Asynchronous Logic

S B Furber (*sfurber@cs.man.ac.uk*),

ICL Professor of Computer Engineering, Department of Computer Science,

The University of Manchester, Oxford Road, Manchester M13 9PL

Abstract

Asynchronous logic is enjoying a resurgence of interest among academic and industrial researchers after two decades of near total neglect. Why is this?

This tutorial presents the reasons for the renewed interest and discusses the current state of development of asynchronous design. There are many ways to design chips without clocks, so some background is offered to the various different asynchronous design methodologies used by various groups around the world today. One such style, called 'micropipelines' was developed by Ivan Sutherland and formed the basis of his 1988 Turing Award Lecture. This approach has been adopted by a group at Manchester University in England and used to develop fully asynchronous implementations of the ARM 32-bit RISC microprocessor. The organization and key implementation details of these AMULET processors will be described. Finally, speculation will be offered on the future of asynchronous design techniques in a world currently dominated by clocked circuits.

1. Motivation

The principle motivation for re-examining asynchronous design is its potential for power-efficiency. The clock in a synchronous circuit runs all the time, causing transitions in the circuit that dissipate electrical power. The clock frequency must be set so that the processor can cope with the peak work-load, and although the clock rate can be adjusted under software control to suit varying demands, this can only be done relatively crudely at a coarse granularity. Therefore most of the time the clock is running faster than is necessary to support the current work-load, resulting in wasted power. An asynchronous design, on the other hand, only causes transitions in the circuit in response to a request to carry out useful work. It can switch instantaneously between zero power dissipation and maximum performance upon demand.

In addition, asynchronous design avoids the problem of clock skew, which is becoming increasingly hard to control in clocked circuits. Propagation delays in clock interconnect wires limit the achievable global synchrony, and at high clock speeds very powerful drivers are required to achieve the necessary edge speeds. As a result, clock drivers are occupying an increasing share of the die area and power budget, compromising the cost-effectiveness and power-efficiency of the design. Asynchronous design replaces the central clock with multiple locally-timed control signals which can be built with much lower area and power costs.

The final benefit is the inherent modularity of self-timed designs. Data encapsulation, modularity and component reuse are of increasing interest in computer software, where such characteristics are necessary to offset the costs of the increasingly sophisticated (and therefore complex) products demanded by the marketplace. In the software domain this has led to the development of novel programming paradigms such as 'object-oriented' languages. In the hardware domain, asynchronous design offers similar benefits which will be increasingly important as chip complexities rise to exploit the billion transistor VLSI technologies which will be available at the end of the millenium.

In summary, therefore, asynchronous technology is attracting renewed interest because of its potential for power-efficiency, the removal of the clock-skew problem and because of its inherent modularity.

2. Asynchronous Logic Styles

In a synchronous (clocked) chip the clock controls all the state changes and communication within the chip. Between active clock edges combinatorial logic generates the next state function, possibly producing many spurious output values (glitches) on the way to the correct value, but so long as the outputs are correct and stable at the next active clock edge the chip will operate correctly. The timing constraints which the design must satisfy may be summarized as follows:

- All logic functions must be correct and stable before the next active clock edge;
- The clock must be clean and the clock period must be longer than the longest logic delay.

In addition, clock skew and slow clock edges can give rise to race conditions which can be avoided by careful latch design and controlling the clock skew and edge speeds carefully.

Asynchronous design styles operate without a clock, so how is data communication controlled and timing managed?

Asynchronous timing

There are two basic ways used in asynchronous logic to determine when the outputs from a combinatorial logic block are valid:

- *Self-timed* logic uses a redundant logic representation where some output values represent valid logic values and others represent invalid values. The output must pass through an invalid value when changing from one valid value to another valid value. The simplest redundant form uses dual-rail encoding, where each boolean uses two wires. ‘00’ is the invalid value, ‘01’ represents ‘false’, ‘10’ represents ‘true’ and ‘11’ is unused.
- *Delay matching* logic employs conventional logic representation but includes an additional control wire which indicates when the output is valid. Usually a transition on the control wire is passed through logic which is known to have a delay no shorter than that through the combinatorial logic function.

Self-timed logic can be designed to be insensitive to delay variations in any of the logic gates or wires in the sense that only the correct valid output will appear, although slow gates or wires will of course cause it to be delayed. This approach is therefore favoured by purists, since the complete decoupling of functionality from performance makes the design insensitive to the characteristics of the implementation technology and very amenable to mathematical proofs of functional correctness. However, the redundant logic structures incur costs which are about twice those of standard single-rail logic (measured, for example, in chip area). Delay matching does not result in delay-insensitivity and requires careful engineering such as detailed SPICE modelling, and therefore is more technology dependent. However the area overhead compared to clocked logic is minimal.

In the short term, therefore, delay matching is favoured by most designers of complex asynchronous circuits. In the longer term, deep submicron technologies may make the control of on-chip delays much trickier and they will also increase the available gate resource, so redundant encoding may become relatively attractive.

Signalling protocols

Conventional logic design typically represents logic values by signal levels; a wire at ground potential represents a logic ‘0’ and at supply potential a logic ‘1’. Many asynchronous styles are similar, but it should be recognised that this is not the only possibility. For instance, in a dual-rail encoded system, a ‘0’ could be represented by a transition on one wire and a ‘1’ by a transition on the other. Since a transition is an event, there is now no need to pass through an invalid code between consecutive values. It is possible, though uncommon, to build complete logic systems on transitions. However, transition signalling is used on the control wires of some of the examples below. It is useful therefore to define two signalling protocols:

- *Level* signalling; here an event is indicated by a level (often a logic ‘1’) and before the next event the wire must return to zero.
- *Transition* signalling; here an event is indicated by a change in logic level (either ‘0’ to ‘1’ or ‘1’ to ‘0’) and the next event is simply the opposite transition, so no recovery phase is required.

Self-timed communication

When data is sent from one place to another, not only must the receiver recognize when the data is valid, but also the sender must recognize when the data has been received before it sends the next data.

The standard approach is to use a *Request-Acknowledge* handshake to control the flow of data. The sequence of actions comprising the communication of data from the Sender to the Receiver is as follows, assuming the matched-delay model (in a self-timed system *Request* is encoded implicitly in the data):

1. The Sender places a valid data value onto a bus.
2. The Sender then issues a *Request* event.
3. The Receiver accepts the data when it is ready to do so.
4. The Receiver issues an *Acknowledge* event to the Sender.
5. The Sender may then remove the data from the bus and begin the next communication when it is ready to do so.

The *Request* and *Acknowledge* events may use transition or level signalling as described above, giving the two communication protocols which are illustrated in Figure 1 and Figure 2. Transition signalling is conceptually cleaner since every transition has a role and its timing is therefore determined by the circuit’s function. It also uses the minimum number of transitions, and should therefore be power-efficient. However, the CMOS circuits used to implement transition control are relatively slow and inefficient, so level signalling is often used employing circuits which are faster and in practice more power-efficient despite using twice the number of transitions. Level signalling leaves somewhat arbitrary decisions to be taken about the timing of the recovery (return-to-zero) phases in the protocol.

The transition signalling protocol is sometimes called the *two-phase* protocol since each com-

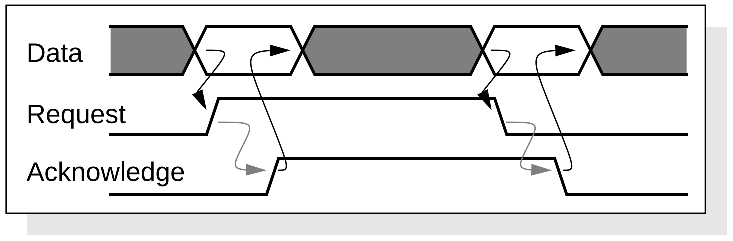

Figure 1 Transition-signalling communication protocol.

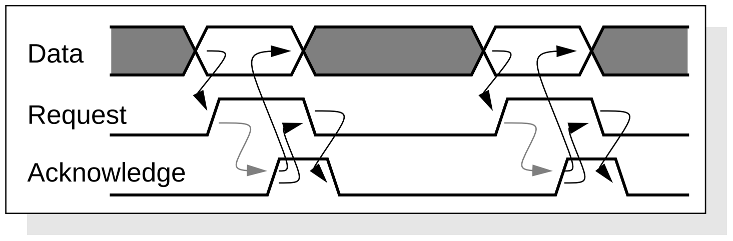

Figure 2 Level-signalling communication protocol.

munication has two phases:

- Firstly the Sender is active preparing the data; this phase is terminated by the *Request* event.
- In the second phase the Receiver is active accepting the data; this phase is terminated by the *Acknowledge* event.

In the level-signalling protocol there are two further phases terminated by the return to zero transitions, so this protocol is sometimes called the *four-phase* signalling protocol.

Self-timed pipelines

An asynchronous pipelined processing unit can be constructed using self-timing techniques to allow for the processing delay in each stage and one of the above protocols to send the result to the next stage.

When the circuit is correctly designed, variable processing delays and arbitrary external delays can be accommodated; all that matters is the local sequencing of events (though long delays will, of course, lead to low performance).

Unlike a clocked pipeline, where the whole pipeline must always be clocked at a rate determined by the slowest stage under worst-case environmental (voltage and temperature) and data conditions, an asynchronous pipeline will operate at a variable rate determined by current conditions. It is possible to allow rare worst-case conditions to cause a processing unit to take a little longer. There will be some performance loss when these conditions do arise, but so long as they are rare enough the impact on overall performance will be small.

Micropipelines

In his 1988 Turing Award Lecture Ivan Sutherland presented an asynchronous design methodology based on matched delay transition signalling pipelines which he called ‘Micropipelines’. Most of what follows was inspired by Sutherland’s work.

3. The AMULET microprocessors

Two asynchronous microprocessors have been developed in the Department of Computer Science at the University of Manchester in the UK using techniques based on Sutherland's micropipelines. The first of these, AMULET1, was designed between 1991 and 1993 and used transition signalling. The second, AMULET2, was designed between 1993 and 1995 and used level signalling.

Both AMULET processor cores have the same high-level organization as illustrated in Figure 3. The design is based upon a set of interacting asynchronous pipelines, all operating in their own time at their own speed. These pipelines might appear to introduce unacceptably long latencies into the processor but, unlike a synchronous pipeline, an asynchronous pipeline can have a very low latency.

The operation of the processor begins with the address interface issuing instruction fetch requests to the memory. The address interface has an autonomous address incrementer (the AMULET2 address interface also incorporates a *Jump Trace Buffer* which attempts to predict branches from past behaviour) which enables it to prefetch instructions as far ahead as the capacities of the various pipeline buffers allow. Fetched instructions then flow through the instruction decoder, access their operands from the register bank and execute. Memory data accesses are interleaved with instruction fetches through the memory pipeline and loaded data is returned directly to the register bank.

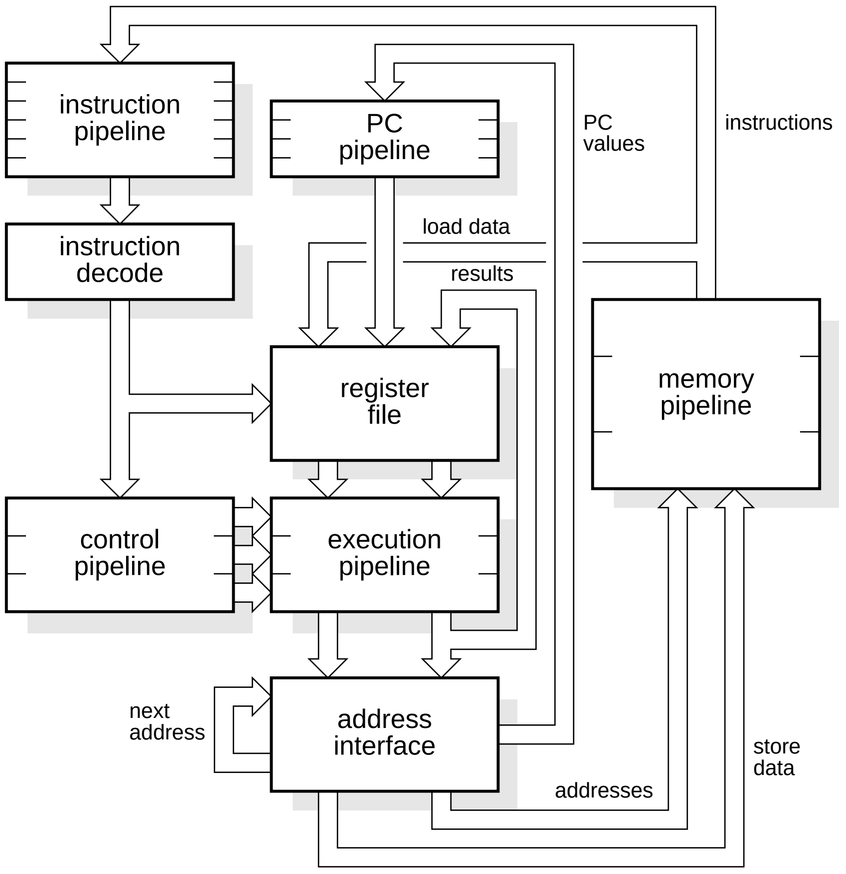

Figure 3 AMULET internal organization.

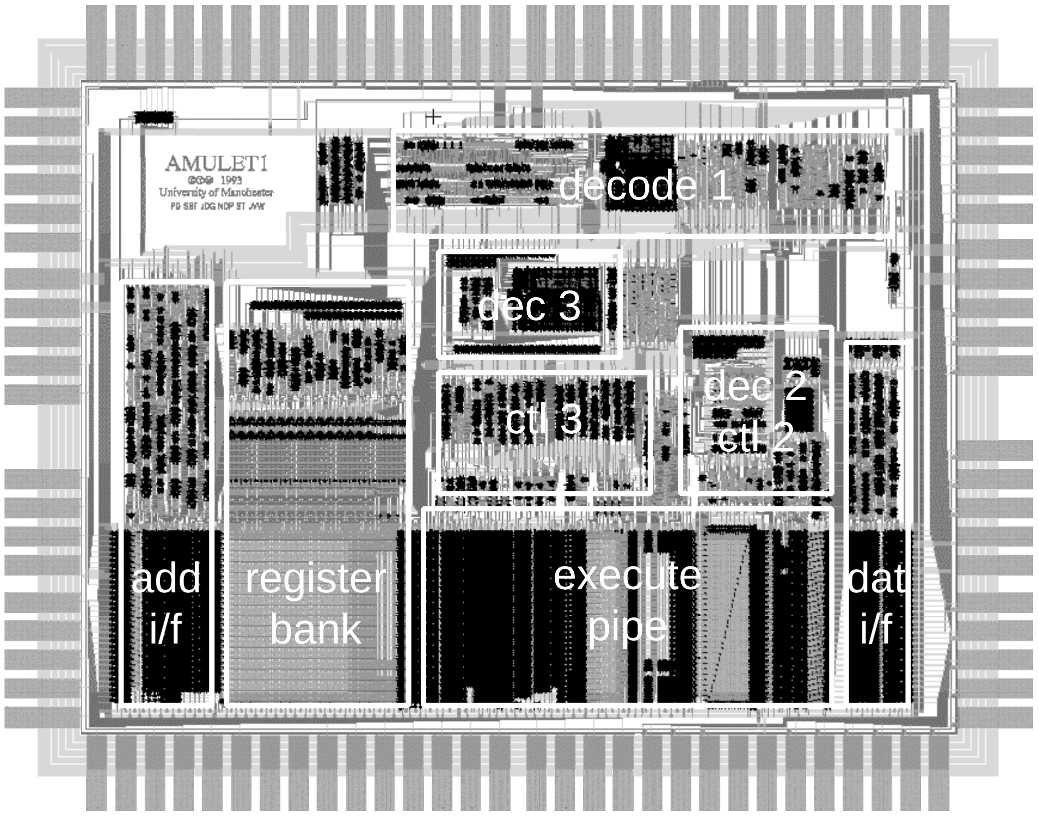

Figure 4 AMULET1 chip layout.

AMULET1 silicon

AMULET1 was developed to demonstrate the feasibility of designing a fully asynchronous implementation of a commercial microprocessor architecture. The prototype chips were functional and ran test programs generated using standard ARM development tools. The layout of the AMULET1 core is shown in Figure 4.

AMULET2e

AMULET2e is an AMULET2 processor core combined with 4 Kbytes of memory, which can be configured either as a cache or a fixed RAM area, and a flexible memory interface (the *funnel*) which allows 8-, 16- or 32-bit external devices to be connected directly, including memories built from DRAM. The internal organization of AMULET2e is illustrated in Figure 5.

AMULET2e cache

The cache comprises four 1 Kbyte blocks, each of which is a fully associative random replacement store with a quad-word line and block size. A pipeline register between the CAM and the RAM sections allows a following access to begin its CAM lookup while the previous access completes within the RAM; this exploits the ability of the AMULET2 core to issue multiple memory requests before the data is returned from the first. Sequential accesses are detected and bypass the CAM lookup, thereby saving power and improving performance.

Cache line fetches are non-blocking, accessing the addressed item first and then allowing the

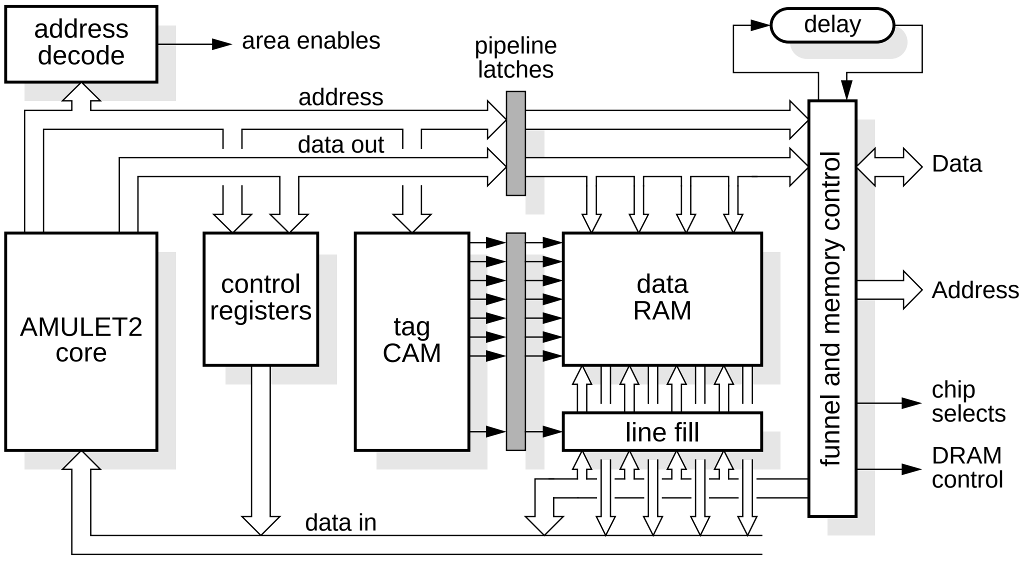

Figure 5 AMULET2e internal organization.

processor to continue while the rest of the line is fetched. The line fetch automaton continues loading the line fetch buffer while the processor accesses the cache. There is an additional CAM entry that identifies references to the data which is stored in the line fetch buffer. Indeed, this data remains in the line fetch buffer where it can be accessed on equal terms to data in the cache until the next cache miss, whereupon the whole buffer is copied into the cache while the new data is loaded from external memory into the line fetch buffer.

AMULET2e systems

AMULET2e has been configured to make building small systems as straightforward as possible. As an example, Figure 6 shows the organization of an evaluation card incorporating AMULET2e. The only components, apart from AMULET2e itself, are four SRAM chips (though the system could equally well have been designed to operate with just one, at lower performance), one ROM chip, a UART and an RS232 line interface. The UART uses a crystal oscillator to control its bit rate and to provide a real-time clock, but all the system timing functions are controlled by AMULET2e using the single reference delay.

This system demonstrates that using an asynchronous processor need be no more difficult than using a conventional clocked processor provided that the memory interface has been carefully thought out.

4. An Asynchronous Future?

The AMULET chips are presently research prototypes and are not about to replace synchronous ARM cores in commercial production. However, there is a resurgence of world-wide interest in the potential of asynchronous design styles to save power and to offer a more modular approach to the design of computing hardware.

The power savings which result from removing the global clock, leaving each subsystem to perform its own timing functions as and when it has useful work to perform, are clear in theory but there are few demonstrations that the benefits can be realized in practice with circuits of

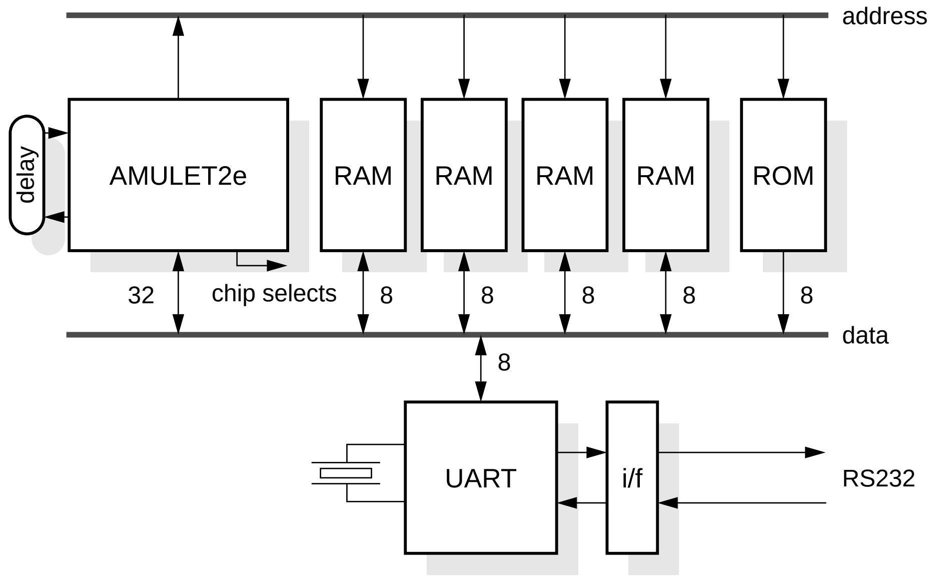

Figure 6 AMULET2e PIE card organization.

sufficient complexity to be commercially interesting. The AMULET research is aimed directly at adding to the body of convincing demonstrations of the merits of asynchronous technology.

An obstacle to the widespread adoption of self-timed design styles is the knowledge-base of the existing design community. Most IC designers have been trained to have a strong aversion to asynchronous circuits because of the difficulties that were experienced by the designers of some early asynchronous computers. These difficulties resulted from an undisciplined approach to self-timed design, and modern developments offer asynchronous design frameworks which overcome most of the problems inherent in what is, admittedly, a more anarchic approach to logic design than that offered within the clocked framework.

The next few years will tell whether or not AMULET and similar developments around the world can demonstrate the sort of advantages that will cause designers to throw away most of their past education and learn a new way to perform their duties.

5. Bibliography

Asynchronous logic

- Birtwistle and Davis (editors), *Asynchronous Circuit Design*, Proceedings of the 1993 VIIth Banff High Order Workshop, Springer, 1995. ISBN 3-540-19901-2, 0-387-19901-2.

  Chapter 5, *Computing without Clocks: Micropipelining the ARM Processor*, pages 211-262, by S. B. Furber, describes the AMULET1 organization in some detail. Other chapters in the book give background on asynchronous logic design and describe alternative approaches to the 'micropipelines' used in the AMULET designs.

Micropipelines

- Sutherland, I. E., *Micropipelines*, Communications of the ACM, vol. 32, no. 6, June 1989, pp. 720-738.