Dynamic Logic in Four-Phase Micropipelines

S. B. Furber and J. Liu

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

Abstract

Micropipelines are self-timed pipelines with characteristics that suggest they may be applicable to low-power circuits. They were originally designed with two-phase control, but four-phase control appears to offer benefits for CMOS implementations.

In low-power applications static circuit behaviour is desirable since it allows activity to cease (and hence power to be saved) without loss of state. However, dynamic circuits offer the benefits of increased speed and lower switched capacitance. Therefore low-power designs often employ dynamic logic with additional latches or charge-retention circuits to give pseudo-static behaviour. These additions increase the cost and power consumption of the dynamic circuits, thereby compromising their potential advantages.

Circuits are proposed in this paper that allow dynamic logic to operate efficiently within a four-phase micropipeline framework without the above-mentioned encumbrances whilst still retaining externally static behaviour.

1: Introduction

Micropipelines were introduced by Ivan Sutherland in his 1988 Turing Award lecture [1]. They are self-timed circuits which employ a two-phase bundled data protocol.

The AMULET1 processor [2, 3, 4], developed at the University of Manchester between 1991 and 1994, used the micropipeline design style, but its successor, AMULET2, abandoned the two-phase control in favour of four-phase control mainly for performance reasons.

Both AMULET processors employ dynamic logic techniques in some parts of their datapaths, but since the delay-insensitive control circuits can stall in any state, extra circuitry is included (such as an additional latch or charge-retention circuitry) to give the dynamic circuits pseudo-static behaviour.

Dynamic logic may easily be used in a self-timed pipeline without additional circuitry at the cost of some performance loss. Leakage causes the output of the circuit to be valid for a short time only; therefore the circuit should not begin evaluation until the output latch is free. It also requires its inputs to be held stable until evaluation is complete, so during evaluation both the input and the output latches are required by the intervening dynamic logic, resulting in at most 50% of the logic being active at any time.

The new idea introduced here is to observe that the above description is slightly over-constrained. It is not strictly necessary that the output latch *is* free before evaluation begins; it is only necessary to know that it will *become* free within a short time (the dynamic storage time of the output nodes). This relaxation of the condition for starting evaluation allows a significant improvement in the pipeline's performance.

2: Four-phase handshake protocols

The four-phase handshake protocol employed in most control circuits on AMULET2 is the 'early' protocol illustrated in figure 1. This uses the rising edge of the Request line to indicate 'data available' and the rising edge of the Acknowledge line to indicate 'data latched'. The falling edges are return to zero actions that carry no meaning.

Other protocols are possible. The 'late' protocol uses the falling edges as active and the 'broad' protocol uses the rising Request and falling Acknowledge edges as active.

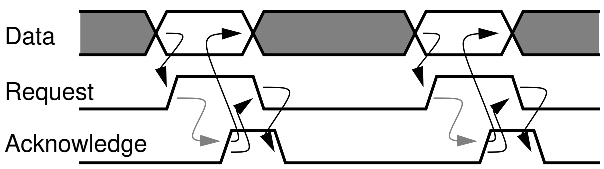

Fig. 1. The 'early' handshake protocol

(All these protocols take the micropipeline view that the sender of the data initiates the transfer; where the receiver is the initiator, yet further protocols are possible.)

For the current work we wish to propagate a signal back up the pipeline to indicate that dynamic evaluation may begin. This could be achieved using an additional wire, but in the interests of efficiency it is desirable to minimize the number of wires, so instead we use one of the inactive transitions in the existing protocol. Since we want the 'evaluate' signal to follow 'data available' and to precede 'data latched', this restricts the choice of protocol to 'broad'. The signalling sequence is illustrated in figure 2.

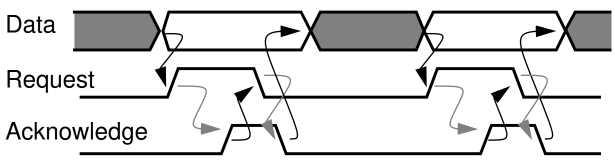

Fig. 2. The 'broad' handshake protocol.

3: Pipeline stage structure

Using the broad protocol outlined above, the general form of a processing pipeline stage is as shown in figure 3.

The dynamic logic and output register are controlled using the input request and acknowledge (*Rin* and *Ain*) and the output request and acknowledge (*Rout* and *Aout*).

The dynamic logic begins evaluation when its enable (*E*) goes high and indicates a valid output on a 'done' signal (*D*). When its enable is low it is precharged, and precharge completion is signalled by the 'done' signal going low.

The output data register may either be a transparent latch which passes data when its latch enable (*Lt*) is low and holds it when high, or it may be a positive-edge triggered flip-flip.

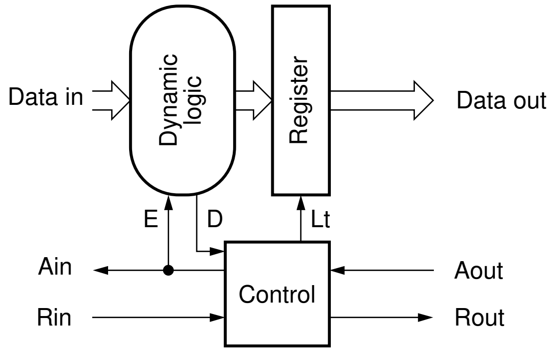

Fig. 3. Pipeline stage structure.

The assumption will be that the stage is connected to similar neighbours. However, there is a discussion on the issues of interfacing to different protocols at the end of the paper.

4: Simple latch controller specification

A simple dynamic pipeline which defers evaluation until the output latch is free is used here as a reference point. The control is specified using a Signal Transition Graph (STG) [5] which shows the orderings of the signal transitions. An STG for a controller for a register built from transparent latches is shown in figure 4. The notation used here follows that used in previous work to develop four-phase micropipeline latch controllers which operated using the early protocol and displayed various different levels of coupling between the input and output handshake sequences [6]. The dashed arrows indicate dependencies which the environment must observe; the solid arrows represent orderings which this circuit must maintain. The solid 'tokens' drawn next to certain arcs represent an initial 'marking', and a particular transition can fire only when there is a token on each of its input arcs (only *Rin*+ can fire in figure 4). When a transition fires, a token is placed on each of its output arcs.

This STG shows that evaluation begins (*E*+) when the input data is valid (*Rin*+) and the output latch is free (*Lt*-). When evalution is complete (*D*+), the data is latched (*Lt*+) and the valid output data signalled (*Rout*+). Note that as the latches are transparent, *Rout*+ need not follow *Lt*+ provided that time is allowed for the data to propagate through the latches. The broad protocol allows the input handshake to proceed through *Ain*+ and *Rin*-, with *Ain*- waiting until the data is safely latched (*Lt*+), and the data is held until the output handshake completes (*Aout*-) before being released (*Lt*-). The state variable (*A*) is not strictly necessary here (it

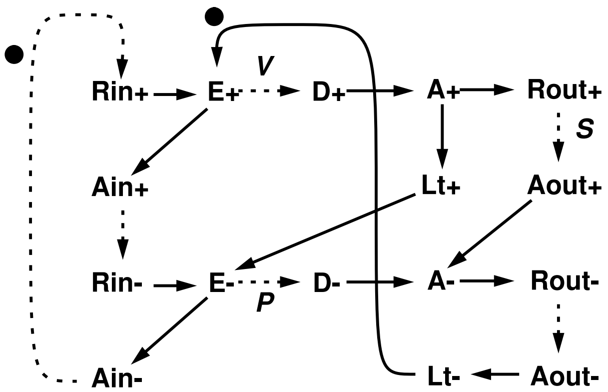

Fig. 4. Simple dynamic pipeline latch controller STG

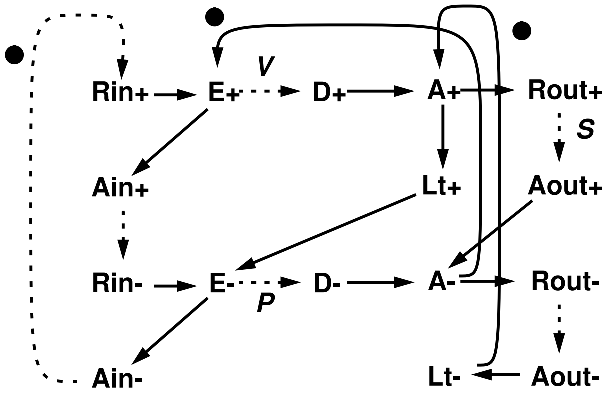

Fig. 5. Enhanced latch controller STG

can be merged into *Rout*) but it will be used in subsequent developments of the circuit.

(Since *E* and *Ain* are wired together in figure 3 they are not strictly ordered, so figure 4 is over-constrained. However, the simple dependence of *Ain*+ on *E*+ and *Ain*- on *E*- will result in a circuit which is just a wire, as required. We could simply omit *Ain* or *E* from the STG, but this makes the STG less clear as a specification, so we have retained both.)

The evaluate phase (*V*) and the precharge phase (*P*), together with a few internal control delays, determine the cycle time of the stage and the throughput of the pipeline.

5: Enhanced latch controller specification

The enhanced controller uses the optimisation mentioned earlier to begin evaluation as soon as it is known that the output latch will become free 'soon' rather than waiting until it is free. A suitable STG is shown in figure 5. The information about the output latch becoming free is propagated back from the next controller on *Aout*+, and this controller must generate a similar signal on *Ain*+.

The STG incorporates a state variable (*A*) which is used to achieve the required operation. In this circuit the data is latched (*Lt*+) when the dynamic logic has completed its evaluation (*D*+) and held until the next stage has finished using it (*Aout*-). Evaluation begins (*E*+) when the input data is ready (*Rin*+) and the previous result has entered processing in the next stage (*Aout*+). This condition guarantees that the output latch will become free 'soon'.

Here 'soon' is interpreted as any period which is not subject to arbitrary external delay, so it is the result of internal self-timed delays only. We argue that, if the next stage is similar to the current one, it can only stall between *Rout*+ and *Aout*+ on the arrow marked *S* in figure 5. If this is true, this property is propagated back to the input, and hence, by induction, along a pipeline of similar stages. Only the end conditions remain to be checked, and we shall return to this later.

6: Edge-triggered latch control specification

The transparent latch controller holds the latch normally open, allowing transients to propagate down the pipeline, wasting power (though the dynamic circuitry in the next stage may make this wastage very low). Where this is undesirable, edge-triggered latches may be used to ensure that only valid data values are propagated.

Normally an edge-triggered latch should have static behaviour when its clock is held either high or low, but a particularly simple form of dynamic single-phase edge-triggered latch becomes rather complex when a full set of weak feedback components is added. Instead, we can design the control circuit so that the clock is never held high and weak feedback is required only when the clock is held low. The latch circuit is shown in figure 6.

The STG specification of a suitable control circuit for this edge-triggered latch is shown in figure 7.

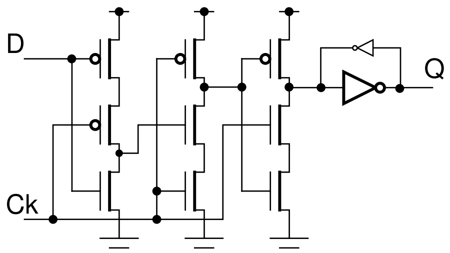

Fig. 6. Edge-triggered latch circuit

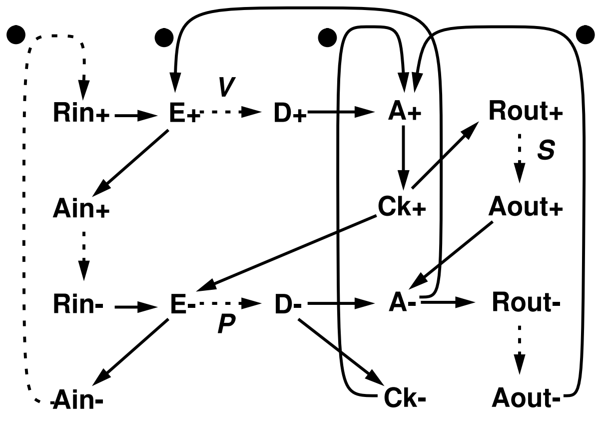

Fig. 7. Edge-triggered latch controller STG

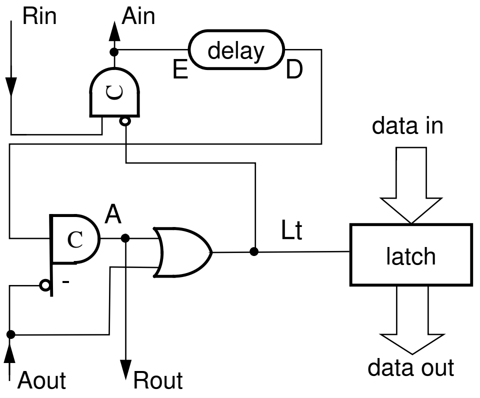

Fig. 8. Simple latch control circuit

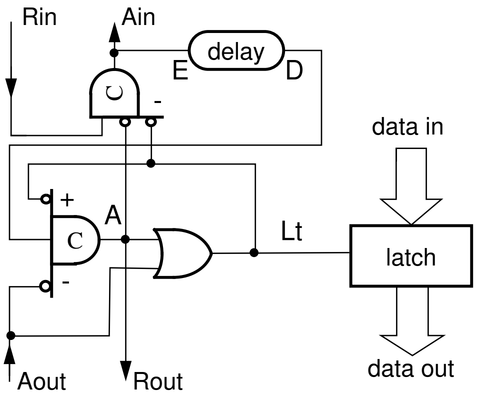

Fig. 9. Enhanced latch control circuit

7: Control circuits

The STG specifications may be used to derive a state graph by following the underlying Petri Net rules [7], and then an implementation derived from the state graph. Alternatively, tools exist which automate this procedure. The tool used here was FORCAGE [8]. The implementation is expressed in the form of logic equations which may be converted into R-S flip-flops or Muller C-gates.

The Muller C-gate implementation of the simple latch controller is shown in figure 8, the enhanced controller in figure 9 and the edge-triggered latch controller in figure 10. It is interesting to observe the small but important differences between figure 8 and figure 9 which lead to significantly different behaviour and performance.

The notation used in these figures 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 illustrated in figure 11 which shows a possible transistor-level implementation of an asymmetric C-gate.

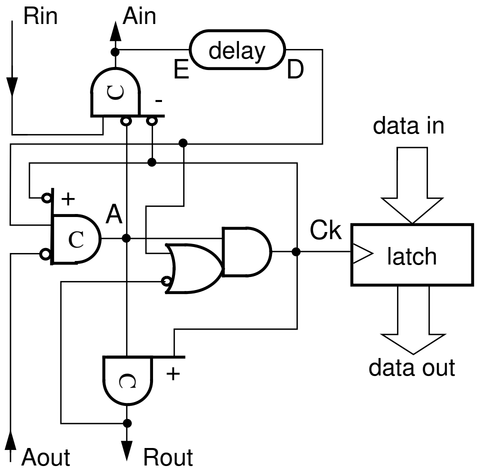

Fig. 10. Edge-triggered latch control circuit

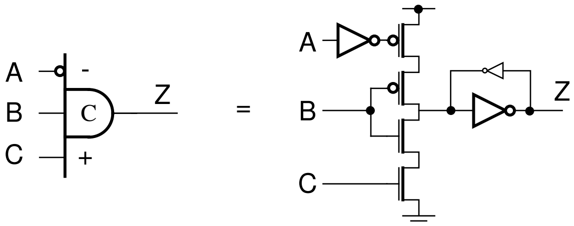

Fig. 11. Asymmetric C-gate notation

8: Performance

The latch control circuits have been laid out using 0.5 micron CMOS design rules and simulated using SPICE [9] operating at worst-case conditions (*Vdd* = 3.3V, *Vss* = 0.1V, slow-slow process corner, at 100˚C) and driving a 32-bit latch. The results of this analysis are shown in Table 1.

The most significant results, shown at the bottom of the table, are the cycle and response times with and without processing logic. The minimum cycle time indicates how rapidly a simple FIFO could propagate data. Here there is no processing logic, so the issue of using dynamic or static circuit techniques does not arise, but this gives an upper bound on the potential throughput. The response time is measured by stalling the output of a 3-stage pipeline until it is full, and then seeing how long it takes from releasing the output until the input starts moving (i.e. the delay from *Aout*+ to the last stage to *Ain*+ from the first stage). Here the enhanced circuits move very quickly.

The corresponding results for a processing pipeline are established by inserting processing logic on the input to each register in a 3-stage pipeline as illustrated in figure 12. The logic used here has an evaluation delay of 5.8nS and a precharge delay of 0.9nS. The cycle time of the simple pipeline increases by approximately twice the processing

Parameter	Dynamic Latch Control Circuit		
	Simple	Enhanced	Edge-Triggered
Rin ↑ to Ain ↑	0.7nS	0.8nS	0.8nS
Rin ↓ to Ain ↓	0.6nS	0.8nS	0.8nS
Aout ↑ to Ain ↑	n/a	2.2nS	2.2nS
Aout ↓ to Ain ↓	n/a	4.9nS	3.7nS
Aout ↑ to Rout ↓	1.0nS	1.0nS	1.7nS
Aout ↓ to Rout ↑	n/a	2.4nS	3.1nS
D ↑ to Rout↑	0.8nS	1.1nS	2.7nS
D ↓ to Rout↓	0.8nS	0.8nS	1.4nS
Min. Cycle Time	7.8nS	7.5nS	8.0nS
Min. Response	16.5nS	7.2nS	7.7nS
Proc. Cycle Time	20.1nS	13.6nS	14.3nS
Proc. Response	29.5ns	7.8nS	7.8nS

Table 1: SPICE Analysis Results

delay, reflecting the 50% duty-cycle that this circuit can achieve, whereas the cycle times of the enhanced and edge-triggered circuits just increase by the evaluation delay. The response times of the enhanced and edge-triggered circuits are largely unchanged since all stages begin evaluation at almost the same time, whereas the simple circuit must wait for each stage to clear before initiating action in the preceding stage.

9: Comparisons with static pipelines

Earlier work presented three different designs of latch controller for pipelines incorporating static or pseudo-static processing logic which were differentiated by the degree of coupling between the control cycles in adjacent pipeline stages [6]. These controllers all use the early handshake protocol, though the 'long-hold' controller effectively supports the broad protocol on its output side.

The reported performance of the older circuits was based on simulations using parameters based on a different process technology from that used here, so the figures are not directly comparable. The fully-decoupled controller was therefore re-simulated on this process technology, where it shows a very similar minimum cycle time (8.2nS) and a considerably longer minimum response time (19.1ns) compared with the dynamic pipeline circuits. The cycle times of the fully-decoupled and long-hold controllers increase in proportion to the processing delay, whereas the cycle time of the semi-decoupled controller increases by twice the processing delay as observed before [6].

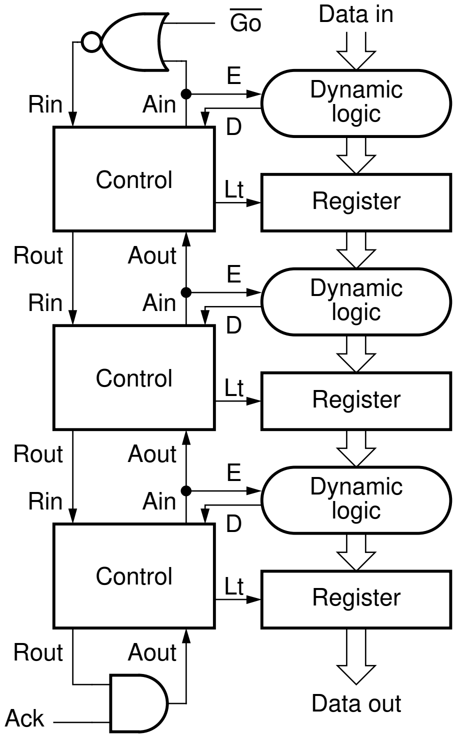

Figure 12: The pipeline structure used for cycle time and response measurements

The new circuits therefore offer similar performance to (and better reponse time than) the older circuits whilst allowing dynamic processing logic to be exploited without any need for circuitry to enforce pseudo-static behaviour.

10: Interfacing to other protocols

Although the broad protocol may be used throughout a design, there are occasions where it may be desirable to interface to alternative protocols such as the early protocol used in the semi-decoupled, fully-decoupled and long-hold latch controllers presented elsewhere [6]. In particular, it is a good idea to ensure that the end conditions of the dynamic pipeline are satisfied when interfacing to less-well controlled regions of the design, such as when going off-chip.

To interface from an early protocol into a dynamic pipeline as presented in this paper, it is necessary only to hold the input data valid until the output acknowledge falls (that is, to use the broad protocol on the output side). The long-hold latch controller has the required characteristics.

To interface a dynamic pipeline into an early protocol latch would appear to be straightforward, since the broad protocol is more than sufficient to cover the input specification of an early protocol latch. However, the dynamic pipeline is not simply using the broad protocol, it is also signalling additional information. In particular, the output device must never stall between *Aout*+ and *Aout*-. This condition is satisfied by the fully-decoupled controller but not the semi-decoupled controller.

11: Conclusions

A framework has been presented which allows dynamic logic to be supported within an asynchronous framework where externally static behaviour is achieved without recourse to charge-retention circuits by enabling evaluation to begin only when it is known that the output latch will shortly become available. This information is propagated back up the pipeline using a previously redundant transition in the 'broad' four-phase protocol to signal 'nearly ready'.

The dynamic pipeline may be interfaced to 'early' protocol static pipelines using a 'long-hold' latch on the input side and a 'fully-decoupled' latch on the output side, and in particular these interfaces may be used to encapsulate the dynamic region to ensure its fully pseudo-static external behaviour.

In addition, a dynamic pipeline controller has been presented which is designed to operate with edge-triggered latches that are static only with their clock inputs low. The controller uses self-timing to ensure that the clock never stalls high, thereby allowing a simpler latch circuit to be employed. This controller uses the same handshake protocol as the level-sensitive latch controller.

Using dynamic logic in an asynchronous framework such as the one presented in this paper is more efficient than in any clocked framework where the clock may be slowed or stopped in any state for power conservation, since the locally self-timed evaluate and latch cycle is well matched to the dynamic storage properties of these circuits.

12: Acknowledgements

This work is the outcome of research prompted by an off-the-cuff comment made by Peter Beerel during a conversation held on a pier in the Sea of Galilee during a workshop held in Israel in March 1995. Peter's seminal contribution is gratefully acknowledged.

13: References

[1] I. E. Sutherland, "Micropipelines", *Communications of the ACM*, Vol. 32, Number 6, June 1989, pp 720-738.

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

[3] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods, "AMULET1: A Micropipelined ARM", *Proceedings of the IEEE Computer Conference*, March 1994.

[4] N. C. Paver, "The Design and Implementation of an Asynchronous Microprocessor", PhD Thesis, 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] S. B. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits", *IEEE Trans. on VLSI Systems*, June 1996.

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

[8] 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.

[9] 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.