and of that from other members of the project particularly Kees van Berkel of Philips Research and Ad Peeters of Eindhoven University and to the ACID-WG. They are also grateful for additional support and advice received from the AMULET1 design team: Steve Furber, Nigel Paver, Paul Day, Jim Garside and Viv Woods.

10: References

[Beni94]Benini L., Favalli M., Ricco B., Analysis of hazard contributions to power dissipation in CMOS ICs. 1994 International Workshop on Low Power Design, NAPA Valley, April 1994.

[Chan92]Chandrakasan A.P., Sheng S., Brodersen R.W., Low Power CMOS Digital Design. IEEE Journal of Solid-State Circuits, Vol. 27-4,April 1992.

[Day94]Day P., Investigations into Micropipeline Latch Design Styles, submitted to IEEE Transactions on VLSI Systems, 1994.

[Dobb92]Dobberpuhl D. et al,
A 200 MHz 64b Dual-Issue CMOS Microprocessor.
IEEE Journal of Solid-State Circuits, 27(11):155-1565, November, 1992.

[Eshr93]Eshraghian K., Weste N. Principles of CMOS Design A Systems Perspective. Second Edition. Addison-Wesley, Wokingham, England, 1993.

[Farn93]Farnsworth C., Micropipeline approach to design examples. ACID-WG workshop on Digital Signal Processing, Barcelona, 1993.

[Farn94]Farnsworth C., Low power Implementations of an I^2C I/O Expander. M.Sc. Thesis, May 1994, University of Manchester.

[Furb93]Furber S.B., Day P., Garside J.D., Paver N.C., Woods J.V., "A Micropipelined ARM." Proceedings of VLSI '93, Grenoble, France, September 1993, best paper award.

[Gars91]Garside J.D. An Asynchronous Adder Design. Internal Amulet Group document, Manchester, June, 1991.

[Gars93]Garside J.D. CMOS VLSI Implementaion of an Asynchronous ALU. Proceedings of the IFIP working conference on Asynchronous Design Methodologies, Manchester, England, 1993.

[Gonc83]N. F. Goncalves, H. J. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures".

[Kram82]R. H. Krambeck, C. M. Lee, H. S. Law. "High-Speed Compact Circuits with CMOS", IEEE J. Solid-State Circuits, vol SC-17, pp. 614-619, June 1982.

[Liu93]Liu D., Svensson C., Trading Speed for Low Power by Choice of Supply and Threshold Voltages. IEEE Journal of Solid-State Circuits. Vol 28-1, Jan. 1993.

[Lyon93]Lyon R.F., Cost, Power, and Parallelism in Speech Signal Processing, IEEE 1993 Custom Integrated Circuits Conference. May 1993.

[Meng91]Meng T.H., Synchronization Design for Digital Systems, Kluwer International series in engineering and computer science, Kluwer Academic Publishers. 1991.

[Pave92]Paver N.C., Day P., Furber S.B., Garside J.D., and Woods J.V.,Register Locking in an Asynchronous Microprocessor.Proceedings of ICCD '92, pp 351-355, October 1992.

[Pave94]Paver N.C., The Design and Implementation of an Asynchronous Microprocessor. PHd thesis, May 1994, University of Manchester.

[Suth89]Sutherland I.E., Micropipelines. Communications of the ACM. 32(6):720-738, January, 1989.

[vBer88]van Berkel C.H., Rem M., Saeijs R.W.J.J., Compilation of Communicating Processes into Delay-Insensitive Circuits. *Proceedings of ICCD*, IEEE, pp. 157-162, 1988.

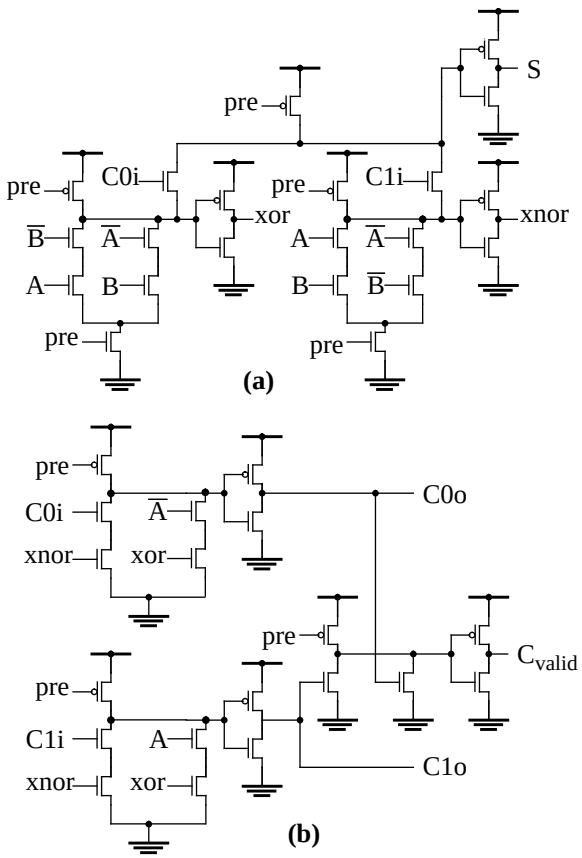

FIGURE 12 Dynamic adder
(a) sum generation (b) carry generation

The dynamic version of the adder cell in contrast relies on dual rail propagation of the carry signal. The addition begins when precharge is released allowing the sum and carry values to be evaluated. The sum generator, shown in figure 12(a), applies both the XOR and XNOR functions to *A* and *B* then conditionally raises Sum when the dual rail carry signal (*C1i* and *C0i*) arrives. The dual rail carry out signals are then generated dynamically (figure 12(b)) by either propagating the carry in, or *A*, subject to whether *A* and *B* are equal. The C_{valid} signal, (generated from the logical OR of the carry out signals), for each stage of the adder is fed into a 32 bit AND gate composed of 4 NORA 8-input AND gates fed into a 4 input domino AND gate, so as to signal completion.

7: Results

All measurements are based on SPICE (Hspice) simulations of physical layout using ES2 1 micron, double layer metal, n-well CMOS fabrication process, for the slowest process corner (slow-slow) at 3.0V supply-voltage. The same set of random set of test vectors were applied to both designs and the energy consumed over the complete period was totalled, the results from which are shown in Table 1 below. The energy measurements include the output stage of a buffer driving the adder so the difference in input capacitance is also considered. The static and dynamic versions of the adder expend almost identical energy when performing an addition on varying inputs. The dynamic adder stage incurs a 27% penalty when the data inputs are fixed because nodes are precharged and discharged redundant to the operation requirements. However, when data inputs change and the addition operation is not required (the input data is broadcast to a number of destinations) the dynamic adder has 25% of the energy expenditure of the static adder. Furthermore, the cycle time of the 32 bit adder composed of the dynamic adder cells operates in approximately half of the time of its static counterpart (Table2). This increase in operation speed occurs as a result of the lower logic threshold of the dynamic gates and hence reduced propagation of the carry signal.

Table 1: Energy

adder cell	dynamic	static	Units
random data energy[a]	67	68	pJ
static energy[b]	22	30	pJ
broadcast energy[c]	7.3	28	pJ

a. random data inputs applied
b. addition operation performed with data inputs constant
c. random data inputs; addition operation not performed

Table 2: Cycle Time

32 bit adder	dynamic	static	Units
worse case cycle time	79	161	nsec
best case cycle time	13	23	nsec

8: Conclusions

A scheme has been presented enabling dynamic design to be utilised in asynchronous circuits in a pseudo static manner. The use of dynamic logic enforces the blocking of data flow down forks or through combinational logic stages without having to introduce another register to filter such unnecessary transitions. This is achieved by exploiting the non-transparency of the precharge phase. In addition, the load presented to data flow is reduced and spurious transitions are removed on the datapath.

9: Acknowledgements

The work here has been carried out as part of ESPIRIT project OMI-EXACT (the Open Microprocessor systems Initiative - Exploitation of Asynchronous Circuit Technologies) and the authors gratefully acknowledge this support

6: Design Examples

To examine the potential of this scheme, consider some of the alternative implementations of a logical function (figure 10). The variable which drives the logical function drives several others which may not be required for all new data written into the variable. The arbitrary logical function implemented by the gates shown in figure 10 is a.b + c.d + e.f. When data is written into the variable storing a to f, all logical functions such as that shown in figure 10(a) will be computed whether or not the function is required, even if normally opaque latches are used. Furthermore, spurious transitions may occur as a result and the load is large since the logical function is implemented twice. This circuit can be adapted by placing pass transistors or transmission gates between the variable being read and the logical function. Unfortunately, short circuit current will flow if the circuit is not initiated regularly (which cannot be guaranteed) since the output of the pass transistors or transmission gates will discharge due to leakage current. To stop this discharge occurring, a storage element, i.e. a data latch, must be added to all data inputs. In cascaded stages of combinational logic normal static logic can be used since the logical function is blocked from the input changes. Alternatively an extra input on the first gates can be used which can be implemented in complex logic. Finally the dynamic version is shown in figure 10(b), here the input capacitance is of course much lower since the p-stack has been removed. Consequently, the output drive strength of the variable can be lower. If further stages are cascaded no spurious transitions will filter through. If desired subsequent stages can also be static, for instance where XOR gates and hence inversions are required between stages.

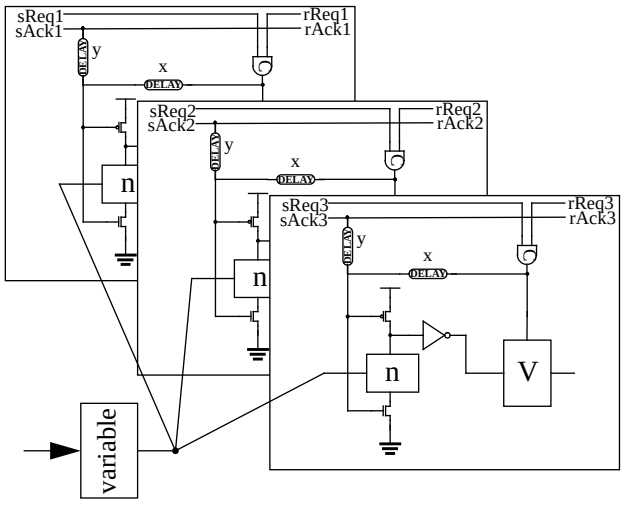

FIGURE 9 Shared variable output on 3 channels

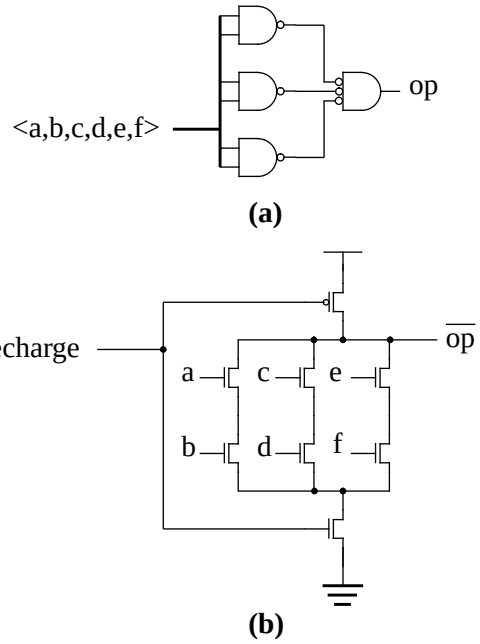

FIGURE 10 a.b + c.d + e.f
(a) static standard cell (b) dynamic

A comparison has been made between a static and a dynamic self-timed adder cell for use in a 32 bit adder. Both designs detect whether the carry signal should be propagated or generated. The static adder cell's operation [Gars91], (figure 11), is initiated by an upgoing transition on *Start*. If the carry can be generated the start signal is propagated on C_{valid} and A is output on C_{out}, otherwise $C_{invalid}$ is propagated to C_{valid} when the previous stage has evaluated. A 32-bit AND gate, constructed from a NAND/NOR tree, collects the C_{valid} signals to determine when the addition is complete. The return to zero phase of *Start* returns all C_{valid} to zero. This is subject to the same carry propagation length as the addition, unless an additional AND gate is placed between *Start* and $C_{invalid}$ to ensure C_{valid} returns to zero within a few gate propagation delays when a downgoing edge on *Start* is received.

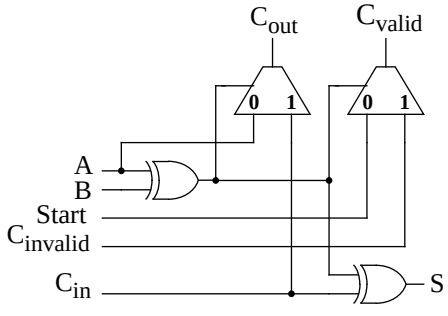

FIGURE 11 Static Adder Cell

complete, is generated by the CSP based general purpose programming language, Tangram [vBer88]. In this case Tangram ensures that a variable (storage component) cannot be read and written concurrently; the variable is first written and then read sequentially, possibly many times before another write. Furthermore, the communication action cannot be interrupted; the reading process and the process containing the variable being read, synchronise along a communication channel. If a logical function is placed between the reading process and the variable it can be applied during the communication. Again the use of 2-phase latch control can be avoided since the control signals return to zero.

Moreover, since the synchronisation of processes ensure uninterrupted communication, the logical function being applied to the read variable can be naturally implemented using dynamic logic. Figure 7 illustrates the scheme used. However, the variables are composed of a register of transparent latches and a buffer capable of driving the register enable signal with the desired edge speed. The transparent latches are normally opaque so a small speed penalty is incurred opening them. A simple domino logic gate represents the dynamic logic in figure 7 and forms part of the communication channel between the two variables. The communication protocol employed in this system is a 4-phase bundled-data protocol which is initiated, from a passive state with all control signals at zero volts, when a *sReq*$\uparrow$ (write data stable request) and a *rReq*$\uparrow$ (read request) have both been received. At this point, the latches inside the right hand variable are driven transparent. A *delay x* later, the dynamic logic enters its evaluation stage (the data stored in the left hand variable has been signalled as stable). A *delay y* later, acknowledgements are returned to the reading process by *rAck*$\uparrow$ and to the process supplying the source data by *sAck*$\uparrow$. Upon return of the down going edges on both *rReq* and *sReq*, the data is stored in the right hand variable and a *delay x*, later the dynamic logic is returned to precharge. Finally, the communication is complete by returning acknowledges *sAck*$\downarrow$ and *rAck*$\downarrow$ to both processes. *Delay x* is required to ensure that the evaluated data is stored before the dynamic logic returns to its precharge phase (after the latch hold time has elapsed) and *delay y* ensures that incorrect data cannot be stored by the early return of *rReq*$\downarrow$ and *sReq*$\downarrow$ by delaying the transmission of *sAck*$\downarrow$ and *rAck*$\downarrow$ until the evaluation phase has completed.

To summarise the sequence of events is as follows:

sReq$\uparrow$|| *rReq*$\uparrow$; right variable transparent ; release precharge ; *delay y ;* *sAck*$\uparrow$ || *rAck*$\uparrow$; *sReq*$\downarrow$ || *rReq*$\downarrow$; store data in right variable ; *delay x* ; return logic to precharge ; *sAck*$\downarrow$ || *rAck*$\downarrow$.

Delay x can be implemented by the load presented by the right hand variable and a small buffer. *Delay y* can either be implemented by a matched path or a completion signal if the logical function is data dependent. If completion signals cannot be naturally included in the combinational function, a DCVSL gate with the outputs logically OR'ed can be used to provide completion for each stage of logic [Meng91]. However, DCVSL gates require the complementary logic function to be evaluated as well, which represents an overhead in silicon area and presents a larger load to the previous stage. Since delay y is also implemented by dynamic logic the return to zero phase occurs concurrently with the logical function's return to precharge and hence the propagation delay of the logical function is not incurred as a penalty.

As a result of the above sequence of events the data supplied by the left variable must therefore be valid when *sReq*$\uparrow$ is sent because precharge is released before the channel acknowledges receipt of the request. This data must remain valid for the complete handshake (until *sAck*$\downarrow$) to ensure the dynamic logic evaluates correctly.

The evaluating function must therefore be valid when *rAck*$\downarrow$ is sent (since this indicates when the data can be read) until the data is latched by *rReq*$\downarrow$.

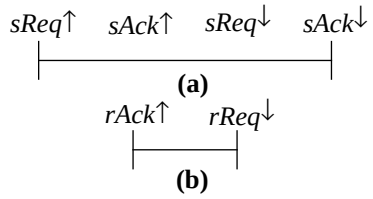

FIGURE 8 Data validity of (a) input data (b) function output

Of course, any function can be placed between two such stages with many stages of logic where required. The potential throughput of such systems is higher than their micropipeline equivalent since the latch control signal can be driven directly without any protocol conversion (this is a result of using 4-phase control rather than the use of dynamic logic).

The power saving merits of using dynamic logic in such a manner are realised when the variable on the left is read on more than one channel containing a logical function (figure 9), a common occurrence in Tangram generated programs as the language supports the sharing of hardware. In this situation each logical operation communicates along a different channel and the only shared item is the data output from the left variable. A communication on a channel will only activate the logical function which is required since the other logical functions will remain in precharge. In this situation, holding a combinational circuit in precharge is the ideal solution for minimising power consumption because the combinational logic behaves like an opaque latch. This fact and the low input capacitance, makes dynamic logic very appealing.

the logic remains in precharge until the function is required at which point the precharge is released and the function is evaluated. After evaluation is complete, the result is stored before the dynamic logic is precharged.

In micropipeline circuits, the scheme used involves holding the dynamic logic in precharge and the storage elements opaque. On arrival of a request the latch stage is driven transparent and the dynamic logic is permitted to evaluate, as the request indicates data validity at the preceding stage. Upon completion of the logical function evaluation (indicated by a matched path or a logical function completion signal), the storage components are driven opaque to store the result. Following result storage the dynamic logic can return to precharge. The stage will return to its initial state when precharge has completed and an acknowledge has been returned from the following stage indicating that the data can be released. The latch control circuit in figure 5 requires a simple modification to provide the desired control functionality, shown in figure 6. The *en* signal is delayed and used to drive the dynamic logic precharge signal allowing the logic to evaluate. The completion signal for the logic function is then used to drive the Toggle element. In this situation dynamic CMOS circuits offer a reduction in silicon area with a potential reduction in power consumption as the input capacitance of each stage has been reduced. In addition, spurious transitions have been removed. However, all outputs have to be precharged on each operation causing unnecessary transitions and hence power consumption.

Since the evaluation phase of the dynamic logic is only the length of the propagation delay through the logic and associated control circuits, the output values have very little time to leak away so power supply voltages can be reduced with similar affect to equivalent static combinational circuits. Furthermore, the short circuit current attributed to leakage current will not occur since the output node voltage level will not degrade sufficiently over such a short period. In addition the opaque latching scheme of figure 5 is enforced, ensuring that the rippling of unnecessary transitions will be localised between stages. Unfortunately, performance (particularly latency) is lost since the latches and dynamic logic are not transparent.

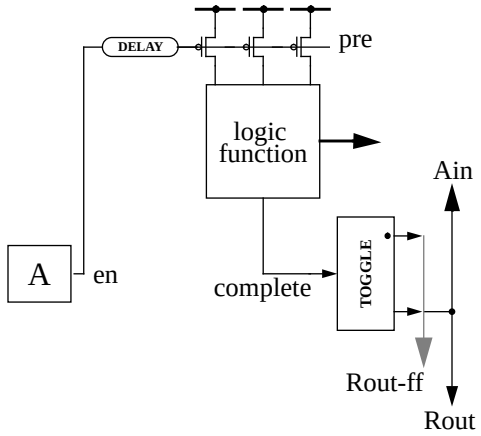

FIGURE 6 Micropipeline control with dynamic logic

5: 4-Phase Control and Dynamic Logic

4-phase bundled-data control can be used to improve the performance of micropipeline circuits whilst retaining a compact silicon area by using level sensitive latches [Day94]. In the scheme suggested, the latch control circuit of a micropipeline stage is not required since the communication signals return to zero and can therefore be used to drive the latch control signals directly. A significant improvement in performance is expected since the XOR gate, Toggle element and 2 Muller C-elements required at each stage for latch control (figure 4 and figure 5), are replaced by two asymmetric Muller C-elements. Again, the latches are transparent on initialisation.

If dynamic logic is used between latch stages of a 4-phase micropipeline, the precharge signal can be driven by the *Rout* signal of the preceding stage. However, the extra transistor (the shaded transistor in figure 1a) is required to ensure that the dynamic output node does not lose its logical value since there is no way of determining whether the following stage will complete its handshake and therefore allow *Rout* to return the dynamic logic to precharge. If dynamic logic is used in these micropipelines, the blocking action whilst in precharge will guarantee that transitions on the datapath will not filter down forks which have not been requested.

A scheme which guarantees that the handshake will

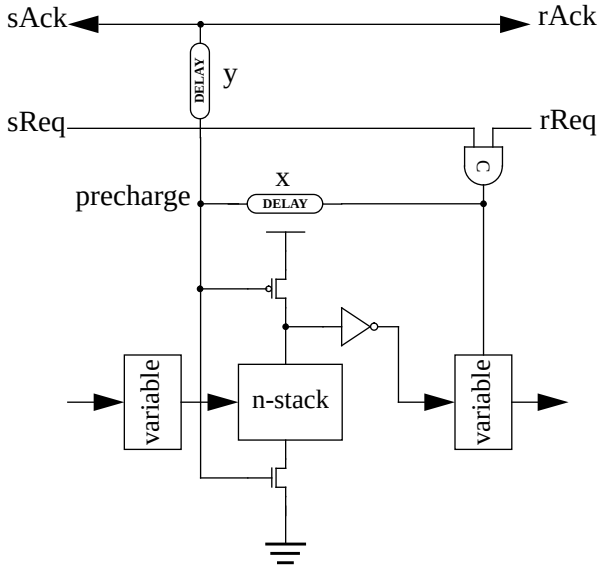

FIGURE 7 4-phase control of dynamic logic.

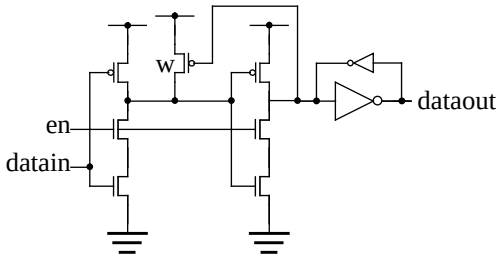

FIGURE 3 Micropipeline latches.

is driven by *en* which are driven by a power buffer (A). The register is initially transparent (*en* = '1'). When a request event is received on *Rin*, the register loads the data on its inputs. An event on *Ain* is sent acknowledging the receipt of data from the previous stage. *Rout* is also sent to the next stage indicating that data is available. When an event is returned from the next stage on *Aout* indicating that the following stage has finished with the data, the register is returned to transparent and the stage becomes receptive to another request on *Rin*. Since the register is normally transparent *Rout* can be sent forward earlier on *Rout-ff*.

Since the register is normally transparent, data transients filter down the datapath through empty pipeline stages causing unnecessary power loss. If the datapath forks into two paths for speculative evaluation later, dramatic increases in power consumption can occur [Farn94] since data in one of the paths will be discarded.

A further latch control circuit has been developed (figure 5), which compromises the latency of the micropipeline, but filters out the static and dynamic hazards arising in the combinational circuit between adjacent stages. Operation of the circuit starts with an event on *Rin* which momentarily makes the data latches transparent, subsequently causing the toggle dot output to fire *Rout-ff*. An event on *Rout-ff* loads the register and releases the previous stage. An event on *Aout* allows the next *Rin* event to be accepted. Unfortunately, the release of *Ain* and hence the previous stage is delayed, although *Rout* (*Rout-ff*) may be sent early when the register becomes transparent. The

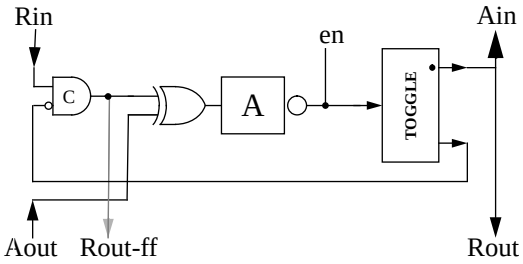

FIGURE 4 Transparent latch micropipeline control

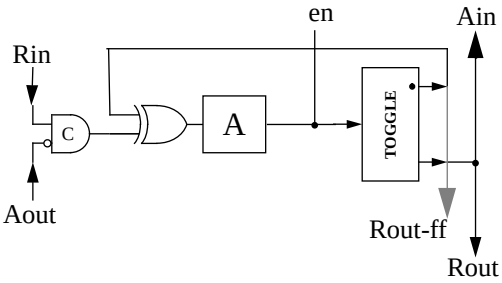

FIGURE 5 Opaque latch micropipeline control

latency of a micropipeline using this type of latch control circuit is therefore greater and the throughput less than that of a micropipeline using the normally open latch control circuit of figure 4.

It should be noted that latching schemes used in synchronous design filter out computation transitions thereby localising redundant transitions. For instance, D-Type edge-triggered flip-flops ensure that data hazards never pass through more than one latching stage. The same situation applies in a system employing a two phase non-overlapping clock or a single phase clock: the latches are never open together, preventing data transitions racing through consecutive latch stages. This implies that by using normally transparent latches, some of the potential advantages of micropipeline design as a low power strategy are lost since the power consumption incurred due to data movement may not be request driven. The power consumption will of course be specific to the application and its state.

As mentioned earlier, combinational logic can be introduced into the micropipeline FIFO by adding the required logic between the latch stages. These combinational circuits can be designed in a similar way to those used in synchronous circuits since they both conform to the bounded-delay model. Delays, matched paths and/or completion signals are used to provide sufficient set-up and hold time. Like all other asynchronous design techniques, micropipelines can exploit average case rather than worse case operation in data dependent computations such as addition. As a consequence, greater composability (even technology migration) can be supported and the additional redundancy employed to reduce the worst case propagation delay in synchronous circuits (e.g carry look-ahead) is not required.

If the static combinational logic is replaced by dynamic logic, the combinational logic can be considered as part of the input latch stage. Unlike a synchronous design, there is no clock which can be guaranteed to arrive to return the dynamic logic to precharge. There is therefore a danger that leakage current will degrade the voltage on the dynamic output nodes. This problem may be solved if dynamic logic is used in a pseudo static manner, whereby

stage of latches or combinational logic and therefore extra power is consumed. If the dynamic logic function is implemented in the n-stack, a reduction of over 60% can be expected during evaluation.

(4) Precharge: The major disadvantage of dynamic circuits is the cost of the precharge phase, particularly since the clock must drive the precharge signal. All nodes that were discharged during the previous evaluation are precharged, and may only be discharged again on the following evaluation phase. The unnecessary transitions of course represent a significant overhead, particularly for logic functions with a high probability of discharging and for sets of data which cause a large number of discharges. When the output node does not discharge the load presented by the precharge transistor represents redundant power-burning. In synchronous systems if clock gating is not employed, this expensive precharge cycle, will occur in all dynamic circuits every clock cycle.

(5) Clock gating and clock frequency management: Clock gating and frequency management are techniques that are used to reduce the power consumption in synchronous circuits [Lyon93] by disabling idle circuits and reducing performance when work loads are low. In static circuits, the effect of extended clock periods or periods without clock signals has no effect since the logic output is always a function of its inputs. However, dynamic circuits rely on the clock to maintain the integrity of their output nodes during the evaluation phase. If the clock frequency is not sufficiently high, the output nodes will gradually discharge because of leakage current, causing the logical output to be lost. To combat this problem, additional transistors (1 per gate, the shaded transistor shown in figure 1a) are required [Eshr93].

4: Micropipelines and Dynamic Logic

In the design of the AMULET1 asynchronous microprocessor [Furb93, Pave94] based on Sutherland's micropipelines, dynamic logic was exploited in three key areas – the ALU [Gars93], some of the finite state machines used to control the device and the register bank [Pave92]. To understand this use of dynamic logic, a brief introduction to micropipeline design techniques is required with particular emphasis on the datapath.

Micropipelines are event driven, elastic pipeline structures devised by Ivan Sutherland. The throughput of a pipeline is the rate at which results emerge from the pipeline and the latency is the time it takes for an individual result to emerge. The communication protocol employed is the 2-phase bundled-data convention.

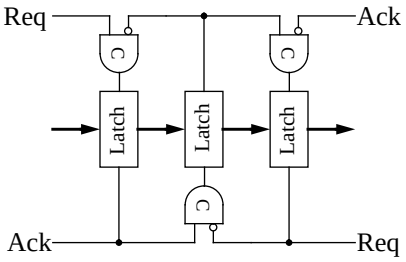

FIGURE 2 A micropipeline FIFO.

The elegance of a micropipeline can be demonstrated when considering the design of a FIFO (figure 2). In the synchronous framework if different clocks are used at each end, arbitration or synchronisation is required as the phase relationship between clocks is unknown. In contrast, the local communication protocol inside a micropipeline, ensures that a stage captures data when the next stage has accepted the previous data and the previous stage has offered new data. This removes the need for arbitration or synchronisation since timing is not taken directly from input or output. The FIFO can be adapted into a pipeline simply by adding combinational logic between the latch stages. With this approach the local communication can be considered a way of generating local clocks where synchronisation of the clocks is controlled by the micropipeline structure [Farn93].

Different latch implementations and control circuits may be used to optimise parameters such as silicon area, power consumption and cycle time between stages. In [Suth89], two alternative capture-pass latch structures were introduced, which capture data when an event (transition) is received. Both are normally transparent and store data upon reception of a request event. Unfortunately these latches are large compared to latches used in synchronous design since two storage elements are required to respond to events. Consequently, the single phase, dynamic latches [Yuan89] used in the DEC Alpha microprocessor [Dobb92] have been adapted for static operation (figure 3) [Day94] and are used in the design examples presented later in this paper. These latches only contain one storage element and are transparent when en = 1. All future references to data latches are to these transparent latches.

Since the latch in figure 3 has level sensitive control, the latch control signal requires conversion from a two-phase protocol to a four-phase protocol. Several such schemes have been developed with different merits. The first, shown in figure 4, is an adaptation of the latch control suggested in [Suth89]. A register consisting of data latches

The success of any technique that claims to reduce power consumption will be judged on its ability to accommodate the scaling of the power supply voltage to the same extent as its competitors.

(3) Architectural refinements: The total energy expended may be reduced by eliminating unnecessary transitions in those parts of the circuit which are not active. In an asynchronous design, this functionality is inherent; distributed control activates a subcircuit only when its functionality is required. Furthermore, the nature of distributed control promotes the use of localised communication which reduces the load presented during communication. In a synchronous system, the clock frequency can be reduced where appropriate or the clock can be blocked to inactive functional units by means of clock gating. Clock gating thus achieves a similar effect to the techniques employed in asynchronous design. However, the relative timing of the clocks are different within each functional unit, hence clock skew problems are emphasised. The non-overlapping section of a two-phase clock can be used to compensate for the propagation delay of the clock gating circuit.

Both synchronous and asynchronous design are well suited to the reduction in power supply voltages as many of the cells are common to both design paradigms.

In synchronous systems, power consumption can be split into two categories, i.e. data movement and clock distribution. For the asynchronous case, power consumption is split between data movement and control. The rest of this paper will concentrate on power consumption minimisation during data movement.

3: Dynamic Logic

In static CMOS design, the logical function is implemented twice, once in the n transistor stack and once in the p transistor stack. In contrast, dynamic logic reduces circuit area by implementing compact “NMOS style” gates without the overhead of static power dissipation.

Despite this obvious attraction, dynamic logic can be difficult to use with some low power techniques in the synchronous framework, such as frequency management and/or clock gating, since data may be lost or corrupted if the logic is not clocked and thereby refreshed regularly.

The most commonly used dynamic logic styles are Domino (figure 1a) [Kram82], NORA[Gonc83] or a combination of the two (figure 1b), since cascading of stages is accommodated using one clock edge to drive the precharge signal. In Domino logic this is allowed since the

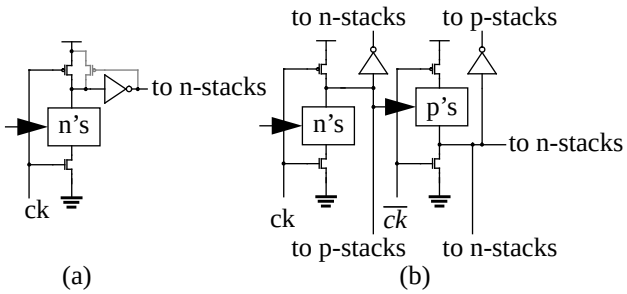

FIGURE 1 Dynamic logic (a) Domino (b) NORA

output of the inverter is driven low during precharge. In further cascaded stages the n-channel precharge transistor may therefore be omitted. However, care must be taken to ensure that precharge occurs in time to remove positive inputs between stages and hence avoid short circuit current dissipation. In NORA logic the function is implemented alternately in n and p stacks to allow inverted outputs which cannot be accommodated in Domino logic.

Chandrakasan et al. [Chan92] highlighted five areas for consideration when comparing the low power properties of dynamic and static logic:

(1) Spurious transitions: In static logic, spurious transitions (static and dynamic hazards) occur when a logic function is evaluated since the propagation delay of different gates is unequal. As a consequence, the gate may assume more than one logical value during the evaluation of the function. According to Benini et al. [Beni94], hazards contribute to between 9% and 38% of the power consumption in their set of static circuits and therefore cannot be ignored. In contrast, dynamic circuit output nodes either remain at the value to which they were precharged (ignoring charge leakage) or are discharged. Consequently, dynamic circuits may potentially save power during logic evaluation since their output nodes make at most one transition during evaluation.

(2) Short-circuit current: In static CMOS combinational circuits, a short circuit current will always flow when the supply voltage is greater than $V_{tn} + |V_{tp}|$, since the logic function of the gate is effectively implemented twice, once in the n stack and its complement in the p stack. (During a change in output, both stacks simultaneously conduct.) In dynamic circuits, the short circuit current will only flow if the node was discharged during the evaluation phase

(3) Input capacitance: The duplication of the logic function in both the n-stack and p-stack in static logic presents an additional load to the previous

Utilising Dynamic Logic for Low Power Consumption in Asynchronous Circuits

C. Farnsworth, D.A. Edwards and S.S. Sikand

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

Abstract

Dynamic logic offers compact, fast solutions for synchronous design. Asynchronous design methodologies which conform to the bounded-delay model can also utilise dynamic logic for combinational circuits obtaining similar benefits to the synchronous case. To achieve these benefits, the logic is held in precharge until it is required and the evaluation phase is completed during a handshake communication action. The resultant power consumption is low since the input capacitance is far smaller than equivalent static CMOS circuits and spurious transitions in the computation are removed.

1: Introduction

Power consumption is becoming an increasingly important criterion in digital design, particularly as more portable products (e.g. laptop and pen based computers) are realised which require compact, low power implementations with relatively high processing abilities. As a consequence, low power techniques are emerging to increase the battery life. In addition, the manufacturing cost and size of a product are reduced since the power supply and cooling requirements may be diminished.

In CMOS circuits, the majority of the power consumption is due to switching activity. In synchronous systems the switching action occurs on the active clock edge regardless of whether the circuit operation is required, whereas asynchronous digital design techniques can reduce the number of unnecessary switching actions because operation is requested only when it is required.

Dynamic logic can also offer potential power consumption savings since the gates are smaller than their static counterparts. However, in spite of this, static gates are often adopted when low power consumption is one of the primary design goals, since the clock has to be distributed to all dynamic logic gates. Furthermore, if clock frequency management or clock gating techniques are adopted, measures must be taken to avoid dynamic nodes discharging.

This paper outlines how asynchronous circuits can exploit the advantages as well as minimise the disadvantages of dynamic logic for low power consumption. Indeed, the precharging of dynamic nodes which is generally considered as a disadvantage may in fact be an advantage in asynchronous circuits.

2: Low Power

Generally, power consumption can be reduced by three main techniques [Lyon93]:

(1) Reducing nodal capacitance: The majority of power consumed in CMOS circuits is caused by dynamic charging and discharging of circuit nodes. The energy expended at each node is:

$$energy = \frac{1}{2} \times C \times \Delta V^2$$

where:

ΔV represents the change in voltage, generally Vdd,

C represents the nodal capacitance.

The total power consumed is the energy expended at all nodes over a specific time period. If nodal capacitance is reduced, an associated reduction in power consumption occurs. The other major source of power consumption in CMOS circuits is the switching current resulting from simultaneous conduction of both the n and p type transistors.

(2) Reducing power supply voltage: The energy expended at each node is proportional to V^2, hence the reduction in power supply voltage can be applied twice to the power consumption. Without major alterations to current processes, 3.3V has emerged as the new industry standard.

and of that from other members of the project particularly Kees van Berkel of Philips Research and Ad Peeters of Eindhoven University and to the ACID-WG. They are also grateful for additional support and advice received from the AMULET1 design team: Steve Furber, Nigel Paver, Paul Day, Jim Garside and Viv Woods.

10: References

[Beni94]Benini L., Favalli M., Ricco B., Analysis of hazard contributions to power dissipation in CMOS ICs. 1994 International Workshop on Low Power Design, NAPA Valley, April 1994.

[Chan92]Chandrakasan A.P., Sheng S., Brodersen R.W., Low Power CMOS Digital Design. IEEE Journal of Solid-State Circuits, Vol. 27-4,April 1992.

[Day94]Day P., Investigations into Micropipeline Latch Design Styles, submitted to IEEE Transactions on VLSI Systems, 1994.

[Dobb92]Dobberpuhl D. et al,
A 200 MHz 64b Dual-Issue CMOS Microprocessor.
IEEE Journal of Solid-State Circuits, 27(11):155-1565, November, 1992.

[Eshr93]Eshraghian K., Weste N. Principles of CMOS Design A Systems Perspective. Second Edition. Addison-Wesley, Wokingham, England, 1993.

[Farn93]Farnsworth C., Micropipeline approach to design examples. ACID-WG workshop on Digital Signal Processing, Barcelona, 1993.

[Farn94]Farnsworth C., Low power Implementations of an I^2C I/O Expander. M.Sc. Thesis, May 1994, University of Manchester.

[Furb93]Furber S.B., Day P., Garside J.D., Paver N.C., Woods J.V., "A Micropipelined ARM." Proceedings of VLSI '93, Grenoble, France, September 1993, best paper award.

[Gars91]Garside J.D. An Asynchronous Adder Design. Internal Amulet Group document, Manchester, June, 1991.

[Gars93]Garside J.D. CMOS VLSI Implementaion of an Asynchronous ALU. Proceedings of the IFIP working conference on Asynchronous Design Methodologies, Manchester, England, 1993.

[Gonc83]N. F. Goncalves, H. J. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures".

[Kram82]R. H. Krambeck, C. M. Lee, H. S. Law. "High-Speed Compact Circuits with CMOS", IEEE J. Solid-State Circuits, vol SC-17, pp. 614-619, June 1982.

[Liu93]Liu D., Svensson C., Trading Speed for Low Power by Choice of Supply and Threshold Voltages. IEEE Journal of Solid-State Circuits. Vol 28-1, Jan. 1993.

[Lyon93]Lyon R.F., Cost, Power, and Parallelism in Speech Signal Processing, IEEE 1993 Custom Integrated Circuits Conference. May 1993.

[Meng91]Meng T.H., Synchronization Design for Digital Systems, Kluwer International series in engineering and computer science, Kluwer Academic Publishers. 1991.

[Pave92]Paver N.C., Day P., Furber S.B., Garside J.D., and Woods J.V.,Register Locking in an Asynchronous Microprocessor.Proceedings of ICCD '92, pp 351-355, October 1992.

[Pave94]Paver N.C., The Design and Implementation of an Asynchronous Microprocessor. PHd thesis, May 1994, University of Manchester.

[Suth89]Sutherland I.E., Micropipelines. Communications of the ACM. 32(6):720-738, January, 1989.

[vBer88]van Berkel C.H., Rem M., Saeijs R.W.J.J., Compilation of Communicating Processes into Delay-Insensitive Circuits. *Proceedings of ICCD*, IEEE, pp. 157-162, 1988.

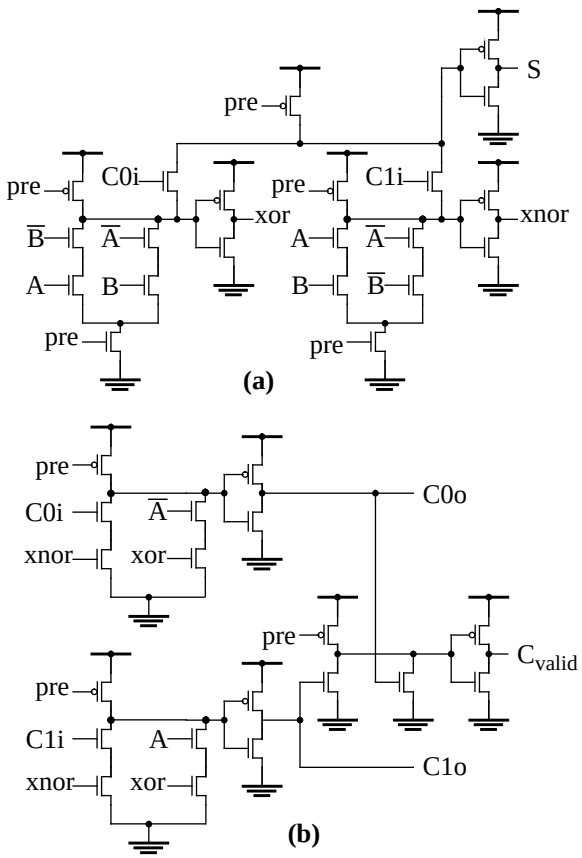

FIGURE 12 Dynamic adder
(a) sum generation (b) carry generation

The dynamic version of the adder cell in contrast relies on dual rail propagation of the carry signal. The addition begins when precharge is released allowing the sum and carry values to be evaluated. The sum generator, shown in figure 12(a), applies both the XOR and XNOR functions to *A* and *B* then conditionally raises Sum when the dual rail carry signal (*C1i* and *C0i*) arrives. The dual rail carry out signals are then generated dynamically (figure 12(b)) by either propagating the carry in, or *A*, subject to whether *A* and *B* are equal. The C_{valid} signal, (generated from the logical OR of the carry out signals), for each stage of the adder is fed into a 32 bit AND gate composed of 4 NORA 8-input AND gates fed into a 4 input domino AND gate, so as to signal completion.

7: Results

All measurements are based on SPICE (Hspice) simulations of physical layout using ES2 1 micron, double layer metal, n-well CMOS fabrication process, for the slowest process corner (slow-slow) at 3.0V supply-voltage. The same set of random set of test vectors were applied to both designs and the energy consumed over the complete period was totalled, the results from which are shown in Table 1 below. The energy measurements include the output stage of a buffer driving the adder so the difference in input capacitance is also considered. The static and dynamic versions of the adder expend almost identical energy when performing an addition on varying inputs. The dynamic adder stage incurs a 27% penalty when the data inputs are fixed because nodes are precharged and discharged redundant to the operation requirements. However, when data inputs change and the addition operation is not required (the input data is broadcast to a number of destinations) the dynamic adder has 25% of the energy expenditure of the static adder. Furthermore, the cycle time of the 32 bit adder composed of the dynamic adder cells operates in approximately half of the time of its static counterpart (Table2). This increase in operation speed occurs as a result of the lower logic threshold of the dynamic gates and hence reduced propagation of the carry signal.

Table 1: Energy

adder cell	dynamic	static	Units
random data energy[a]	67	68	pJ
static energy[b]	22	30	pJ
broadcast energy[c]	7.3	28	pJ

a. random data inputs applied
b. addition operation performed with data inputs constant
c. random data inputs; addition operation not performed

Table 2: Cycle Time

32 bit adder	dynamic	static	Units
worse case cycle time	79	161	nsec
best case cycle time	13	23	nsec

8: Conclusions

A scheme has been presented enabling dynamic design to be utilised in asynchronous circuits in a pseudo static manner. The use of dynamic logic enforces the blocking of data flow down forks or through combinational logic stages without having to introduce another register to filter such unnecessary transitions. This is achieved by exploiting the non-transparency of the precharge phase. In addition, the load presented to data flow is reduced and spurious transitions are removed on the datapath.

9: Acknowledgements

The work here has been carried out as part of ESPIRIT project OMI-EXACT (the Open Microprocessor systems Initiative - Exploitation of Asynchronous Circuit Technologies) and the authors gratefully acknowledge this support

6: Design Examples

To examine the potential of this scheme, consider some of the alternative implementations of a logical function (figure 10). The variable which drives the logical function drives several others which may not be required for all new data written into the variable. The arbitrary logical function implemented by the gates shown in figure 10 is a.b + c.d + e.f. When data is written into the variable storing a to f, all logical functions such as that shown in figure 10(a) will be computed whether or not the function is required, even if normally opaque latches are used. Furthermore, spurious transitions may occur as a result and the load is large since the logical function is implemented twice. This circuit can be adapted by placing pass transistors or transmission gates between the variable being read and the logical function. Unfortunately, short circuit current will flow if the circuit is not initiated regularly (which cannot be guaranteed) since the output of the pass transistors or transmission gates will discharge due to leakage current. To stop this discharge occurring, a storage element, i.e. a data latch, must be added to all data inputs. In cascaded stages of combinational logic normal static logic can be used since the logical function is blocked from the input changes. Alternatively an extra input on the first gates can be used which can be implemented in complex logic. Finally the dynamic version is shown in figure 10(b), here the input capacitance is of course much lower since the p-stack has been removed. Consequently, the output drive strength of the variable can be lower. If further stages are cascaded no spurious transitions will filter through. If desired subsequent stages can also be static, for instance where XOR gates and hence inversions are required between stages.

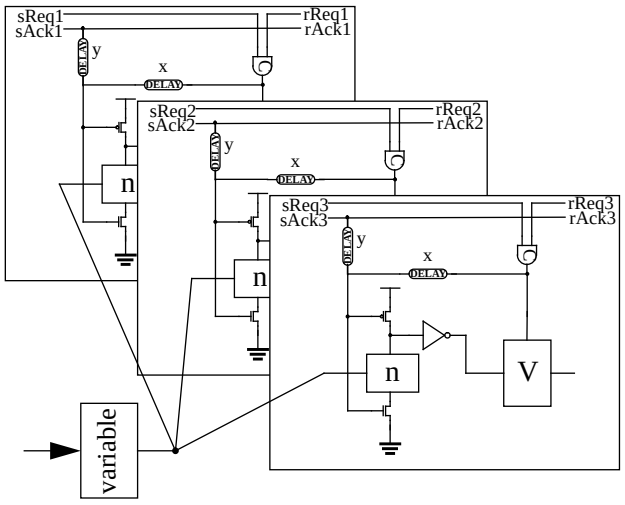

FIGURE 9 Shared variable output on 3 channels

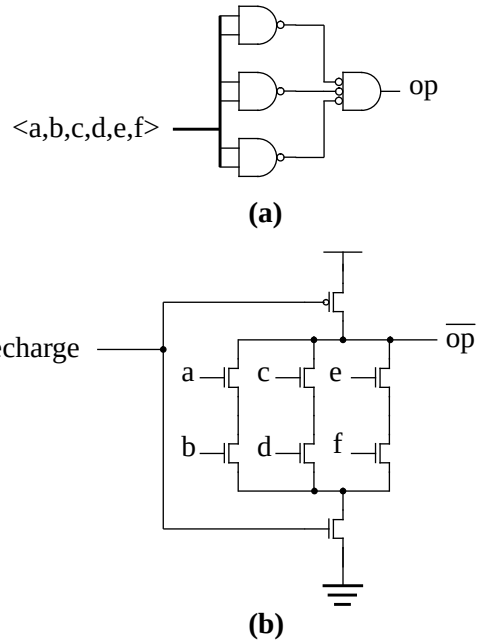

FIGURE 10 a.b + c.d + e.f
(a) static standard cell (b) dynamic

A comparison has been made between a static and a dynamic self-timed adder cell for use in a 32 bit adder. Both designs detect whether the carry signal should be propagated or generated. The static adder cell's operation [Gars91], (figure 11), is initiated by an upgoing transition on *Start*. If the carry can be generated the start signal is propagated on C_{valid} and A is output on C_{out}, otherwise $C_{invalid}$ is propagated to C_{valid} when the previous stage has evaluated. A 32-bit AND gate, constructed from a NAND/NOR tree, collects the C_{valid} signals to determine when the addition is complete. The return to zero phase of *Start* returns all C_{valid} to zero. This is subject to the same carry propagation length as the addition, unless an additional AND gate is placed between *Start* and $C_{invalid}$ to ensure C_{valid} returns to zero within a few gate propagation delays when a downgoing edge on *Start* is received.

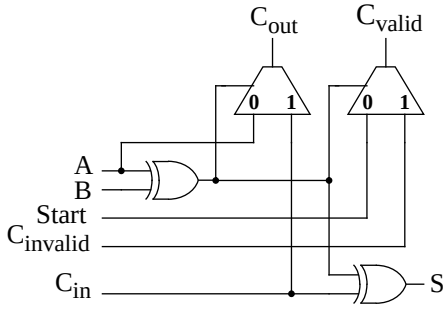

FIGURE 11 Static Adder Cell

complete, is generated by the CSP based general purpose programming language, Tangram [vBer88]. In this case Tangram ensures that a variable (storage component) cannot be read and written concurrently; the variable is first written and then read sequentially, possibly many times before another write. Furthermore, the communication action cannot be interrupted; the reading process and the process containing the variable being read, synchronise along a communication channel. If a logical function is placed between the reading process and the variable it can be applied during the communication. Again the use of 2-phase latch control can be avoided since the control signals return to zero.

Moreover, since the synchronisation of processes ensure uninterrupted communication, the logical function being applied to the read variable can be naturally implemented using dynamic logic. Figure 7 illustrates the scheme used. However, the variables are composed of a register of transparent latches and a buffer capable of driving the register enable signal with the desired edge speed. The transparent latches are normally opaque so a small speed penalty is incurred opening them. A simple domino logic gate represents the dynamic logic in figure 7 and forms part of the communication channel between the two variables. The communication protocol employed in this system is a 4-phase bundled-data protocol which is initiated, from a passive state with all control signals at zero volts, when a *sReq*$\uparrow$ (write data stable request) and a *rReq*$\uparrow$ (read request) have both been received. At this point, the latches inside the right hand variable are driven transparent. A *delay x* later, the dynamic logic enters its evaluation stage (the data stored in the left hand variable has been signalled as stable). A *delay y* later, acknowledgements are returned to the reading process by *rAck*$\uparrow$ and to the process supplying the source data by *sAck*$\uparrow$. Upon return of the down going edges on both *rReq* and *sReq*, the data is stored in the right hand variable and a *delay x*, later the dynamic logic is returned to precharge. Finally, the communication is complete by returning acknowledges *sAck*$\downarrow$ and *rAck*$\downarrow$ to both processes. *Delay x* is required to ensure that the evaluated data is stored before the dynamic logic returns to its precharge phase (after the latch hold time has elapsed) and *delay y* ensures that incorrect data cannot be stored by the early return of *rReq*$\downarrow$ and *sReq*$\downarrow$ by delaying the transmission of *sAck*$\downarrow$ and *rAck*$\downarrow$ until the evaluation phase has completed.

To summarise the sequence of events is as follows:

sReq$\uparrow$|| *rReq*$\uparrow$; right variable transparent ; release precharge ; *delay y ;* *sAck*$\uparrow$ || *rAck*$\uparrow$; *sReq*$\downarrow$ || *rReq*$\downarrow$; store data in right variable ; *delay x* ; return logic to precharge ; *sAck*$\downarrow$ || *rAck*$\downarrow$.

Delay x can be implemented by the load presented by the right hand variable and a small buffer. *Delay y* can either be implemented by a matched path or a completion signal if the logical function is data dependent. If completion signals cannot be naturally included in the combinational function, a DCVSL gate with the outputs logically OR'ed can be used to provide completion for each stage of logic [Meng91]. However, DCVSL gates require the complementary logic function to be evaluated as well, which represents an overhead in silicon area and presents a larger load to the previous stage. Since delay y is also implemented by dynamic logic the return to zero phase occurs concurrently with the logical function's return to precharge and hence the propagation delay of the logical function is not incurred as a penalty.

As a result of the above sequence of events the data supplied by the left variable must therefore be valid when *sReq*$\uparrow$ is sent because precharge is released before the channel acknowledges receipt of the request. This data must remain valid for the complete handshake (until *sAck*$\downarrow$) to ensure the dynamic logic evaluates correctly.

The evaluating function must therefore be valid when *rAck*$\downarrow$ is sent (since this indicates when the data can be read) until the data is latched by *rReq*$\downarrow$.

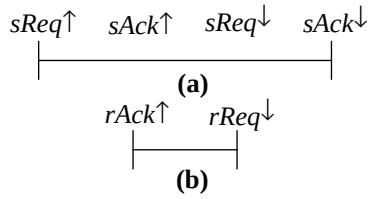

FIGURE 8 Data validity of (a) input data (b) function output

Of course, any function can be placed between two such stages with many stages of logic where required. The potential throughput of such systems is higher than their micropipeline equivalent since the latch control signal can be driven directly without any protocol conversion (this is a result of using 4-phase control rather than the use of dynamic logic).

The power saving merits of using dynamic logic in such a manner are realised when the variable on the left is read on more than one channel containing a logical function (figure 9), a common occurrence in Tangram generated programs as the language supports the sharing of hardware. In this situation each logical operation communicates along a different channel and the only shared item is the data output from the left variable. A communication on a channel will only activate the logical function which is required since the other logical functions will remain in precharge. In this situation, holding a combinational circuit in precharge is the ideal solution for minimising power consumption because the combinational logic behaves like an opaque latch. This fact and the low input capacitance, makes dynamic logic very appealing.

the logic remains in precharge until the function is required at which point the precharge is released and the function is evaluated. After evaluation is complete, the result is stored before the dynamic logic is precharged.

In micropipeline circuits, the scheme used involves holding the dynamic logic in precharge and the storage elements opaque. On arrival of a request the latch stage is driven transparent and the dynamic logic is permitted to evaluate, as the request indicates data validity at the preceding stage. Upon completion of the logical function evaluation (indicated by a matched path or a logical function completion signal), the storage components are driven opaque to store the result. Following result storage the dynamic logic can return to precharge. The stage will return to its initial state when precharge has completed and an acknowledge has been returned from the following stage indicating that the data can be released. The latch control circuit in figure 5 requires a simple modification to provide the desired control functionality, shown in figure 6. The *en* signal is delayed and used to drive the dynamic logic precharge signal allowing the logic to evaluate. The completion signal for the logic function is then used to drive the Toggle element. In this situation dynamic CMOS circuits offer a reduction in silicon area with a potential reduction in power consumption as the input capacitance of each stage has been reduced. In addition, spurious transitions have been removed. However, all outputs have to be precharged on each operation causing unnecessary transitions and hence power consumption.

Since the evaluation phase of the dynamic logic is only the length of the propagation delay through the logic and associated control circuits, the output values have very little time to leak away so power supply voltages can be reduced with similar affect to equivalent static combinational circuits. Furthermore, the short circuit current attributed to leakage current will not occur since the output node voltage level will not degrade sufficiently over such a short period. In addition the opaque latching scheme of figure 5 is enforced, ensuring that the rippling of unnecessary transitions will be localised between stages. Unfortunately, performance (particularly latency) is lost since the latches and dynamic logic are not transparent.

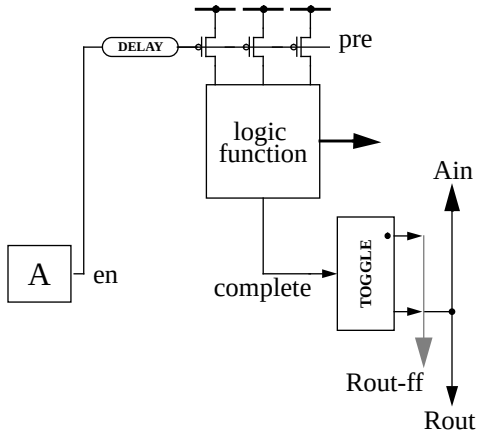

FIGURE 6 Micropipeline control with dynamic logic

5: 4-Phase Control and Dynamic Logic

4-phase bundled-data control can be used to improve the performance of micropipeline circuits whilst retaining a compact silicon area by using level sensitive latches [Day94]. In the scheme suggested, the latch control circuit of a micropipeline stage is not required since the communication signals return to zero and can therefore be used to drive the latch control signals directly. A significant improvement in performance is expected since the XOR gate, Toggle element and 2 Muller C-elements required at each stage for latch control (figure 4 and figure 5), are replaced by two asymmetric Muller C-elements. Again, the latches are transparent on initialisation.

If dynamic logic is used between latch stages of a 4-phase micropipeline, the precharge signal can be driven by the *Rout* signal of the preceding stage. However, the extra transistor (the shaded transistor in figure 1a) is required to ensure that the dynamic output node does not lose its logical value since there is no way of determining whether the following stage will complete its handshake and therefore allow *Rout* to return the dynamic logic to precharge. If dynamic logic is used in these micropipelines, the blocking action whilst in precharge will guarantee that transitions on the datapath will not filter down forks which have not been requested.

A scheme which guarantees that the handshake will

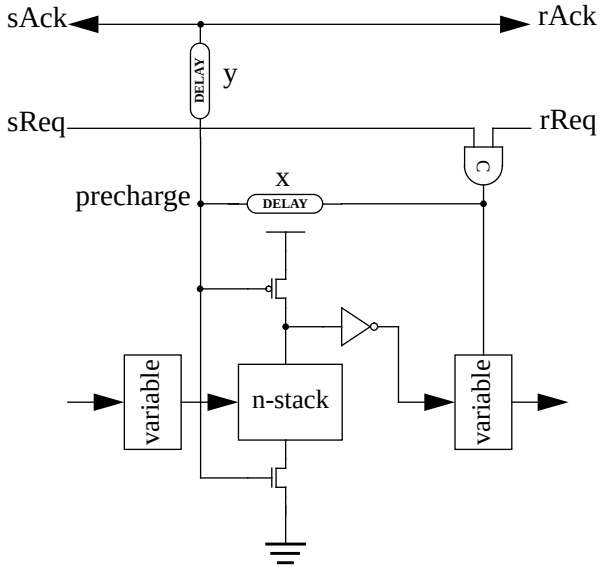

FIGURE 7 4-phase control of dynamic logic.

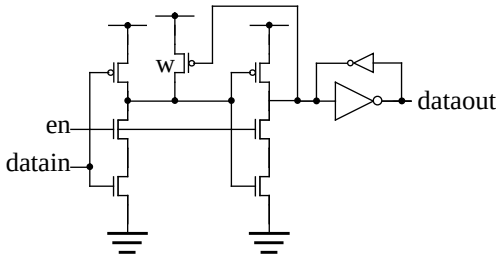

FIGURE 3 Micropipeline latches.

is driven by *en* which are driven by a power buffer (A). The register is initially transparent (*en* = '1'). When a request event is received on *Rin*, the register loads the data on its inputs. An event on *Ain* is sent acknowledging the receipt of data from the previous stage. *Rout* is also sent to the next stage indicating that data is available. When an event is returned from the next stage on *Aout* indicating that the following stage has finished with the data, the register is returned to transparent and the stage becomes receptive to another request on *Rin*. Since the register is normally transparent *Rout* can be sent forward earlier on *Rout-ff*.

Since the register is normally transparent, data transients filter down the datapath through empty pipeline stages causing unnecessary power loss. If the datapath forks into two paths for speculative evaluation later, dramatic increases in power consumption can occur [Farn94] since data in one of the paths will be discarded.

A further latch control circuit has been developed (figure 5), which compromises the latency of the micropipeline, but filters out the static and dynamic hazards arising in the combinational circuit between adjacent stages. Operation of the circuit starts with an event on *Rin* which momentarily makes the data latches transparent, subsequently causing the toggle dot output to fire *Rout-ff*. An event on *Rout-ff* loads the register and releases the previous stage. An event on *Aout* allows the next *Rin* event to be accepted. Unfortunately, the release of *Ain* and hence the previous stage is delayed, although *Rout* (*Rout-ff*) may be sent early when the register becomes transparent. The

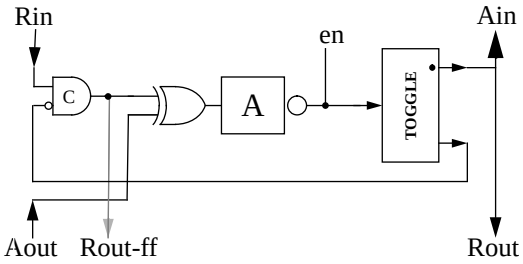

FIGURE 4 Transparent latch micropipeline control

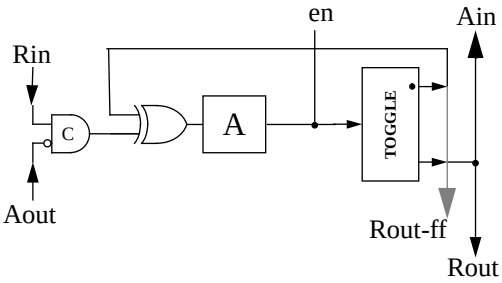

FIGURE 5 Opaque latch micropipeline control

latency of a micropipeline using this type of latch control circuit is therefore greater and the throughput less than that of a micropipeline using the normally open latch control circuit of figure 4.

It should be noted that latching schemes used in synchronous design filter out computation transitions thereby localising redundant transitions. For instance, D-Type edge-triggered flip-flops ensure that data hazards never pass through more than one latching stage. The same situation applies in a system employing a two phase non-overlapping clock or a single phase clock: the latches are never open together, preventing data transitions racing through consecutive latch stages. This implies that by using normally transparent latches, some of the potential advantages of micropipeline design as a low power strategy are lost since the power consumption incurred due to data movement may not be request driven. The power consumption will of course be specific to the application and its state.

As mentioned earlier, combinational logic can be introduced into the micropipeline FIFO by adding the required logic between the latch stages. These combinational circuits can be designed in a similar way to those used in synchronous circuits since they both conform to the bounded-delay model. Delays, matched paths and/or completion signals are used to provide sufficient set-up and hold time. Like all other asynchronous design techniques, micropipelines can exploit average case rather than worse case operation in data dependent computations such as addition. As a consequence, greater composability (even technology migration) can be supported and the additional redundancy employed to reduce the worst case propagation delay in synchronous circuits (e.g carry look-ahead) is not required.

If the static combinational logic is replaced by dynamic logic, the combinational logic can be considered as part of the input latch stage. Unlike a synchronous design, there is no clock which can be guaranteed to arrive to return the dynamic logic to precharge. There is therefore a danger that leakage current will degrade the voltage on the dynamic output nodes. This problem may be solved if dynamic logic is used in a pseudo static manner, whereby

stage of latches or combinational logic and therefore extra power is consumed. If the dynamic logic function is implemented in the n-stack, a reduction of over 60% can be expected during evaluation.

(4) Precharge: The major disadvantage of dynamic circuits is the cost of the precharge phase, particularly since the clock must drive the precharge signal. All nodes that were discharged during the previous evaluation are precharged, and may only be discharged again on the following evaluation phase. The unnecessary transitions of course represent a significant overhead, particularly for logic functions with a high probability of discharging and for sets of data which cause a large number of discharges. When the output node does not discharge the load presented by the precharge transistor represents redundant power-burning. In synchronous systems if clock gating is not employed, this expensive precharge cycle, will occur in all dynamic circuits every clock cycle.

(5) Clock gating and clock frequency management: Clock gating and frequency management are techniques that are used to reduce the power consumption in synchronous circuits [Lyon93] by disabling idle circuits and reducing performance when work loads are low. In static circuits, the effect of extended clock periods or periods without clock signals has no effect since the logic output is always a function of its inputs. However, dynamic circuits rely on the clock to maintain the integrity of their output nodes during the evaluation phase. If the clock frequency is not sufficiently high, the output nodes will gradually discharge because of leakage current, causing the logical output to be lost. To combat this problem, additional transistors (1 per gate, the shaded transistor shown in figure 1a) are required [Eshr93].

4: Micropipelines and Dynamic Logic

In the design of the AMULET1 asynchronous microprocessor [Furb93, Pave94] based on Sutherland's micropipelines, dynamic logic was exploited in three key areas – the ALU [Gars93], some of the finite state machines used to control the device and the register bank [Pave92]. To understand this use of dynamic logic, a brief introduction to micropipeline design techniques is required with particular emphasis on the datapath.

Micropipelines are event driven, elastic pipeline structures devised by Ivan Sutherland. The throughput of a pipeline is the rate at which results emerge from the pipeline and the latency is the time it takes for an individual result to emerge. The communication protocol employed is the 2-phase bundled-data convention.

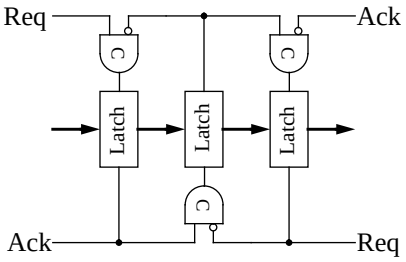

FIGURE 2 A micropipeline FIFO.

The elegance of a micropipeline can be demonstrated when considering the design of a FIFO (figure 2). In the synchronous framework if different clocks are used at each end, arbitration or synchronisation is required as the phase relationship between clocks is unknown. In contrast, the local communication protocol inside a micropipeline, ensures that a stage captures data when the next stage has accepted the previous data and the previous stage has offered new data. This removes the need for arbitration or synchronisation since timing is not taken directly from input or output. The FIFO can be adapted into a pipeline simply by adding combinational logic between the latch stages. With this approach the local communication can be considered a way of generating local clocks where synchronisation of the clocks is controlled by the micropipeline structure [Farn93].

Different latch implementations and control circuits may be used to optimise parameters such as silicon area, power consumption and cycle time between stages. In [Suth89], two alternative capture-pass latch structures were introduced, which capture data when an event (transition) is received. Both are normally transparent and store data upon reception of a request event. Unfortunately these latches are large compared to latches used in synchronous design since two storage elements are required to respond to events. Consequently, the single phase, dynamic latches [Yuan89] used in the DEC Alpha microprocessor [Dobb92] have been adapted for static operation (figure 3) [Day94] and are used in the design examples presented later in this paper. These latches only contain one storage element and are transparent when en = 1. All future references to data latches are to these transparent latches.

Since the latch in figure 3 has level sensitive control, the latch control signal requires conversion from a two-phase protocol to a four-phase protocol. Several such schemes have been developed with different merits. The first, shown in figure 4, is an adaptation of the latch control suggested in [Suth89]. A register consisting of data latches

The success of any technique that claims to reduce power consumption will be judged on its ability to accommodate the scaling of the power supply voltage to the same extent as its competitors.

(3) Architectural refinements: The total energy expended may be reduced by eliminating unnecessary transitions in those parts of the circuit which are not active. In an asynchronous design, this functionality is inherent; distributed control activates a subcircuit only when its functionality is required. Furthermore, the nature of distributed control promotes the use of localised communication which reduces the load presented during communication. In a synchronous system, the clock frequency can be reduced where appropriate or the clock can be blocked to inactive functional units by means of clock gating. Clock gating thus achieves a similar effect to the techniques employed in asynchronous design. However, the relative timing of the clocks are different within each functional unit, hence clock skew problems are emphasised. The non-overlapping section of a two-phase clock can be used to compensate for the propagation delay of the clock gating circuit.

Both synchronous and asynchronous design are well suited to the reduction in power supply voltages as many of the cells are common to both design paradigms.

In synchronous systems, power consumption can be split into two categories, i.e. data movement and clock distribution. For the asynchronous case, power consumption is split between data movement and control. The rest of this paper will concentrate on power consumption minimisation during data movement.

3: Dynamic Logic

In static CMOS design, the logical function is implemented twice, once in the n transistor stack and once in the p transistor stack. In contrast, dynamic logic reduces circuit area by implementing compact “NMOS style” gates without the overhead of static power dissipation.

Despite this obvious attraction, dynamic logic can be difficult to use with some low power techniques in the synchronous framework, such as frequency management and/or clock gating, since data may be lost or corrupted if the logic is not clocked and thereby refreshed regularly.

The most commonly used dynamic logic styles are Domino (figure 1a) [Kram82], NORA[Gonc83] or a combination of the two (figure 1b), since cascading of stages is accommodated using one clock edge to drive the precharge signal. In Domino logic this is allowed since the

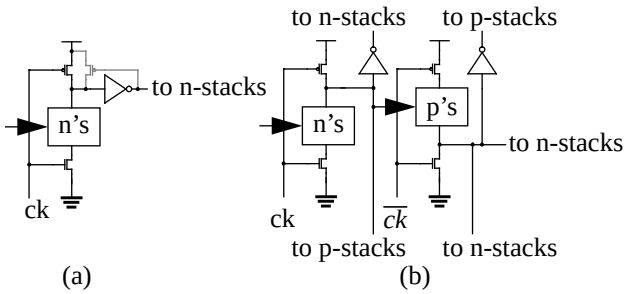

FIGURE 1 Dynamic logic (a) Domino (b) NORA

output of the inverter is driven low during precharge. In further cascaded stages the n-channel precharge transistor may therefore be omitted. However, care must be taken to ensure that precharge occurs in time to remove positive inputs between stages and hence avoid short circuit current dissipation. In NORA logic the function is implemented alternately in n and p stacks to allow inverted outputs which cannot be accommodated in Domino logic.

Chandrakasan et al. [Chan92] highlighted five areas for consideration when comparing the low power properties of dynamic and static logic:

(1) Spurious transitions: In static logic, spurious transitions (static and dynamic hazards) occur when a logic function is evaluated since the propagation delay of different gates is unequal. As a consequence, the gate may assume more than one logical value during the evaluation of the function. According to Benini et al. [Beni94], hazards contribute to between 9% and 38% of the power consumption in their set of static circuits and therefore cannot be ignored. In contrast, dynamic circuit output nodes either remain at the value to which they were precharged (ignoring charge leakage) or are discharged. Consequently, dynamic circuits may potentially save power during logic evaluation since their output nodes make at most one transition during evaluation.

(2) Short-circuit current: In static CMOS combinational circuits, a short circuit current will always flow when the supply voltage is greater than $V_{tn} + |V_{tp}|$, since the logic function of the gate is effectively implemented twice, once in the n stack and its complement in the p stack. (During a change in output, both stacks simultaneously conduct.) In dynamic circuits, the short circuit current will only flow if the node was discharged during the evaluation phase

(3) Input capacitance: The duplication of the logic function in both the n-stack and p-stack in static logic presents an additional load to the previous

Utilising Dynamic Logic for Low Power Consumption in Asynchronous Circuits

C. Farnsworth, D.A. Edwards and S.S. Sikand

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

Abstract

Dynamic logic offers compact, fast solutions for synchronous design. Asynchronous design methodologies which conform to the bounded-delay model can also utilise dynamic logic for combinational circuits obtaining similar benefits to the synchronous case. To achieve these benefits, the logic is held in precharge until it is required and the evaluation phase is completed during a handshake communication action. The resultant power consumption is low since the input capacitance is far smaller than equivalent static CMOS circuits and spurious transitions in the computation are removed.

1: Introduction

Power consumption is becoming an increasingly important criterion in digital design, particularly as more portable products (e.g. laptop and pen based computers) are realised which require compact, low power implementations with relatively high processing abilities. As a consequence, low power techniques are emerging to increase the battery life. In addition, the manufacturing cost and size of a product are reduced since the power supply and cooling requirements may be diminished.

In CMOS circuits, the majority of the power consumption is due to switching activity. In synchronous systems the switching action occurs on the active clock edge regardless of whether the circuit operation is required, whereas asynchronous digital design techniques can reduce the number of unnecessary switching actions because operation is requested only when it is required.

Dynamic logic can also offer potential power consumption savings since the gates are smaller than their static counterparts. However, in spite of this, static gates are often adopted when low power consumption is one of the primary design goals, since the clock has to be distributed to all dynamic logic gates. Furthermore, if clock frequency management or clock gating techniques are adopted, measures must be taken to avoid dynamic nodes discharging.

This paper outlines how asynchronous circuits can exploit the advantages as well as minimise the disadvantages of dynamic logic for low power consumption. Indeed, the precharging of dynamic nodes which is generally considered as a disadvantage may in fact be an advantage in asynchronous circuits.

2: Low Power

Generally, power consumption can be reduced by three main techniques [Lyon93]:

(1) Reducing nodal capacitance: The majority of power consumed in CMOS circuits is caused by dynamic charging and discharging of circuit nodes. The energy expended at each node is:

$$energy = \frac{1}{2} \times C \times \Delta V^2$$

where:

ΔV represents the change in voltage, generally Vdd,

C represents the nodal capacitance.

The total power consumed is the energy expended at all nodes over a specific time period. If nodal capacitance is reduced, an associated reduction in power consumption occurs. The other major source of power consumption in CMOS circuits is the switching current resulting from simultaneous conduction of both the n and p type transistors.

(2) Reducing power supply voltage: The energy expended at each node is proportional to V^2, hence the reduction in power supply voltage can be applied twice to the power consumption. Without major alterations to current processes, 3.3V has emerged as the new industry standard.