fetching instructions that are not executed decreases both performance and power-efficiency. The major reason for erroneous speculation is that code is punctuated by branch instructions, many of which are taken after

instructions that follow have been prefetched. Prefetching only executed instructions by using some form of branch prediction saves power.

Branch prediction schemes vary from crude to highly sophisticated. In pure performance terms, the more sophisticated methods are better because they produce more accurate predictions. This may not be the case from an energy-efficiency viewpoint because the predictors themselves can consume considerable power.

Amulet2 uses a relatively simple branch predictor, a branch target buffer (BTB), which is shown in Figure 4. This branch predictor recognizes previously encountered branch source addresses and assumes that they jump to the same place again. Despite its relative simplicity, this predictor can halve the number of wasted instruction prefetches. With about 15 branches in every 100 useful instructions—and surmising a prefetch depth of two (Amulet2's prefetch depth is nondeterministic)—this represents an improvement from 120/100 to 110/100 in the ratio of fetched to useful instructions. It yields approximately a 10% speedup and saves a similar amount of power, less the overhead required to perform the prediction.

The BTB operates with a CAM-RAM structure, caching previously taken branches. The CAM tries to associate each outgoing fetch address with a branch and, when it finds one, substitutes the previous target address. This means that a CAM cycle is performed for every instruction fetch, and even a small CAM is quite power hungry. The power used here can be almost as great as that saved by preventing wasted instruction fetches. However, it is possible to greatly reduce the CAM power by exploiting the highly sequential nature of instruction addresses.

The BTB CAM is asymmetrically split into sets of address bits; the lower four form one set and the upper 26 another, as shown in Figure 5. (Instructions are word aligned so only 30 of the 32 address bits are used.) The upper comparison is only performed when needed, which is comparatively rare. At other times, it is suppressed and the previous result used. This mechanism cuts the BTB's power needs by around 70%, and it can truly claim to give significant power as well as performance benefits.

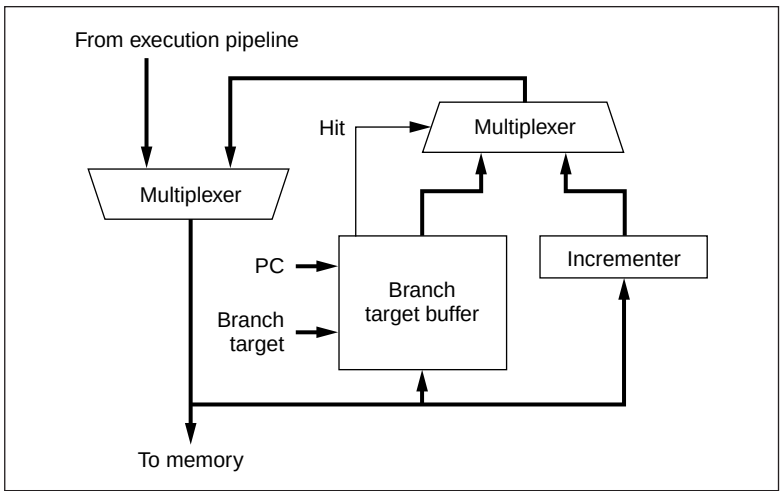


**Figure 4. Branch target buffer organization.**

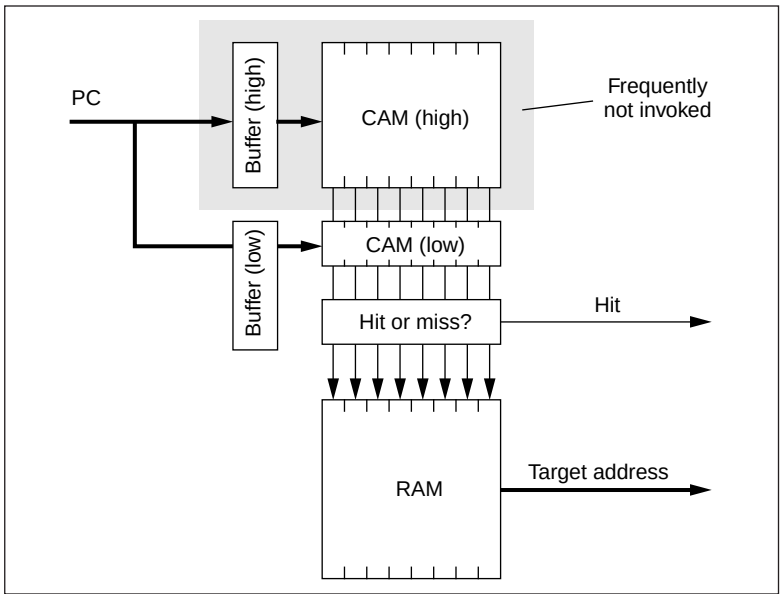


**Figure 5. Branch target buffer structure. CAM = content addressable memory. RAM = random access memory.**

## Halt

Very few microprocessors—whether they are in PCs, laptops, or embedded controllers—work at full capacity all the time. Instead, a typical work pattern may have intense activity followed by idle periods.

A clocked system has a continuously running clock that must meet performance targets in periods of high demand. Clocking it rapidly when idle wastes power. Instead, either the clock frequency should decrease when the processor is idle or some form of sleep mode is needed.

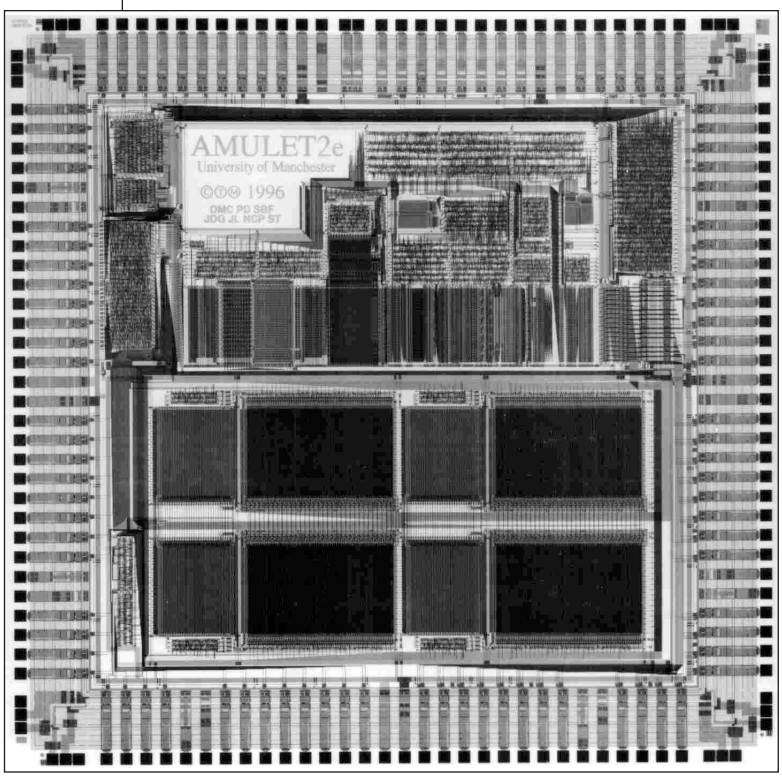


**Figure 6. AMULET2e die plot.**

Reducing the clock frequency saves energy but requires control, usually using software that is relatively slow and incurs its own energy cost. Furthermore, it takes time to bring the clock back to full speed when work arrives.

Putting the processor into a sleep mode is more efficient. In this mode, it is normal to stop the clock until an interrupt occurs. A crystal oscillator or a phase-locked loop (PLL) supplies the clock. Stopping the clock means either stopping the oscillator or gating its output. The first choice is low power but imposes a significant latency penalty when the oscillator restarts. The second option leaves a small but significant residual power drain.

In an asynchronous environment, the choices are simpler. Any part of the system that can run will do so; any part waiting for input or output will be halted. The only two speeds available are therefore full ahead and stopped, the full-ahead speed dictated by the prevailing conditions of voltage, temperature, and so on.

Because a self-timed system relies on all processes cooperating to exchange data, it is very easy to cause the whole system to stop. For example, artificially inserting a check into any point in an asynchronous pipeline will cause downstream stages to drain and starve. Those upstream will back up and halt. When the system halts, dynamic power consumption—the dominant factor in many CMOS processes—drops to nothing.

The Amulet2 microprocessor,[5] while implementing the ARM instruction set (which has no specific halt or sleep instructions) does precisely this. It detects an instruction—a branch back to itself (which would normally define an endless loop)—and uses this to suspend execution. Processing halts in one pipeline stage—and therefore the whole processor—purely through this local action.

The only method of escape from such an endless loop is an interrupt, so Amulet2 allows the assertion of an interrupt input to cause execution to resume. Naturally, the full processing speed is restored immediately.

### Amulet2e test results

Amulet2e, shown in Figure 6, has been fabricated and successfully runs standard ARM code. It was produced on a 0.5-µm, three-metal-layer process. It uses 454,000 transistors (93,000 of which are in the processor core) on a die that is 6.4 mm square. In the fastest available configuration, the device delivers 42 Dhrystone 2.1 MIPS (million instructions per second) with a power consumption of about 150 mW in the core logic—which includes the processor core and cache memory, but excludes the I/O pads—running at 3.3 V. This is faster than the older ARM 710 but slower than the ARM 810, which was fabricated at about the same time. It represents a power-efficiency of 280 MIPS/W, which is as good as either of the clocked CPUs.

### BTB and cache power-save measurements

Two benchmark programs were used to measure the performance and power-efficiency benefits of the architectural features:

- Dhrystone 2.1 is widely used to evaluate embedded cores, but it has very unusual branching characteristics compared to any real programs used during the BTB's archi-

Table 1. Amulet2e results.

| Feature | Dhrystone 2.1 | | Sort | |
|---|---|---|---|---|
| | Performance (percentage change in MIPS) | Power efficiency (percentage change in MIPS/W) | Performance (percentage change in MIPS) | Power efficiency (percentage change in MIPS/W) |
| BTB on | +5 | +0 | +22 | +15 |
| BTB power–save | +5 | +3 | +25 | +22 |
| CAM bypass | +6 | +15 | +5 | +14 |
| All on | +11 | +18 | +32 | +35 |

tectural evaluation. (The BTB was not optimized for Dhrystone.)

- A set of sorting routines that combine insertion sort, shell-sort, and quick-sort algorithms. We ran these benchmarks on a randomized data set to generate alternative results. This program has very tight looping behavior.

Table 1 summarizes the results of these tests. The base case is the processor running from cache with the BTB and the CAM bypass disabled. Table 1 shows the improvement in performance and power-efficiency delivered by enabling each feature in turn. It also records the total effect when all features are enabled.

The cache CAM bypass mechanism delivers about a 5% performance improvement and 15% power-efficiency improvment on both benchmark programs. This is a useful contribution from a simple circuit addition.

The BTB gives a modest 5% performance improvement on Dhrystone and a dramatic 22% on the sort benchmark. Real programs should fall somewhere between these two. The BTB power-save feature makes little difference to the Dhrystone performance but improves its power-efficiency by 3%. In all cases where we observed the BTB to have a negative impact on power-efficiency, the power-save feature has cancelled the loss and converted it into a power-efficiency gain.

### Halt measurements

The Halt feature's effect is different in principle from the cache CAM bypass and the BTB power save, because it has no impact on power consumption when the processor is running. It only comes into play when the processor is idle.

Table 2. Amulet2e core idle currents.

| Idle condition | IDD (mA) |
|---|---|
| Halt disabled | 50 |
| Halt enabled; centisecond timer interrupts | 1 |
| Halt enabled; interrupts disabled | 0.001 |

To judge the halt's effectiveness, we can observe the current consumed by the processor core logic when idling with halt disabled. This current is around 50 mA if the idle loop is in the cache. With halt enabled, this current decreases to around 1 mA on the test card when centisecond timer interrupts are being handled. It becomes about 1 µA (leakage current) when interrupts are disabled. Table 2 summarizes these results.

## Amulet3i

Amulet3i is a processor subsystem based on the Amulet3 core. It was designed for system-on-chip applications, the first of which is Draco, shown in Figure 2. A telecommunications controller, Draco was fabricated in a 0.35-µm, three-metal-layer CMOS process. Amulet3i retains many of the power-saving features of Amulet2e; it also incorporates several additional features.

### Amulet3 BTB

The Amulet3 BTB is similar to that of Amulet2, though it has only 16 entries instead of 20. A new feature further reduces the system power consumption. In Amulet2, the branch instruction is fetched to determine its condition

Table 3. AMULET3i BTB test results.

| Feature | Dhrystone 2.1 | | Sort | |
|---|---|---|---|---|
| | Performance (percentage change in MIPS) | Power efficiency (percentage change in MIPS/W) | Performance (percentage change in MIPS) | Power efficiency (percentage change in MIPS/W) |
| BTB on | +5.8 | +3.7 | +21.5 | +13.9 |
| BTB power save | +6.0 | +6.3 | +21.5 | +19.7 |

code and whether to save a return address. In contrast, Amulet3 stores the condition and return information in the BTB RAM along with the target address, so the branch need not be fetched from memory.

Branches form about 15% of instructions, and the BTB captures many of these. This represents a saving of around 10% in instruction fetch cycles. The suppressed cycles are internal to the processor and faster than external accesses, a fact exploited by the processor's asynchronous pipeline.

The effect of the BTB and its power-save feature (which is similar to that in Amulet2e), are shown in Table 3. Although we used the same test programs as we used for Amulet2e, the results are not directly comparable because Amulet3i has no cache, and the programs were run from external memory. The power-save feature has minimal effect on overall performance but does improve power efficiency substantially.

### Counterflow pipeline color

Despite the BTB, unpredicted branches still occur, and it is then necessary to discard any speculatively fetched instructions. The default mechanism is to allow the pipeline to keep flowing up to the instruction commit point and invalidate instructions there. The commit point is in the execution stage. This is after the instruction decoder, which spawns many internal operations, including several register reads and commits to writing back results. So operations must continue until the instruction can be discarded cleanly. Abandoning prefetched instructions before they are decoded saves a lot of power, but requires information only available further down the pipeline.

All Amulet processors manage the problem of discarding unwanted instructions by *coloring*. Each section of code is prefetched with a known color that is matched to a color in the execution unit. When a branch is taken, this color is changed, and mismatching instructions are discarded until the new stream arrives. Because only one branch is outstanding at any time, two colors suffice and coloring requires only a single color bit. The problem of early discarding reduces to that of propagating the color bit, which originates in the execution unit, to the instruction decoder.

The decode and execution stages are adjacent in the pipeline and they synchronize with each other to communicate. Normally, this communication is one way, however it is possible to use the synchronization to pass information in both directions at once. In this way, the color can reach the instruction decoder and be used to prevent the further flow of unwanted instructions.

This is not an ideal solution. Counterflow data leapfrog one instruction as they cross the pipeline stage interface, so an instruction always leaks through. However, it does prevent another one or two instructions from entering the processor's power-hungry parts.

This mechanism becomes more significant when running 16-bit Thumb code. Thumb instructions are fetched as 32-bit pairs, which significantly increases the effective instruction prefetch depth. The penalty for erroneous prefetch is also increased, but the counterflow color mechanism removes much of the power overhead.

In testing this feature, we ran the Dhrystone benchmark in Thumb code from on-chip RAM. Enabling the counterflow color mechanism

improved performance by 3.9% and power efficiency by 8.6%. With ARM code, the figures were 0.5% and 0.6%.

### Amulet3 halt

A halting mechanism very similar to that used in Amulet2 is employed in Amulet3. A branch instruction that loops to itself is detected, and the processor stops. Experiments suggest that it is equally effective at reducing current drain to very low levels in Amulet3. This is difficult to show in practice, because of other circuits' current draw on Draco.

### Amulet3i on-chip memory

Amulet3i does not have a cache, although it does contain 8 Kbytes of memory-mapped RAM. This RAM employs some speed/power optimization techniques, but these are different from those in Amulet2e's cache.

The RAM is implemented as eight interleaved 1-Kbyte blocks, minimizing access conflicts from the separate instruction and data buses. Instead of exploiting sequential access, each block of RAM has a block buffer, essentially a single line of cache that holds the last-read RAM line (four words). This has its own tag that recognizes not only sequential accesses but also repeat accesses to this line. Furthermore, there are separate block buffers for instruction and data accesses, resulting in a 32-word instruction cache and a 32-word data cache. Hits avert power-hungry RAM accesses; they are also faster, a characteristic the asynchronous environment readily exploits.

Interestingly, the buffers grew out of another power-saving feature. To reduce the power requirement of a RAM read, a self-timed mechanism delays sense amplifiers' activation until after the amplifiers' (differential) inputs are expected to have diverged. Enforcing this delay ensures outputs will switch quickly. When the sense amplifiers switch, they are immediately deactivated with their outputs latched until the processor can consume the data. These latches, being already present, adapt into the block buffers at very little cost and provide considerable benefit. When running real code, they can comfortably intercept more than half of the instruction fetches and a considerable proportion of data operations.

**Table 4. Energy per operation against pipeline occupancy for latch operating modes.**

| No. of instructions in pipeline | Normally open, energy per operation (nJ) | Normally closed, energy per operation (nJ) | Decrease in energy per operation (percentage) |
|---|---|---|---|
| 1 | 2.47 | 1.95 | 21 |
| 2 | 2.06 | 1.96 | 5 |
| 3 | 2.0 | 1.96 | 2 |
| 4 | 2.0 | 1.96 | 2 |

### Dynamic Pipeline Power Management

As functional units in a data path evaluate their inputs, they output glitches, causing unwanted switching activity. The power wasted can become significant if the glitches are allowed to propagate to subsequent pipeline stages.

For maximum throughput, the latches in an asynchronous pipeline should operate in normally open mode, allowing data to pass down the pipeline as quickly as possible. Unfortunately, glitches will also propagate down the pipeline, wasting power. Using normally closed pipeline latches can block the transitions. This reduces wasted power, but increases the cycle time by the time taken to open the data latches.

Ideally, normally open or normally closed mode would be selected according to computational demand. Latch controllers have been designed that switch operating mode in response to a control signal.[6] This capability facilitates a variety of dynamic power management schemes selecting speed or economy mode as appropriate. Amulet3i provides software control for the operating mode.

To illustrate the operating mode's influence on asynchronous pipeline power consumption, we obtained the results shown in Table 4 for a $32 \times 32$-bit multiplier data path consisting of five pipeline stages. The first four stages generate the partial products and calculate the partial sum and partial carry; a final adder stage resolves the carries. We used Synopsys' EPIC Powermill to analyze the relative energy consumptions of the circuits.

Crucially, the size of the power saving is determined by pipeline occupancy. At maximum throughput, the pipeline is full and the energy per operation only decreases by 2% when running in normally closed mode. In contrast, when running with only a single input value at a time, the difference in energy per operation increases to 21% (and is even greater if the inputs are skewed).

## Future power management strategies

Work is still underway to improve the power efficiency of Amulet3 systems, now with contactless smartcard applications in mind. Here the available power is very limited and varies according to the card's distance from the base station. Dynamic power management techniques are therefore of great importance.

### Dynamic pipeline depth control

An unpipelined processor version consumes less power than a pipelined version, but pipelining delivers considerable performance improvement with modest power-efficiency losses.[7] Pipelining implies speculation as operations are issued before the previous instruction has committed. Speculative operations affect the power consumption in two ways: the overall processor activity at a given time is increased, so the power consumed is higher, and in some cases the speculation is wrong, so the energy spent in these operations is wasted.

The energy wasted through incorrect speculation clearly leads to a trade-off between the accuracy of predicting the program flow, the energy this prediction requires, and the energy that the operations need.

The increase in overall power consumption is an issue when the power budget available for the processor is restricted, as in a contactless smartcard. One approach is to slow the processor down, which will happen automatically if the supply voltage is reduced. However, below a certain voltage, CMOS circuits do not operate correctly and further power reductions may still be necessary. In asynchronous circuits this could be achieved by switching additional delays into control circuits. A better approach is to slow the processor indirectly by disabling speculation, for example by dynamically reducing the processor pipeline depth.

There are many ways to do this. One method is to have a circular token buffer with as many places as the processor has pipeline stages. The total number of tokens present in the token buffer regulates the effective pipeline depth of the processor. This can be controlled dynamically by power management hardware or by software.[8]

A less general approach is to join pipeline stages dynamically when specific instructions pass through them. For example, following most compare instructions there is a conditional instruction (usually a branch) that uses the comparison's results. The decode stage can stall the pipeline upstream until the comparison is finished, so that fewer instructions are fetched (potentially wrongly) from memory. With a lower branch penalty, branch prediction can be disabled, thus saving more energy.

A similar approach has been proposed for synchronous superscalar processors operating at a much coarser level.[9] In this approach, the number of uncommitted branches that are followed is restricted if their predictions are not trusted.

**SEVERAL POWER MANAGEMENT** techniques have been introduced as the Amulet processor series has developed. Some of these are straightforward circuit improvements; some are architectural design features that exploit the device's asynchronous nature. Others would be equally applicable in the synchronous world.

Some techniques allow trading off power efficiency against performance. A normally open pipeline is faster but less power-efficient than a normally closed pipeline. Our results show that the power savings are large when the pipeline is lightly loaded but small when the pipeline is busy. Other techniques improve both performance and power efficiency. It might have been expected that branch prediction would enhance performance but cost power. It turns out that the reduction in wasted instruction prefetches more than offsets the power cost of prediction (at least with our prediction scheme), so there is a win all round.

The Amulet2e cache sequential optimization and the Amulet3 RAM line buffers and color counterflow system likewise yield improvements in both performance and power-efficiency. Removing unnecessary activity often saves both power and time.

By far the most significant power saving technique employed on Amulet2e and Amulet3 is the ability to halt and restart all processor activity without other penalties. Energy can be conserved across very short idle periods without any power management overheads—the operating system merely executes a halt as its idle task. While the Amulet designs are not the fastest processors available, they are very good at doing nothing!

For further information on asynchronous technology and the Amulet microprocessors, visit the Amulet Web site at http://www.cs.man.ac.uk/amulet/. ■

## ■ References


1. T.R. Halfhill, "Transmeta Breaks x86 Low-Power Barrier," *Microprocessor Report*, Feb. 2000.
2. S.B. Furber, *ARM System-on-Chip Architecture*, Addison Wesley Longman, London, 2000. pp. 105-150.
3. J.D. Garside, S. Temple, and R. Mehra, "The AMULET2e Cache System," *Proc. Async 96*, Aizu-Wakamatsu, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 208–217.
4. R. Mehra, and J.D. Garside, "A Cache Line Fill Circuit for a Micropipelined Asynchronous Microprocessor," *TCCA Newsletter*, IEEE Computer Society Press, Los Alamitos, Calif., Oct. 1995.
5. S.B. Furber et al., "AMULET2e: An Asynchronous Embedded Controller," *Proc.* IEEE, IEEE Press, Piscataway, N.J., 1999, pp. 243–256.
6. M. Lewis, J.D. Garside, and L. Brackenbury, "Reconfigurable Latch Controllers for Low Power Asynchronous Circuits," *Proc. Async 99*, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 27–35
7. R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Processors," *Proc. Int'l Symp. on Low Power Electronics*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 12–13.
8. L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. Very Large-Scale Integration Systems*, vol. 8, no. 3, Jun. 2000, pp. 299–316.
9. S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction," *Proc. 25th Int'l Symp. Computer Architecture*, ACM, New York, 1998, pp. 132-141.



**Steve B. Furber** is the ICL chair of computer engineering at the University of Manchester. His research interests include microprocessor design (he was a principal designer of the ARM 32-bit RISC microprocessor) and asynchronous logic. Furber has a BA in mathematics and PhD in aerodynamics from the University of Cambridge, England. He is a Fellow of the Royal Academy of Engineering, a Fellow of the British Computer Society, a Chartered Engineer, and a member of the IEEE.



**Aristides Efthymiou** is a PhD student at the University of Manchester. His research interests are in the area of low-power circuit design and processor architecture. He received a BSc and MSc degree in computer science from the University of Crete, Greece.



**Jim D. Garside** has worked on hardware systems using Inmos Transputers for investigation into parallel computer architectures and as a programmer on air traffic control systems. He is now a lecturer in the Department of Computer Science at the University of Manchester. His research interests include asynchronous logic systems, especially in the design of the AMULET series of asynchronous microprocessors. Garside has a BSc in physics from the University of Manchester and a PhD in computer science from the same institution.

**David W. Lloyd** is the Amulet Research Fellow at the University of Manchester, where he has worked on the Amulet3 core. His research interests range from architectural design for asynchronous systems to high-performance dynamic circuits. Lloyd has a BSc in physical electronics from Newcastle-upon-Tyne Polytechnic and a PhD in electrical and electronic engineering from the University of Nottingham.

**Mike J.G. Lewis** is a researcher with the Amulet Group, and has recently received his PhD on the application of asynchronous techniques to low-power digital signal processing. He has a MEng degree in electronic engineering from the University of Cambridge. He is a student member of the IEEE.

**Steve Temple** is a Research Fellow in the Department of Computer Science at the University of Manchester. His research interests include microprocessor system design and asynchronous logic. He has a BA in computer science from the University of Cambridge, UK, and a PhD for research into local area networks from the University of Cambridge Computer Laboratory.

■ Direct questions and comments about this article to Steve Furber at sfurber@cs.man.ac.uk.