Four-Phase Micropipeline Latch Control Circuits

Stephen B. Furber and Paul Day

Department of Computer Science, The University,
Oxford Road, Manchester M13 9PL, England.

Abstract

Standard micropipelines use simple two-phase control circuits. The latches employed on AMULET1 are level-sensitive, so two- to four-phase converters are required in each latch controller. To avoid this overhead an investigation has been carried out into four-phase micropipeline control circuits; this has thrown up several design issues relating to cost, performance and safety, and forms a useful illustration of asynchronous design techniques.

1: Introduction

The AMULET1 asynchronous implementation of the ARM microprocessor [1,2] employs two-phase control circuits as advocated by Sutherland in his seminal paper on micropipelines [3]. The two-phase communication protocol is illustrated in Fig. 1. Data is set up by the sender, which then issues a transition on the request line to signal its validity. When the receiver has accepted the data, it issues a transition on the acknowledge line. Rising and falling transitions alternate and carry the same meaning, so the signal level has no significance.

However, many internal functions require level-sensitive (four-phase) control, so the design has several instances of two- to four-phase signalling converters. Perhaps the most prominent case in point is in FIFO structures where standard transparent latches are used in preference to Sutherland's 'capture-pass' latches on the grounds of area efficiency [4]. The control logic for each stage then comprises a Muller C-element for two-phase synchronisation and an exclusive-OR and 'toggle' [3] for two- to four-phase conversion as shown in Fig. 2; the conversion circuit dominates the logic cost of the control.

An obvious way to avoid the conversion circuit cost is to implement the micropipeline handshake with four-phase signalling. This introduces redundant 'return to zero' events on the handshake lines, as shown in Fig. 1, which cause sequencing problems to which there are several solutions as presented in the remainder of this paper.

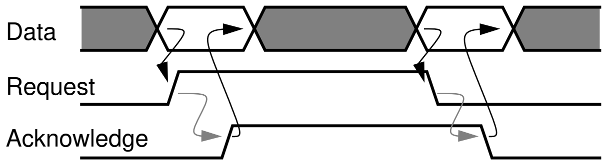

Fig. 1. The two-phase handshake protocol

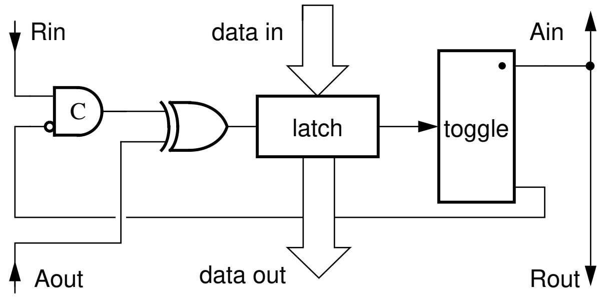

Fig. 2. A two-phase latch control circuit

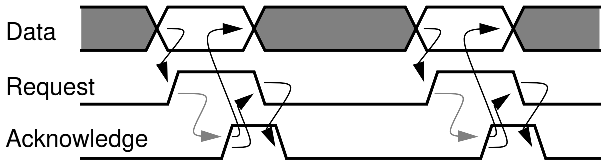

Fig. 3. The four-phase handshake protocol

The design process used, and described in some detail in this paper, is based on Signal Transition Graphs (STGs) [5] which represent the sequencing of circuit transitions in a formal, but intuitively understandable, way.

The resulting circuits offer a range of performance and cost options depending on the requirements of the application, and these will be employed in future designs to improve their speed and cost relative to AMULET1.

2: Two- and Four-Phase Handshakes

The control logic for a micropipeline register must support the handshake protocol on both its input and its output ports. It is therefore defined in terms of input and output requests (*Rin*, *Rout*) and acknowledges (*Ain*, *Aout*), and its internal latching function.

In the standard two-phase protocol an event (rising or falling edge) on *Rin* signals the availability of input data and the register issues an event on *Ain* to indicate to the source of the data that it has been captured and may be removed (see Fig. 1). The latch also issues an event on *Rout* to indicate that its output data is now valid and will be held stable until an event on *Aout* signals that it has been accepted by the next stage in the pipeline.

When four-phase signalling is used there is a choice to be made as to which edge (rising or falling) of each handshake signal is active and takes the place of the event described above; the other edge is inactive and is part of the recovery phase during which the circuit prepares for the next cycle. In this paper we will choose rising edges to be active in every case, as shown in Fig. 1.

Previous work in this area [6,7] included four-phase latch controller designs which employed edge-triggered data registers. Transparent latches require roughly half the number of transistors used in edge-triggered registers and therefore allow more compact datapaths to be constructed.

3: Four-phase Latch Control

In addition to the handshake signals, the latch circuit also has an internal control signal (*Lt*) which causes the data latches to be open (transparent) when low and closed when high.

The specification of the latch control circuit is illustrated in STG form in Fig. 4. The dashed arrows indicate orderings which must be maintained by the environment; the solid arrows represent orderings which the circuit must itself ensure. These STG fragments reflect the handshake sequence on the input and output sides of the circuit, together with some behavioural properties: *Rout*+ indicates the availability of output data and must therefore follow *Rin*+; when input data is available (signalled by *Rin*+) the latches may close (*Lt*+) and then the input may be acknowledged (*Ain*+); when the output data has been acknowledged (*Aout*+) the latches may open again (*Lt*-); the latch must alternate between open and closed.

The design problem is to merge these STG fragments into a single dependency diagram where all the partial orderings required by the specification are preserved (either directly, or indirectly through other transitions), and then to check that the resulting unified STG has the properties which are required to give a reliable implementation. The principle such property is that of persistency, which means that a signal transition must depend on all the transitions that depend on its inverse transition. This ensures that once a signal has switched it cannot switch back until the first transition has been stable long enough to allow all dependent transitions to occur.

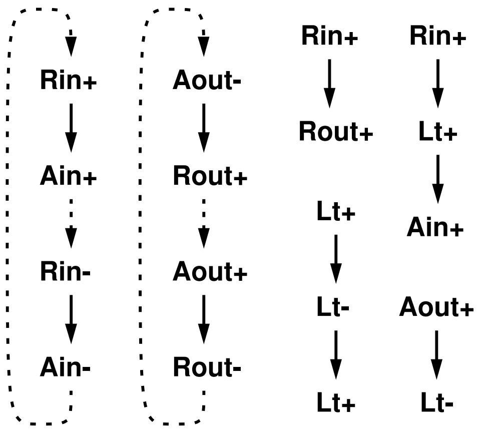

Fig. 4. Specification of 4-phase control circuit

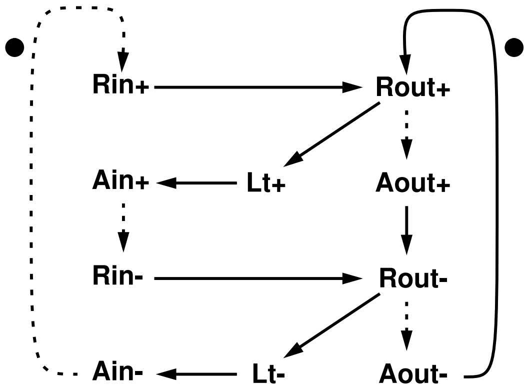

Fig. 5. STG of simple 4-phase control circuit

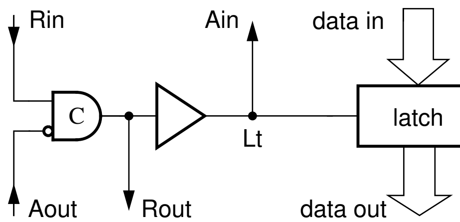

Fig. 6. Simple 4-phase latch control circuit

4: Simple 4-Phase Latch Controller

The simplest merged STG is shown in Fig. 5. The state graph may be derived from the STG and an implementation from the state graph, but in this case it may be seen by inspection that the circuit in Fig. 6 is an implementation of this STG.

Whilst this circuit operates correctly, it has some undesirable properties. Most notable of these is that when several such latches are formed into a FIFO, at most alternate stages can be occupied at any time. This is because *Aout* must be low (and therefore the next latch empty) before *Lt* can go high (and this latch become full).

Certain engineering assumptions are built into Fig. 6. It is expected that there will be several bits of latched data, so *Lt* must have reasonable drive buffering. The path from *Rin* to *Ain* reflects the need for the latch to close before input data may be removed, and the buffer delay is built into this path. The buffer delay is not, however, built into the path from *Rin* to *Rout*, since there is no need for the latch to close before *Rout* is signalled so long as the data has propagated through the latches. Therefore the C-gate delay must be no shorter than the latch data-in to data-out delay for the correct operation of this circuit. This assumption is, in fact, also built into the specification in Fig. 4.

5: Semi-Decoupled 4-Phase Latch Controller

To increase the decoupling between the input and output sides of the latch, and in particular to allow a FIFO to fill all its stages, the STG must be revised to allow the latch to close before the output handshake has completed. A suitably modified STG is shown in Fig. 7.

Note here that in order to achieve the desired decoupling, an internal variable (*A*) has been introduced to record when the input side is ready to proceed. Now the latch can close before the adjacent latch on the output side is empty, since *Lt*+ is concurrent with *Aout*-.

To simplify the derivation of a state graph for this STG it may be observed that *Lt* and *Ain* follow directly from *A* with internal dependencies only; *Lt* and *Ain* may therefore be omitted from the state graph.

The STG can be transformed into the state graph in Fig. 8 (without the dashed transitions and the '1111' state between them; these will be referred to later) by applying the underlying Petri net rules [8] to construct the reachability tree. The state graph has a unique state at each node, so the logic equations for the output variables may be derived from it directly:

$$
\begin{aligned}
&A+ && = 0101 + 0100 && = Rin.\overline{Rout} \\
&A- && = 1011 && = \overline{Rin}.Rout.Aout \\
&Rout+ && = 1000 + 1100 && = A.\overline{Aout} \\
&Rout- && = 0011 + 0111 && = \overline{A}.Aout
\end{aligned}
$$

Since the states 0010 and 0110 are not reachable it is possible to simplify the last equation further:

$$Rout- \ = \overline{A}.$$

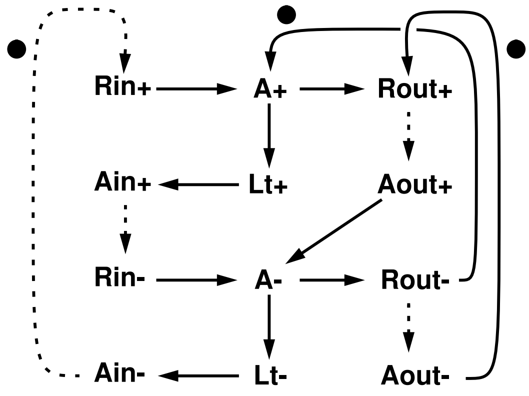

Fig. 7. Semi-decoupled 4-phase STG

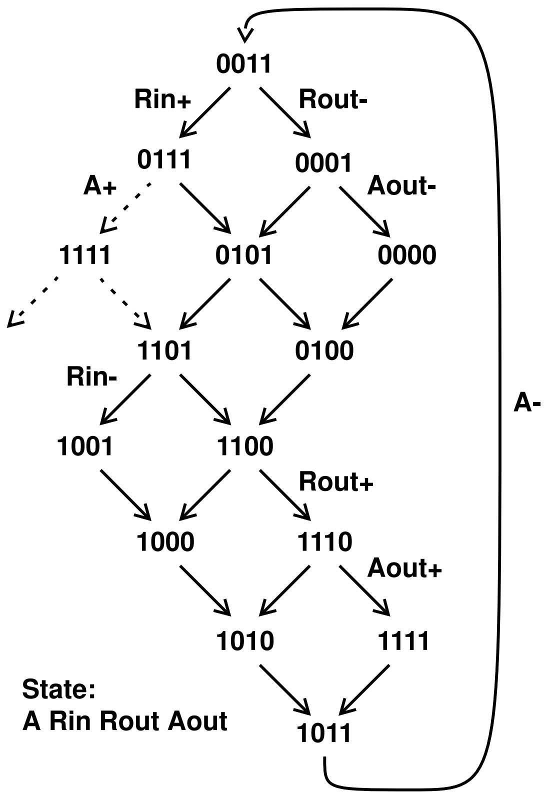

Fig. 8. Semi-decoupled state graph

These equations may then be used to construct an implementation based on R-S flip-flops or, equivalently, asymmetric Muller C-gates. The C-gate implementation is illustrated in Fig. 9.

The notation used in this figure for asymmetric C-gates indicates that an input controls both edges of the output when it is connected to the main body of the gate; it controls only the rising edge when connected to the extension marked '+', and it controls only the falling edge when connected to the extension marked '-'. This notation is illus-

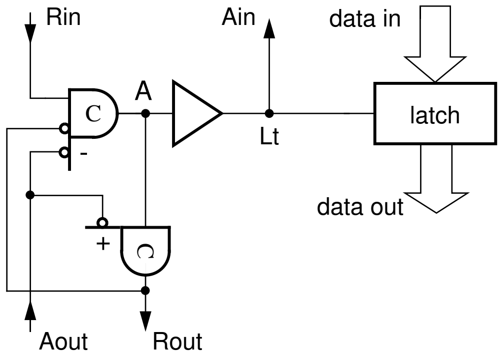

Fig. 9. Semi-decoupled control circuit

trated in Fig. 10 which shows a possible transistor-level implementation of an asymmetric C-gate.

6: Fully decoupled 4-phase latch controller

The circuit in Fig. 9 can be used to construct a four-phase micropipeline FIFO with good performance, and every stage can be filled if the output process stalls. However the recovery cycles on the two sides of the stage are still linked; *Ain* cannot return to zero until *Aout* has gone high. In a simple FIFO this does not present a problem, but if the pipeline incorporates processing logic there may be a performance loss due to this partial coupling.

Full decoupling may be achieved by introducing another state variable as shown in Fig. 11. Here, as soon as the new data has been latched (*Lt+*) the handshake on the input side can complete and return to zero.

This STG is highly concurrent and the state diagram is correspondingly complex. To simplify it slightly we omit *Lt* since it always follows *A* directly via an internal path. The resulting state diagram is shown in Fig. 12.

The equations for the outputs and state variables generated by the circuit are:

$$
\begin{aligned}
A+ &= \overline{B}.\overline{Rout}.Rin \\
A- &= B.Rout.Aout \\
B+ &= Ain \\
B- &= \overline{A}.\overline{Ain} \\
Ain+ &= A.\overline{B} \\
Ain- &= \overline{Rin}.B \\
Rout+ &= A.\overline{Aout} \\
Rout- &= \overline{A}
\end{aligned}
$$

These equations may be implemented as the control circuit shown in Fig. 13.

7: Fully decoupled controller with long hold

In certain applications it is useful for the latch to hold data until *Aout* goes low, rather than releasing it when *Aout*

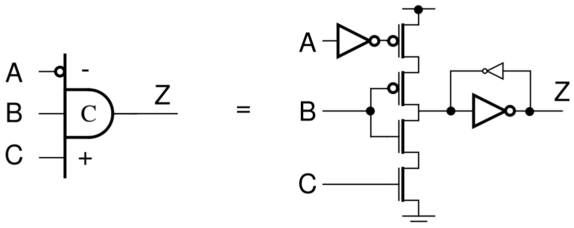

Fig. 10. Asymmetric C-gate notation

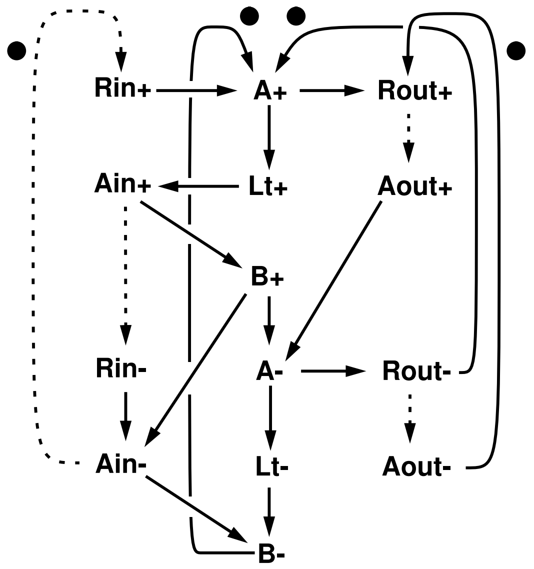

Fig. 11. Fully decoupled 4-phase STG

goes high as does the circuit in Fig. 13. If, for example, the latch is holding the read address for a register bank, then *Rout* may be used to enable the address decoder. *Aout+* indicates that the data has been read from the register bank and *Aout-* indicates that the decoder has been disabled.

An STG for a fully decoupled latch controller which holds the output data until *Aout-* is shown in Fig. 14 and the corresponding circuit is shown in Fig. 15.

8: Engineering optimisations

The STGs in Figs. 7, 11 and 14 were designed to produce speed-independent circuits, and therefore have the property of persistency: when a signal makes a transition it cannot subsequently make the inverse transition until all immediate successor transitions have occurred.

The circuit in Fig. 9 is very similar to that designed earlier by Paul Day using an intuitive approach [9]. The only difference is that the circuit in Fig. 9 causes *A+* to wait for $\overline{Rout}$. This is necessary for speed-independent operation, but is not necessary for correct operation of the circuit under realistic conditions since the path from *A-* to *Rout-* is internal to the control circuit and represents a single C-gate

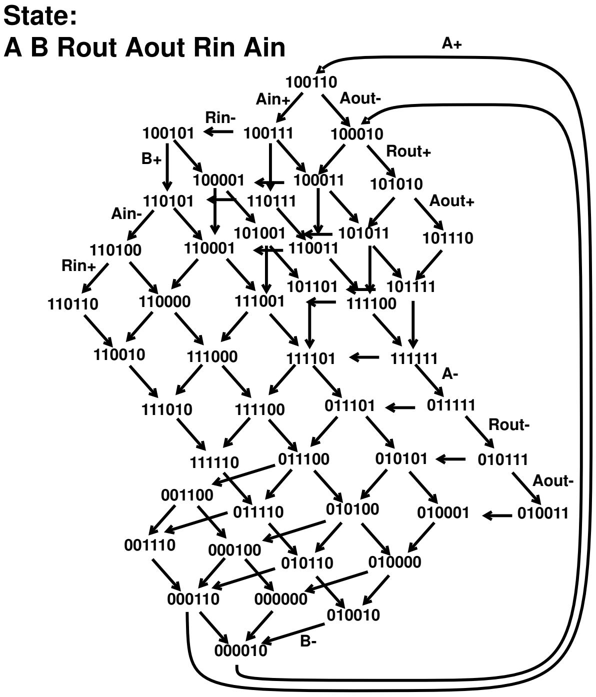

Fig. 12. Fully decoupled state graph

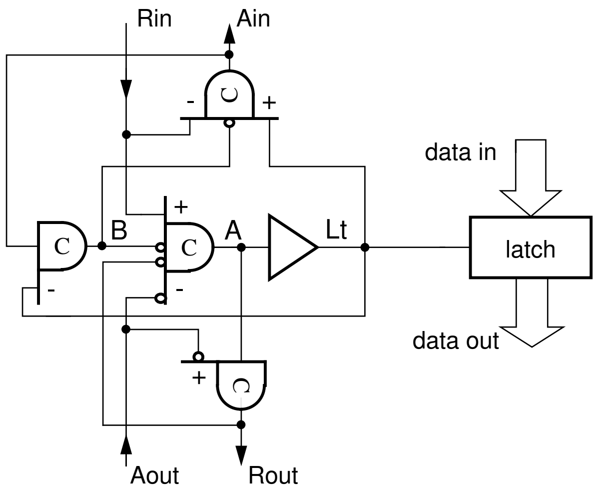

Fig. 13. Fully decoupled control circuit

delay, whereas the path from *A-* to *Lt-* to *Ain-* to *Rin+* includes the buffer path to drive the latches. The removal of this dependency corresponds to removing the arc from *Rout-* to *A+* in Fig. 7, whereupon the STG no longer has the persistency property. In principle this then allows the dashed transitions in Fig. 8 which lead to duplicated states, but in practice this will never happen since the *Rout-* transition will happen first and state 0111 will never be reached.

The non-persistency property of Day's original circuit was first exposed by Alex Yakovlev at Newcastle University using the FORCAGE tool [10].

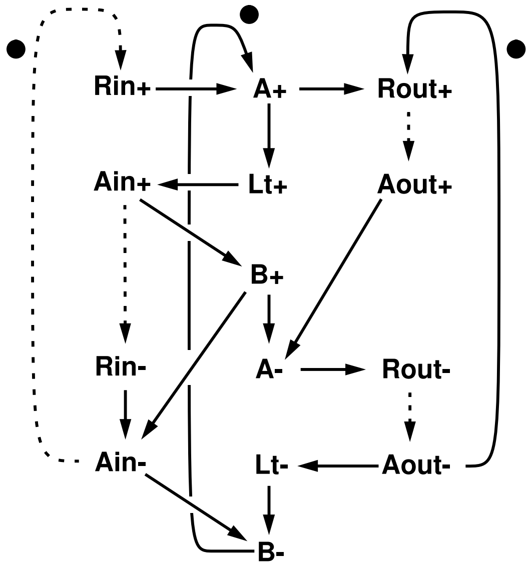

Fig. 14. Fully decoupled long hold STG

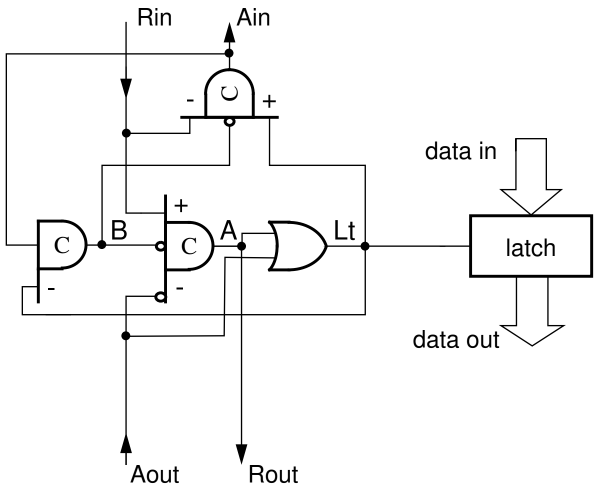

Fig. 15. Long hold control circuit

A similar argument may be made for the removal of the connection from *Rout* to *A+* (but not to *A-*) in Fig. 11. The resulting circuit will operate safely under normal conditions and will be slightly faster than the original.

9: VLSI implementations

All of the proposed latch control circuits (apart from the simple case) have subsequently been optimised for driving a 32-bit, single-phase latch control line with a total load of 1pF. Corresponding layout for these control circuits has been generated using a 1µm, double layer metal CMOS standard cell library containing standard gates plus the required asymmetric Muller C-gates. Figure 11 shows the

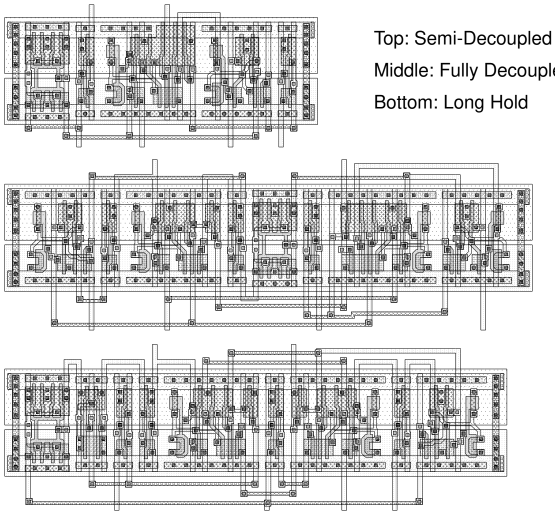

Fig. 16. Standard cell layout of latch control circuits

VLSI layout for the three different latch control circuits.

To analyse the relative performance of the various latch control circuits SPICE analysis [11] has been performed on extracted layout for worst case conditions (*Vdd*=4.6V, slow-slow process corner, at 100˚C). Table 1 shows the simulation results for the three latch types.

The table also shows the minimum cycle time for each type of latch. The cycle time was measured by constructing a 3-stage pipeline for each latch circuit type and hard-wiring the input request to the input acknowledge and the output request to the output acknowledge. These figures therefore represent the minimum cycle times achievable with these circuits.

Parameter	Latch Control Circuit		
	Semi-Decoupled	Fully Decoupled	Long Hold
Rin ↑ to Rout ↑	4.1nS	3.8nS	1.4nS
Rin ↑ to Ain ↑	4.1nS	5.0nS	6.1nS
Rin ↓ to Ain ↓	4.2nS	2.0nS	2.0nS
Aout ↑ to Ain ↓	4.7nS	n/a	n/a
Aout ↑ to Rout ↓	4.3nS	4.8nS	2.3nS
Aout ↓ to Rout ↑	2.4nS	2.4nS	6.3nS
Min. Cycle Time	14.1nS	15.5nS	16.2nS
Processing Pipe Cycle Time	54.7nS	39.7nS	42.3nS

Table 1: SPICE Analysis Results

The minimum cycle time for the semi-decoupled latch

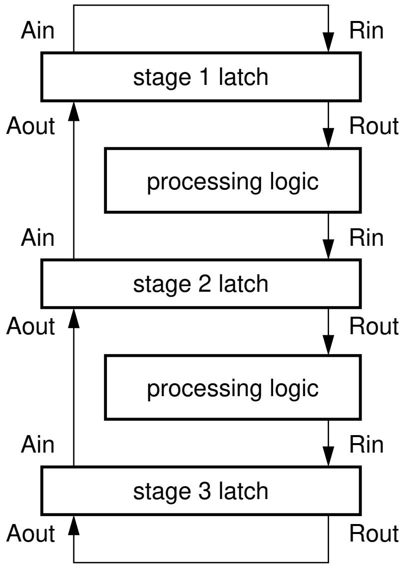

Fig. 17. Pipeline structure used for cycle time measurements

controller can be further reduced to 12.3nS by removing two inversion stages from the circuit (resulting in both input and output acknowledge interfaces being inverted). This may be compared with the minimum cycle time for a two-phase control circuit using similar latches of 16.8nS, reported earlier [9].

The minimum cycle time measurements above represent a pipeline with no data processing functions between latch stages. The inclusion of processing logic between stages can be simulated using the same technique as above but with dynamic processing elements inserted either side of the second latch stage (see Fig. 17). Pipeline simulations have been carried out modelling a dynamic processing element with an evaluate (*Rin* rising to *Rout* rising) delay of 20.3nS and a precharge (*Rin* falling to *Rout* falling) delay of 5.4nS. The bottom row of Table 1 shows the resulting cycle time measurements for each of the latch types. Here we now see how the differing decoupling techniques have affected the resulting cycle times with the fully decoupled latch control showing a clear performance advantage over the semi-decoupled latch controller.

When the semi-decoupled controller is used, the cycle time of the latch includes both the processing delay on the input side and that on the output side. This may be seen by considering three copies of Fig. 7 joined together to form the STG of the 3-stage pipeline, as shown in Fig. 18. (Note how the internal 'environmental' constraints are replaced by existing constraints when the STGs are joined.) The processing and recovery delays occur on arcs marked 'P' and 'R' respectively in this figure, and the heavier arcs

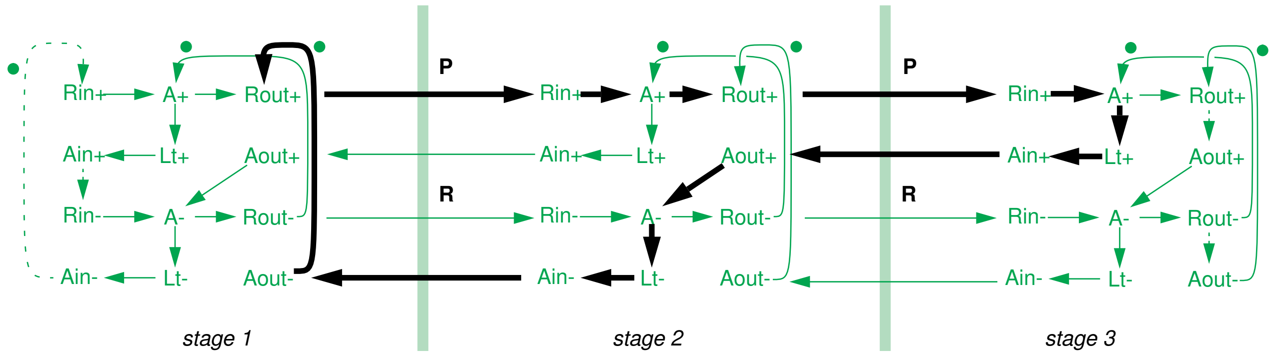

Fig. 18. STG of a 3-stage semi-decoupled pipeline, showing critical cycle

highlight a cycle which includes two 'P' arcs. The recovery time is concurrent with the processing time on the other side of the latch, so the cycle time is increased by the sum of the two processing delays, which is approximately 40nS in this case.

The fully-decoupled controller allows the processing delays on either side to run concurrently since the long cycle shown in Fig. 18 is broken, so the recovery delay is now visible, and the cycle time is increased by the sum of the processing delay and the recovery delay, which is approximately 25nS.

The times chosen here are representative of dynamic processing logic. Static logic has no recovery time, so the advantage of the fully-decoupled controller will be greater.

Although the 4-phase circuits look complex, they all use fewer transistors than the 2-phase circuit shown in Fig. 2 since the 2-phase 'toggle' is a complex component [4].

As a point of reference, a synchronous pipeline with these processing delays could cycle in around 30nS (processing plus recovery plus latch delay) but would require edge-triggered latches. To operate with level-sensitive latches a synchronous pipeline would be restricted to 50% occupancy and a cycle time around 50nS (two processing and two latch delays).

10: Conclusions

Four-phase control circuits offer higher performance and lower cost than two-phase circuits for micropipelines which use conventional level-sensitive data latches.

Four different designs of four-phase circuit have been presented with different cost and performance properties. The design process has been described for each of these circuits and some optimisations presented which simplify the circuits whilst not seriously compromising their safe operation under normal conditions. These circuits have been checked out by conventional simulation, but they have not yet been implemented in silicon. The fully persistent variants have also been produced from their STGs using FORCAGE [10] to confirm their correctness.

In FIFO applications it is expected that the semi-decoupled control circuit will give the best performance at the lowest cost; only when the pipeline includes processing logic between successive stages will the fully decoupled circuit offer an advantage. Where the circuit requires the data to be held throughout the cycle the long hold variant of the fully decoupled latch may have advantages.

The simple controller shown in Fig. 6 is not functionally equivalent to the other three, since it only allows a 50% pipeline occupancy. Likewise, earlier designs [6,7] are not strictly comparable since they depend on the use of edge-triggered latches, although despite employing a circuit which appears similar to the semi-decoupled controller presented here, they are, in fact, fully decoupled.

An equivalent synchronous processing pipeline could operate 25% faster, but would require more expensive edge-triggered latches to do so. With level-sensitive latches it, too, would be restricted to 50% occupancy and would deliver a 25% lower throughput than the fully-decoupled controller circuit.

The design methodology was based on Signal Transition Graphs which are a formal (but intuitive) way of representing transition orderings. From the STG a state graph may be generated, and from that an implementation expressed in asymmetric C-gates (or, equivalently, R-S flip-flops). This approach gives a framework within which engineering optimisations are possible. The complexity of the state graphs produced in the course of this work suggests that this methodology is approaching its limit of usefulness with the examples presented here, at least without computer assistance. Tools such as FORCAGE extend the potential scope of similar approaches to more complex circuits at the cost of losing some of the engineering 'feel' of the manual approach.

These circuits will be incorporated into the next version of the AMULET asynchronous implementation of the ARM microprocessor, and are expected to make a significant contribution to its performance and power-efficiency.

11: Acknowledgments

The work described in this paper was supported as part of ESPRIT project 6909, OMI/DE-ARM (the Open Microprocessor systems Initiative - Deeply Embedded ARM Applications project). The authors are grateful for this support from the CEC.

12: References

[1] S. B. Furber, P. Day, J. D. Garside, N. C. Paver and J. V. Woods, "A Micropipelined ARM", in *Proceedings of the IFIP TC 10/WG 10.5 International Conference on Very Large Scale Integration* (VLSI'93), Ed. T. Yanagawa and P. A. Ivey, North Holland, pp. 211-220, September 1993.

[2] S. B. Furber, P. Day, J. D. Garside, N. C. Paver and J. V. Woods, "AMULET1: A Micropipelined ARM", in *Proceedings of the IEEE Computer Conference* (CompCon'94), pp. 476-485, March 1994.

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

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

[5] T.-A. Chu, "Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications", in *Proceedings of ICCD'87*, IEEE, pp. 220-223, October 1987.

[6] T. H.-Y. Meng, R. W. Brodersen and D. G. Messerschmitt, "Asynchronous Design for Programmable Digital Signal Processors", in *IEEE Transactions on Signal Processing*, vol. 39, no. 4, pp. 939-952, April 1991.

[7] G. M. Jacobs and R. W. Brodersen, "A Fully Asynchronous Digital Signal Processor Using Self-Timed Circuits", in *IEEE J. Solid-State Circuits*, vol. 25, no. 6, pp. 1526-1537, December 1990.

[8] J. Peterson, *Petri Net Theory and Modelling of Systems*, Prentice Hall, 1981.

[9] P. Day and J. V. Woods, "Investigation into Micropipeline Latch Design Styles", in *IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 264-272, June 1995.

[10] M. Kishinevsky, A. Kondratyev, A. Taubin and V. Varshavsky, *Concurrent Hardware - The Theory and Practice of Self-Timed Circuits*, Wiley Series in Parallel Computing, 1994.

[11] L. W. Nagel and D. O. Pederson, *Simulation Program with Integrated Circuit Emphasis (SPICE)*, University of California, Berkeley, Electronics Research Lab. Report No. ERL-M383, 12th April 1983.