2. G. Gopalakrishnan, & P . Jain “Some Recent Asynchronous System Design Methodologies”
Technical Report Number UU-CS-TR-90-016, University of Utah, October 1990

3. I.E. Sutherland, “Micropipelines”, Communications of the ACM,
Vol. 32, Number 6, June 1989, pp 720-738.

4. S.B. Furber , “VLSI RISC Architecture and Organization”,
Marcel Dekker Inc., New Y ork, 1989.

5. N. Paver , P. Day , S.B. Furber , J.D. Garside, J.V . Woods
“Register Locking in an Asynchronous Microprocessor”,
ICCD ‘92: IEEE International Conference on Computer Design (1992)

6. VLSI T echnology Inc., “VL86C010 RISC Family Data Manual”
VLSI T echnology Inc., 1987

7. D.V. Jaggar “A Performance Study of the Acorn RISC Machine”
M.Sc. Thesis, University of Canterbury , 1990

8. N. W este & K. Eshraghian “Principles of CMOS VLSI Design”
Addison-W esley , 1985

data path are reduced by about 0.5 pF per signal. Thus the saving in power is extended to other surrounding functions.

4. CONCLUSIONS

This paper describes an asynchronous implementation of an ARM ALU. The unit exploits the fact that most operations can be completed quickly to discard the hardware typically used in an ALU to speed up a few worst-case operations; these operations are rare and so are allowed to take longer . Although it appears that a typical addition in a real programme is somewhat slower than a completely random distribution would suggest it is still much faster than the worst case. The average time for an addition is still about half that of the worst case, with logical operations faster still. Although not as fast as can be achieved by elaborate carry prediction schemes the unit gives a good throughput for a minimal number of transistors.

An asynchronous unit can function with smaller timing safety margins than an externally clocked one as the unit' s self-timed nature means that processing and environmental variations will affect the speeds of the logic and timing circuitry equally . The design should also function at reduced supply voltages. This would reduce instruction throughput but would be more than compensated for by the reduction in power dissipation giving more MIPs per watt.

The resultant design is a simple, compact ALU. It' s lack of carry prediction logic makes it somewhat slower than other designs but this should be compensated for by its ability to deliver most results in less than the worst case time and a reduction in power consumption. It is believed that an ALU in this form will be able to perform significantly more calculations at a reasonable speed with a given amount of energy than a comparable, synchronous unit.

5. ACKNOWLEDGEMENTS

The work described here has been carried out as part of ESPRIT project 5386 (the Open Microprocessor systems Initiative - Microprocessor Architecture Project, OMI-MAP), and the author gratefully acknowledges this support. He is also grateful for additional specific support received during the course of this work from Acorn Computers Limited, Advanced RISC Machines Limited, VLSI T echnology Limited and Compass Design Automation.

REFERENCES

1. A.J. Martin, S. Burns, T .K. Lee, D. Borkovie, P .J. Hazewindus, "The Design of an Asynchronous Microprocessor", Advanced Research in VLSI: Proceedings of the Decennial CalT ech Conference on VLSI, (1989) MIT Press, pp 351-373.

Table 3
Simulated timings for representative operations

	Vdd = 4.6V Temp.= 100 °C			Vdd = 5.0V Temp.= 40 °C			Vdd = 3.0V Temp.= 40 °C		
	slow	typ.	fast	slow	typ.	fast	slow	typ.	fast
logical	7	5	4	6	5	4	9	6	5
addition (typ.)	23	13	9	17	12	8	31	19	12
addition (worst case)	45	22	17	33	21	14	64	38	23

All numbers are timings in ns

This is based on the specified cycle time of the processor and the relative complexity of the various units which must operate in this time. This figure will be a worst case estimate and should be compared to the 45ns figure in table 3. It is interesting to note that the typical operation for the asynchronous unit is actually faster than the synchronous unit.

In practice the fabricated synchronous ARM chips are capable of sustaining much higher clock rates by ignoring the derating allowance for manufacture. However this means neglecting the 'safety margin' which allows for the variation in manufacturing and operating conditions of the device. In contrast the self-timed unit accommodates these variations automatically and therefore operates much closer to its true speed. In this case a typical addition is about twice as fast as the (derated) time for the synchronous ALU and even the worst case, including recovery time, is slightly faster .

3.2. Power consumption

The primary aim of the project has been to use asynchronous techniques to reduce power consumption. This is addressed in several ways. Firstly by omitting any carry look ahead scheme the number of transistors in the ALU can be reduced. This means that there will be inherently fewer switching edges within the unit and consequently - in CMOS technology - a reduction in the power required.

The transistor count is also reduced by the use of dynamic 'domino' logic. This reduces the loading on many of the internal nodes and permits fast operation with small transistors. Dynamic logic was chosen both for these reasons and because it gives simple, glitch free operation which is essential for reliable completion detection.

Finally the compactness of the unit in comparison with its synchronous counterpart means that parasitic capacitances of the buses crossing through the unit in the

circuitry used for the logic operations so that its propagation delay will be similar - it is fabricated close by on the same chip and is subject to the same physical conditions - and an extra delay is then added to allow a reasonable safety margin. The final completion signal is selected from this or the adder circuit, depending on the function being executed.

3.1. Performance

Ideally a power estimation for a CMOS circuit should be extracted from the number of switching transitions each net makes and the load capacitance of that network. Unfortunately it is difficult to derive the number of transitions from existing tools, a problem which is exacerbated by the extreme data dependencies exhibited in the ALU. However with two similar designs it is possible to make some rough estimates by comparing the transistor count and size of the two units.

The self-timed unit contains less than 2300 transistors with an additional 140 for timing purposes, a total well under 2500. A comparable synchronous ARM ALU based on carry selection comprises almost 3000 transistors. In both cases the ALU is 32 bits wide with a data pitch of 36 µm per bit. The asynchronous ALU occupies a section of the data path approximately 340 µm in length as opposed to the synchronous implementation in the same technology which is over 850 µm long; the asynchronous unit is therefore only 40% of the size of the synchronous implementation. Much of this size difference is accounted for by the irregularity of the layout used for carry selection, although the need for larger transistors and more control lines in parts of the synchronous unit also has an effect. A reduction in size is beneficial in decreasing the capacitance of both wiring internal to the ALU and of buses which traverse the unit.

Performance estimates for the asynchronous ALU have been made using SPICE simulation of the extracted layout. The important measurements are the propagation delay for an operation directly through the ALU from input activation to output and the carry propagation time between adjacent bits. Simulation has been performed under a range of conditions, some results of which are shown in table 3. Estimates are given for logical (including 'move') operations and both typical and 'worst-case' additions. These include simulation at different power supply voltages which suggest that the ALU will operate correctly - albeit slower - at supply voltages lower than 2V , an important consideration for power conservation.

In practice these figures will be less than the actual cycle time required for the operation as they do not take into account either the precharging of the dynamic logic or the overhead required for control functions. A reasonable *estimate* for the control overhead would be something similar to the logical operation evaluation time; thus a typical addition cycle would take about 17ns at 5V and 40 °C. With an instruction mix of 20% move/logical instructions this yields an average execution time of around 14ns or 70 million operations/s.

An estimate of the propagation delay of the worst case addition in the synchronous processor is that, under poor conditions (100 °C, 4.6V), it requires about 30 ns.

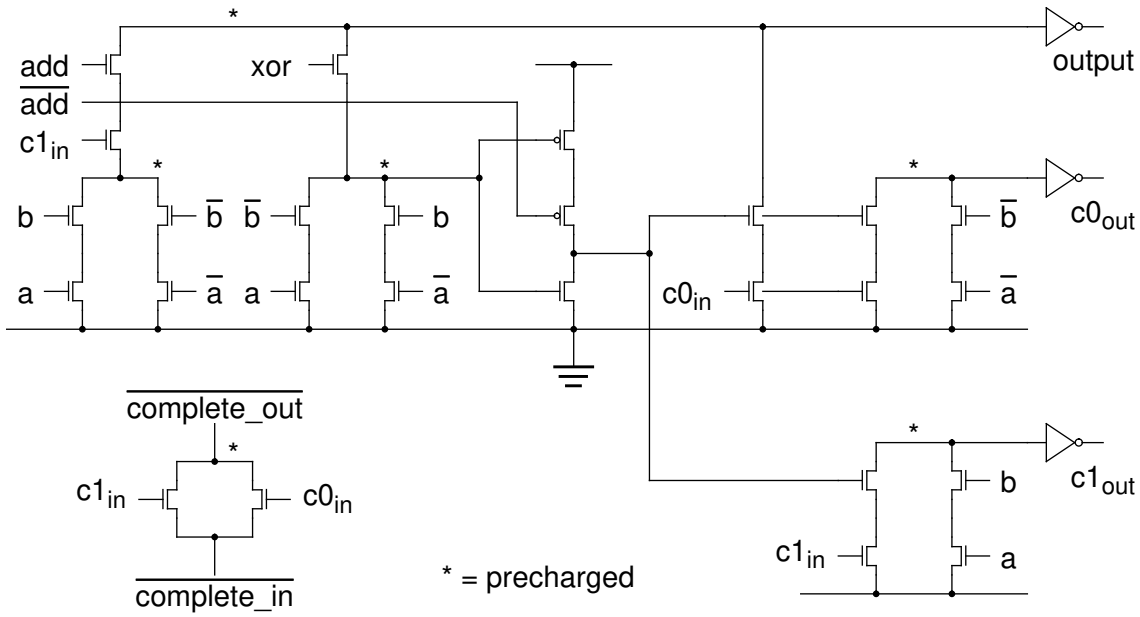

Figure 4: The adder circuit in the ALU

This ALU has no special fast carry logic and performs addition with a chain of thirty two full adders. The only concession to the asynchronous nature of the unit made at the design stage was that the carries between the individual bits are encoded onto pairs of wires signalling the “0” and “1” states of the carry bit respectively . In this fashion it is possible to detect that a carry signal has arrived at a given bit position by observing a change in state of one of these signals. ALU completion is signalled when a carry has been transmitted to all 32 bits in the word.

The inset on figure 4 shows an adder completion stage. Logically all thirty two of these are wired in series and the completion signal is generated when the arrival of one or other of the carry states arrives at that stage. This gate would be impractical and the completion signal is staged and passed through a tree of gates. Because completion is detected from the carry input it occurs slightly before the result becomes valid; this is however a constant, known delay and is accounted for by the 32-way fan-in of the completion signal and subsequent control gating. This delay modelling is characteristic of the bounded delay approach to asynchronous circuit design, as opposed to the more ‘pure’ delay-insensitive approach.

The bitwise (move and logical) operations are not significantly data dependent and take a constant time - typically less than addition operations. The timing mechanism for these operations differs from that of the addition operation in that it uses a constant delay element to signal completion. This delay is in the form of a ‘thirty third bit’ on the data path which is physically located furthest from the activation signals to ensure that it is the last to operate. This circuit is designed to produce an output state change for every logical operation. It closely resembles the

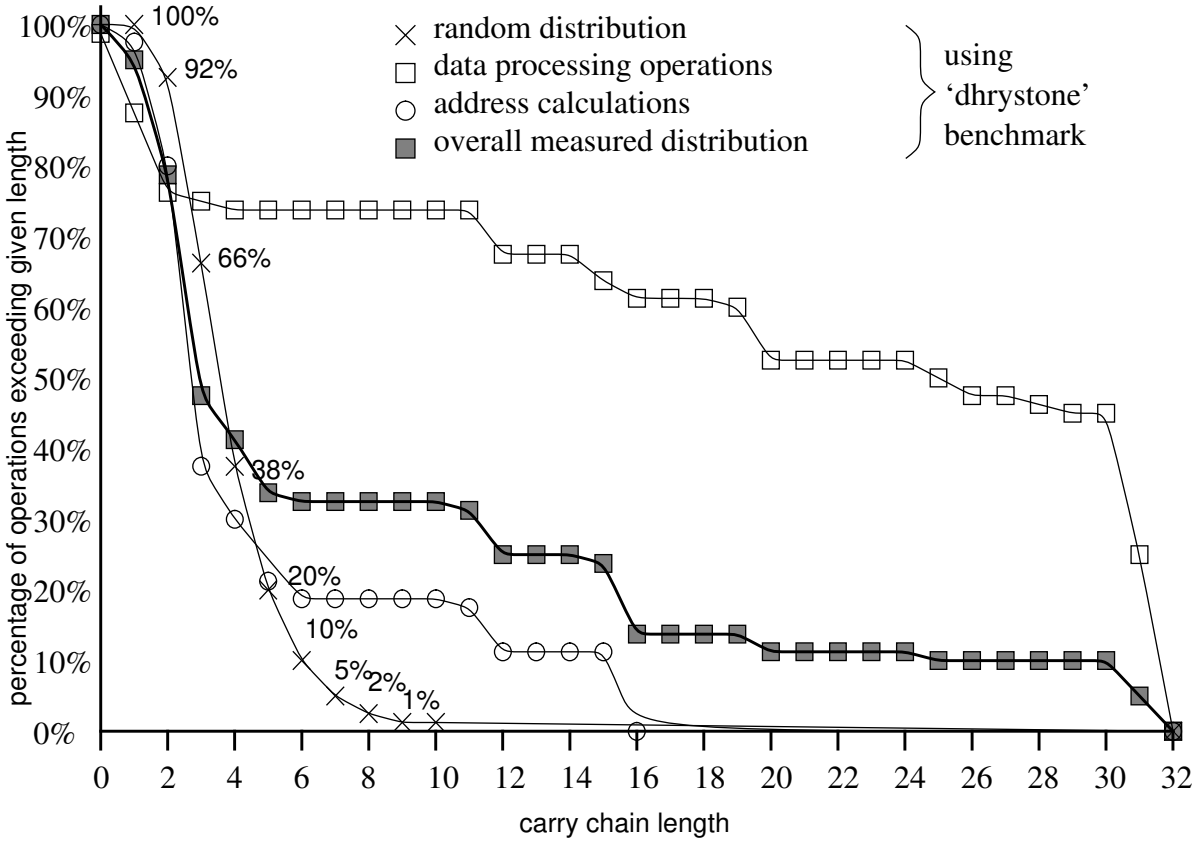

Figure 3: Proportion of longest carry chains exceeding given length

are not required for a given operation are inhibited from operating and thus consume essentially no power; for example carry propagation is inhibited except during addition operations.

The input signals are converted into both true and complement forms at the input stage (with conditional inversion). This means that a signal will be propagated on one of two wires carrying the state of the bit. The use of dynamic logic ensures that the internal operation gives at most one edge on each signal so that the completion signals can be detected unambiguously .

It is interesting to note that - in part - the internal operation thus relies on delay-insensitive two-rail 4-phase (i.e. level-driven) logic. This form of logic is used primarily for convenience, but also carries timing information in the case of the addition operation.

From the input stage the input signals are fed to the evaluation stage; the adder part of the evaluation logic is shown in figure 4. In this diagram either a or $\overline{a}$ and either b or $\overline{b}$ go active (logic high), but never both. The control signals add and $\overline{add}$ are set active before this operation. The precharge transistors are omitted for clarity on this diagram, as are the logical functions AND and OR; the XOR function has been included as it requires only a single control transistor extra to the adder and serves to illustrate how the different functions are multiplexed onto the output.

from this study have indicated that typical data operations have carry propagation chains approximately 18 bits long whereas those for address calculations are about 9 bits. The preponderance of address calculations means that the overall average appears to lie between 12 and 13 bits; this is subject to confirmation by running larger test programmes.

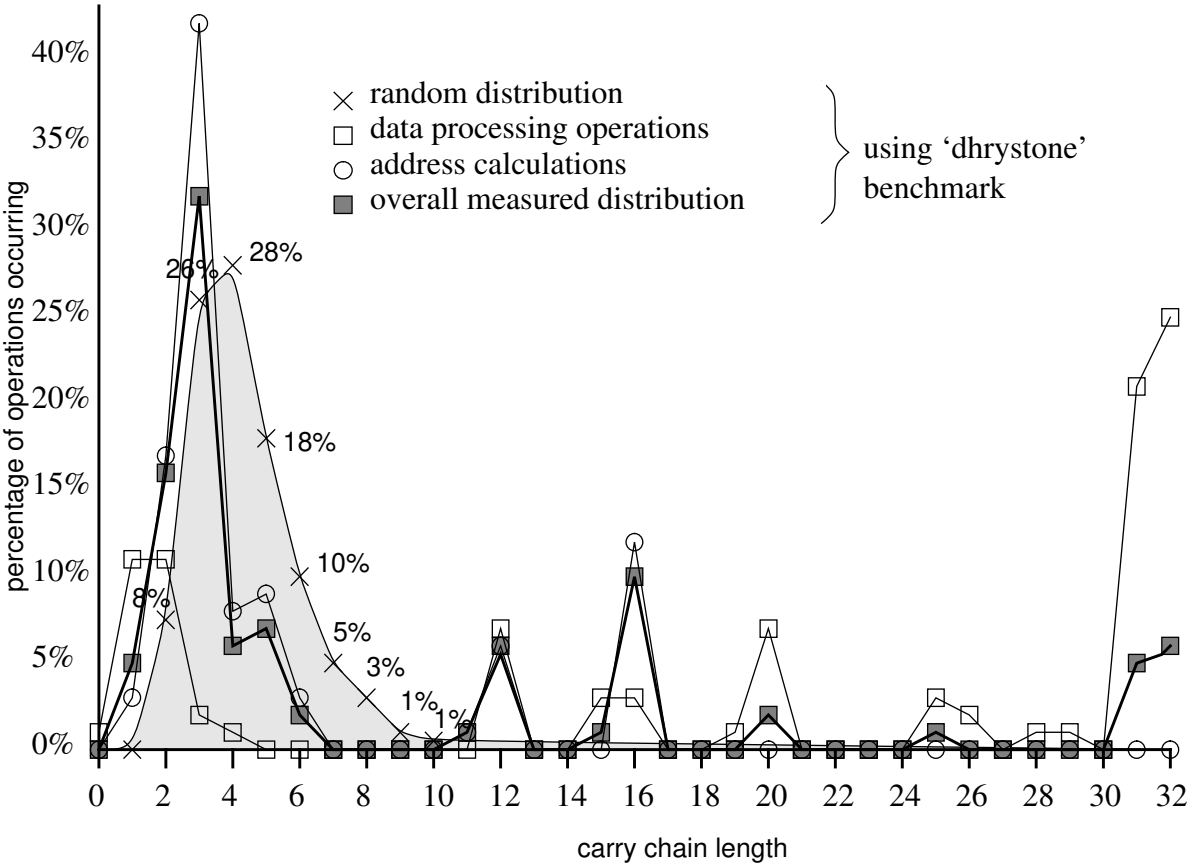

Figure 2: Longest carry chain length distribution - 32-bit word

3. IMPLEMENT ATION

Using the principles outlined above an ALU has been designed for integration with the asynchronous ARM implementation currently under development. The addition function is performed by single bit stages with a ripple carry running between them whilst the logical operations are conventionally implemented, wherever possible sharing gates with the adder .

The design is targeted at a 1.2 µm CMOS process for direct comparison with an existing synchronous processor . The asynchronous ALU is implemented in dynamic CMOS 'domino' logic [8]. This both reduces the transistor count and ensures that any signal propagation through the unit is glitch-free. Gates which

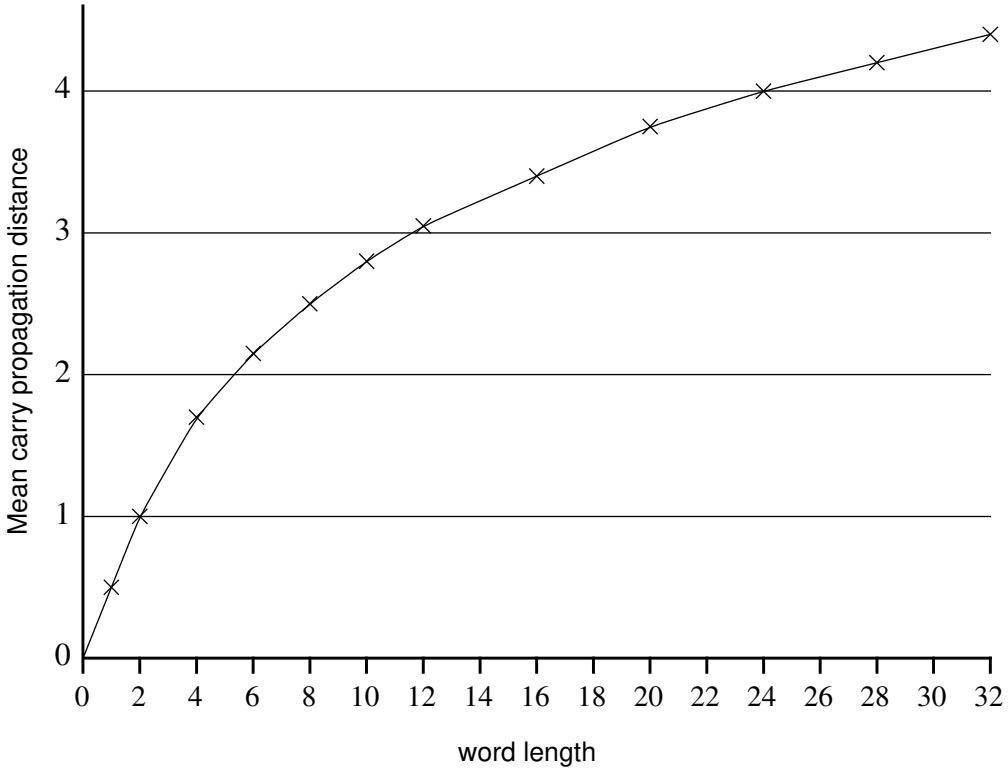

Figure 1: A verage size of longest carry propagation chain
for different word lengths assuming random data distribution

provides an acceptable model. In a dynamic trace of typical compiled ARM code [7] over 50% of all instructions are branches or memory accesses and require a 'random' addition which compares favourably with the 20% of operations which are comparisons or arithmetic operations and have 'predictable' data input.

One potential penalty of relying on this approach is that address and branch calculations are often highly repetitive. Most programmes spend the majority of their time executing a few short loops. If this critical section contains some pathological cases of the addition operation then the software execution speed could be significantly affected by minor changes to the source code altering branch lengths and positions. For this reason this design of ALU may be less than ideal for address calculations; in such circumstances it may be preferable to use some extra logic to reduce the worst case-timings. This is an important distinction because many processors use separate ALUs for data and address calculation.

An internal study at Manchester University has examined the remaining cases of the data processing operations. This has produced a distribution of maximum carry propagation chains for the address calculations which is not dissimilar from the random model although with some significant extra peaks{fig.s 2,3}. The distribution for data operations however appears radically different. Preliminary results

Table 2a
Full adder complete truth table

Inputs			Outputs	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2b
Full adder carry output

A	B	Cout
0	0	0
0	1	Cin
1	0	Cin
1	1	1

approaches require a large quantity of circuitry to accommodate the few pathological cases.

In an asynchronous ALU addition operations may take different times depending on the input data, providing that some means of detecting completion is included. If the cases with long carry propagation 'chains' are relatively rare a simple adder may be used which - despite poor worst-case performance - can deliver 'typical' results in a reasonable time. This allows a reduction in size and complexity of the ALU, with a consequent reduction in power consumption, without radically altering the typical performance.

2.3. T ypical addition operations

The degree of exploitation of short 'typical' addition times is dependent on what a typical operation is. It can be observed from table 2b that a carry is propagated through a bit position if the two operand input bits are different i.e. the exclusive-OR of the two bits is equal to '1'. The carry propagation chains are therefore represented by contiguous strings of '1' s if the input operands are XOR'ed together . As the carry signals will propagate along these chains in parallel only the length of the longest chain is significant in determining the addition time. If addition operands are assumed to be random then the longest chain has a mean value much less than the maximum - for example the mean carry propagation length is about 4.4 bits for 32-bit operands - over seven times less than the 32-bit worst case - and this benefit improves with increasing word length {fig. 1}.

In practice the data are not random and it is necessary to derive statistics from a dynamic instruction trace. However there is still merit in figures derived from random input data; for example the ARM processor performs operations such as address offsets for relative branches and memory address calculations with the main ALU. As these depend on variables such as programme size and load address - which are unpredictable for any particular circumstance - assuming random data

Table 1
ARM ALU data processing functions

op-code	mnemonic	operation	'A' input	'B' input	function
0	AND	and	true	true	AND
1	EOR	exclusive-OR	true	true	XOR
2	SUB	subtract	true	complement	add
3	RSB	reverse subtract	complement	true	add
4	ADD	add	true	true	add
5	ADC	add with carry	true	true	add
6	SBC	subtract with carry	true	complement	add
7	RSC	reverse sub w/ carry	complement	true	add
8	TST	test bits	true	true	AND
9	TEQ	test equal	true	true	XOR
A	CMP	compare	true	complement	add
B	CMN	compare negative	true	true	add
C	ORR	OR	true	true	OR
D	MOV	move	zero	true	OR
E	BIC	bit clear	true	complement	AND
F	MVN	move NOT	zero	complement	OR

Note: functions 8 to B do not produce a result and are only used to set the flags

A study of ARM instruction frequencies [7] reveals that about 20% of all instructions perform arithmetic data processing with a similar proportion of logical operations (including 'move'). However the ARM also uses its ALU to perform calculations for 'load', 'store' and 'branch' operations, so that about 80% of the operations performed by the ALU require some form of addition. It is therefore important to ensure that addition is performed rapidly , either by the use of complex hardware or by exploiting its inherent data dependency .

2.2. The addition operation

In an individual bit addition three inputs are required: the two operand bits and a carry input from the previous stage. The addition operation is limited by the speed of the propagation of the carry signal across the word. However the carry output from a single bit addition does not always depend on the carry input and in half of the possible input cases it may be generated before the input carry state is known {table 2}. It is therefore unlikely that a carry signal will have to propagate across many bit positions before it reaches one where its state has already been correctly predicted.

In conventional (synchronous) ALUs all operations must take the same amount of time; a considerable effort has been expended in schemes such as carry look ahead and carry steering in order to speed up the addition operation; these

Several approaches to asynchronous circuit design are currently being investigated [2]. The technique used here is based upon Sutherland's "Micropipelines" [3], a bundled-data, bounded-delay model. In this schema data buses with common timing characteristics are 'bundled' together and their validity is indicated by local handshake signals whose delays are crafted to model the data's timing characteristics. Delay modelling is similar to the technique used in conventional synchronous design to ensure that data will be valid at a fixed time such as a clock edge. The micropipeline approach was chosen over other asynchronous techniques for its economy in silicon area and thus its potential for low power consumption, although parts of the ALU borrow techniques from other asynchronous design methodologies.

2. THE ALU

The ARM architecture [4] defines a register-based RISC processor in which data processing operations require one or two operands to be read from the register bank and a single result value to be returned; the Arithmetic Logic Unit (ALU) performs these operations. The ALU is also used to perform other operations on the processor such as branches and address calculations. The architecture also requires a barrel shifter on one of the ALU input buses. In existing synchronous ARM implementations there is a three-stage pipeline consisting of fetch, decode and execute stages; instruction *execution* is not further pipelined and an arithmetic operation is completed within a single clock cycle. In the asynchronous implementation instruction execution is decomposed into a number of pipeline stages; the ALU forms one of these stages, as does the register bank [5] and the shifter.

2.1. ARM ALU functions

The set of functions performed by the primary ALU of various microprocessors is fairly consistent and can normally be decomposed into 'move' operations, the basic logic functions (AND, OR and exclusive-OR) and an addition operation all with the optional complementing or forcing-to-zero of one or both of the input buses. The ARM instruction set provides sixteen data processing operations [6] and a method of synthesizing these is shown in table 1. In addition to these operations the ALU can perform functions which are implicit in other instructions, such as moving data from the 'A' bus to its output or providing a zero value output.

In an asynchronous architecture the time consumed in processing an operation is not constrained to a fixed cycle but may depend on both the operation being performed and the data it is performed on. Of the four *functions* listed in table 1 addition is potentially the most time consuming because logical operations are performed in a bitwise fashion whereas addition requires interaction across the entire word length. The addition operation may therefore be allowed a longer processing time than the logical operations.

A CMOS VLSI Implementation of an Asynchronous ALU

J. D. Garside

Department of Computer Science, Manchester University ,
Oxford Road, Manchester , M13 9PL, U.K.
e-mail: jgarside@cs.man.ac.uk

Abstract

A CMOS self-timed ALU has been developed as part of an asynchronous implementation of the ARM microprocessor . This unit exploits the data dependency inherent in many arithmetic operations to enable a small, simple ALU to deliver a mean performance comparable with that of a more sophisticated synchronous one with consequent reductions in both silicon area and electrical power consumption. The self-timed nature of the unit means that the majority of operations complete quickly whilst allowing rare 'worst-case' operations to take longer , maintaining a high average throughput.

This paper presents instruction usage statistics to justify the claimed performance and SPICE simulation results of measurements taken from the layout.

Keyword Codes: B.2.0; B.2.1; B.7.1
Keywords: Arithmetic and Logic Structures, General; Arithmetic and Logic Structures, Design Styles,Integrated Circuits, T ypes and Design Styles

1. INTRODUCTION

The ever -increasing availability of high performance processing has led to greater expectations from computer equipment. One offshoot from this is a requirement for powerful processors in such devices as portable computers and embedded controllers. In such circumstances low electrical power consumption is often an important consideration, especially when the equipment is battery powered. One approach which promises high performance with low power consumption is the use of asynchronous computing techniques. T o investigate these a self-timed implementation of the ARM microprocessor - a 32-bit RISC architecture developed by Advanced RISC Machines Limited - is being produced as a commercially realistic technology demonstrator . Other researchers [1] have demonstrated the feasibility of building a complete asynchronous microprocessor; the current project addresses the problems associated with the re-implementation of an existing commercial architecture with the specific goal of minimising power consumption.