Investigation into Micropipeline Latch Design Styles

Paul Day and J.Viv. Woods

***Abstract*— An asynchronous implementation of the ARM microprocessor has been designed and fabricated based on Sutherland's Micropipeline approach. Reviews of this work have shown that considerable performance improvement may be possible in a number of key design areas. This paper assesses the effects of different design styles on the micropipeline latch structures used.**

The original design has latch structures based on pass-transistor transparent latches. An evaluation of the use of single-phase transparent latch structures is given plus the application of 2-phase and 4-phase control techniques.

I. Introduction

The present increase in awareness of power dissipation of high performance CMOS microprocessors has led to an upsurge of interest in asynchronous design as a low-power technology. From high performance processors having a power consumption of 20-30 Watts to the growth in consumer demand for hand-held battery-powered equipment, the power issue is now one which cannot be ignored.

Asynchronous design is not new [1-6] but has largely been neglected by contemporary digital designers who instead have opted for the clocked, globally synchronised, approach. With the observation that synchronous logic design is beginning to reach serious limits with regard to clock distribution and skew, asynchronous design (where global synchrony is abandoned) would seem to offer significant benefits, being free from these design problems. Also, by their very nature, asynchronous circuits only use energy when doing useful work.

To investigate whether an asynchronous approach would offer significant advantages in the design of RISC microprocessors over more conventional methods, an asynchronous implementation of the ARM processor [7] has been designed and fabricated [8,9]. The asynchronous methodology applied to this design was based on Sutherland's "Micropipelines" [10], this being chosen over other asynchronous methodologies as being the most practical, with the right balance of engineering cost and performance.

Resulting silicon has been proved to be functional, executing programs generated by standard ARM development tools such as the assembler and C compiler. Performance figures, however, from this first prototype implementation do not show any major benefits over the equivalent synchronous design [11]. Evaluation of the design has shown that considerable performance improvements may be possible in a number of key design areas. The micropipeline latch design style chosen for the asynchronous ARM implementation is considered as an area where improvements can be made.

The authors are with the Department of Computer Science, The University, Oxford Road, Manchester, M13 9PL, U.K.

II. Micropipelines

The micropipeline approach uses bundled data with a 2-phase transition signalled handshake protocol to control data transfers, see Fig. 1.

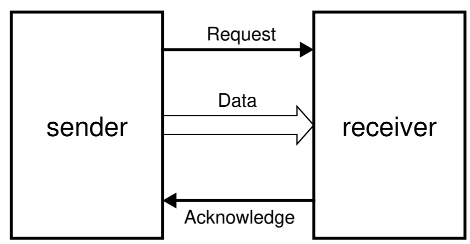

Fig. 1. A two-phase bundled data interface

The interface between sender and receiver consists of a bundle of data which carries information (using one wire for each bit) and two control wires; *Request* from the sender to the receiver carries a transition when the data is valid; *Acknowledge* from the receiver to the sender carries a transition when the data has been used. The protocol for this sequence is illustrated in Fig. 2. This defines the sequence in which events must occur, there is no upper bound on the delays between consecutive events.

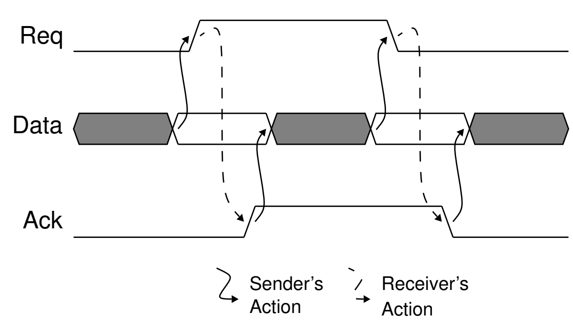

Fig. 2. Bundled data interface protocol

Other asynchronous design styles such as dual-rail encoding, where each Boolean is implemented on two wires to allow the timing information to be communicated for each data bit, are considered to be delay-insensitive [12]; that is they are insensitive to variations in the delays of logic gates and wiring. Once the data bundling constraints are met the micropipeline approach can be considered delay-insensitive.

This basic concept of a ‘2-phase bundled data protocol’ can be expanded to build FIFO structures known as micropipelines. Sutherland’s approach describes the use of a capture-pass latch as a data storage element. Fig. 3 shows the basic structure of a capture-pass latch.

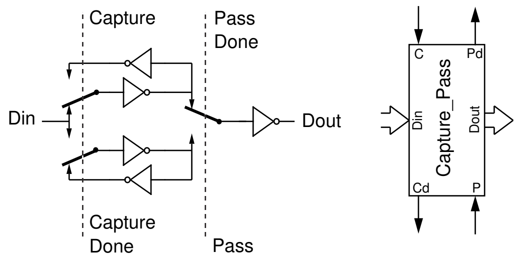

Fig. 3. Capture-pass data storage element

The capture-pass latch is transparent until an event occurs on its *Capture* line. This causes the latch to hold any data that was on its input line, *Din*, at that time. The *Capture Done* event signals that the capture operation has completed. *Dout* now represents the captured data, any change of data on *Din* will have no effect on this value. An event on *Pass* signals that the latch contents have been consumed and that the latch can return to its transparent state, ready for the next data value and input event. The event *Pass Done* signals the completion of the pass operation.

Capture-pass latch structures can be chained to form a FIFO or micropipeline structure with the use of the Muller C-gate [13] to ensure correct operation of the bundled data protocol or handshake control. The Muller C-gate acts as an AND function for events. Each input of the Muller C-gate must receive an event before an event is propagated to its output. Fig. 4 shows a basic micropipeline structure.

Here the Muller C-gates are shown with an inversion at one of their inputs. On initialisation all C-gate outputs will be zero, the inversion therefore primes the C-gate for firing on the first event received on *Rin*. When valid data is presented at *Din* an input request will be generated by the sender in the form of an event on *Rin*. This will cause the first stage to capture the data. On completion of the capture an acknowledge is returned to the sender via an event on *Ain*. The sender can now prepare the next data for the pipeline.

This event is also propagated forward down the pipeline through a delay element to the C-gate controlling the second capture-pass stage. The delay element represents any required control delay to ensure that the data is valid at the data input of the second stage prior to a capture event being issued.

As the second stage C-gate will be primed after initialisation a capture event will be generated and thus the data will be latched. On *Capture Done* the second stage latch will forward an event to the next stage and also send an event back to the preceding stage to signal that the input data can be removed. This event activates *Pass* on the first stage latch thus returning it to its transparent state and, on completion, the *Pass Done* event primes the input C-gate ready for the next input event. This process continues down the pipeline until an output request is generated at *Rout*.

Further data can be input into the pipeline; however, the pipeline will gradually fill if the data is not removed from the output and will eventually stall at the input C-gate at the first stage after four data elements have been latched. New input data at *Din* with a corresponding event on *Rin* cannot progress until the first stage capture-pass latch becomes transparent which is similarly stalled on the second stage and so on. An acknowledge event from the receiver on *Aout* signals that the data on *Dout* has been consumed thus enabling the pipeline data to progress a stage further and generate a new *Rout* event with its corresponding *Dout*. This describes the basic First-In First-Out property of micropipelines.

A. Micropipeline Implementation in the Asynchronous ARM

One of the main areas of design in the asynchronous ARM which incorporates the micropipeline technique is the 32-bit wide datapath. The datapath is a high density custom layout cell design where transistor count and density are at a premium. For this reason the building of latch circuits based on the capture-pass approach was considered too costly in area and transistor count and a more conventional latching circuit structure was chosen.

Fig. 5 shows the latching structure chosen, a pass transistor transparent latch structure, along with a proposed CMOS capture-pass latch structure. Note that this capture-pass implementation has four transmission gate structures compared

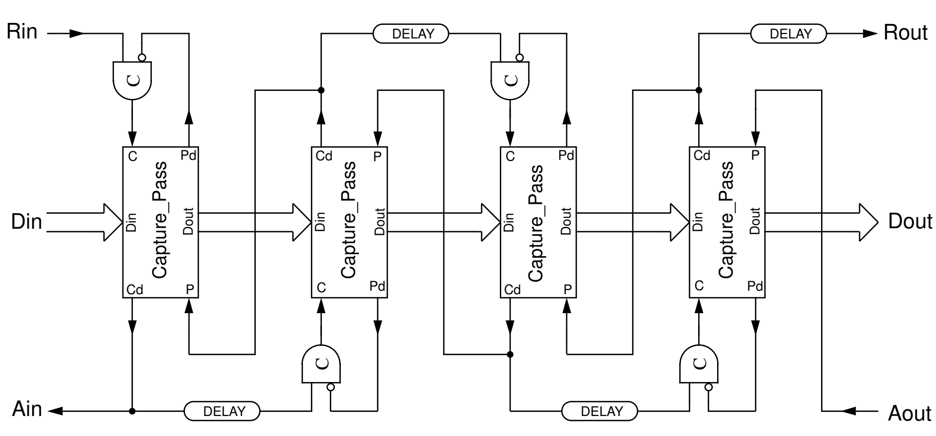

Fig. 4. Sutherland micropipeline structure

with the single instance in the conventional latch. The total gate capacitance loading on the *C*, *P*, *nC* and *nP* lines will therefore be four times that of the total gate capacitance loading on the *En* and *nEn* lines; the former will however change state only once per data transfer whereas for the conventional latch structure, two state changes are required. The conventional latch therefore offers considerable energy savings, switching half the gate capacitive load on its control wires each cycle, compared with the capture-pass latch.

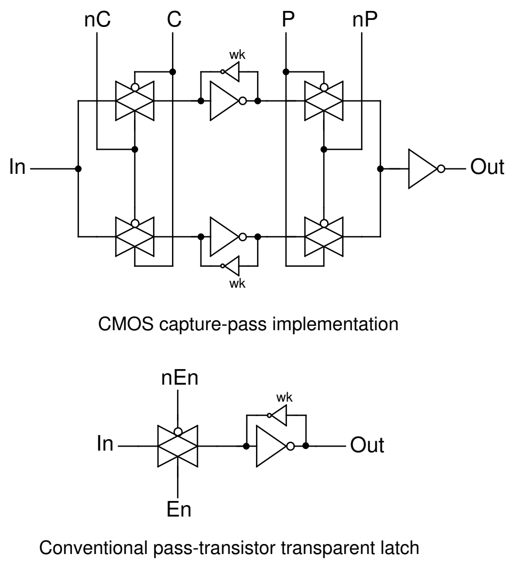

Fig. 5. Pass transistor and capture-pass latch structures

The conventional transparent latch approach leads to a very efficient 32-bit wide datapath implementation. However the required control for this latch requires a 4-phase protocol, compared to the capture-pass style approach which uses a 2-phase protocol. Therefore to meet the interface protocol required for the 2-phase transition signalling approach, extra components are required over the Sutherland micropipeline approach to implement the 2-phase to 4-phase conversions. Fig. 6 shows the micropipeline control circuit implemented on the asynchronous ARM, including the buffering circuits required to drive the full 32-bit wide datapath registers.

The 2-phase transition signal protocol is preserved with *Rin* and *Ain* performing the latch input handshake protocol and *Rout* and *Aout* performing the latch output handshake protocol. The control signals *En* and *nEn* are used to drive the latch enable lines of the 32-bit datapath registers.

This circuit introduces two new event control blocks. The exclusive-OR gate acts as a merge for events, an event on either of its inputs will generate a corresponding event on its output. The Toggle circuit acts as an event steer; after initialisation the first event is steered to its dot output, the second to the blank output, the third to the dot and so on with input events being steered to alternate outputs.

After initialisation the latch will be transparent with all event lines low. An event on *Rin* will therefore propagate through the primed C-gate, exclusive-OR and drive buffer circuitry closing the transparent latches. To sense that these latches have fully closed the *nEn* and inverted *En* lines are connected to a C-gate. When both latch control signals have changed state an event is propagated through the C-gate and Toggle generating an output request *Rout*, stating that the latch output data is now valid, and an input acknowledge *Ain*, stating that the input data can be removed. Any subsequent input requests will now be stalled by the input C-gate.

An output acknowledge on *Aout* signals that the latch data has been consumed, this event propagating through the exclusive-OR and drive buffer circuitry to open the latch. The C-gate again detects that both latch control signals have changed state, the subsequent event being steered through the Toggle to its blank output to prime the input C-gate ready for the next input request. This signals that the latch is now transparent and therefore the cycle can begin again. Fig. 7 shows a timing diagram illustrating the above.

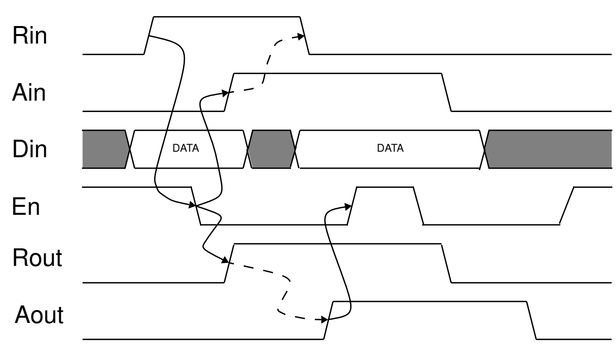

Fig. 7. Micropipeline control circuit timing diagram

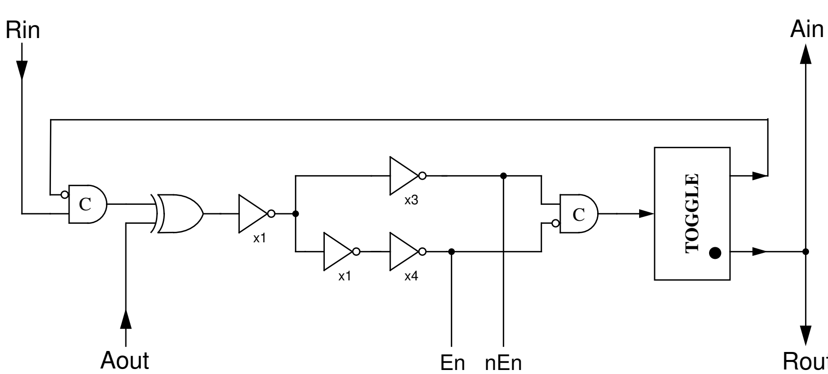

Fig. 6. Asynchronous ARM micropipeline control circuit

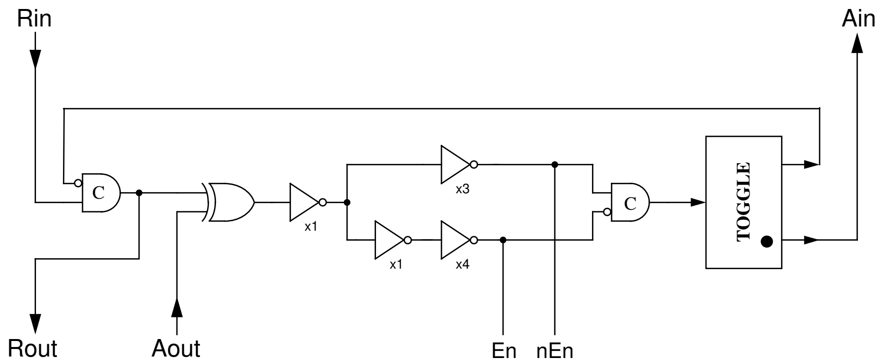

Fig. 8. Asynchronous ARM fast-forward micropipeline control circuit

Any depth of pipeline can easily be constructed by cascading these latches and control circuits.

A variation on the control circuit shown in Fig. 6 is shown in Fig. 8. Here the *Rout* signal is directly wired from the output of the input C-gate. For this circuit to meet the bundling constraint the latch data throughput must be faster than the *Rin* to *Rout* propagation time. This control circuit must therefore be used with care to ensure that this constraint is met. Note that the original control circuit shown in Fig. 6 has a large safety margin which easily meets the bundling constraint.

The benefit this circuit has over the original control circuit is that the input to output request propagation delay is faster, giving a lower latency latch circuit. The latch cycle time is also reduced though the input acknowledge event must still wait for the latch to close to ensure that the latch set-up and hold times are not violated.

B. The Asynchronous ARM Micropipeline Performance

To analyse the performance of the asynchronous ARM micropipeline control circuits SPICE [14] analyses have been performed on extracted layout from the design for worst case conditions (Vdd=4.6V, slow-slow process corner, at 100˚C temperature), the design being implemented on a 1µm, double layer metal CMOS process.

Table I shows the simulated delay through a single datapath latch element with appropriate output loading.

TABLE I
DATAPATH LATCH DELAY

Path	Delay
Data in to Data out	1.4nS

Table II shows the micropipeline control simulated delay, again with appropriate output loading and full loading capacitance for a 32-bit datapath latch on the *En* and *nEn* lines.

These results show that once valid data is presented at the latch input this data will be propagated to the latch output in 1.4nS. For a normal micropipeline control circuit the request forward propagation time is actually 9.1nS, more than 7nS behind the data for a single micropipeline stage.

TABLE II
MICROPIPELINE CONTROL DELAYS

Path	Normal Delay	Fast-Forward Delay
Rin to *Rout*	9.1nS	2.0nS
Rin to *Ain*	9.1nS	9.1nS
Aout to C-gate primed	9.3nS	9.3nS
Cycle Time	27.5nS	20.4nS

However in many cases, some form of data processing takes place between micropipeline stages and it is thus possible to “hide” some of this processing delay within the micropipeline latch control timing. Any cases where the data propagation due to data processing is longer than, or of the order of, the control forward propagation delay will require extra delay in the form of matched path elements or a delay line, to ensure adequate safety margins.

Where a pipeline has no processing between stages the fast-forward micropipeline control circuit can be used to achieve minimum latency.

The minimum cycle times for micropipelines constructed using the control circuits of Figs. 6 and 8 are also shown in Table II. The minimum cycle time possible is the sum of the forward event propagation delay (*Rin* to *Rout*), the latch delay for the next pipeline stage (*Rin* to *Ain*) and the latch recovery time to prime the C-gate ready for the next input data event (*Aout* to C-gate primed).

One of the main concerns of the present asynchronous ARM design is the cycle time of the micropipeline latch stages. A micropipeline constructed with the normal latch control stages has a cycle time under worst case simulation conditions of 27.5nS, giving an effective maximum frequency of operation of 36.4MHz. A micropipeline constructed with the fast-forward style control stages has a cycle time of 20.4nS, giving an effective maximum frequency of operation of 49.0MHz.

The performance predictions of SPICE simulations carried out on various areas of the design have shown good correlation with the performance of the actual silicon. To achieve performance goals set for future versions of the asynchronous ARM microprocessor, significant improvements in the above figures will be required.

III. Single-Phase Transparent Latch Structures

From the control circuits shown in Figs. 6 and 8 an improvement in cycle time could be achieved by removing the C-gate which detects the changes on both latch enable wires and connecting the *nEn* signal directly to the Toggle. Although this is not a purely "delay-insensitive" implementation, as the Toggle may possibly be activated before the *En* line has fully switched, one could argue that safety margins would be adequately met.

The use of single phase latches would legitimately remove the need for this C-gate and also simplify the drive buffer circuits as only one phase would be required. The use of single-phase latch designs has been extolled by Yuan and Svensson [15] whose true single-phase transparent latch structures have been shown to produce high-speed CMOS latch designs with clock speeds of the order of 200MHz [16]. Fig. 9 shows static versions of Yuan and Svensson's true single-phase transparent latch structures.

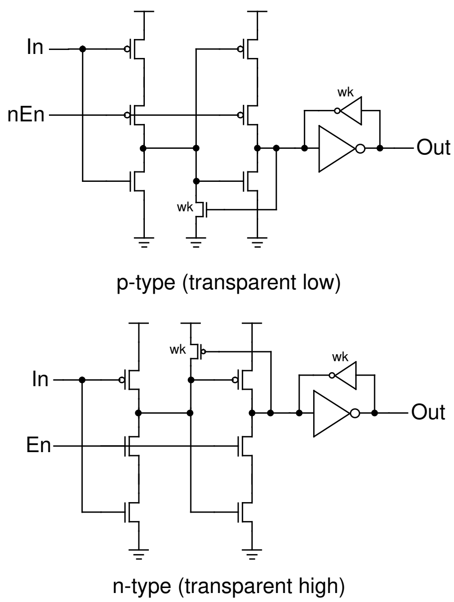

Fig. 9. Single-phase static transparent latch structures

Two versions of single-phase latch structures are shown; a p-type latch, which is transparent when *nEn* is low, and an n-type latch, which is transparent when *En* is high. When transparent, input data will propagate through the latch structures to their outputs. When the latch control switches, for example *nEn* goes high for the p-type latch, input data flow to the inverter and weak feedback data retention circuit is disabled by the double input stack and data is stored. Further changes on the input signal will therefore have no effect on the stored data.

A weak pull-down transistor is connected to the data input node of the second transistor stack on the p-type latch. When this stack is disabled in the latched state this transistor prevents this node from drifting high and spuriously changing the latched data state by turning on the n-type pull-down transistor in the second transistor stack.

The weak pull-up transistor in the n-type latch performs a similar function, preventing the second stack data input node from drifting low.

Compared to the conventional pass-transistor latch shown in Fig. 5 the single-phase transparent latches have a greater transistor component count (almost double). For comparison, the n-type single-phase transparent latch has been laid out within the asynchronous ARM datapath pitch, using minimum size transistors on the latch enable line and identical size transistors for the inverter output driver as used in the asynchronous ARM datapath pass-transistor latches. Table III shows a comparison of the capacitive loading of the latch control lines for a 32-bit wide datapath register.

TABLE III
Latch Control Capacitive Loading For 32-Bit Datapath Register

Latch	Control	Capacitance
Pass Transistor Latch	*En* *nEn*	0.93pF 1.15pF
Single-Phase Latch	*En*	1.04pF

The single-phase transparent latch structure has half the effective control load capacitance of its pass-transistor equivalent on its latch enable line, due to the removal of the requirement of complementary lines and use of minimum size transistors. For the pass-transistor latch circuit, any further reduction in the size of the pass circuit transistors will result in increased edge times on its internal node, thus decreasing the circuit performance. This reduction in capacitance will have significance for the energy consumption of the two latch styles as will be shown later.

Table IV shows comparative worst case SPICE simulated results for the two latch types with similar output load. As expected the single-phase transparent latch has a slower data throughput than that of the pass-transistor latch design.

TABLE IV
Comparison of Latch Data Propagation Delays

Path	Delay
Pass Transistor Latch: Data in to Data out	1.4nS
Single-Phase Latch: Data in to Data out	3.7nS

By comparing the data route node capacitance (ignoring intermediate nodes on stacks) of the two latch styles an approximation of their respective energy consumption can be calculated. Summing the nodal capacitance for the pass-transistor latch circuit gives a total capacitance of 0.31pF compared with a total nodal capacitance of 0.40pF for the

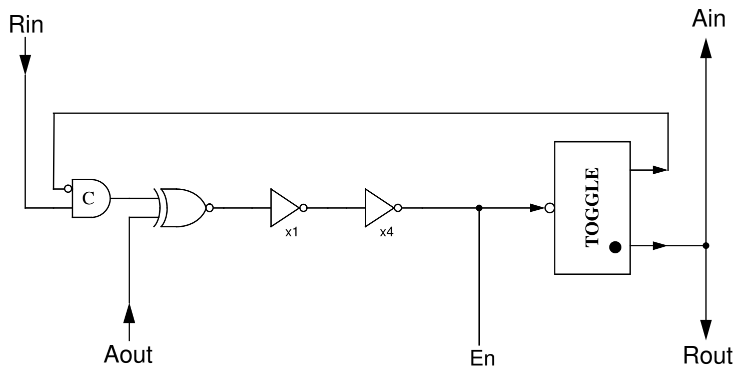

Fig. 10. Single-phase latch micropipeline control circuit

single-phase transparent latch (ignoring external output loads). This figure suggests that replacing pass-transistor transparent latches with single-phase transparent latches on the asynchronous ARM datapath will result in a 30% increase in dynamic power related to datapath data flow based on the $^1/_2CV^2$ formula.

We have already seen that a 32-bit wide datapath constructed using single-phase transparent latches has effectively half the capacitive control loading of its pass-transistor equivalent. If we consider the energy for latching a single 32-bit data value where all bits change state, with the latch then returning to a transparent state, at a supply voltage of 5V, we obtain the following;

Pass Transistor Latch:-

latch control	= 52pJ
datapath	= 124pJ
total	= 176pJ

Single-Phase Latch:-

latch control	= 26pJ
datapath	= 160pJ
total	= 186pJ

This assumption gives remarkably similar total figures for the two latch styles. However, it is highly unlikely that all data bits will change for every data value as data flows through the datapath pipelines. Assuming that, on average, half the data bits will change gives energy figures of 106pJ for the single-phase latch structures compared with 114pJ for the pass-transistor structure tilting the balance in energy saving in favour of single-phase latch structures. This figure may well be further improved on for a particular application, for example in an incrementing loop only one bit will toggle in 50% of all data changes.

A. Micropipeline Implementation for Single-Phase Transparent Latch Structures

The use of single-phase transparent latches for datapath registers greatly simplifies the required control circuit. Fig. 10 shows the control circuit for the n-type single-phase transparent latch style.

On initialisation the latch will be transparent with *En* high. An inversion is applied to the input of the Toggle circuit to correct the polarity of the first, and subsequent, events. This inversion is hidden within the Toggle circuit design and produces no extra delay over a non-inverted input Toggle design.

The circuit operates in the same way as that shown in Fig. 6, but benefits in speed due to the simplified driver buffer circuit and removal of the C-gate. A fast-forward version can again be implemented by wiring *Rout* directly from the output of the input C-gate, however, the single-phase transparent latch data delay is more than twice that of its pass transistor equivalent and some form of delay line may be required to meet the bundled data constraint.

SPICE simulation results under worst case conditions of extracted layout of the single-phase micropipeline control circuit are shown in Table V, with appropriate output loading and full loading for a 32-bit single-phase datapath latch on the *En* line included.

TABLE V
SINGLE-PHASE LATCH MICROPIPELINE CONTROL DELAYS

Path	Normal Delay	Fast-Forward Delay
Rin to *Rout*	6.3nS	*4.0nS*
Rin to *Ain*	6.3nS	6.3nS
Aout to C-gate primed	6.5nS	6.5nS
Cycle Time	19.1nS	16.8nS

The results show estimated values for a fast-forward version of the single-phase latch micropipeline control with a request in to request out figure of 4nS, which is of the order of the data in to data out delay of the n-type single-phase transparent latch design.

These figures show a considerable improvement over those shown in Table II. Both request forward propagation time and cycle time for the normal pipeline structure have been reduced by 30%, using single-phase latch structures.

The simplified control circuit also has the added benefit of being more energy efficient as there are now fewer nodes to toggle as data flows through the pipeline control circuitry.

IV. 4-PHASE MICROPIPELINE CONTROL

So far, and also in all design areas of the asynchronous

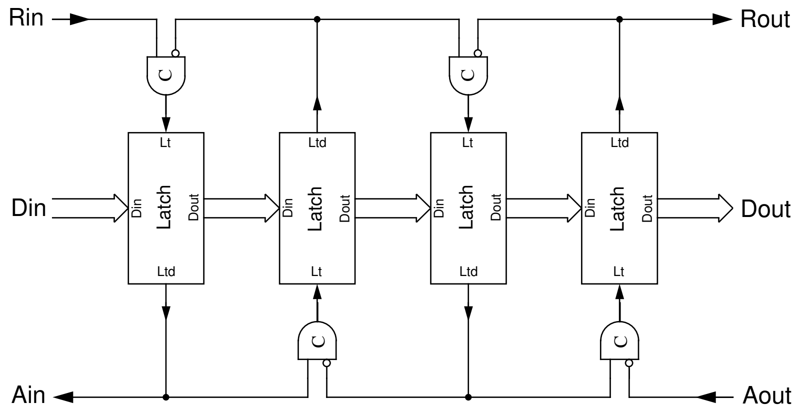

Fig. 11. Simple 4-phase micropipeline structure

ARM chip, all asynchronous control has been based on a 2-phase, bundled data, transition signalling interface protocol, whether the asynchronous block elements be a micropipeline latch stage or a 32-bit by 32-bit multiplier. To construct efficient latch structures for datapath design some form of 2-phase to 4-phase conversion is required for their control, with a corresponding 4-phase to 2-phase conversion to return back to the 2-phase transition signalling interface. The exclusive-OR gate and Toggle, shown in Figs. 6, 8 and 10, basically perform this 2-phase to 4-phase and 4-phase to 2-phase conversion, respectively.

The use of a 4-phase data bundled interface would remove the need for these elements. The 4-phase protocol however requires a reset phase where control signals must return to zero. This reset phase may affect performance and also make the control at the asynchronous interface more difficult to follow than the 2-phase transition interface, which can be considered somewhat cleaner.

Other asynchronous design methodologies such as that adopted by van Berkel [17] use only 4-phase handshake protocols and are based on fully delay-insensitive approaches.

A. 4-Phase Micropipeline Control Circuit

Fig. 11 shows a simple 4-phase micropipeline circuit. Here C-gates are used to drive the latch control signal, *Lt*, of a single-phase transparent latch directly. The signal, *Ltd*, is used to detect latch completion.

On initialisation all latch stages will be transparent with all control signals low. When *Din* is valid a request will be generated on the pipeline input with *Rin* going high. The first stage C-gate will be primed after initialisation and thus the latch signal, *Lt*, will go high latching the input data.

After latch completion, *Ltd* will go high, this being steered to the second stage C-gate and back to the sender via *Ain*. This signals that the data has been latched and can be removed, and that the reset phase, returning *Rin* to zero, can begin.

The second stage will then latch as the first, with the second stage latch completion signal being forwarded to the next stage C-gate and being fed back to the first stage input C-gate. This will rendezvous with the resetting of *Rin*, setting the first latch stage transparent and on completion, resetting *Ain* low and completing the 4-phase handshake protocol.

This simple 4-phase micropipeline circuit behaves in a very similar manner to that of the Sutherland micropipeline circuit shown in Fig. 4. However this circuit has the following restriction; a latch can only be occupied if the following adjacent latch is transparent. Comparing the component count with that shown in Fig. 10 this would at first appear to be a small sacrifice, in the simple 4-phase micropipeline each stage requires only a C-gate and inverter buffer to drive a 32-bit datapath latch constructed of the n-type single-phase transparent latches shown in Fig. 9.

However, if the pipeline backlogs, which will eventually occur if data is fed into the pipeline at a greater rate than data is removed, this property will mean that only every other pipeline stage can be occupied, effectively halving the pipeline depth. Therefore to obtain the equivalent pipeline depth to that of a 2-phase transition signal controlled micropipeline, the number of pipeline stages would need to be doubled, doubling the area occupied, pipeline latency and cycle time.

This simple micropipeline approach is therefore undesirable for the micropipeline structures used in the asynchronous ARM. The following section looks at the design of a micropipeline control circuit which, although not fully delay-insensitive, could be used under certain engineering constraints to build micropipelines with a 4-phase bundled data interface. This implementation allows the simultaneous occupation of adjacent pipeline stages thus allowing full occupation of all pipeline stages when a pipeline becomes backlogged.

B. Optimised 4-Phase Micropipeline Control Circuit

Fig. 12 shows an optimised 4-phase micropipeline control circuit which overcomes the problem described above of the simple micropipeline circuit of Fig. 11.

The control circuit shown has been designed to enable direct comparison with the 2-phase transition signal circuit shown in Fig. 10 and therefore has enough drive capability for a 32-bit datapath latch constructed using n-type single-phase transparent latches. Note that the acknowledge signals, *nAin* and *nAout*, are inverted. This circuit uses C-gates with unbalanced input stacks to provide the required interlocks for the 4-phase control to enable adjacent latch stages to hold data simultaneously. Fig. 13 shows the transistor circuits for gates C1 and C2.

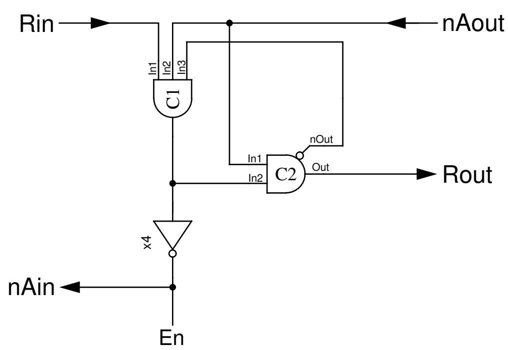

Fig. 12. Optimised 4-phase micropipeline control circuit

The C1 gate has a reset low signal, *Cdn*, this being required for initialisation. This gate has the following production rules:-

`IF` $\overline{In1}\bullet\overline{In2}\bullet\overline{In3}$ `THEN` *Out* → low,
`ELSE IF` *In1* `THEN` *Out* → high,
`ELSE` no change in *Out*.

Similarly the production rules for C2 are:-

`IF` $\overline{In2}$ `THEN` *Out* → low,
`ELSE IF` $In1\bullet In2$ `THEN` *Out* → high,
`ELSE` no change in *Out*.

The C2 output signal, *nOut*, provides an early inversion of *Out*. From the above production rule it can be seen that reset is not required for C2 as on initialisation of C1, *In2* of C2 will be reset low forcing *Out* of C2 low, as required.

Therefore after initialisation the optimised 4-phase control circuit will have *Rin* and *Rout* low, *nAin* and *nAout* high, and *En* high, meaning the corresponding datapath latch circuit will be transparent.

A valid request to the control circuit will be signalled by *Rin* going high. This will fire the gate C1, force *En* low thus closing the data latch and generate an acknowledge back to the sender, signalled by *nAin* going low. The output of C1 going high will also fire C2, which has been primed by *nAout* being high. This then generates a valid request out, *Rout* going high, signalling that data is ready to the next pipeline stage.

The firing of *Rout* then primes the C1 gate by setting *In3* low, this interlock will be explained later. C1 then acts as a rendezvous for the resetting of the input request, *Rin*, and a valid output acknowledge, *nAout* going low, which signals that the latch data has been consumed by the following stage. After this rendezvous the latch is reset to its transparent state with *En* going high, *nAin* is reset high and *Rout* reset low. The input stage is therefore now ready to accept new input data requests, when *Rin* goes high the next latching action will occur.

New data can therefore reside in the latch stage even though the output acknowledge signal, *nAout*, has yet to reset thus allowing adjacent latch stages to hold data. A valid *Rout* cannot be generated until *nAout* returns high due to the interlocks in C2. If the interlock between *Rout* and C1 did not exist then in this condition it would be possible for input requests to carry on cycling at the pipeline input, thus losing data. This interlock ensures that a valid *Rout* must be generated, as well as a valid *nAout*, before C1 can be reset and thus making the latch transparent.

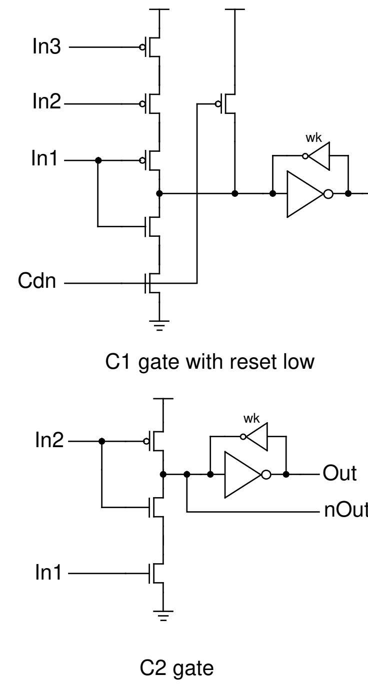

Fig. 13. Unbalanced C-gate transistor circuits

This interlock enforces a timing constraint on the latch circuit and thus this circuit cannot be deemed truly delay-insensitive. On resetting *Rout* to zero the low signal to *In3* on the gate C1 will be removed disabling the upper stack. This however must occur before *Rin* is reset after new data has been latched into the pipeline stage. If this was not the case then input data into the latch would be lost before being passed on to the next pipeline stage. It can be seen that for this to occur *Rin* must first go high, thus latching new data, and then low, that is twice around the input loop. Compared with the internal inversion in C2 from the C1 output to the C1 input stack, this is a much longer data route. Thus by keeping these components closely coupled this phenomenon can be avoided.

Fig. 14 shows the basic bundled data protocol sequence for passing one bundle of data through a 4-phase micropipeline control stage. Note that the acknowledge signals, *Ain* and *Aout*, are shown non-inverted for clarity.

C. Performance of the Optimised 4-Phase Micropipeline Control Circuit

Observing the circuits shown in Figs. 10 and 12, one obvious benefit of the 4-phase micropipeline control circuit is the reduced component count. Comparing standard cell layout for both these circuits shows that the 4-phase control circuit area is half that of its 2-phase equivalent.

Again SPICE simulations have been run on extracted lay-

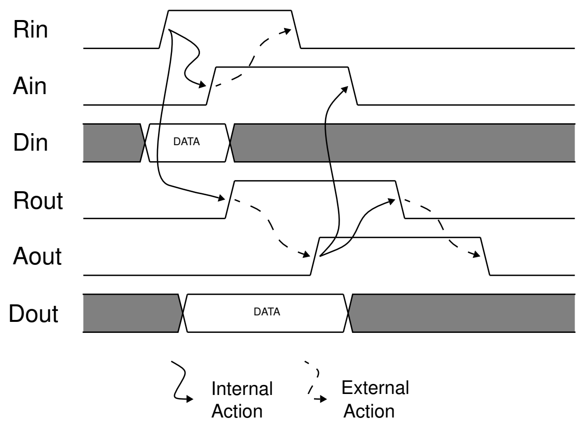

Fig. 14. 4-Phase bundled data interface protocol

out for worst case conditions. Table VI shows the timing results obtained for a 32-bit datapath latch constructed using single-phase transparent latches and with the appropriate output loading on the *nAin* and *Rout* signal wires.

TABLE VI
4-PHASE MICROPIPELINE CONTROL DELAYS

Path	Delay
Rin ↑ to Rout ↑	4.0nS
Rin ↑ to nAin ↓	3.2nS
nAout ↓ to nAin ↑	3.8nS
nAin ↑ to Rin ↑ (nAout ↑ to Rout ↑)	1.7nS
Cycle Time	12.7nS

The results show the minimum cycle time for the 4-phase micropipeline control circuit, this being the sum of the propagation times shown. Fig. 15 shows how the cycle time was derived, beginning at stage n, *Rin*, with timing figures shown for each component. The inner loop, between stage n and stage n-1, shows the reset loop which sets *Rin* low after an *nAin* acknowledge signal.

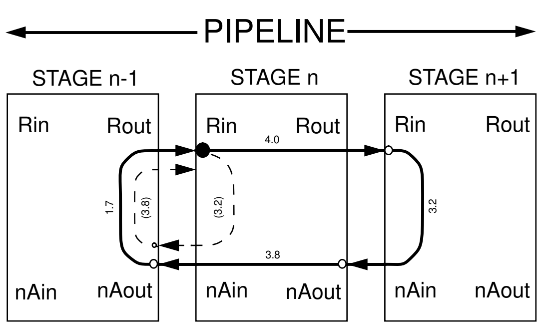

Fig. 15. 4-Phase control circuit cycle time

The figures in Table VI show a major performance improvement over those of the 2-phase micropipeline control circuit of Fig. 10 (see Table V), even though a reset phase, where all control signals must return to zero, is now required. Forward propagation of an event is of the order of the n-type single-phase transparent latch delay shown in Table IV, which is equivalent to that of a fast-forward 2-phase latch control structure. Minimum cycle time however has been reduced to just 12.7nS compared with 19.1nS and 16.8nS for the normal and fast-forward versions of the 2-phase micropipeline control circuit, respectively. It would seem therefore that the 4-phase control circuit offers greater benefits over its 2-phase equivalent in both performance and component count and area. However, here the main area of interest in asynchronous logic is in its power saving features, so energy figures for both circuits must also be compared.

By adding the total nodal capacitance switched (ignoring intermediate nodes on stacks and external loading capacitance) for the latching and passing of a single data packet, an approximate comparison of energy used in each control circuit can be made. Adding the switched node capacitance for the 2-phase micropipeline control circuit, shown in Fig. 10, gives a total switched capacitance figure of 5.20pF, for the passing of a single data packet. For the 4-phase micropipeline control circuit, shown in Fig. 12, the total switched capacitance for the passing of a single data packet is 4.21pF. This figure shows that the 4-phase micropipeline control circuit will have an energy consumption 20% lower than that of its 2-phase control equivalent.

These results suggest that the construction of a 4-phase micropipeline based on the circuit shown in Fig. 12 will have a greater performance, lower energy consumption and reduced control area and component count over an equivalent length 2-phase micropipeline based on the circuit shown in Fig. 10.

V. CONCLUSIONS

The design of the asynchronous ARM microprocessor has shown the feasibility of constructing a complex commercial microprocessor architecture using Sutherland's micropipeline approach. However, analysis of the resulting design has shown that there are many areas of this design where performance improvements can be made. This paper has considered the basic micropipeline structure employed in the asynchronous ARM and has looked at alternative design methods to see how this structure, fundamental to the whole design approach, can be improved.

The introduction of single-phase transparent latch structures, rather than the more conventional pass-transistor transparent latch approach originally adopted, has been shown to reap benefits in control speed and component count at the cost of increased latch size and slower data propagation speed. These latch structures have however been shown to have the potential for significant power savings, this being the original project aim for the design of the asynchronous ARM microprocessor.

A more fundamental design issue is raised with the comparison of performance of a 4-phase micropipeline control approach with that of a 2-phase approach. A 4-phase control circuit, optimised for single-phase transparent latch structures, was found to have improved performance, reduced component count and reduced energy consumption over its 2-phase equivalent.

This result opens up the issue of 2-phase versus 4-phase asynchronous design. The present asynchronous ARM's interfaces at asynchronous boundaries have been routinely designed in a 2-phase protocol. From the results shown here it seems likely that for future versions of the asynchronous ARM many design areas, if not all, will be predominately 4-phase.

Acknowledgments

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 ESPRIT project 7249 (the Open Microprocessor systems Initiative - Highly Optimised Reusable Nucleus Project, OMI-HORN), and the authors gratefully acknowledge this support. They are also grateful for additional specific support received during the course of this work from Acorn Computers Limited, Advanced RISC Machines Limited, VLSI Technology Limited and Compass Design Automation.

References

[1] D.E. Muller and W.S. Bartky, "A theory of asynchronous circuits," in *Proc. of an International Symposium of the Theory of Switching*, pp. 204-243, 1959.

[2] W.A. Clark, "Macromodular computer systems," *AFIPS Conference Proceedings: 1967 Spring Joint Computer Conference*, Academic Press, Atlantic City, pp. 335-336, 1967.

[3] S.H. Unger, "Asynchronous sequential switching circuits," Wiley-Interscience, New York, 1969.

[4] R.M. Kelly, "Towards a theory of universal speed-independent modules," *IEEE Trans. Comput.*, vol. 32, no. 1, pp. 21-33, June 1974.

[5] C.L. Seitz, "System timing," Chapter 7, in *Introduction to VLSI Systems*, C. Mead and L. Conway, Addison-Wesley, 1980.

[6] G. Gopalakrishnan and P. Jain, "Some recent asynchronous system design methodologies," *Technical Report Number UU-CD-TR-90-016*, Dept. Computer Science, University of Utah, Oct. 1990.

[7] S.B. Furber, "VLSI RISC architecture and organisation," Marcel Dekker, New York, 1989.

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

[9] S.B. Furber, P. Day, J.D. Garside, N.C. Paver, J.V. Woods, "AMULET1: A micropipelined ARM," *Proc. IEEE Comput. Conf.*, March 1994.

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

[11] S.B. Furber, P. Day, J.D. Garside, N.C. Paver, S. Temple, J.V. Woods, "The design and evaluation of an asynchronous microprocessor," to be presented at the *IEEE International Conf. on Computer Design*, Oct. 1994.

[12] M. Rem, "The nature of delay-insensitive computing," *{IV} Higher Order Workshop, Banff 1990*, Springer-Verlag, pp. 105-122, 1991.

[13] R.E. Miller, "Sequential circuits," Chapter 10, in *Switching Theory*, Vol. 2, Wiley, New York, 1965.

[14] 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.

[15] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *IEEE J. Solid-State Circuits*, vol. 24, no. 1, pp. 62-70, Feb. 1989.

[16] D.W. Dobberpuhl et al., "A 200-MHz 64-b dual-issue CMOS microprocessor", *IEEE J. Solid-State Circuits*, vol. 27, no. 11, pp. 1555-1565, Nov. 1992.

[17] C.H. van Berkel., M. Rem, R.W.J.J. Saejis, "VLSI programming," in *Proc. of ICCD'88*, pp. 152-156, 1988.