AMULET3: a 100 MIPS Asynchronous Embedded Processor

S. B. Furber, D. A. Edwards and J. D. Garside,
Department of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UK.
sfurber, dedwards, jgarside@cs.man.ac.uk

Abstract

AMULET3 is a 32-bit asynchronous processor core that is fully instruction set compatible with the clocked ARM cores. It represents the culmination of ten years of research and development into asynchronous processor design at the University of Manchester, and is the first step into commercial use for this technology.

AMULET3 shows that asynchronous technology is commercially viable, and is competitive in terms of performance, area and power-efficiency with clocked design. In addition, asynchronous design offers significant advantages in terms of reduced electromagnetic interference and unique power management capabilities.

1. Introduction

The AMULET group in the Department of Computer Science at the University of Manchester, U.K., has spent a decade researching the commercial potential of asynchronous design techniques. The focus of this work has been the design of asynchronous implementations of the ARM 32-bit RISC architecture [1, 2].

AMULET1 silicon was delivered in 1994. This basic processor core showed that asynchronous design was feasible, though its performance and power-efficiency were not as good as the contemporary ARM6 [3-5].

AMULET2e silicon was delivered in 1996. This chip comprised an improved processor core together with a cache memory and a flexible external memory interface. This part was competitive with the ARM7 cores, and in addition it showed that asynchronous operation offers advantages in terms of reduced electromagnetic interference and very simple power management [6]. Although it is in use in a number of applications, such as robotics and radio networking, AMULET2e is a research prototype and it is not suitable for commercial production due to the difficulty of testing certain on-chip circuits.

AMULET3 silicon will be delivered in 2000 as part of the commercial DRACO chip (described in section 2). AMULET3 is competitive with the ARM9TDMI core and retains the advantages of asynchronous operation demonstrated by AMULET2e. It shows that asynchronous processor cores are commercially viable and are ready for use in market niches where their advantages can be exploited effectively.

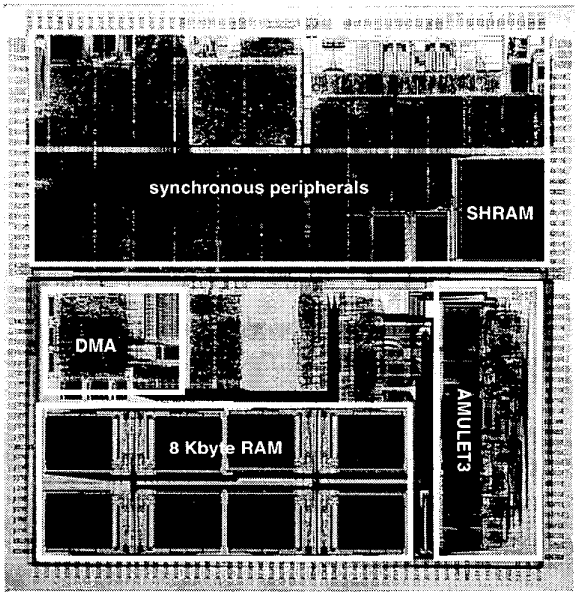

Figure 1. The DRACO chip

2. DRACO

The DRACO (DECT Radio Communications Controller) chip is a telecommunications controller intended for ISDN (*Integrated Services Digital Network*) DECT (*Digital European Cordless Telephone*) base station applications. The chip area is divided equally between the AMULET3H asynchronous processing subsystem (described in section 3) and a synchronous telecommunications peripheral subsystem. The layout of the chip is shown in figure 1.

0-7695-0801-4/00 $10.00 © 2000 IEEE

The synchronous telecommunications subsystem includes a large number of functions in the synchronous peripheral subsystem. ISDN is supported by an ISDN controller with a 16 Kbit/s HDLC controller and transformerless analogue ISDN interfaces. The DECT radio interface includes a baseband controller, an analogue interface to the DECT radio subsystem and a DECT encryption engine.

There is a four-channel full-duplex ADPCM/PCM conversion signal processor and a telecommunications codec with an analogue front-end for speech input and output which could alternatively be used for an analogue telecommunications port. 8 Kbytes of shared RAM are used for buffer space by the DECT controller. (This is in addition to the 8 Kbytes of dual-port RAM in the AMULET3H subsystem.)

General-purpose peripherals include two high-speed UARTs with an IrDA interface that can be used by either UART, an interrupt controller, counter-timers, a watchdog timer, and 65 flexible I/O ports including an 8-bit parallel port with handshake capability. Other interfaces include a 2 Mbit/s IOM2 highway controller with programmable switching functionality, an I^2C interface, an analogue-to-digital converter (ADC) interface, and two general-purpose pulse-width modulation controllers.

An on-chip clock oscillator produces 38.864 MHz, and on-chip phase-locked loops produce the 12.288 MHz master clock required by the ISDN interface and the 13.824 MHz master clock required by the DECT controller. An ISDN-DECT synchronizer phase-locked loop avoids bit loss in data transfers between the ISDN and DECT clock domains. The clocked system has power-down features.

Overall the DRACO chip is a state-of-the-art telecommunications SoC (system-on-chip) with one very unusual feature: its processing subsystem operates fully asynchronously.

3. The AMULET3H SoC subsystem

DRACO's asynchronous processing subsystem is based around the MARBLE self-timed on-chip bus [7]. This bus is a full functionality multi-master on-chip interconnect with a central arbiter and address decoder. It supports split transactions and, in its current manifestation, can perform up to 83 million 32-bit data transfers per second [8].

The processor core is a 120 MIPS AMULET3 32-bit core with on-chip debug hardware support (described further in section 4). 8 Kbytes of dual-port high-speed RAM are connected to the processor's instruction and data memory interfaces and both memory ports then connect to the MARBLE bus (see figure 2). The bus also hosts a 32-channel DMA controller (synthesized using Balsa - see section 5.2), 16 Kbytes of ROM and an external memory interface. The programmable external memory interface supports the

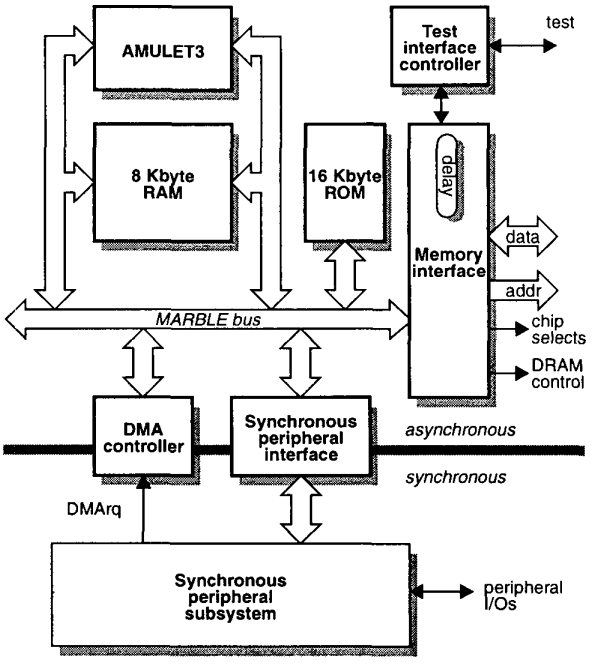

Figure 2. AMULET3H organization

direct connection of SRAM, DRAM and flash memory. An on-chip reference delay line calibrated by software is used to control off-chip memory access timings [9].

A bridge connects MARBLE to the synchronous on-chip bus that supports the synchronous peripherals. The bridge is the sole master of the synchronous bus, which therefore has a very straightforward structure.

A particular feature of the asynchronous subsystem is an asynchronous event driven load module that holds the processor in its zero-power wait mode while an analogue-to-digital conversion completes, thereby synchronizing the software to external data rates.

4. The AMULET3 core

The AMULET3 processor core supports ARM architecture version 4T [2], including the Thumb 16-bit compressed instruction set [10]. The organization of the processor – in coarse outline – is shown in figure 3.

As implemented the pipeline is somewhat deeper than implied in the figure – for example the instruction decoder has two stages – but at this level an asynchronous processing pipeline appears very like its 'conventional' synchronous counterpart. The only difference is that the timing is localized such that only units that need to communicate are ever in step with each other, and then only for the duration of the communication.

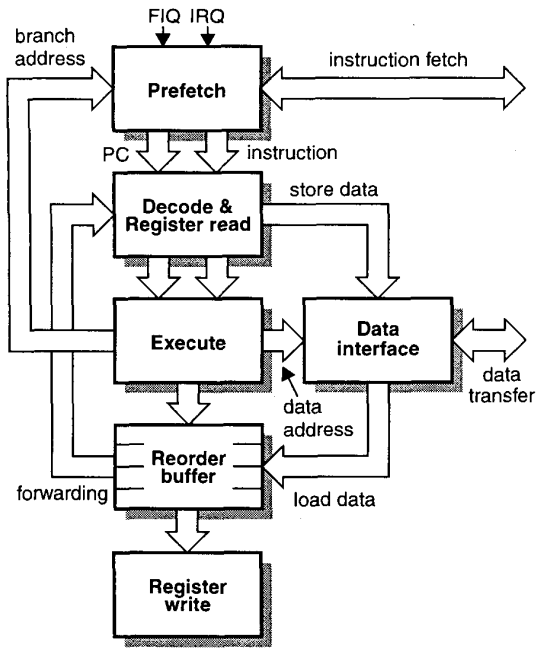

Figure 3. AMULET3 organization

This gives considerable freedom in design. For example the prefetch unit fetches instructions 32 bits at a time (representing one ARM instruction or a pair of Thumb instructions). The decoder is then responsible for decoding and issuing these packets. In the normal course of executing ARM code each packet entering the decoder will result in another being generated, some time later. However when running Thumb code two instructions will be issued in sequence. This slows down the consumption of instruction packets and may stall the prefetch unit, but there is no need for global control.

The ability to expand instructions locally is exploited in the few multi-cycle instructions (notably multi-register PUSH/POP) in the ARM/Thumb instruction sets. The converse is also possible: all ARM instructions may be conditional and an instruction which is not going to produce a result may be removed without it travelling the length of the pipeline.

The disadvantage – from an architect's viewpoint – in running pipeline stages asynchronously is that many processor design 'tricks' rely on global synchronisation. A notable example is result forwarding where the instruction decoder can track the progress of previously issued instructions and pirate copies of their results before they are complete. This requires knowledge of where results are at certain times, knowledge that is unavailable to an asynchronous designer. Instead the architect must derive other techniques; three examples are given below.

4.1. PC tracking

A feature of the ARM is that the programme counter (PC) is available as a general purpose register, despite 'logically' residing in the prefetch unit. Synchronous ARMs resolve this by reading the PC and using timing information to know how many times it has been incremented since the instruction was fetched.

In AMULET3 register reading has no synchronisation with the prefetch unit; instead the PC value is sent down the pipeline in parallel with the instruction fetch. Although this widens the pipeline considerably the actual cost is low both in area (the few latches are grouped adjacently in the datapath) and in power (on average very few bits in the PC alter from one instruction to the next).

4.2. The reorder buffer

Another example of architectural innovation in AMULET3 is the implementation of register forwarding where, as noted above, conventional techniques based on the use of global information available in a synchronous implementation are inapplicable.

AMULET3 issues instructions in order but the stream then splits into two:

- internal operations are retired immediately into a reorder buffer where their results are available for forwarding;
- load and store instructions are sidelined to the data interface where the data memory access is completed, any loaded data being returned to the reorder buffer out of order with the internal results.

The reorder buffer thus receives inputs in an arbitrary order at random, possibly overlapping, times. However some vital information is retained:

- Each result carries with it the identity of its own location in the reorder buffer. Locations are assigned by the decoder which therefore knows what will arrive where (but not when).
- Every outstanding result will arrive eventually.
- The incoming streams can never contend for the same reorder buffer location.

All results are copied back from the reorder buffer to the register file in their order of issue by a simple process that waits for each result in turn, copies it back and moves to the next. Memory faults are identified during this process, giving an exact exception model and allowing the abandonment of unwanted, speculative results [11].

A reorder buffer is not very useful without the ability to forward values to subsequent operations. This is possible in the asynchronous domain because the decoder has assigned the results to the reorder buffer locations and it knows

which registers are awaiting new results. When decoding an instruction the decoder can therefore check whether or not a register operand was assigned a reorder buffer location in the preceding few instructions; if so the appropriate result can be intercepted at the reorder buffer.

The key insight is that the result may be garnered as soon as it arrives (it may already be present). Waiting until an event occurs is straightforward in the asynchronous domain when it is guaranteed that the event has happened, is happening, or will happen.

Of course the forwarding is independent of the register copy back process so the result may already have reached the register bank too; however, transferring the result from the reorder buffer to the register file is a *copy* back process, not a move process, so the value is still present in the reorder buffer and will remain until it is overwritten. Only the decoder can reassign that location and it will only do so when any expected values have been read safely. Thus the forwarding process, both internal and data memory result arrival and the register write process can all proceed safely with minimal synchronization.

4.3. Power management

AMULET3 supports power management through the inclusion of an interruptible 'Halt' instruction. Halting causes a stall in the asynchronous handshake network to propagate rapidly throughout the asynchronous subsystem, bringing the power consumption down to zero. A real-time operating system can simply place a Halt instruction in its idle task to obtain optimal results – the processor will use no power when there is no work to do. There is no clock to scale, no crystal oscillator or PLL to turn off, and no hard decisions as to when each of these steps should be taken.

4.4. AMULET3 performance

The AMULET3 core uses 113,000 transistors and occupies 3 mm^2 on a 3.3 V 0.35 µm process. The core is capable of operating at up to 120 MIPS (Dhrystone 2.1) and at maximum throughput consumes about 155 mW, giving a power efficiency of 780 MIPS/W.

The AMULET3H system core uses 825,000 transistors and occupies 21 mm^2. The memory system reduces the maximum throughput to 100 MIPS at 215 mW; the overall system power-efficiency is 465 MIPS/W.

The processor core performance, area and power-efficiency are very similar to the clocked ARM9TDMI core [12], showing that asynchronous processors need not incur any penalty in these areas. In addition, asynchronous operation offers significant benefits in terms of reduced electromagnetic interference and simplified power management.

The down-side, as discussed in the next section, is the poor tools support for asynchronous design offered by CAD vendors. Some of these short-comings have been addressed by academic tool development, but the asynchronous design flow is far from being as well-integrated as that for conventional clocked systems.

5. Design flow

Support for asynchronous design within a conventional CAD design flow is limited. The Balsa synthesis tool [13] was used to develop the 32-channel DMA controller but, apart from the use of Petrify [14] for the synthesis of small asynchronous control circuits, the rest of the AMULET3H subsystem was designed manually.

The processor's datapath uses full custom hand layout with control entered as schematics and then laid out using automatic place and route tools. The standard cell library was supplied by ARM Limited, significantly augmented with specialized cells such as C gates and arbiters as required for efficient asynchronous design.

5.1. Controller synthesis

AMULET3 incorporates many independent small control circuits, ranging from the pipeline latch controllers that manage data propagation from one stage to the next using Request-Acknowledge handshaking [15] to the specialised and complex asynchronous token-passing controller that handles synchronization for forwarding and sequencing for copy back within the reorder buffer.

Extensive use was made of Petrify [14] in the design and verification of these circuits. Petrify takes a circuit specification in the form of a Petri Net and converts it into a speed-independent circuit.

5.2. High-level synthesis

The AMULET designs have grown significantly in complexity since AMULET1. Balsa has been developed in response to the evident need to handle this increased complexity by moving away from 'hand' design to a more automated approach, at least for the non speed-critical parts of the design. Balsa is a hardware description language that can be used to specify descriptions in a conventional style of programming language. These descriptions are then compiled into asynchronous networks of macromodular components which may be implemented in standard cell logic or on FPGAs. In this sense, Balsa is more like existing silicon compilers than synthesis with HDLs such as VHDL or Verilog. Unlike silicon compilers, research into which tends to be focused on automated resource allocation and program manipulation, Balsa makes use of syntax-directed compilation to produce circuits that reflect the structure and resource management decisions of the designer's original description. This 'transparent' compilation approach

allows the designer to make changes to a circuit's description with the goal of improving some desired property such as speed, area, power etc., whilst being able to predict the effect that the change will have on the structure of the final circuit.

The synthesis mechanism, handshake component set and the use of transparent compilation are modelled on the Philips Tangram system [16]. Balsa differs from Tangram in a number of respects:

- The channel semantics have been extended to include the enclosure of commands within a channel-receive command allowing the synthesis of Micropipeline [17], data-driven push-like structures, in addition to the normal pull-style control-driven circuits normally produced.
- The handshake component library has been extended to include components with a greater degree of parameterization in order to take advantage of conventional forms of automated circuit optimization wherever possible.
- The expressiveness of the language in enhanced by a number of features such as the support for recursive process descriptions.

Transparent compilation and high-level optimization require a fast loop of edit - synthesize - simulate - edit in order to be an effective technique for optimizing designs. Balsa's compilation mechanism is very fast, making it an excellent language for rapid prototyping. The asynchronous modelling language LARD [18] is used as a simulation environment allowing designs to be simulated before they are committed to a gate level implementation.

The DMA controller is an ideal vehicle for the application of the Balsa system. The controller is not speed-critical, all accesses being across, and therefore limited by, the MARBLE bus. The specification of the controller was initially ill-defined and the controller straddles the boundary of the synchronous and asynchronous domains on DRACO. Furthermore its behaviour is algorithmically quite complex and ideally suited to be described in a high-level HDL. The controller supports:

- 16 request clients and 32 channels;
- 3 channel types with a complicated register structure;
- programmable client to channel mapping;
- chaining of channel requests;
- interfaces both to synchronous devices and to memory.

Balsa allows for the inclusion of 'foreign', non-Balsa parts as long as they are provided with a handshake wrapper. The DMA controller is a mixture of Balsa synthesized circuits and custom layout (for the register banks). Initial simulation took place entirely within the Balsa system. Major changes were made in the specification of the controller during the development of the design without adversely impacting the overall design time (the Balsa description was developed in parallel with the custom layout of the register banks). Transistor-level simulation revealed that the design was not able to saturate the MARBLE bus. Only two days were required to make architectural changes to the design in order to increase its speed by introducing more concurrency, and to produce a new fully simulated layout. The Balsa description of the controller comprises 900 lines of code and contributes to about 50% of the area of the controller which in total occupies an area of 2.1 mm^2 and consists of 70K transistors.

Balsa was not used in the design of the AMULET3 processor core itself for performance reasons, however current research is aimed at improving the efficiency and safety of Balsa's datapath compilation. Techniques for removing superfluous datapath-related control overhead generated by the current compilation mechanism are being investigated. New approaches for delay-insensitive back-ends are also being investigated with a view to alleviating the increasingly difficult problem of timing closure.

5.3. Timing verification

Timing verification on the AMULET3H subsystem was carried out using TimeMill on an extracted transistor netlist. Additional tools written in PERL were used to trawl the TimeMill trace file to find any set-up and hold timing violations, thereby ensuring that the asynchronous control was operating with adequate margins. The coverage obtained by this method is only as good as the test program run under simulation, and this is not as rigorous as the static timing analysis used to verify the timing of clocked designs. While there is no fundamental reason why static timing analysis techniques could not be applied to asynchronous designs, there is no commercial support for such use at present.

5.4. Production test strategy

The DRACO chip has an external memory interface that can become a MARBLE bus master for production test purposes. The general test strategy is to use the tester to load a program into on-chip RAM and then to execute the program on the AMULET3 core. The on-chip RAM and processor core are viewed as a general-purpose programmable BIST engine. Once loaded the test program can be run at full processor speed without tester intervention, so the tester merely has to wait for the worst-case test time and it can then access a signature placed by the test program in a particular memory location to see whether the test was passed.

In order to ensure the effectiveness of this test strategy a number of features of the system had to be designed to support it. For example, the branch target buffer in the proces-

sor core is very hard to test through functional operation, so it has a special test access port from the MARBLE bus. The processor core can test its own branch prediction hardware through this port (though branch prediction must be disabled at the time).

6. Conclusions

AMULET3 shows that asynchronous processor cores are commercially viable, and potentially advantageous when low electromagnetic emissions and/or very flexible power management are required.

Together with the MARBLE bus, fully asynchronous system-on-chip design is possible and mixed clocked/asynchronous systems are practical. DRACO is a commercial example of a mixed timing device.

Asynchronous design is not well supported by commercial CAD tools, though it is possible to implement asynchronous designs with existing tools. Once asynchronous technology gains a significant foothold in commercial design it is likely that the CAD companies will develop suitable tools, and academic tools such as Balsa show that there are no serious barriers to the development of effective design automation for asynchronous systems.

7. Acknowledgments

The development of AMULET3 has been support primarily within the EU-funded OMI-DE2 and OMI-ATOM projects, and authors are grateful to the European Commission for their continuing support for this work. ARM Limited coordinated these projects; their support, and that of the other project partners, is also acknowledged.

Aspects of the work have benefited from support from the UK government through the EPSRC.

The VLSI design work has leant heavily on CAD tools from Compass Design Automation (now part of Avant!) and EPIC Design Technology, Inc. (now part of Synopsis).

8. References

[1] Furber, S.B., *ARM System-on-Chip Architecture*, Addison Wesley Longman, 2000. ISBN 0-201-67519-6

[2] Jaggar, D., *Advanced RISC Machines Architecture Reference Manual*, Prentice Hall, 1996. ISBN 0-13-736299-4

[3] Furber, S.B., Day, P., Garside, J.D., Paver N.C. and Woods, J.V., "The Design and Evaluation of an Asynchronous Microprocessor", *Proc. ICCD'94*, Boston, October 1994, pp. 217-220.

[4] Paver, N.C., *The Design and Implementation of an Asynchronous Microprocessor*, PhD Thesis, University of Manchester, June 1994.

[5] Woods, J.V., Day, P., Furber, S.B., Garside, J.D., Paver N.C. and Temple, S., "AMULET1: An Asynchronous ARM Microprocessor", *IEEE Trans. Computers*, **46**(4), April 1997, pp. 385-398.

[6] Furber, S.B., Garside, J.D., Riocreux, P., Temple, S., Liu, J., Day, P., Liu, J. and Paver, N.C., "AMULET2e: An Asynchronous Embedded Controller", *Proc. IEEE*, **87**(2), February 1999, pp. 243-256.

[7] Bainbridge, W.J. and Furber, S.B., "Asynchronous Macrocell Interconnect using MARBLE", *Proc. Async'98*, San Diego, April 1998.

[8] Bainbridge, W.J., *Asynchronous System-on-Chip Interconnect*, PhD Thesis, University of Manchester, 2000.

[9] Temple, S. and Furber, S.B., "On-Chip Timing Reference for Self-Timed Microprocessor", *Electronics Letters*, **36**(11) May 2000, pp. 942-943.

[10] Segars, S., Clarke and Goudge, "Embedded Control Problems, Thumb, and the ARM7TDMI", *IEEE Micro*, **15**(5), October 1995, pp. 22-30.

[11] Gilbert, D.A., *Dependency and Exception Handling in an Asynchronous Microprocessor*, PhD Thesis, University of Manchester, 1997.

[12] Segars, S., "The ARM9 Family - High Performance Microprocessors for Embedded Applications", *Proc. ICCD'98*, Austin, October 1998, pp. 230-235.

[13] Bardsley, A and Edwards, D.A., "Compiling the Language Balsa to Delay Insensitive Hardware", Proc. CHDL'97, Toledo, April 1997. Published in Kloos, C.D. and Cerny, E. (eds.) *Hardware Descriptions Languages and their Applications*, IFIP & Chapman Hall, ISBN 01412 78810 1, 1997 pp. 89-91.

[14] Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L. and Yakovlev, A. "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers" *IEICE Trans. on Information and Systems*, **E80-D**(3) March 1997, pp. 315-325.

[15] Furber, S.B. and Day, P., "Four-Phase Micropipeline Latch Control Circuits", *IEEE Trans. on VLSI*, **4**(2), June 1996, pp. 247-253.

[16] van Berkel, K., *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, vol. 5, Intl. Series on Parallel Computation, Cambridge University Press, 1993.

[17] Sutherland, I.E., "Micropipelines". *Communications of the ACM*, **32**(6), June 1989, pp. 720-738.

[18] Endecott, P.B. and Furber, S.B., "Modelling and Simulation of Asynchronous Systems using the LARD Hardware Description Language", *Proc. 12th European Simulation Multiconference*, Manchester, June 1998, Society for Computer Simulation International, pp. 39-43. ISBN 1-56555-148-6.