DIVISION OF COMPUTER SCIENCE

Potential for Asynchronous Microprocessor Design

C. J. Elston

Technical Report No.182

March 1994

Potential for asynchronous microprocessor design

C.J.Elston
Research Student
School of Information Sciences
Division of Computer Science
Technical Report

Abstract

Asynchronous (non-globally clocked) design and realisation of processors is undergoing a resurgence of interest. The difficulties of producing complex, high performance synchronous (globally clocked) processors are becoming severe and these problems will only be compounded by technological advancement. Asynchronous design has its own associated difficulties in producing functionally correct, high performance processors but there is the potential to increase performance and simplify design. This report outlines a number of promising approaches to asynchronous design giving the advantages and disadvantages of each in the context of present and future environments.

Key words: Microprocessor design, asynchronous systems, delay-insensitive circuits, speed-independent circuits, Micropipelines, design methodologies.

1 *Introduction*

The majority of digital systems are designed and realised using synchronous techniques. A synchronous system is one in which all steps in computation and communication are performed in lockstep with a global, "master" clock; global in the sense that the clock must cover the spatial extent of the system with a nominally identical signal at all points. The clock provides all sequencing and scheduling required for the correct operation of the system, determining when an operation may be started and when it may be regarded as having finished. There are problems associated with generating and distributing high frequency global clock signals; principal among these are clock skew, wire delay and power consumption [Elston94]. The problems associated with synchronous systems are becoming so severe that without the unlikely event of a dramatic change in the direction of technological advancement the future for synchronous systems looks bleak. The rate of increase of performance of synchronous systems will be impaired as the difficulties of design and fabrication become a limiting factor.

An asynchronous system is one in which there is no global clock. All scheduling and sequencing required by the system is provided by the interconnection of components of the system. A system can be constructed such that the cessation of activity in one module can initiate activity in modules that logically follow. While a synchronous system performs all operations in concert with the global clock an asynchronous system is made up of a network of components that order each others activity.

An analogy that can be employed to illustrate the differences between synchronous and asynchronous systems is that of transportation. A synchronous system is analogous to a public transport system such as a bus service. Individual buses work in lockstep with the bus station clock. Regardless of the presence or absense of passengers the bus will arrive and depart from the bus stops along the bus route at regular intervals, each journey taking a fixed length of time regardless of the load, in terms of the number of passengers, placed on the service. On the other hand an asynchronous system is analogous to a private transport system such as a cab service. Cabs do not depart at fixed intervals but only when they are carrying passengers. There is no notion of time in this system, a cab will depart depending on the availability of passengers and a journey will take the length of time required to traverse the distance from A to B. The analogy given breaks down under examination but suffices to show the essential differences between the two schemes.

2 *Asynchronous systems*

The correct operation of a digital system such as a processor can be defined by specifying an ordering, or partial ordering of operations. This abstraction results in a design in the sequence domain and simplifies the design process by disregarding the physical time metric. At some point in the design life cycle, to allow the design to be realised in silicon, the metric free sequence abstraction must be tied to the physical reality of time. The way in which the connection is made between sequence and time is what distinguishes synchronous and asynchronous systems. In a synchronous system there is a global clock signal to divide time into discrete intervals. During these intervals the next state of the system is evaluated based on the present state of the system plus control signals as additional inputs. In an asynchronous system there is no global clock and time takes on the physical reality of a continuum. There is no implicit synchronisation in the asynchronous system and explicit synchronisation must be provided in the interior of the system. The dominance of synchronous systems can be traced back to the earliest electronic computers [Turing47] and arises because of difficulties in asynchronous design. However the technological climate has changed beyond all recognition and the clocked design framework is beginning to limit the complexity and performance possibilities of new systems.

The explicit synchronisation required by an asynchronous system is imposed by the components of the system. Communication between the components of the system must be performed in such a way that the correct operation of the system is assured. Therefore in an asynchronous system sequence and time are connected inside the component modules of the system or through their interaction. The connection between sequence and time is made using signals. Signals are exchanged between modules to ensure that the correct sequence behaviour of the system is maintained. The manner in which these signals are produced and transmitted can distinguish asynchronous systems from one another.

3 *Classifying asynchronous systems*

Within the sphere of asynchronous systems there are a number of classes that attract particular attention in logic design. Two of these are speed independent systems and delay insensitive systems. Speed independent systems are those whose correct behaviour does not depend on delays in the constituent modules of the system and delay insensitive systems are those whose correct behaviour does not depend on delays in the constituent modules or delays introduced by interconnecting wires. The class of delay insensitive circuits is rather limited [Martin90]. Many common circuits are not delay insensitive and there are certain functions that cannot be realised in a delay insensitive manner. Another interesting type of asynchronous system is the micropipeline [Sutherland89] which compromises generality in the hope of obtaining high performance and design simplicity.

There are many further high level classifications used to further subdivide asynchronous circuits. For example a circuit can be data insensitive (where correct operation does not depend on the delay in the availability of data) or control insensitive (where correct operation does not depend on the delay in the availability of control). Furthermore, there are asynchronous circuits that are realised using Boolean gates [David89][Seitz80][Martin89a][Martin89b][Nordmann77] and there are asynchronous circuits that are realised using macromodules [Sutherland89][Costa93] [Ebergen91][DeGloria93][Meng89][Lam90]. A macromodule is a hand crafted module built to mimic procedure call, control flow merging, rendezvous etc. Sutherlands work on micropipelines [Sutherland89] is an example of the use of macromodules.

4 *Speed independent systems*

A circuit is speed independent when the correct operation of the circuit does not depend on any delays in the constituent modules of the circuit [Keller74]. The gate delays in the circuit are assumed to be unbounded but the delays in interconnecting wires are assumed to be negligable. Speed independent circuits are based on theoretical work done by Muller in the 1950's and 60's [Muller59][Muller63] whose work led to the control logic used on the Illiac III computer at the University of Illinois. Since the origins of speed independent circuit design technology and techniques for integrated circuit realisation have made enormous leaps forward and the assumption on which it is based, namely wire delays being negligible in comparison to component delays, is no longer valid.

Speed independent design concentrates on eliminating hazards (such as the appearance of an incorrect output either permanent or transitory) in circuits that arise from differences in the speed of operation of the components of the circuit. For example care is taken to eliminate the case where incorrect outputs from the circuit appear in the environment because the outputs from certain gates appear earlier than outputs from other gates. One way to ensure that hazards do not occur in the operation of a circuit is to use a two level AND and OR gate network and make the prime implicants of the function mutually exclusive. This approach leads to large, complex circuits when a function has many inputs.

Improvements in VLSI design allow higher speed logic gates to be incorporated into CMOS design; gate delays in advanced CMOS circuits can be below 1ns.

However, as will be shown in section 5, wire delays do not scale with technology [Seitz80]. As the scale of integration increases wire delays will become even more important and the silicon area within which the negligable wire delay assumption can be made will be further restricted. Speed independent design is an interesting design discipline but the physical size of a speed independent circuit is being continually reduced. The reduction in the area a speed independent circuit can occupy as the scale of integration increases is counterbalanced to some extent because the density of components within that area is increased.

5 *Wire Delays*

Wire delays are becoming increasingly important. The following argument illustrates the trend in the behaviour of wire delays as the scale of integration is increased.The time taken for a signal to propagate along a wire in MOS technology is not limited by "speed of light" considerations except in the very shortest wires where propagation times are negligable anyway. For the majority of wires it is the resistivity of the wire in combination with the parasitic capacitance that determines the rate at which the voltage driven onto the wire will equalise along its length. This process is governed by the diffusion equation (and will be referred to as the diffusion delay).

$$\mathrm{RC}\frac{d\mathrm{V}}{dt} = \frac{d^2\mathrm{V}}{dx^2} \qquad (1)$$

Solutions to the diffusion equation are complex but in general the time required for a transient to propagate a distance x along a wire is proportional to x^2. The quadratic diffusion delay is independent of wire width since a wider line has a lower resistance but a proportionately higher capacitance and vice versa. Wire delay is due to the combined effects of the velocity of electromagnetic wave propagation and diffusion within the wire itself.

As the physical size of an integrated circuit is reduced the resistance per unit length scales up quadratically since it is not only made narrower but also thinner to maintain the same relative surface flatness. Capacitance per unit area stays the same since reduced capacitance due to decreased width just balances increased capacitance per unit area due to thinner oxides. Thus in scaling down the physical size the diffusion delay will increase quadratically. This is in comparison with gate delays in integrated circuits where the trend is to decrease with decreasing component size. Hence as the dimensions of integrated circuits are scaled down and the scale of integration is increased the delays in wires will become increasingly important relative to gate delays. It is not so much the absolute magnitude of the delay that is important but the relative magnitudes of delays in wires and gates. When gate delays are much bigger than wire delays the speed independent assumption can be validly made. When delays in wires become comparable to delays in gates the assumption breaks down and speed independent design is no longer of use.

6 *Delay insensitive systems*

A delay insensitive system is one where the correct operation of the system does not depend on the delays in the modules that compose the system or on the delays in the wires that interconnect the modules in the system [Udding86]. The correct operation of the system means that it conforms to the ordering, or partial ordering relation of the sequence specification from the design process. The term delay insensitive was coined by a group at Washington University led by Charles Molnar and Wesley Clark [Clark74] around 1970 whilst working on macromodular computer systems. Essentially delay insensitivity entails that no simplifying assumptions can be made about component or wire delays except that they be finite and positive.

It has been shown by Alain Martin [Martin90] that the class of circuits that are delay insensitive is very small and that for some functions there is no delay insensitive realisation of the function. However the limitations of delay insensitivity are not as bad as they may sound at first. If simple assumptions are made about the delays in small physical regions then any arbitrary circuit can be realised in a functionally correct manner. The nature of the assumptions and their implictions is discussed in section 9.

There is another point that is pertinent to a discussion of delay insensitive systems. Although at some point the delay insensitivity of a system must be comprimised to allow arbitrary functions to be realised there may still be delay insensitivity at a higher level. If a system is composed of modules that each present a delay insensitive interface then the system will be delay insensitive in its operation. Udding gives a formal classification of delay insensitivite systems [Udding1986] as free from computation and communication interference. Computation interference means a signal arriving when the acceptor is not ready for it and cannot assimilate it meaning the signal will have an undefined effect. Communication interference occurs when two signals on the same wire interfere; an example is when a signal is asserted and then negated before it can be assimilated by a module so that the two signals interfere and the effect on the module is undefined.

7 *The limitations of delay insensitivity*

This section contains an explanation of why and how the limitations in delay insensitivity that have been referred to before arise in delay insensitive circuits. The argument follows the form given by Martin in [Martin90].

A delay insensitive circuit is a network of gates; a gate being a Boolean logical operator with one or more inputs and one output (a multiple output gate is constructed by adding a fork to the output of a gate). The state of a circuit is characterised by the values of the input and output variables of the gates composing the circuit. The assumption is made that all circuits are closed, each variable of the circuit is the input of a gate and also the output of a gate; an open circuit can be transformed into a closed one by representing the environment of the circuit as gates.

A gate is defined by a pair of production rules each with a Boolean guard, one causing a true transition (representing Boolean 1) and one causing a false transition (representing Boolean 0). For example, an AND gate with inputs x and y and output z can be defined as follows.

$$x \wedge y \text{ -> } z_t$$
$$\text{-}x \vee \text{-}y \text{ -> } z_f$$

where z_t and z_f denote a true transition on z and a false transition on z is respectively and $x \wedge y$ is an example of a guard expression.

Martin uses an equivalent definition for delay insensitivity to Udding [Udding86]. The two conditons that a circuit must adhere to for delay insensitivity are non-interference and stability. Non-interference means that the two production rules defining the gate cannot be true at the same time and so it is not possible to attempt a true and a false transition at the same time (computational interference). Stability refers to the guard of a production rule; the guard is stable if it is falsified only in states where the result of the production rule holds; that is a transition cannot be withdrawn once it has been initiated (communication interference).

To advance it is necessary to construct a partial order relation for transitions. Firstly a pre-order relation $<$ is constructed that is transitive and anti-reflective. From this the partial order $\leq$ is defined as follows, $a \leq b$ meaning $a<b$ or $a=b$.

For a circuit to exhibit stability, and hence satisfy one of the conditions for delay insensitivity, a guard cannot be falsified until it holds true (in the example if $x \wedge y$ becomes true it cannot be falsified again until the transition z_t is complete) . Consider the execution of a production rule with guard B and transition t. Either B is never falsified once it holds (t is the final transition of the variable) or B is falsified after a finite number of transitions following t. In the case of B being falsified it is necessary for t to complete before B is falsified.

For all transitions i that falsifiy B it must be guaranteed that $t<i$. Hence by definition of the order relation there must be a transition s such that s is a successor of t and $s \leq i$; s acknowledges t. Hence a theorum, called the acknowledgement theorum can be stated: In a delay insensitive circuit, each non-final transition has a successor transition. Also by corollary, a non-final transition on the input of a gate has a succesor transition on each output of the gate.

Using the above argument it can be shown that there can be no delay insensitive circuit that contains an AND gate, OR gate, XOR gate or flip-flop except where the number and order of the input transitions is constrained to the point where the gate becomes of limited use in general application. For the example of an AND gate with inputs p and q and output r. Consider the case (starting from an inital state where all inputs and outputs are false (Boolean 0)) where p goes to true and some time later q goes to true, after a delay r will go to true and both p and q will have been acknowledged. Now the gate can be falsified by making either p or q false (simultaneous inputs, however unlikely will have an undefined effect). If p goes to false r will go to false and p will have been acknowledged. In this case q has not and cannot be acknowledged and so the gate cannot be used in a delay insensitive circuit. Similarly for an OR gate the only way in which transitions can be acknowledged is if one input goes to true then the output goes to true which acknowledges the input then the same input goes to false and the output will then acknowledge this; at no time can both inputs be true at the same time or one will not receive an acknowledge.

It can be reasoned in a similar way to the above that for a delay insensitive circuit all gates except the inverter and the Muller C-element can be eliminated. This means that a delay insensitive circuit can contain only C-elements. A Muller C-element is a gate where the output becomes the same as the inputs when both inputs attain the same state, for any other case the C-element retains its previous state.

There is a theorum that states that if a delay insensitive circuit has only one computation, and the computation contains at least three transitions on each variable, then the circuit can be constructed in a delay insensitive manner using C-elements only. The proof of this is straightforward but long and the interested reader is directed to a thorough explanation in [Martin90].

8 *Current trends in asynchronous design methodologies*

Current trends in asynchronous design methodologies find a direct parellel in high level languages. The trend in high level languages towards object oriented design encompassing abstraction and encapsulation are carried over into asynchronous design methodologies. The common factor in many methodologies [Ebergen91][Costa93] [DeGloria93][Meng89][Lam90] is that systems of arbitrary complexity can be designed and synthesised from high level descriptions and can be built using a set of primitive building blocks. Any notion of layout, timing or any low level issues can be neglected because as long as the building block interfaces function in a delay insensitive manner as specified and the high level description is correctly compiled into primitives the system will function correctly. This abstracts away superfluous low level detail. Also because all the building blocks communicate in a delay insensitive manner a logical interconnection of the building blocks will also have a delay insensitive interface and can therefore be encapsulated and reused. This leads to a hierarcical design process. This is a major advantage of delay insensitive systems; if you have a set of delay insensitive building blocks then a specification using these building blocks will function correctly regardless of the layout of the blocks or on the internal composition of the blocks [Ebergen91][Udding86]. Hence one block can be substituted for a block with the same interface but a more efficient implementation or better performance without affecting the correct operation of the system. This argument also applies to higher level encapsulated modules, any implementation with the same interface can be substituted into the system without affecting the operation of the system; this leads to modular expansion and incremental improvement of systems [Ganesh92].

However the layout of the delay insensitive building blocks will have a decisive influence on the performance of the system and the composition of the blocks will similarly affect the performance. This aspect of asynchronous design has been neglected in favour of simplified design and automated synthesis of systems. The emphasis on object oriented design may not ultimately be an entirely healthy trend unless sufficient attention is also applied to optimising asynchronous designs at a lower level.

The trend in high level languages is to allow a decrease in performance, as a direct result of an increase in code size, if the resulting code allows complexity to be managed efficiently. Processor design should not follow suit concentrating primarily on managing the complexity of design at the expense of performance. The success of a processor is intimately tied to performance and ultimately this is the feature by which a processor will be judged. Hence processor design must at some stage take account of low level issues to produce designs that perform optimally; it is of no use to say that two components can just be connected up and placed on a chip anywhere and the system will perform correctly, the placement of those two components must be carefully analysed in relation to all other components in the system to produce a design that will perform optimally.

9 *Asynchronous design methodologies*

This section examines some of the most important and widely used approaches to designing asynchronous circuits. Techniques for designing delay insensitive circuits are examined as well as micropipelining which is not delay insensitive but offers promising performance increases.

9.1 *Micropipelines*

Micropipelining is a technique based on the work of Ivan Sutherland [Sutherland1989]. It is an interesting technique that has the potential to outstrip the performance of a synchronous equivalent because the transition signalling conceptual framework leads to a potential for doubling the speed compared to conventional clocking. There is also an appeal due to the inherent simplicity and elegance of a micropipeline. However the simplicity arises from a relaxation of the delay insensitivity of the system. There is a rigorous condition on correct operation that must be met by a micropipeline that data lines must be stable and valid before an initiation signal arrives. This condition is neither new nor surprising since similar constraints apply in synchronous systems.

Micropipelines introduce a dependence on the final layout of the system, for correct operation all data lines must be stable and valid when the initiation signal arrives. Hence the layout of the final system is constrained such that the routing of the initiation line must alway have a longer delay associated with it than every data line. This makes automated layout more difficult but it would be hoped by no means impossible. Also when substituting one module in the system for another the physical size and shape of the new module relative to the old module would be important.

The completion signal for a pipeline stage in a micropipeline is controlled by a delay. When the initiation signal arrives the data is already valid and ready to be operated on, after a certain time dictated by the delay the operation is assumed to be complete and a completion signal is issued. This means that the delay between receiving an initiation signal and issuing a completion signal must be a worst case time for the operation performed. However this would still do far better than a synchronous system where instead of having a worst case for the operation to be performed the worst case is for the entire range of operations possible. For example the worst case time for an add operation is much greater than the worst case time for a shift operation.

There is another important advantage of a micropipeline compared to a synchronous design. The worst case time in the synchronous design (the clock cycle time) must take into account the worst case enviromental conditions which can lead to a non-optimum cycle time. In a micropipeline the delay can be implemented as a string of gates chosen to give the correct delay, and is called a hardwired delay. Whereas in a synchronous pipeline there is no way for the clock cycle to track enviromental conditions in a micropipeline the delay through the hardwired delay would track enviromental conditions very well because the gate string would be in close physical proximity to the logic circuit and would be subject to an enviroment very similar to the logic circuit ensuring that the delay through the hardwired would closely track the delay through the logic circuit. Therefore a micropipeline would perform optimally across the whole range of temperature, humidity and voltage. Further a micropipelined implememtation would be less sensitive than other alternatives to process variation. Any variation introduced during fabrication would apply to both the circuit and its associated delay and so would track well. This would lead to more robust processors and a higher yield per silicon wafer.

Overall a micropipeline structure would produce an elegant, compact design. The area penalty for asynchronous organisation would be minimal and there would be the possibility of a substantial performance enhancement. Increase performance is the major factor in favour of micropipelines. The signalling convention is as fast and efficient as possible so that, in theory, a micropipeline has the potential to outperform both a synchronous system and a delay insensitive system. A further advantage is that micropipelines would scale well with technology. Because a micropipelined system depends only quite weakly on the delays in wires scaling down the dimensions of the system should not cause major upheavals in design or implementation.

9.2 *Delay insensitive design*

There are several delay insensitive design methodologies proposed at present [Meng89][Lam90][Ebergen91][DeGloria93][Costa93] and all follow the basic structure of decomposing a high level description into a network of basic modules (building blocks). The building blocks are delay insensitive and communication takes place by means of channels utilising a 4-phase (return to zero) signalling convention [Seitz80]. This method of communication involves twice the number of transitions compared to 2-phase signalling [Seitz80]. Also the system must include completion detection which entails an overhead in detecting when an operation has completed. Dual rail coding of data is necessary to allow completion detection. Each bit of information is coded into two bits on data lines. This allows the data to have three states; these states are Boolean 1, Boolean 0 and an undefined state. This enables completion of an operation to be detected by sensing when all the output data values have made a transition from the undefined state to a defined Boolean 1 or 0. Sensing the transition is straightforward and entails an XOR gate for each dual rail coded bit, the XOR gates in turn are connected to a multiple input Muller C-element; when each dual rail bit has made a transition all the XOR gates will go to 1 and hence the C-element will go to 1 signalling completion. For a sequence of operations the circuit must start from the undefined state each time. Starting from the undefined state means that the system must be cleared back to the undefined state between computations. There are also additional problems in detecting when a circuit has completed clearing back to zero which leads to the use of a worst case delay for clearing down. This involves a substantial overhead and amounts to a major performance penalty. Techniques to overcome this overhead include duplicating functional units or alternating code sets but each solution implies a large increase in the area of the circuit. One promising technique is that proposed by Martin [Martin89] where the time spent waiting for the system to clear down is hidden to some extent. The basis for this technique is that the final acknowledge for the completion of the clearing down process is overlapped with a subsequent operation. Hence although the system will be delay insensitive and design of the system need take no account of any implememtation issues such as layout, substantial overheads will be introduced by using a delay insensitive signalling convention. The problem that must be resolved is whether the greater design freedom of delay insensitivity and the possible performance benefits of asynchronisity are outweighed by the additional communication and computation overheads.

An advantage of delay insensitivity is that changes in technology will not affect this technique and that the performance will be enhanced by technological advance. As the problems associated with synchronous processor increase due to design and fabrication issues the delay insensitive processor will be going from strength to strength due to lower gate delays and increasing integration. Another advantage is that a delay insensitive system is arbitrarily expandable and can be altered at will meaning that increasing the lifetime of a design by midlife improvement is relatively easy.

Delay insensitive systems will also pay a high price in terms of silicon area. Dual rail encoding of data using both phases of the input data operated on separately will mean an approximate doubling of silicon area. Also for performance reasons if the overhead in resetting the system to the undefined state is to be disguised the whole functional unit must be duplicated so that while one unit is clearing down the other unit can start another operation; this will mean another approximate doubling of silicon area. Therefore taken together dual rail encoding of data and duplicating functional unit will mean a four fold increase in silicon area. Again it must be decided whether this price is too high to pay for the increased generality and design simplicity of delay insensitivity.

10 *Assumptions*

This section outlines a few of the assumptions that can be made to allow arbitrary functions to be realised asynchronously. These can either be used to build entire systems or to construct a set of building blocks. The selection is not a definitive guide but serves to illustrate the choices that can be made in designing asynchronous circuits and systems. At some level the delay insensitivity of the system must be compromised to enable arbitrary functions to be realised. The decision to be made is at what level to make compromises; should they be made at as low a level as possible to retain as much generality as possible or should they be made at a higher level to aid in design and implementation. There is also the point that at some level assumptions can be validly made. Some wires are short enough that the time for a voltage change to equalise along the length of the wire is negligable. The skew between signals on different branches of a fork will be negligable when the branches are of a similar length. The difference in delays of gates in close physical proximity will be negligable and so on. The search for generality may become blinding in the sense that in designing and building systems of the complexity of modern processors sensible and justifiable assumptions are essential.

10.1 *Equipotential regions*

Self timed systems were introduced by Charles Seitz [Seitz1980]. A self timed system makes the assumption that a system can be divided into small regions, called equipotential regions, and inside these regions the wire delays can be assumed to be negligable. The conditions placed on the system are that a component must reside completely inside one equipotential region and that the whole of the system must be covered by a number of equipotential regions. Hence inside an equipotential region speed independent design can be used assuming wire delays to be negligable compared to gate delays and the signalling convention need not be delay insensitive. Across equipotential regions wire delays cannot be assumed to be negligable and so a signalling convention must be used that is delay insensitive. Taking the example of a delay insensitive system constructed using a set of building blocks, each building block will reside inside an equipotential region and can be designed assuming negligable wire delays whereas different building blocks reside in different regions and so wire delays cannot be assumed to be negligable and delay insensitive signalling must be used. A disadvantage of this assumption is that it brings in the hazy notion of a region on a chip inside which delays are negligable. How big is the region? Does it vary from system to system or is it constant? This assumption requires careful and correct low level knowledge.

10.2 *Isochronoic forks*

An isochronoic fork is another assumption that can be made to allow arbitrary circuit functions to be realised in an asynchronous manner [Martin90]. The isochronic fork assumption essentially makes an assumption about the skew in the arrival of signals on lines. An isochronic fork is a local fork introduced to distribute a variable as the input to several gates. The assumption made is that the delays in the various branches of the fork are small compared to the delays in the various gates receiving the variable as input. In terms of acknowledgements it is assumed that when a fork distributes a signal to a number of gates only one acknowledge will be required; when one branch has been acknowledged it is assumed that the other branches of the fork will also have finished.

11 *Mixed design styles*

One possibility for further work is on mixed asynchronous/synchronous systems. These would be locally synchronous, globally asynchronous systems [Rosenberger88]. They would be composed of a number of synchronous modules to perform computation in an area and time efficient manner. The clocks inside the modules could be stopped and started asynchronously by initiation and completion signals. The overall composition of the system would be asynchronous but the component modules would be locally clocked.

This approach would probably be regarded as the next step forward as it represents an evolutionary approach; if clocks cannot be handled on a scale as large as a silicon chip simply break the chip up into more managable chunks and clock these. Evidence of this approach is already becoming apparent on the larger and more complex processors available today; the more complicated the system the more asynchronous communication must be introduced on chip to cope with that compexity.

The motivation for this approach is apparent. It would allow the 'synchronous fraternity' to carry on and contribute to new designs without having to tackle the underlying problems with synchronous design. It would be an approach that would put off the moment when synchronous design must face its nemisis in the form of generating and distributing global clock signals. However this approach does have merit. The design would be managable; if the synchronous regions were small and simple enough they could be designed and tested quite easily using existing technology. There is also a wealth of knowledge and sophisticated tools associated with synchronous design; it is efficient and would lead to small, high performance clocked modules.

Does this represent the way forward? Probably on a commercial basis but there are strong motivations to avoiding the use of clocks in any form. The locally clocked modules will not scale well with technology; these modules will face the same problems faced by processors today when the scale of integration reaches a sufficient level. At this point the modules will either have to be further subdivided or abandoned. The subdivision of modules could not go on indefinately since there must be a limit on the number of independent clocks that can be supported by a system without the area required for clock generation and distribution becoming too large. Also concerted effort on asynchronous design will benefit all design styles and will help make hybrid systems more efficient, both in terms of performance and silicon area.

12 *Conclusions and ways forward*

Asynchronous design is motivated by a desire to provide solutions to underlying problems faced by synchronous design. It is not an attempt to paper over the cracks or hold back the tide of physical reality but an effort to bring harmony to the discipline of integrated circuit design. The trend at present is to wrestle with technological advance; synchronous design means that designers are having to battle with the problems posed by advancing technology to take advantage of the benifits of higher levels of integration and higher frequency clocks. The benefits offered will to some extent be compromised by this battle. Asynchronous design holds the promise of harmony between the two aspects of integrated circuits, design and technology. It is possible, and likely, that asynchronous design will allow synergy between technology and application. New applications will take the best advantage of available technology and will have a longer lifetime due to increased ability to scale with technology and easier midlife improvement.

Regarding the best approach to asynchronous design there is a clear decision to be made. Two choices that become apparent are a pure delay insensitive approach or a more compromised approach such as micropipelines. The delay insensitive approach seems to be more general and places less constraint on layout issues for correct operation. However layout issues will still be of vital importance because they will have a strong effect on performance. Micropipelines are a simple and elegant idea but they are less general than delay insensitive systems. Rigorous conditions are placed on the layout of the system and any automation of the design process must take this into account. However it was shown before though that even a delay insensitive system will need to be optimised at the layout level and therefore does the constraint on micropipelines amount to a major disadvantage?

Regarding performance more work is needed. A micropipeline approach offers high performance at relatively low cost and in all probability would substantially outperform a synchronous equivalent. The 2-phase signalling convention is as fast and efficient as possible. However the completion signal is generated by a delay in the module. This leads to the use of a worst case delay for the circuit considered. In a delay insensitive system an operation will complete in the time required based on the relative complexity of the operands but there is an overhead, in terms of performance and area, introduced in the form of detecting and signalling completion. Also the need to return to an undefined state between computations will also degrade performance. Based on the evidence an intuitive estimate is that micropipelines will substantially increase performance whereas a delay insensitive system would probably be of comparable performance. This does not mean that delay insensitive systems do not have advantages over both synchronous systems and micropipelines; they will scale better with technology and will take advantage of the increasing scale of integration and the reduction in gate delays. Micropipelines will scale well with technology also but the constraints placed on the ordering of data and initiation signals may prove to be problematic as systems become increasingly more complex.

Hence the initial question raised by this article that needs to be answered is whether to use a micropipeline approach or a delay insensitive approach? Both have distinct advantages and can only be differentiated by making a decision for the overall objectives of the final system. Higher performance and simplicity or increased generality and scalability?

Generally it is the case that a high performance processor will be more desirable than a lower performance alternative. Applications produced today demand ever greater processing power (for example digital signal processing). Therefore if an asynchronous approach can be made to yield high performance at reasonable cost it will be an attractive alternative to synchronous systems. Such an approach is offered by micropipelining or some derivative of it. The problems of micropipelines are that a delay associated with each module must be fabricated and that layout must take into account the delays of the interconnecting wires. Both of these problems have reasonably straightforward solutions. The delay associated with each module is the worst case delay for the circuit and therefore circuits should be chosen of a managable size to facilitate this. The problem of managing delays on wires is more difficult but could be solved by automated layout tecniques. These would probably be time consuming to produce an optimum solution. Overall micropipelining offers the best performance possibilities along with the lowest cost in terms of silicon area increases.

References

[Clark74] W.A. Clark and C.E. Molnar, **Macromodular computer systems**, in **Computers in biomedical research**, R.W. Stacy and B.D. Waxman (Eds.), Vol IV, 1974, Academic Press: New York and London, pp45-83.

[Costa93] A. Costa, A. DeGloria, P. Faraboschi, G. Nateri and M.Olivieri, **An asynchronous approach to the RISC design of a micro-controller**, Microprocessing and Microprogramming, 38(1993), pp447-454

[David89] I. David, R. Ginosar and M. Yoeli, **An efficient implementation of Boolean functions and finite state machines as self-timed circuits**, Computer Architecture News, 17, 6 (December 1989), pp 91-104.

[DeGloria93] A. DeGloria, P. Faraboschi and M. Olivieri, **Delay insensitive micro-pipelined combinational logic**, Microprocessing and Microprogramming, 36(1993), pp225-241.

[Ebergen91] J.C. Ebergen, **A formal approach to designing delay insensitive circuits**, Distributed Computing, 51(5), 1991, pp107-119.

[Elston93] C.J. Elston, **Difficulties with synchronous systems**, in preparation

[Keller74] R.M. Keller, **Towards a theory of universal speed-independent modules**, IEEE Transactions on Computers, C-23, 1, January 1974, pp21-33.

[Lam90] P.N. Lam and H.F. Li, **Hierarchical design of delay-insensitive systems**, IEE Proceedings, Vol. 137, part E, No. 1, January 1990, pp41-56.

[Martin89a] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic and P.J. Hazewindus, **The design of an asynchronous microprocessor**, Technical report CS-TR-89-02, Caltech, 1989.

[Martin89b] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic and P.J. Hazewindus, **The first asynchronous microprocessor: the test results**, Technical report CS-TR-89-06, Caltech, 1989.

[Martin90] A.J. Martin, **The limitations to delay insensitivity in asynchronous circuits**, in W.J. Dally (ed), Sixth MIT Conference on Advanced Research in VLSI, MIT Press, Cambridge, MA, 1990, pp 263-278.

[Meng89] T.H.Y Meng, R.W. Broderson and D.G. Messerschmitt, **Automatic synthesis of asynchronous circuits from high-level specifications**, IEEE Transactions on Computer-aided design, 8, 11, November 1989, pp1185-1205

[Molnar85] C.E. Molnar, T. Fang and F.U. Rosenberger, **Synthesis of delay insensitive modules**, 1985 Chapel Hill Conference on VLSI, Chapel Hill, NC, May 1985, pp 67-86.

[Muller59] D.E. Muller and W.S. Bartky, **A theory of asynchronous circuits**, Ann. Computation Lab. of Harvard Univ., Vol 29, 1959, pp204-243.

[Muller63] D.E. Muller, **Asynchronous logic and application to information processing**, Switching theory in space technology, Aitken and Main (Eds.), Stanford, CA: Stanford Univ Press, 1963, pp289-297.

[Nordmann77] B.J. Nordmann, **Modular asynchronous control design**, IEEE Transactions on Computers, C-26, 3, March 1977, pp196-207.

[Seitz80] C.L. Seitz, **System timing**, in **Introduction to VLSI systems**, Addison-Wesley, Reading, MA, 1980, pp 218-262.

[Sutherland89] I.E. Sutherland, **Micropipelines**, Communications of the ACM, V32(6), June 1989, pp 720-738.

[Turing47] A.M. Turing, **Lecture to the London Mathematical Society on 20th February 1947**, in B.E. Carpenter and R.W. Doran (Eds), Charles Babbage Reprint Series for the History of Computing, Vol. 10, MIT Press, Cambridge, Massechusetts, 1986.

[Udding86] J.T. Udding, **A formal model for defining and classifying delay insensitive circuits and systems**, Distributed Computing (1986)1, pp197-204.