

Simple Circuits that Work for Complicated Reasons

Charles E. Molnar and Ian W. Jones
Sun Microsystems Laboratories, Palo Alto, California, USA

Abstract

This paper brings together a selection of creative circuit designs and ideas that Charles Molnar devised while working at Sun Microsystems Laboratories. The circuits offer fast implementations of functions that were severe speed bottlenecks in our existing systems. Charlie strongly believed that reliable and very fast circuit module implementations must make use of local delay constraints, and that the interfaces between these modules can then employ more robust and delay-insensitive signaling protocols. These circuit designs provided us with: an unusual 2-phase arbiter, known as the propellor arbiter; a method of using arbiters to measure on-chip delays very precisely; and three versions of a latch control circuit, known as a Charlie Box.

1. Introduction

This paper summarises some of the interesting and unusual circuits that Charles Molnar devised while working at Sun Microsystems Laboratories. The theme behind each design was to find a circuit that was fast. To achieve high speed, Charlie strongly believed that local delay constraints were both necessary and manageable, whereas taking a purist stance—requiring every part of the circuit to be speed-independent if not fully delay-insensitive—was cumbersome and complex. Charlie was always most interested in the hard challenges, and revelled in the design of circuits such as arbiters. This was an area where he was a leading expert, and yet he was the first to say that he still did not understand the problem fully and that there was further knowledge to be mined. Charlie always tried to dig down to the core of the problem, to make no assumptions, and to understand the meaning of each signal and the purpose of each component in the circuit. For circuits commonly used in clocked designs, he would gain a significant speed improvement in the circuit by identifying and eliminating the design criteria that apply

to the clocked domain but are no longer applicable in an asynchronous environment.

Many ideas seemed to be waiting up Charlie's sleeve and he would suddenly find a use for an idea that he'd long been mulling over, often for years. This sometimes resulted in quite unusual yet elegant circuit solutions. Other ideas would come to him while travelling; an example is the propellor arbiter design that I first read about in a letter that he sent to us in December 1992 from the M. S. Orchid while on a cruise on the Nile. Like so many of us, Charlie often needed a sparring partner, or sometimes just a listener, to help develop his ideas. Not only could he rarely resist a design challenge himself, but he also loved to issue challenges to his colleagues, saying with a twinkle in his eye, "and a dollar if you can solve this before I do." The title of this paper comes from Charlie's design style which nicely complemented that of his career-long friend and co-asynchronologist, Ivan Sutherland. Ivan is, quite typically, as he himself puts it, "the first to get it nearly right," and though generally ground-breaking in impact, these "first" circuit implementations are sometimes complicated and require careful study to refine and simplify them, usually resulting in greater speed. Charlie, with characteristically well-intentioned good humour, referred to Ivan's first circuit designs as "complicated circuits that almost work for complicated reasons."

Although some parts of the material in this paper have been publicly presented, I believe that none of it has been officially published. It is my intention here to help complete the record on Charlie's behalf. I have also specifically selected these designs because, although Charlie was the principal designer, I had the pleasure of working alongside him during the whole design process and contributing to the development of the component specifications and the evaluation of the circuit implementations.

2. Propellor Arbiter

The propellor arbiter design is one of Charlie's great works of art. The circuit is simple and yet has just enough

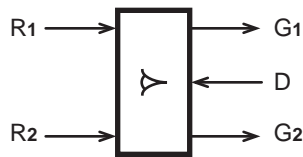


Figure 1. 2-phase, 5-wire arbiter.

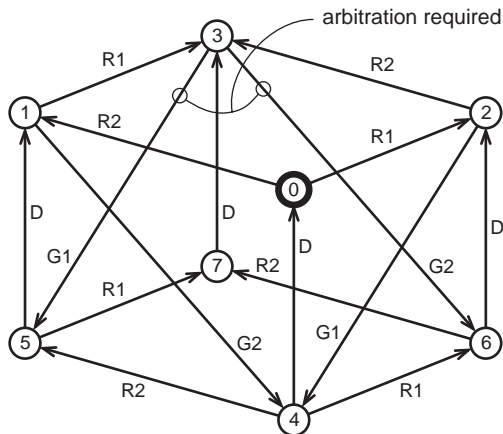


Figure 2. State diagram of 5-wire arbiter specification.

transistors to deal with the complex conditions it has to handle.

Charlie actually presented a circuit diagram of his propeller arbiter design during the Special Invited Session, “An Extended Industrial Design Example: The Counterflow Pipeline Processor,” at Async 1994, [7]. However, he did not go into much detail regarding the design during his talk, nor was the talk material included in the conference proceedings, being supplied instead as a collection of hand-outs to the attendees. What follows here is a fuller description of this remarkable circuit.

In 1992, Ivan challenged Charlie to design a faster 5-wire transition, or 2-phase, arbiter. A block diagram showing the interface signals of such a 5-wire arbiter is shown in Figure 1, and its state graph specification is shown in Figure 2. This graph has 8 states that I have drawn as the vertices of a cube. In this state graph each event corresponds to a transition in the level of the signal. The start state, drawn with a thicker O, is in the middle of the figure, at the top-front corner of the cube. Initially one request from R1 or R2 will be granted by the corresponding event G1 or G2, and a second grant cannot be issued until a done event D has been received. If, from the initial state, both R1 and R2 events have been received, the arbiter is in state 3 at the top of the figure, and the arbiter must decide to issue either a G1 or a G2, but not

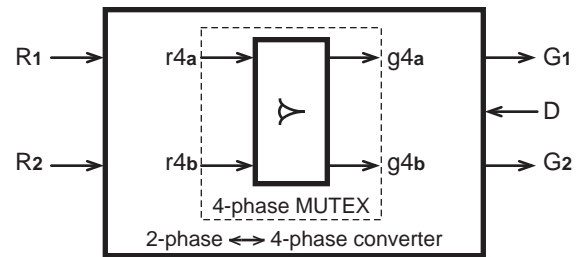


Figure 3. 5-wire arbiter implemented with a 4-phase MUTEX.

both. Only after a done event D has been received may the arbiter issue the second grant event.

We needed 5-wire arbiters between adjacent stages of the Counterflow Pipeline Processor design that we were working on at the time [8]. Throughout the design we used 2-phase control circuits. Our arbiter circuits employed a single classic 4-phase MUTEX circuit, as explored in the Washington University Macromodule Project [5] and described in [1]. These arbiters required 2-phase to 4-phase conversion of the request and done inputs, and 4-phase to 2-phase conversion to generate the grant outputs, as illustrated in Figure 3. In the Counterflow Pipeline application, we expected that the most common situation was that the arbiter would be receiving input requests relatively widely separated in time. As you can imagine, our existing arbiters were slow, and were a major bottleneck in this application. The challenge was to devise an arbiter circuit that did not require the slow 2-phase to 4-phase and 4-phase to 2-phase converter circuits.

Both Charlie and Ivan had come up with a variety of 2-phase arbiter designs that were unsatisfactory, either because they were very complicated or because they relied on some tricky local timing constraints for correct operation. In December 1992, Charlie went on a cruise on the Nile and sent us a letter describing a potential solution described in terms of the phase portraits of a dynamical system. Neither Ivan nor I fully understood Charlie’s idea until he returned from Africa and more carefully explained it to us. By this time Charlie had fleshed out the idea to the level of providing a set of logic equations.

The theme behind the propeller arbiter is that there are four circuit “blades,” N, E, S, and W, each cross-tied to its two neighbouring blades, as shown in Figure 4. The “hub” of the propeller is connected to the positive supply, and the tip of each blade is grounded. There are four output nodes, which I have labelled Nwins, Ewins, Swins, and Wwins. If one of these outputs is pulled Lo, it declares that blade to be the winner. Each of these output nodes is dynamic, as I have shown in the figure, and for static operation, each of these nodes must be weakly pulled-up to the positive

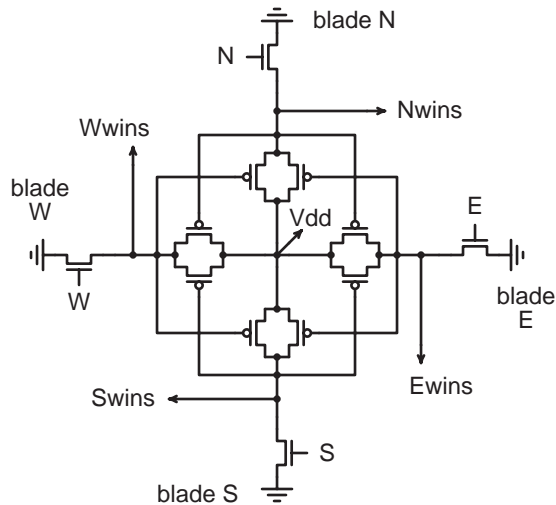


Figure 4. Basic circuit of propellor arbiter.

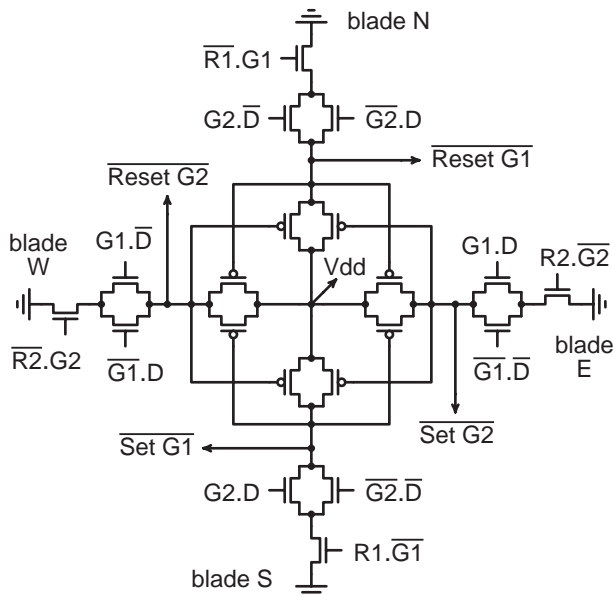


Figure 5. 2-phase propellor arbiter circuit.

supply by components not shown in this figure. The cross-coupling between each pair of adjacent blades forms a MUTEX, and thus there are a total of four MUTEXes in this propellor arbiter circuit, each of which shares the transistors in its blades with the adjacent MUTEXes.

Let's take an initial condition where all four inputs are Lo, then none of the outputs will be driven. Now if input N is driven Hi, Nwins will be pulled Lo, and via the cross-couplings Wwins and Ewins will be pulled Hi. In this state, the only output that is not actively driven is the node labelled Swins. This situation corresponds to a single uncontested request to the arbiter. Now take the situation

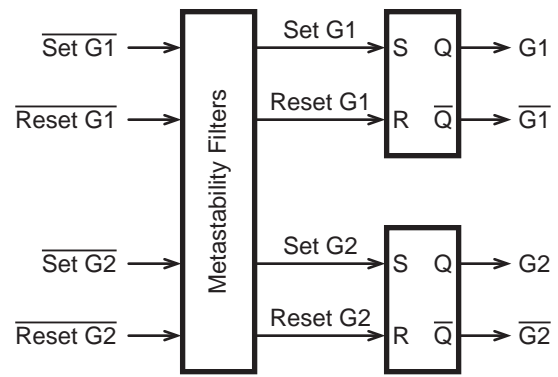


Figure 6. Inverters with metastability filtering and S-R flip-flops.

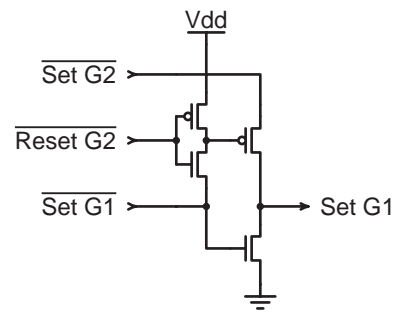


Figure 7. Inverter circuit with metastability filtering.

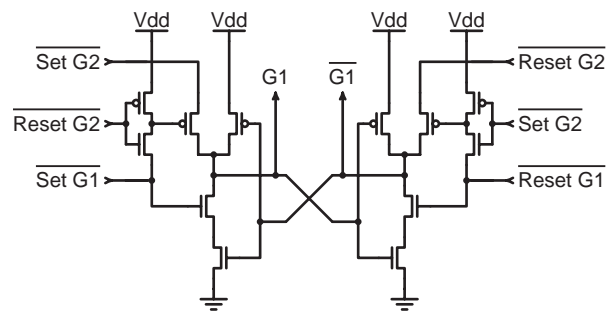


Figure 8. S-R flip-flop circuit with metastability filtering.

where inputs N and E are driven Hi at about the same time—a contested pair of requests to the arbiter. Under these circumstances the cross-tie between the N and E blades comes into play and only one of the nodes Nwins and Ewins will be pulled Lo. Say Ewins is pulled Lo, then E is declared as winning the arbitration, and node Nwins will be pulled Hi. Swins will also be pulled Hi, and Wwins will remain undriven—Wwins will be pulled Hi by the weak pull-up components not shown.

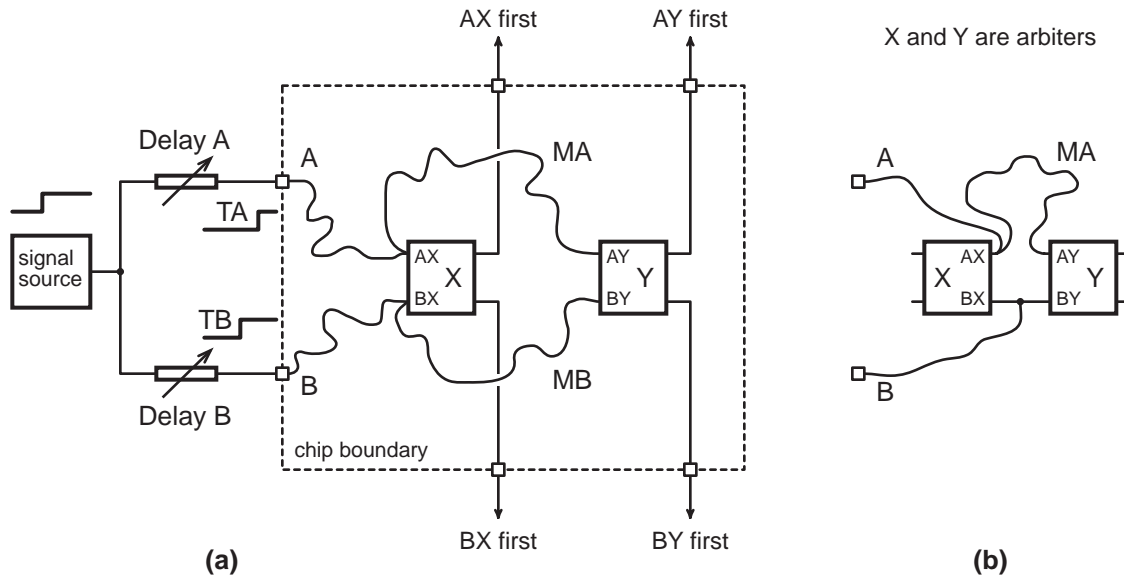


Figure 9. Test circuit using arbiters for measuring delay.

This basic propellor circuit provides exactly the necessary operation to implement the transition arbiter. What is required are a few extra transistors to establish when each of the propellor blades should be pulled down. Figure 5 shows these extra transistors and their gate signals that apply for an initial state of the arbiter with all of the input and output signals Lo.

The output signals G1 and G2 are generated and maintained by independent S-R flip-flops, as shown in Figure 6. The set and reset input signals to these flip-flops must be filtered to avoid state changes until after any metastability has been resolved. The filtering can be achieved by placing a metastability filter between the propellor elements and the flip-flops as shown in Figure 6. A sample metastability filter circuit is shown in Figure 7, which, with careful transistor sizing, produces a clean Set G1 output signal only after $\overline{\text{Set G1}}$ is pulled Lo and both $\overline{\text{Set G2}}$ and $\overline{\text{Reset G2}}$ are Hi. Alternatively, the metastability filter can be incorporated into the S-R flip-flop, as illustrated in Figure 8.

Referring back to Figure 5, in the initial state, input D will be Lo and fed-back output signals G1 and G2 will be Lo, and the first permitted input changes are R1 and/or R2 transitioning from Lo to Hi. In this initial state only the S and E blades are active, and their output nodes, labelled $\overline{\text{Set G1}}$ and $\overline{\text{Set G2}}$, are used to set either G1 or G2 Hi. The signals $\overline{\text{Set G1}}$ and $\overline{\text{Reset G1}}$ set and reset the flip-flop that generates outputs G1 and $\overline{\text{G1}}$. Any arbitration required from the initial state will thus be handled by the S and E blades. R1 transitions are handled alternately by blades S and N, while R2 transitions are handled alternately by blades E and W. Once a blade has been pulled down, say

the S blade with an R1, then no other blade can be pulled down until two further events occur: the grant event is issued, G1 goes Hi and partially enables blade N, and the done event D is received, which causes blade N to be fully enabled, and blade E to be re-enabled. Thus the next arbitration will be handled by the N and E blades. In this manner, at most two adjacent blades will be enabled to carry out the next arbitration.

When two request inputs arrive at about the same time, these input signals will attempt to pull down two adjacent blades. The cross-coupling between these adjacent blades will result in only one of the blades being pulled down, and the metastability filter will pass the winning signal on to the grant latches only when any metastability has been resolved. Once the winning grant signal is generated, it is fed back to the propellor arbiter, allowing the winning blade to be pulled back Hi.

The performance of this propellor arbiter circuit is about twice as fast as our original arbiter design that incorporated the 2-to-4 phase conversions—a substantial improvement.

3. Using Arbiters to Measure Delays

In 1996 Charlie came up with a very practical way of measuring on-chip delays from off-chip. The motivation came from our quest for speed in our asynchronous control circuits, leading us to rely on control of the relative delays of signals propagating on different paths. We wanted to make precise measurements of the way in which circuit delays vary among different circuits on the same chip. With sub-nanosecond on-chip propagation times, and severe

limits on the bandwidth of signal measurements via output pads, accurate measurements are very difficult. Even with on-chip probing, accurate measurements of small delays present serious difficulties.

Charlie had developed the viewpoint that synchronisers and arbiters act as time amplifiers, and this led to his idea to use arbiters for making delay measurements. Arbiter circuits provide a highly precise and accurate method for declaring which of two signaling events occurred first. Once such a determination has been made, the outcome can be communicated at leisure. This permits easy passage of such outcome results through output pads, and multiplexers or other steering circuits can easily be accommodated to minimise the number of circuit pins needed for making these measurements.

The measurement technique using arbiters can measure delay values and differences of the order of a picosecond, without requiring any special techniques for providing signals on-chip or bringing signals off-chip for external measurement. This measurement method may prove useful for the study of synchronous as well as asynchronous circuits, for example for measuring the skew in propagation along clock distribution nets.

Figure 9 shows the delay measurement circuit arrangement. It includes an off-chip source of two signal transitions, at times T_A and T_B , that drive paths A and B. Signals A and B enter the chip through two package pins, and are conducted by separate paths and conditioning circuits to two on-chip path segments MA and MB, whose delays are to be compared. Two on-chip arbiters observe the relative order of arrival of signals at the entry to and at the exit from, respectively, path segments MA and MB.

Arbiter X observes the two signals at two adjacent points, AX and BX, where each enters one of the segments MA and MB whose relative delay $DA-DB$ is to be measured. The times T_A and T_B of the two input signals are then varied until a very small change in T_A-T_B alters the order of their arrival at AX and BX as reported by the arbiter X. We label these times T_{AX} and T_{BX} . We conclude that, with these input delays, the two signals arrive at AX and BX and enter their respective segments MA and MB simultaneously.

Arbiter Y determines the order of arrival of signals A and B at the ends AY and BY of the segments MA and MB that they have taken. Once again, T_A and T_B are adjusted until a very small change in T_A-T_B inverts the order of arrival of the signals at AY and BY. We label these times T_{AY} and T_{BY} . We conclude that, with these input delays, the signals arrive at AY and BY upon leaving their respective segments MA and MB simultaneously.

Two sets of measurements are needed, the first to make the arrival of signals A and B at points AX and BX simultaneous, and the second to make the arrival of signals

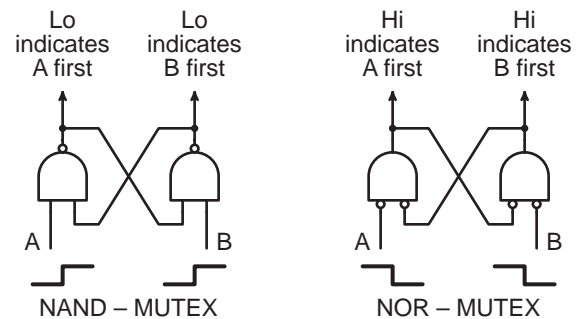


Figure 10. NAND and NOR MUX circuits.

A and B at points AY and BY simultaneous. Each set of measurements requires multiple trials to find the input delay settings at which the order of arrival at the appropriate test point change. These measurements give us T_{AX} , T_{BX} , T_{AY} , and T_{BY} . Only the differences $(T_{AX}-T_{BX})$ and $(T_{AY}-T_{BY})$ are needed; $(T_{AX}-T_{BX})-(T_{AY}-T_{BY})$ gives us $(DA-DB)$, the difference in the delays through segments MA and MB. If nodes BX and BY are made to coincide, and only delay T_A is adjusted, then $T_{BX}=T_{BY}$, and our measurement yields the absolute delay in segment MA as $T_{AX}-T_{AY}$, as illustrated in Figure 9b.

Since the input signals can be generated off-chip, their control should present no special problems. The measurement accuracy depends on the assumption that the relative propagation delays from the sources of signals A and B to AX and BX are constant and independent of the time relation between signals A and B. We would expect the difference of their propagation delays to be quite stable, although one must be wary of unintended coupling of the two paths by power supply or ground sharing, or by mutual capacitance, which would cause interactions of the timing of the two signals. History-dependent effects might also produce drifting of thresholds or other effects that would appear as instability of delay values. If feasible, making the input signal waveforms periodic at a constant frequency, but with adjustable phase, should eliminate first-order effects of this kind.

Notice that signal conditioning circuitry can be included in the input paths leading to points AX and BX without compromising the measurement method, provided that they do not add new opportunities for the kinds of problems mentioned above.

The simple MUX form of arbiter circuit made up of two NAND or NOR gates with their outputs cross-coupled, as shown in Figure 10, is very suitable for making the delay measurements. Such circuits are capable of declaring the relative arrival order of two positive-going (NAND) or two negative-going (NOR) signal transitions. By their symmetry, these circuits should have relatively little bias. It

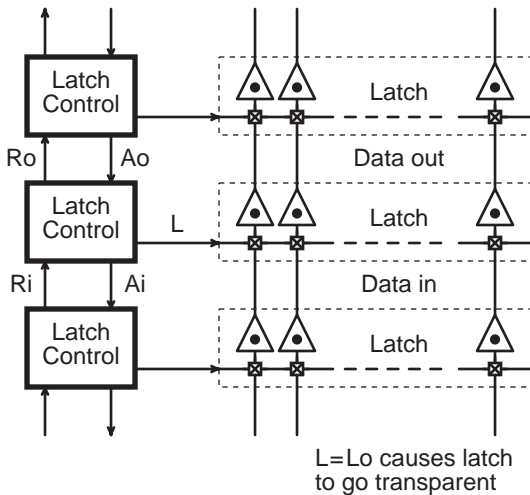


Figure 11. Latch control and data path.

is more difficult to see how arbiter circuits might be built to compare positive-going to negative-going signals without bias. Metastability-caused delay in the response of either arbiter is not likely to be a problem, since there is no need to observe the arbiter immediately after it is presented with input changes.

4. Charlie Boxes

Latch control circuits were another bottleneck in our Counterflow Pipeline Processor design. Once again, Charlie came to the rescue with a variety of novel alternative control circuits that more closely match the speed of the latches in the data path. These control circuits became known within our group as “Charlie Boxes.” I presented three Charlie Box circuit designs at Async 1994, though these were not published in the proceedings but were provided as a collection of hand-outs to the attendees.

The existing latch control scheme that we were using is shown in Figure 11. This is the control with bundled-data scheme, and three stages of a FIFO pipeline are shown in the figure. Each data path latch consists of a pass-gate (a square with an X inside) followed by a sticky buffer (a triangle with a • inside). The sticky buffer holds the data value when the pass-gate is opaque, and can be implemented in a variety of ways. One common implementation, shown at the top of Figure 13, consists of a pair of inverters, with a weak feedback inverter from the center node back to the input to provide the storage; input values from the pass-gate easily override the fed back value. The latch control circuits, to the left of Figure 11, provide the level signal L that makes the pass-gates opaque and transparent. With an initial condition where all pass-gates are transparent, the data path will be transparent.

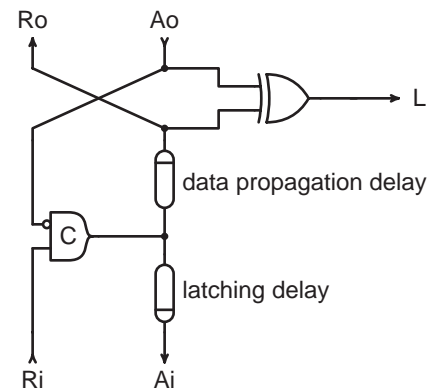


Figure 12. Basic micropipeline latch control circuit.

Charlie’s first step was to introduce the notion of data validity and non-validity messages and to define the association between these messages and the control signal events. Thus transition input signal Ri to the latch control circuit announces that the data presented at the input of the pass-gates is valid and that it may be latched. The control circuit then changes the level of the latch control signal L, thus changing the state of the pass-gates from transparent to opaque. The Ri event is acknowledged with an Ai event when the data are safely latched. Valid data on the output of the latch is announced by the event Ro. Receipt of an Ao event indicates that the output data have been received by the following stage, and that the present stage may make its output data invalid.

Referring to Figure 12, a more precise interpretation of the transition events in each of the control signals output from the current stage to the two adjacent stages follows. Note that the two output signals Ro and Ai have quite distinct meanings, reflecting the notions of data validity and non-validity. These meanings are also reflected in the meanings given to the inputs Ri and Ao which are the outputs of the adjacent stages:

Ri - Indicates that a next instance of valid input data is guaranteed by the previous stage to be available to the current stage.

Ro - Indicates that a next instance of valid input data is guaranteed by the current stage to be available to the following stage.

Ai - Indicates that the previous stage may allow its output data to become invalid, and announce new valid data when available.

Ao - Indicates that the current stage may allow its output data to become invalid, and announce new valid data when available.

The mnemonics are as follows: R indicates request, A indicates acknowledge, and i and o denote input and output interfaces of the current stage with respect to the data. Note

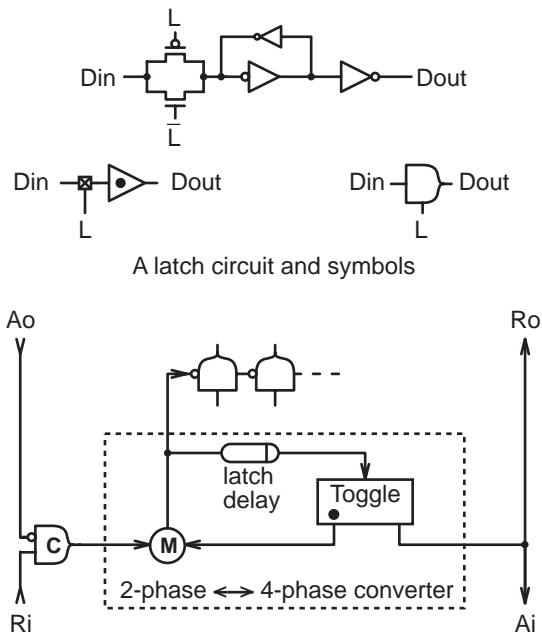


Figure 13. Micropipeline FIFO control circuit with 2-phase \leftrightarrow 4-phase conversion.

that R_i and A_o are inputs, and that R_o and A_i are outputs of the current stage.

The basic latch control circuit consists of a C-element, with its A_o input initially primed, as shown in Figure 12. This control circuit appears in Wann and Ellis, 1967 [2], and may have been used even earlier, and has become the familiar 2-phase micropipeline control circuit after being published by Ivan in his Micropipelines paper [6]. This control circuit is intended to operate 4-phase data latches, such as the circuit shown at the top of Figure 13. The latch control signal L is produced from an XOR gate that monitors the C-element output, suitably delayed to match the data propagation delay through the latches and the A_o input. The XOR gate causes the latch to go transparent whenever an A_o has been received and the C-element has not yet fired. Thus, the latch is made opaque after the C-element has fired, and will remain opaque until the output data is acknowledged with an A_o input. The use of this XOR element requires considerable care to ensure that the setup and hold times and minimum latching times of the data latches are satisfied. Typically this is done by managing the delays in various control and data paths. To illustrate this, in Figure 12, I have shown delays between the C-element output and the A_i and R_o outputs that match the latching delay and the data propagation delay through the latch, respectively.

A more conservative 4-phase latch control circuit used to operate normally opaque latches is shown at the bottom of Figure 13. This method of incorporating 4-phase data

latches into a 2-phase control environment is inefficient because of the delay overhead of the 2-phase to 4-phase converter employed.

A key observation that Charlie made was that latching does not change the data, but unlatching does—a concept that was opposite to our established design style, and one that was inherited from standard design practice of clocked circuits. Charlie used this new observation to obtain greater concurrency in operation between the latch control circuit and the flow of data through the latches. The design approach taken by Charlie was driven by the desire to achieve better performance, both by allowing earlier announcement of data availability to the next stage, and by more complete and careful characterization of the data latching element and its environment.

The requirements for the latch and associated control circuit were: the use of transition control signaling between stages; simple 4-phase data latches as the data storage elements; use of levels to encode data, with “bundling constraints” between control signals and data validity. Also, we wanted to announce when new output data was available, with R_o , as soon as the data propagated through the latch and are valid at its output.

Charlie designed several latch control circuits, each optimised for a different trade-off between functionality, speed and circuit complexity. I will present three of the designs here: the Full Charlie Box, “XXL,” the Simplified Charlie Box, “M,” and the Oversimplified Charlie Box, “S,” where the labels “XXL,” “M,” and “S” hint at the size and complexity of the circuits. These are the same three designs that I presented at Async94 [7]. Then I emphasised their functionality, expressed in terms of Petri net snippets, while here I will concentrate more on the circuit designs. Charlie developed the circuits starting with a specification of the largest and most complicated design, and then carefully pruning the functionality and increasing the number of local delay constraints to achieve greater speed and smaller circuit implementations. However, I will present them in the order “M,” “S,” and “XXL,” as this lends itself to a more gradual explanation of the trade-offs between functionality and performance.

I will start with the Simplified Charlie Box, “M.” The latch control design idea that Charlie came up with was unusual because he chose to treat the latch as not only a data path element, but also as part of the control circuit, with feedback signals from the latch circuit to the control circuit to report the state of the latch, as illustrated in Figure 14. This figure shows three pipeline stages that form part of a FIFO, with the data path shown on the right, and the associated latch control stages on the left of the figure. The latch control circuit has the expected R_i , A_i and R_o , A_o interface signals, and the latch control signal, L , but in addition receives an acknowledge signal, G , back from the

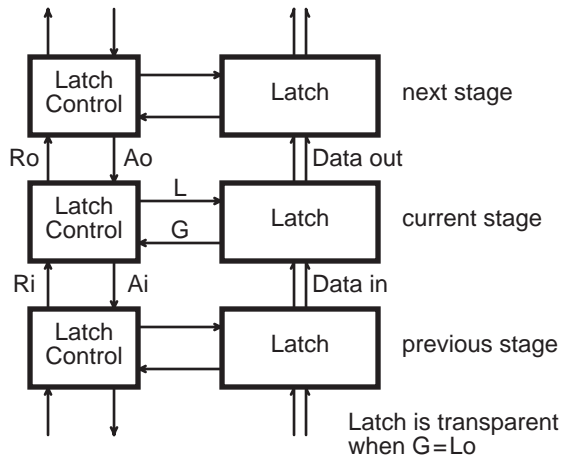
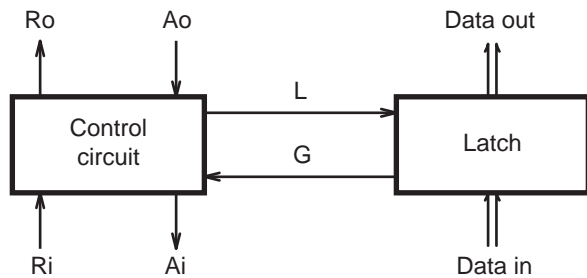


Figure 14. Latch control block diagram—with one feedback signal, G.



- L+ causes latch to go opaque
- G+ announces latch is opaque
- L- causes latch to go transparent
- G- announces latch is transparent

Figure 15. Latch control block diagram—messages associated with L and G.

latch. One stage is shown in Figure 15 along with the messages associated with the rising and falling edges of signals L and G. Starting in an initial state when both L and G are Lo, the transition of L from Lo to Hi, denoted L+, causes the data latch to change from transparent to opaque. Once the latch is safely opaque, the L+ event is acknowledged by the G+ transition. The G signal exhibits hysteresis, and in response to a falling L transition, G- is announced only when the latch has become fully transparent once more.

The specification, in the form of a state diagram, for this Simplified Charlie Box latch control circuit is given in

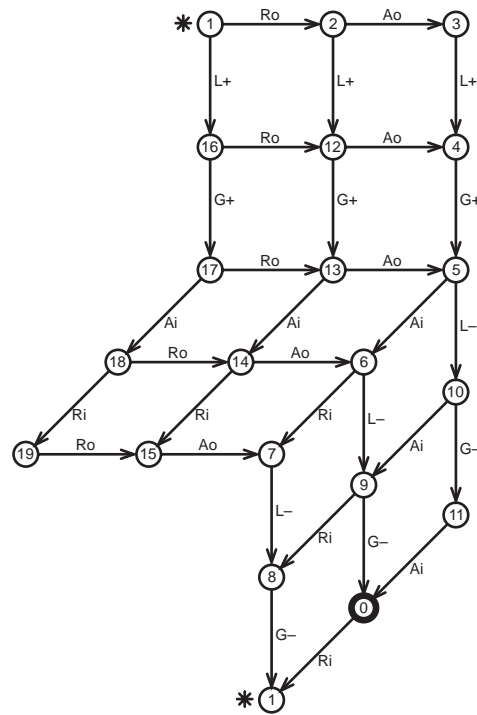


Figure 16. Latch control circuit specification for the Simplified Charlie Box, M.

Figure 16. The start state is 0 at the bottom of the diagram, and after an Ri input reaches state 1 that is duplicated at the top of the diagram and indicated by a “*”. Events Ro, Ao are now concurrent with L+, G+, indicating that the process of latching the data does not need to be completed before announcing valid output data to the next stage. This concurrency is represented by the squares formed by states 1, 2, 3, and 16, 12, 4, and 17, 13, 5, in the top of the state diagram. Once the data are latched, as indicated by the receipt of the G+ event, in states 17, 13, and 5, then Ai can be issued informing the previous stage that it is no longer obliged to maintain valid data to the input of this stage. Events Ai, Ri are thus also concurrent with events Ro and Ao, as specified by the squares with states 17, 13, 5, and 18, 14, 6, and 19, 15, 7, in the middle of the diagram. Finally, once the data latches are opaque, and an Ao has been received, then the unlatching process, L-, G-, can take place concurrently with the Ai, and the next Ri event. This is specified by the squares with states 7, 6, 5, and 8, 9, 10, and 1, 0, 11, at the bottom of the diagram.

Charlie used a combination of EISG [4] and Unger Synthesis [3] methods to, by hand, come up with several circuit implementations that meet this specification. One implementation of this Simplified Charlie Box is shown in Figure 17. Notice the element used between the Ri input and the Ro output—it is a wait-on element. A wait-on

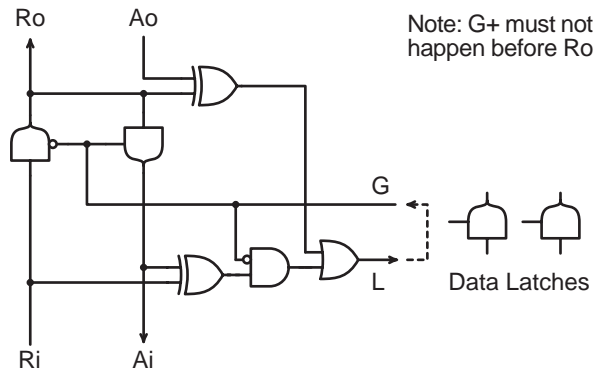


Figure 17. Simplified Charlie Box circuit, M.

element can be thought of as a 1-input C-element with an enable input. When enabled, a transition on the input generates a transition on its output, while when disabled no output transitions will be generated, and the output retains its present level. A wait-on element between R_i and R_o is significant because a latch is an implementation of a wait-on element. This wait-on element can be identical to the data latches used, as illustrated at the top of Figure 13, thus providing an excellent delay model of the data propagation delay through the data latches. A second observation is that the XOR element found in Figure 12 also appears in the circuit of Figure 17. The output of this upper XOR element generates the $L+$ latch control signal when the input request R_i has propagated via the wait-on element to R_o . However, the $L+$ event can be generated slightly earlier, by tapping off from the R_i signal with the lower XOR in the diagram. Deriving the $L+$ signal from input R_i places an additional delay constraint on the bundling of control signal R_i with valid input data. This new constraint is that the data has propagated through the pass-gates of the data latch before $L+$ is produced via the lower XOR and the AND and OR gates. Meeting this extra delay constraint presents no difficulty. The AND gate is present to ensure that the data latches have become fully transparent before the R_i signal is permitted to make them opaque with an $L+$ signal. In this implementation there is a further delay constraint to be satisfied: $G+$ must not arrive before R_o is generated, as otherwise the arrival of $G+$ could turn off the wait-on and prevent R_o from ever being generated—resulting in a deadlock. In practice this is an easy delay constraint to meet, as there is only the wait-on delay from R_i to R_o , while there is substantially more delay from R_i to $G+$, even if the XOR, AND, and OR gates are combined into a single complex gate. In the implementation this delay constraint implies that states 17, 18, and 19 of the specification given in Figure 16 cannot be reached, because in state 16 the $G+$ event cannot occur. Although this implementation does not exploit the full

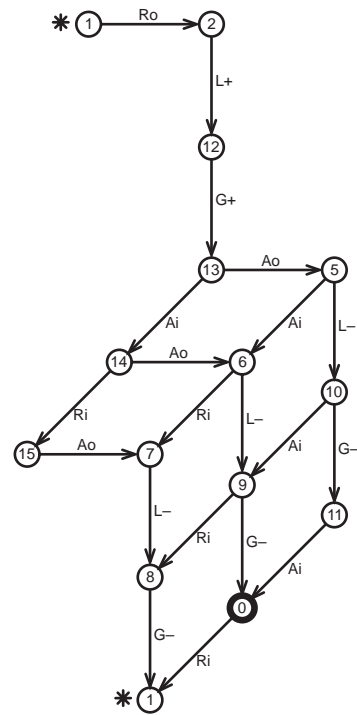


Figure 18. Specification for the Oversimplified Charlie Box, S.

concurrency permitted by the specification, we chose this design for its compactness and speed.

Next I will describe the Oversimplified Charlie Box, “S.” By specifying slightly less concurrency than in the Simplified Charlie Box described above, an even simpler and faster latch control circuit results, as shown in Figure 18. The reduced concurrency is a consequence of applying the additional delay constraints that 1) in the circuit, $L+$ must follow R_o , and 2) in the environment, $G+$ must occur before A_o can be received, i.e., the receiver of the output data must be slower than the latching process. This first delay constraint is easily met by making $L+$ dependent upon R_o , while the environmental delay constraint can be met by ensuring that the current latch stage is followed by an identical stage, or one that is slower. This follows from the specification that A_i in the following stage can be issued only after its $G+$ has been received, and A_i in the following stage becomes A_o in the current stage. Thus if the latches latch the data in the current stage before it is latched in the following stage, then this delay constraint is met. Although there is only slightly less concurrency permitted in this specification, the circuit implementation is quite a bit smaller, as seen in Figure 19. Although this circuit resembles the basic latch control circuit shown in Figure 12, it offers substantial speed improvement because R_o is generated concurrently

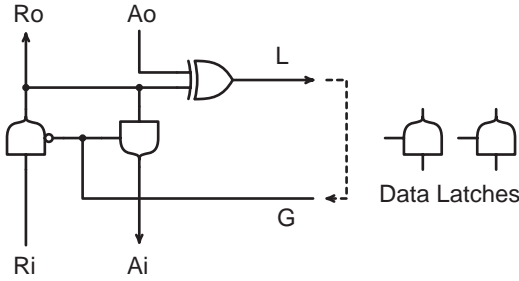
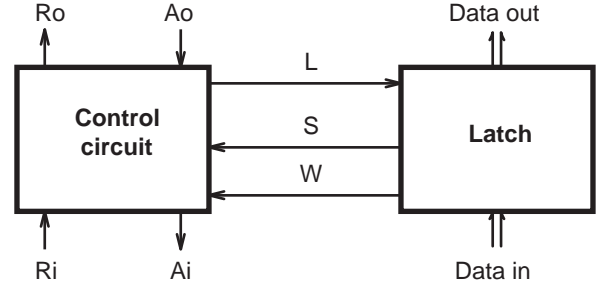


Figure 19. Oversimplified Charlie Box circuit, S.

with the data latches changing from transparent to opaque. Furthermore, there is better control signal and data bundle matching because the Ri-to-Ro control path flows through a wait-on element, which can be implemented by exactly the same circuit as the data path latch elements. Of course, Charlie’s observation that latching does not change the data, but unlatching does, suggests that the circuit of Figure 12 can be altered so that the XOR inputs are attached to the Ao input and the C-element output, appropriately delayed by the data propagation delay through the pass-gates of the latch rather than the propagation delay to the outputs of the latch.

The final Charlie box circuit, Charlie’s starting point, is the Full Charlie Box, or the extra extra large design, “XXL.” In this design, Charlie was seeking the maximum concurrency between the control circuit operation and the data movement. Furthermore, in our Counterflow pipeline application, data items could be deleted from the pipeline. An item was deleted in a stage by acknowledging an Ri input by an Ai without latching the data or generating an Ro output. The previous stage would thus receive its Ao very shortly after having sent its Ro output. The Full Charlie Box has the additional feature that it permits this early acknowledgment of the output data before the latch has completed latching the data. In response to such an early acknowledgment, the control circuit can abort the latching process. For a wide data path, where changing the latch from transparent to opaque, and from opaque to transparent, is a substantial portion of the latch operation cycle, early abort of the latching process can offer substantial speed improvement, because the current stage can immediately issue an acknowledgment of the input data with Ai.

To achieve this extra capability, an additional acknowledgment signal is required back from the latch circuit to the control circuit, as illustrated in Figure 20. The two acknowledge signals, S and W, convey the following messages: In response to L+, which causes the latch to go opaque, W+ announces the acknowledgment of L+ and conveys the message that the latch is no longer transparent. S+ is issued when the latch has become opaque and the



- L+ causes latch to go opaque
- W+ announces latch is no longer transparent
- S+ announces latch is opaque
- L- causes latch to go transparent
- S- announces latch is no longer opaque
- W- announces latch is transparent

Figure 20. Feedback Signals S and W from Latch.

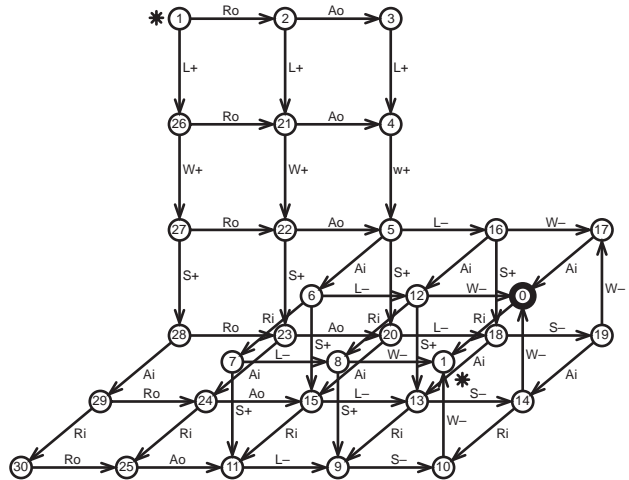


Figure 21. Specification for the Full Charlie Box, XXL.

data are latched. Similarly, an L- causes the latch to go transparent. S- is issued when the latch is no longer opaque, and W- is issued once the latch has become transparent. If the output data are acknowledged by Ao before the latch has announced that the data are latched, by signal S+, then the latching process can be aborted, permitting the shorter latch-unlatch cycle of L+; W+; L-; W-. These long and short latch cycles are illustrated by the waveforms in Figure 20.

The interface state graph (ISG) specification of this latch controller is given in Figure 21. Notice that this graph

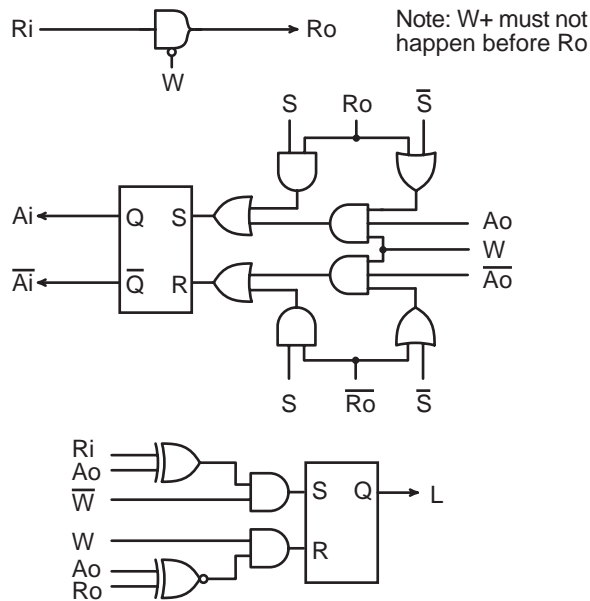


Figure 22. Full Charlie Box circuit, XXL—Flip-Flop implementation.

exhibits quite a bit more concurrency than the Simplified and Oversimplified Charlie Box specifications given in Figures 16 and 18, respectively. One event sequence in this graph that illustrates the latch abort feature is: starting at state 0, R_i ; R_o ; A_o ; $L+$; $W+$; $L-$; $W-$; A_i . An implementation of a circuit that conforms to this ISG is shown in Figure 22. This implementation was derived by-hand using a combination of EISG [4] and Unger Synthesis [3] methods and is not a particularly fast implementation. Note also that this implementation, like the Simplified Charlie Box circuit implementation, “M,” requires that $W+$ not happen before R_o is issued. A faster circuit implementation, given in Figure 23, uses a wait-on element to produce the A_i output rather than producing A_i from the output of an S-R flip-flop. With this implementation also, $W+$ must not happen before R_o is issued. Thus R_o can be generated whenever the latch is transparent, i.e., W is L_o . There are two conditions when A_i can be issued: either when the data are latched, or upon an early abort of the latching. These conditions appear in the circuit implementation as the control signal that enables the wait-on element that generates A_i . Thus A_i can be issued when either S is H_i , i.e., the latch has latched and is opaque, OR when an early A_o has been received, i.e., R_o and A_o have the same level, AND the latch is no longer transparent, i.e., W is H_i . There are also two conditions when the latch should be driven/held opaque: it can be driven opaque when fresh input data are available AND the latch is transparent, OR the latch must be held opaque until

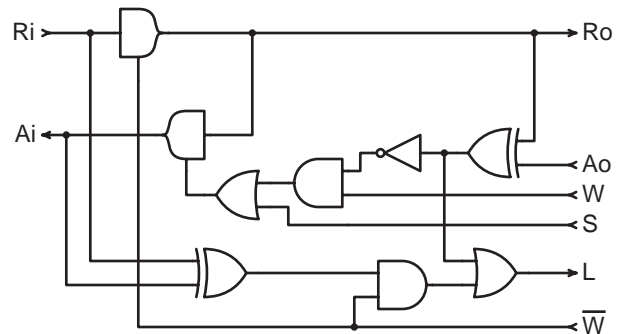


Figure 23. Full Charlie Box circuit, XXL—Wait-On implementation.

the output request R_o has been acknowledged by an A_o . These conditions appear in the circuit implementation of Figure 23 as the logic gates that generate the latch control signal L .

5. Conclusions

Charlie’s belief that fast asynchronous circuit modules must rely upon the use of local delay matching is now becoming more widely accepted among other asynchronous circuit designers. This is being helped by the development of supporting design tools, as well as by results from the “ultimate verification method” of building and measuring the performance of real chips. Charlie believed in looking at a circuit from a fresh perspective and re-evaluating any assumptions that might apply to the use of the circuit. Often such assumptions are so deeply rooted in design practice that we have forgotten that they are, after all, merely assumptions, and instead treat them as immutable facts. An example of this is Charlie’s observation that the data become valid at the output of a latch before the latch has been latched. Charlie also strongly felt that there was much benefit in viewing the behaviour of asynchronous circuits from a dynamical systems perspective. His propellor arbiter design is a direct outcome of this belief. Charlie thought of synchronisers and arbiters as time amplifiers, and thus came up with the idea of using arbiters for making accurate delay measurements. This is important not only for local delay matching, but also for comparing the delay values obtained from simulation with real measurements obtained from experimental chips. Charlie’s delay measurement technique enables circuit designers to explore the edges of their designs in both the asynchronous and clocked domains.

6. References

- [1] C. L. Seitz; Chapter 7: System Timing in *Introduction to VLSI Systems* by Carver Mead and Lynn Conway, Addison-Wesley, 1980.
- [2] D. F. Wann and R. A. Ellis; Implementation of Time Dependent Algorithms in Asynchronous Computer Systems, *Technical Memorandum 32*, Computer Systems Laboratory, Washington University, July, 1967.
- [3] S. H. Unger; *Asynchronous Sequential Circuits*. Wiley-Interscience, New York, 1969.
- [4] C. E. Molnar, T. P. Fang, and F. U. Rosenberger; Synthesis of Delay-Insensitive Modules, *Proceedings of the 1985 Chapel Hill Conference on Advanced Research in VLSI*, 1985.
- [5] C. E. Molnar (Chief Editor); *Macromodular Computer Design, Final Report*, Computer Systems Laboratory, Washington University, St. Louis, Missouri, 1974.
- [6] I. E. Sutherland; Micropipelines, *Communications of the ACM*, Vol 32. #6, p. 720-738, June 1989 (1988 Turing Award Lecture).
- [7] *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Async94, Salt Lake City, Utah, IEEE Computer Society Press, November 3-5, 1994.
- [8] R. F. Sproull, I. E. Sutherland, and C. E. Molnar; The Counterflow Pipeline Architecture, *Design and Test of Computers*, IEEE, Fall 1994.