AN ABSTRACT OF THE THESIS OF

Donald C. Kirkpatrick for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented 25 April 1985.

Title: Design of Self-Synchronized Asynchronous Sequential State Machines Using Asymmetrical Delay Elements

Abstract approved: Redacted for Privacy
V. M. Powers

A design style is presented for a self-synchronized, multiple input change, asynchronous state machine which processes input changes at a speed limited only by the required machine behavior and implementation technology. This state machine will operate at this ultimate speed because of a new asynchronous delay element with unequal rising and falling propagation delays. This new delay element is used in a clock generator circuit which monitors the machine's inputs to generate a clock pulse for each input state change. Two new functions, based on the machine's required behavior, are defined for a multiple output change machine. The first function specifies the time between intermediate state transitions in a multiple output change sequence. The second function indicates when the next state is a final stable state. The clock generator, new delay element, and new functions are used in two design examples. This design style is extended to the unbounded input change mode, pulse mode, and speed independent mode.

Design of Self-Synchronized Asynchronous Sequential State Machines Using Asymmetrical Delay Elements

by

Donald C. Kirkpatrick

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirement for the
degree of

Doctor of Philosophy

Completed 25 April 1985

Commencement June 1985

APPROVED:

Redacted for Privacy

Associate Professor of Electrical and Computer Engineering
in charge of major

Redacted for Privacy

Head of department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented 25 April 1985

Typed by researcher for Donald C. Kirkpatrick

TABLE OF CONTENTS

LIST OF FIGURES

DESIGN OF SELF-SYNCHRONIZED ASYNCHRONOUS SEQUENTIAL STATE MACHINES USING ASYMMETRICAL DELAY ELEMENTS

INTRODUCTION

The logical structure and design style selected for an asynchronous sequential state machine implementation will affect the performance of the resulting circuit realization. An optimum structure and design style will result in the ultimate speed of the final circuit being limited only by required machine behavior and implementation technology.

This dissertation emphasizes design of asynchronous state machines operating in multiple input change mode. In general, such machines cannot be realized without delay elements (Friedman and Menon, 1968). A new delay element, with unequal delay of the rising and falling edges, is used in a circuit to monitor a machine's inputs and, when a change is detected, generate a clock pulse. This design style is shown to be an optimum solution, permitting simple design techniques, yet requiring little added circuitry. This design style is extended to operate in unbounded input change mode, pulse mode, and speed independent mode.

The Problem

One of the most compelling rationales for embarking upon an asynchronous design is to maximize the operating speed of a sequential state machine. The measure of operating speed will be the maximum rate at which the machine can process input state changes. The ultimate operating speed of any asynchronous machine is bounded by the fundamental limitations imposed by the required machine behavior and implementation technology. The machine is required to perform a sequence of transitions as determined by its behavioral description, and these transitions proceed at a pace which is limited by the speed of the circuits used to realize the design. Except for a normal fundamental mode machine, every design methodology presented to date imposes additional restrictions on operating speed beyond these fundamental limitations. The task is to develop a design style wherein the ultimate operating speed is limited only by these fundamental constraints.

Previous sequential machines do not achieve this ultimate speed. For the multiple input change mode machine, the choice has always been either a fundamentally flawed structure that can never reach the ultimate speed, or a structure that could achieve ultimate speed but is prevented from doing so by limitations of available delay elements.

The search for a circuit realization that will achieve the ultimate operating speed can be divided into two phases. The first is a careful analysis of the possible machine structures to determine which have the potential to realize this ultimate operating speed. The second is the development of a method to design and augment the structure as required so that the final circuit realization does in fact achieve this ultimate operating speed.

Motivation

Technology is continually improving; operating speeds that were only dreams yesterday are commonplace today. These gains should not be squandered on a mediocre machine design or implementation strategy. Achieving ultimate operating speed is especially important to the test equipment manufacturer. His customers are building newer and faster circuits every day. The manufacturer must stay one step ahead so he can offer his customers products capable of testing and measuring their circuits.

Quite often the interface between digital test equipment and the customers circuit is asynchronous; the customer's circuit and the test equipment each have their own clocks. This kind of interface can be most difficult because of interactions between the two different clocks. The test equipment manufacturer strives to squeeze all the

speed possible (consistent with other goals such as cost) into his equipment to maximize his potential market.

Why Self-Synchronized Asynchronous Design

There exists a class of design problems that can only be solved using asynchronous design methods. In many practical problems, the clock pulse that characterizes synchronous design is not available. Even when it is, greater overall speed can sometimes be achieved by designing asynchronous sub-circuits. The interface between two synchronous circuits with different clocks is always an asynchronous design problem. For problems where speed is critical, an asynchronous machine has the distinct advantage of not being required to wait for the next clock pulse.

Synchronous machines have many advantages over asynchronous machines. By using a self-synchronized asynchronous design, the inherent speed advantage of an asynchronous machine is retained while the advantages of efficient state assignment and logic reduction normally associated with synchronous machines is obtained.

The state assignment process involves designating a unique state-variable value for each state of the machine. Any state assignment imposes structure on the machine (Hartmanis and Stearns, 1966) and influences the logic complexity (Kohavi,1978), but for a synchronous design,

proper operation of the resulting machine will result with any state assignment. However, a necessary condition for proper asynchronous machine operation is a proper state assignment (Liu, 1963; Tracey, 1966; Tan, 1971). By using a self-synchronized design, the state assignment problem is transformed to the synchronous case (Chuang and Das, 1973) and failures due to improper state assignment are avoided.

A necessary condition for proper operation of an asynchronous machine is the proper design of the transition function combinational logic (Unger, 1969). A proper design requires the addition of logic gates to an otherwise minimal circuit for the sole purpose of suppressing spurious output pulses. A synchronous machine is unaffected by these spurious pulses because they are not present when the clock occurs. Again, self-synchronization transforms this asynchronous design problem into a synchronous problem and renders these additional logic gates unnecessary.

The price for this design simplification and hardware complexity reduction is the addition of a clock generator. In the following chapters, previously proposed clock generators are discussed, their assumptions, advantages, and limitations are presented, and their timing requirements are analyzed. Once the problems are explored, an optimum clock generator is presented. The self-synchronized asynchronous machine is then extended to operate in the unbounded input change mode, pulse mode, and speed independent mode.

BASIC CONCEPTS AND DEFINITIONS

A physical circuit can be abstractly modeled using the mathematical concepts of sets and mapping functions.

Definition: A sequential machine, M, is a quintuple,

$$M=(S,I,O,\delta,\lambda),$$

where:

i) S is a finite nonempty set of internal states.

ii) I is a finite nonempty set of input states.

iii) O is a finite nonempty set of output states.

iv) $\delta:S\times I\rightarrow S$ is called the transition function.

v) $\lambda:S\times I\rightarrow O$ is called the output function.

When the output function is of secondary importance, the abstract model can be simplified.

Definition: A state machine, M, is a triplet,

$$M=(S,I,\delta),$$

where S, I and δ are defined above.

The input signal combination presented to the machine is called the input state. The output signal combination

produced by the machine is called the output state. The state variable signal combination is called the internal state. The individual input, output, or internal state variable signals themselves will be referred to as inputs, outputs, or state variables. Together, the internal and input states form the total state (or just state). This notation can be clarified by examining the Huffman-Moore model of a finite state machine shown in Figure 1.

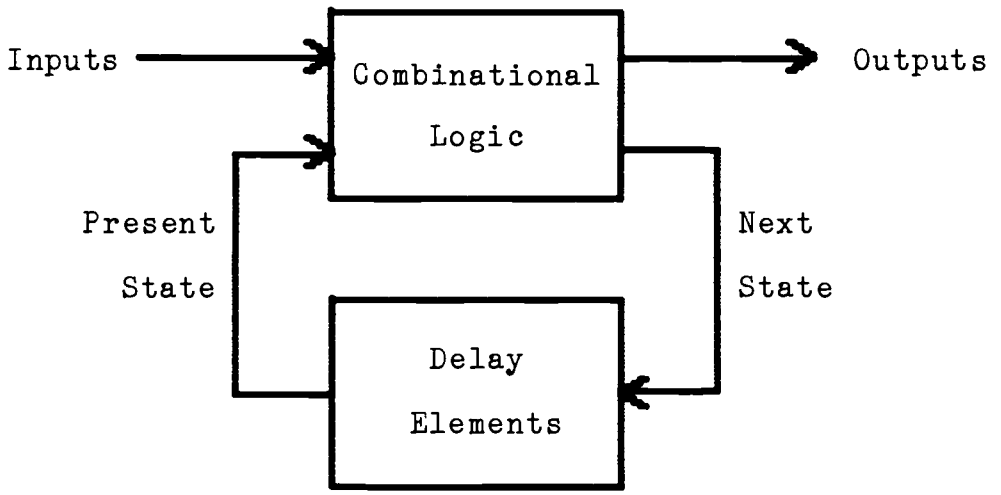


Figure 1

Huffman-Moore Model Finite State Machine

This model is completely general; any sequential state machine can be built in this form. The combinational logic block contains no memory and the delay element block contains only memory devices. As indicated in the introduction, the state assignment must be based on the transition function or the machine can malfunction. Machines

with a special kind of transition function can be built delay free (utilizing only the stray delays), but the vast majority of machines require at least one delay element.

A sequential machine may be designed to operate in one of six basic modes. These modes are characterized by the constraints placed on the input signals.

> Definition: The input signals of a synchronous machine are permitted to change only during the period between changes of a special input signal called the clock. All input signals must be stable for a specified time prior to a change in the clock and must remain stable for a specified time after the clock changes.

> Definition: Only one input of an asynchronous - single input change machine (SIC) is allowed to change at a time. Consecutive input changes must be separated by some specified minimum time.

> Definition: Several inputs of an asynchronous - multiple input change machine (MIC) are permitted to change within a specified period of time. The state machine is to consider all these changes to be simultaneous. After this first period, no further input changes are permitted for a second time period while the machine processes this input.

Definition: Any input of an asynchronous - unrestricted input change machine (UIC) may change at any time. Sometimes the mild restriction that the same input may not change twice within some minimum time is included.

Definition: For a pulse mode machine, the input changes always occur in pairs. The same input signal must change twice within some specified minimum time. Each input state corresponds to a single input change pair (pulse) and consecutive input states must be separated by a specified minimum time.

Definition: Input changes for a speed independent machine are permitted only when the machine indicates, through special outputs, that it is ready to accept the next input change.

If the restrictions and conditions are all properly met by the circuit providing input to the sequential machine, then all is well and the machine will operate as it should. If, for some reason, the circuit providing the inputs does not meet the constraints, unpredictable operation will result. The sequential machine could produce spurious output pulses, wrong output states, or go to the wrong next state. It is impossible to predict a priori the effects of assumption violations.

In addition to the six basic modes of operation, an asynchronous sequential machine may be classified into one of three categories based on its output behavior.

Definition: A machine is classified as single output change (SOC) if no more than one output state change results from every single input state change.

Definition: A machine is classified as multiple output change (MOC) if at least one input state change produces more than one output state and there exists some fixed upper bound on the number of output state changes that result from every single input state change.

Definition: A machine is classified as unbounded output change (UOC) if there is no uniform finite number bounding the number of output state changes that result from one or more single input state change.

There is only one output state for every total state of the sequential machine. If a sequential state machine is to produce more than one output state in response to a single input state change, the sequential machine must perambulate through a sequence of internal states. There will be one internal state for each output state. For both SOC and MOC operation, the machine will always reach a final stable

total state. If the transition function maps a total state to its own internal state, that total state is stable. The UOC behavior results when no final stable total state exists for a given input state.

> Definition: A machine operates in fundamental mode if a final stable state is always reached between input state changes.

Non-fundamental mode operation will not be considered herein; it will be assumed all state transition sequences terminate in a final stable state. A machine operating in fundamental mode with single input change and single output change is said to be a normal fundamental mode machine.

The process of encoding the states of a machine as binary numbers is called state assignment. Choosing a state assignment for a synchronous machine influences the complexity of the combinational logic, but proper operation of the machine is not affected.

An asynchronous sequential design using level-sensitive logic suffers from several potential failure modes that are not found in synchronous designs (Unger, 1969). In asynchronous level-sensitive design, an improper state assignment may cause the machine to fail to reach the proper next state. An asynchronous machine is said to have a critical race if the proper operation of the machine depends upon the relative speed of the state variable changes.

There exists a class of asynchronous machines for which a race-free state assignment is a necessary, but not sufficient, condition for proper operation. Machines of this class have an essential hazard - so called essential because its presence or absence is determined by the machine's required functional behavior. The final circuit realization must have at least one delay element to assure proper operation. A machine has an essential hazard if any state requires the following behavior; starting in state s, the input changes to x and the machine reaches a new total state. If this new total state is not re-established by now changing the input to what it was in s and back to x again, then the machine has an essential hazard.

Even when a proper state assignment is chosen, the machine may still fail to achieve the proper next state due to delays in the combinational logic. These delays will cause spurious output pulses in the transition function logic unless additional circuits are introduced to suppress them. Any combinational logic realization that has the potential for spurious outputs is said to have a logic hazard.

Synchronous machines do not suffer from malfunctions due to critical races, essential hazards, or logic hazards. There is no clock pulse when these spurious output pulses occur or when the machine would be susceptible to a critical race or essential hazard. It would be advantageous to

transform an asynchronous machine into a synchronous machine yet retain the asynchronous speed advantage. This is possible if the machine can be built to generate its own clock pulse at the appropriate time. The entire self-synchronizing problem then revolves around generating this clock pulse without giving up the inherent speed advantage an asynchronous machine has over a synchronous machine.

REVIEW OF LITERATURE

The foundation of sequential state machine analysis and design was set in place by three classic papers. D. A. Huffman presented the first orderly method for state machine synthesis (Huffman, 1954). He introduced the flow chart concept and presented a method for reducing the number of storage elements. E. F. Moore published his studies of the abstract properties of sequential machines (Moore, 1956). He had independently developed essentially the same method for reducing the number of storage elements. G. H. Mealy presented a formal procedure for the synthesis of sequential machines (Mealy, 1955). From these three papers come the basic concepts of sequential machines - present state, next state, inputs, outputs, state equivalence, flow charts, state diagrams, Huffman-Moore Model, Moore Machine, Mealy Machine, and much more.

In the late 1950's, D. E. Muller and W. S. Bartky developed the theory of speed independent circuits. Numerous papers were presented at conferences and published. These papers have been condensed into chapters in the texts (Unger, 1969; Miller, 1965). The original papers are reportedly not easy to read. The chapters in these texts

were the source for the speed independent information presented herein.

The importance of circuit delays was known very early. S. H. Unger proved that if the required machine behavior has an essential hazard, then there is no delay free realization that will operate properly under all conditions (Unger, 1959). He then proved any circuit can be realized hazard-free with at most a single delay element (under the single input change assumption).

The state assignment is crucial to proper asynchronous state machine operation. C. N. Liu published a method of state variable assignment for asynchronous circuits. A Liu assignment solves the critical race problem while allowing concurrent changes in the state variables (Liu, 1963). He also was the first to prove the conditions necessary for a race-free assignment. J. H. Tracey improved upon the Liu assignment (Tracey, 1966). The advantage of a Tracey assignment is that the number of state variables is bounded above by the number required for the Liu assignment; in many cases the number required is less. C. J. Tan extended the work of Liu and Tracey (Tan, 1971). A Tan assignment results in reduced complexity for the transition function combinational logic at the expense of additional state variables.

In addition to the Liu, Tracey, and Tan assignments, there are several "universal" state assignments possible

(Unger, 1969). All of these state assignments suffer the same problem. The number of state variables required to make a race-free assignment is usually greater than the minimum number required to encode the states of the machine.

S. H. Unger published the first formal definition of proper behavior for a machine operating in the unbounded input change mode (Unger, 1971). The paper then detailed the design of a Huffman-Moore machine to function properly in this mode.

As a first step toward a self-synchronized machine, D. Friedman and P. R. Menon published the first practical solutions to the problem of multiple input change mode design (Friedman and Menon, 1968). This paper demonstrated that any circuit operating in multiple input change mode has a hazard-free realization with, at most, a single delay element. Three solutions are presented: source box in the input path, Huffman-Moore design with proper state assignment, and delay box in the feedback path.

J. G. Bredeson and P. T. Hulina published the first method to use the input transitions to generate a self-synchronizing clock pulse (Bredeson and Hulina, 1971). This paper describes how the normal problems of critical races and logic hazards are avoided for a self-synchronized asynchronous machine. The design method in this paper is strictly limited to single input change mode.

A solution to the multiple input change problem took

another two years to surface. H. Y. H. Chuang and S. Das published a method for designing a self-synchronized machine operating in the multiple input change mode (Chuang and Das, 1973). A short time later, C. A. Rey and J. Vaucher published another method for designing a self-synchronized machine (Rey and Vaucher, 1974). This design used a general purpose clocking circuit and allowed multiple output changes. The delay elements were digital differentiators and monostable multivibrators.

The two machines of Rey and Vaucher and Chuang and Das were compared by Unger to the Huffman-Moore machine (Unger, 1977). Unger found several problems with the Rey and Vaucher machine and suggested an improved clock generator in the same spirit. This paper was primarily concerned with asynchronous machines operating in the UIC mode. Unger demonstrated that the Huffman-Moore approach was suitable for UIC operation if a proper state assignment was made, but all previous self-synchronized approaches are unsuitable for UIC operation.

The state transition function can be realized using many different kinds of logic components. A read only memory device has been a popular choice in synchronous designs for many years, but was not used in early asynchronous machines due to possible spurious outputs. H. A. Sholl and S. C. Yang published the first asynchronous machine using memory devices to realize the transition function (Sholl and Yang,

1975). The design is not self-synchronized; the unavoidable memory access delay is used to control critical races. Memory devices are also attractive because of the inherent matching of delays. B. Thomas and P. C. Chandrasekharan presented a design methodology using the matched delays in memory devices (Thomas and Chandrasekharan, 1981).

There are several structural configurations that an asynchronous machine can assume. J. L. Huertas and J. I. Acha were the first to recognize this and they published three models for self-synchronizing asynchronous machines (Huertas and Acha, 1976). This paper is the first to use a comparator in the clock generator. G. L. Chiesa published a method of constructing a larger asynchronous machine from a collection of smaller self-synchronized asynchronous machines (Chiesa, 1979). A. B. Hayes published the first self-synchronized machine to operate in the speed independent mode (Hayes, 1981).

The above references provide a historical path from the beginnings of switching theory to the present. There are also several texts that provide a valuable reference source. The key ideas in these texts were first presented in the papers previously mentioned herein, but the texts provide a context and unity not possible in these individual papers.

One of the earlier texts was written by J. Hartmanis and R. E. Stearns in 1966. This text models the structure of sequential machines using the mathematics of groups. It is

this text that serves as the guideline for the mathematical notation used herein. Another such text was written by H. S. Stone in 1973. Both texts are invaluable references for any work relating algebraic structures to machine architectures.

An early two-volume text on sequential machine design was written by R. E. Miller. Published in 1965, it is a remarkably complete text on switching theory. The second volume contains one chapter on asynchronous switching networks and one chapter on speed independent circuit theory. These two chapters formed the most complete reference work on asynchronous circuits and machines prior to the classic text written by S. H. Unger and published in 1969. There still is no better text on asynchronous design than Unger. He covers virtually every design aspect with many theorems and proofs. Since his book was published, the major advances in the field have been in synthesis of unbounded input change mode machines and self-synchronized machines.

There are several other texts that have a chapter on asynchronous design: F. J. Hill and G. R. Peterson, Z. Kohavi, C. R. Clare, and W. I. Fletcher for example. These texts have little to offer beyond what is already in Unger or Miller.

THE DELAY ELEMENT

In the real world, all circuits exhibit delay. When a machine is designed under the synchronous assumption, the time between successive clock pulses must be greater than the sum of the delays in the circuit. If this condition is met, the machine will operate as if all the components are ideal. In the asynchronous machine, the delays must be carefully analyzed or malfunctions may occur.

Delay Element Types

Circuit delays may be either deliberately introduced or unavoidable due to physical device characteristics.

Definition: A stray delay is the unavoidable delay cause by the physical limitations of the device.

Definition: A delay element is a delay that has been deliberately introduced by the designer.

Definition: A delay-free circuit is one that has only stray delays.

Delay-free does not mean the circuit has no delay. It just means there are no intentionally introduced delays. In general, the location or magnitude of a stray delay is not assumed to be known. However, the upper bound on the magnitude of the stray delay is specified.

In all previous work, three types of delay elements have been used.

> Definition: A pure delay only transforms or shifts the input signal in time by amount D.

> Definition: An inertial delay outputs a signal only after it has persisted for the delay time D.

> Definition: A monostable multivibrator outputs a pulse of fixed duration D in response to a positive (or negative) input transition.

The pure delay only shifts the input signal in time. It is best approximated in the real world by a transmission line. The inertial delay not only delays the input by amount D, but it also filters the input. Any pulse of duration less than D does not propagate to the output. A monostable multivibrator is a delay in the sense it produces an edge a fixed time after a reference edge. The multivibrator is also known as a one-shot. If the multivibrator time period can be extended by additional edges occurring before the end of the period, then the multivibrator is said to be retriggerable.

Other kinds of delay behavior are possible. One such behavior introduced here is called an asymmetrical delay.

> Definition: An asymmetrical delay output changes to a one (or zero) only after the input change has persisted for the delay time D. The opposite signal change is output without delay.

An asymmetrical delay operates as an inertial delay on one edge of a signal only. The other edge is ideally passed through with no delay.

Figure 2 compares the behavior of the four different types of delays on the same input waveform. All four delays have the same delay time (D). The monostable multivibrator (one-shot) delay and the asymmetrical delay are shown operating on the positive edge, but could just have easily been designed to operate on the negative edge.

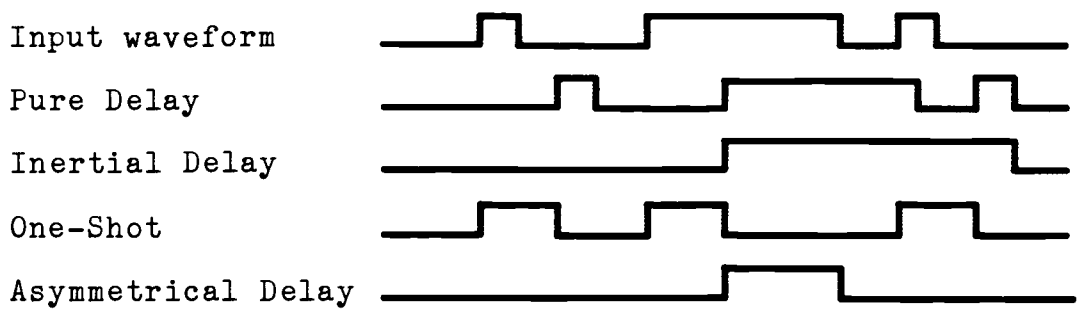


Figure 2

Delay Element Waveforms

Asymmetrical Delay Element Design

Several methods exist for creating an asymmetrical delay element. One such method is the resistor-capacitor-diode combination shown in Figure 3.

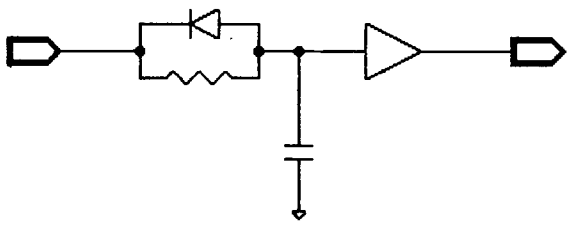


Figure 3

Simple Asymmetrical Delay

This method has the advantage of simplicity. However, there are serious limitations to its usefulness. If large delays are required, the capacitor value can be large enough that the driving circuit output impedance is important. The rise time of the capacitor voltage is slow so a buffer with hysteresis must be present. With a low input, the noise immunity of the circuit is degraded by the diode forward drop. With a high input, noise is more easily coupled into the node due to the high source impedance. The nominal delay time is difficult to control since it is a function of driving circuit output impedance, logic high voltage level, buffer threshold, and temperature.

Sometimes it is also desirable to make the delay programmable. One reason for making the delay programmable is to provide a method for calibrating the delay. A second reason is to increase the throughput of the machine by customizing the delay to the state of the machine. This second reason will be detailed later. The key to building a programmable asymmetrical delay is illustrated by the circuit in Figure 4.

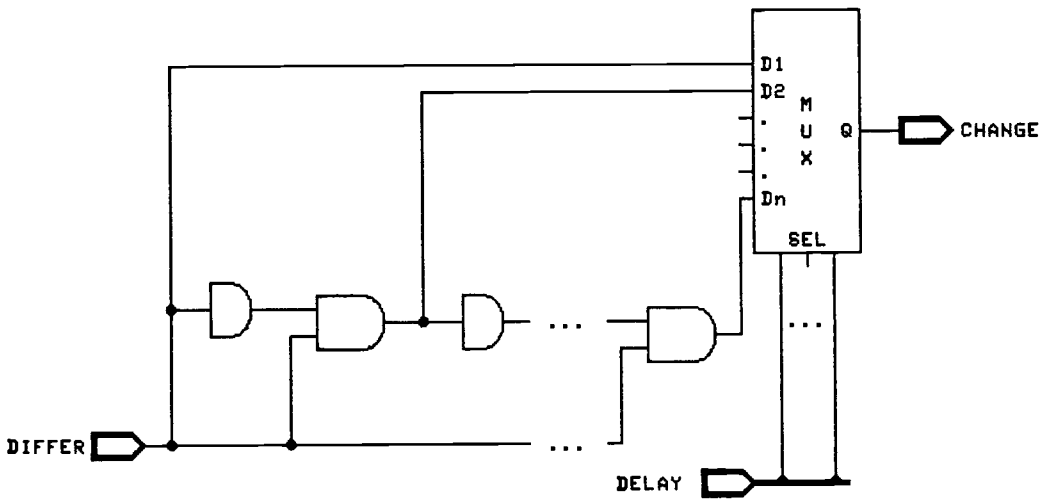


Figure 4

Programmable Asymmetrical Delay

For reasons which will be obvious later, the delay element input and output have been named DIFFER and CHANGE. The total delay is achieved through a series of smaller individual delay elements, shown above using the traditional "D" symbol. The delayed edge propagates using a serial path while the non-delayed edge propagates using a parallel path.

An individual delay element can be realized using the resistor-capacitor-diode circuit shown in Figure 3. AND gates serve as buffers. The delay is programmed using the binary vector DELAY to select the proper input on the N to 1 multiplexer.

This circuit solves many of the problems associated with the circuit shown in Figure 3. Since the total delay is spread over many smaller delays, the capacitors and resistors are smaller. This helps solve problems caused by noise and slow rise time. The input low noise margin is restored since the second input of each AND gate is connected to DIFFER. The scheme of Figure 4 was built and tested using 74F08's as the AND gates. With one exception, the performance was excellent. There was a significant variation in propagation delay with temperature due to the 74F08 input threshold drift. The actual drift was -4.4 millivolts per degree C. This produced a 20% variation in propagation delay over the commercial temperature range of 0 to 70 degrees C.

For many applications this amount of drift is not acceptable and another solution must be found. Such a solution is to replace the DELAY-AND gate string with a shift register, as shown in Figure 5 on the next page. A low on DIFFER holds the shift register in a reset condition. When DIFFER changes to a high, the reset is removed and the register begins to shift the high on the serial input down

the register. The accuracy of the delay is limited only by the accuracy of the oscillator. DIFFER also turns the oscillator on and off so the time to the first shift is known and constant.

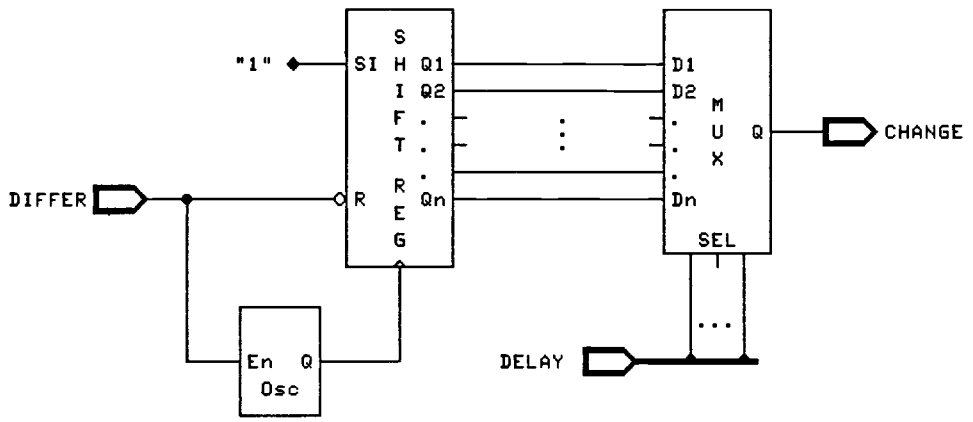


Figure 5

Improved Asymmetrical Delay

This structure is very suitable for delays much longer than would be reasonable for the resistor-capacitor-diode network. For extremely long delays, the shift register would be replaced by a counter. The counter would be preloaded with a count value and decremented by the oscillator. When the counter reached zero, the delay time would be over. A shift register is chosen here because it is extremely easy to decode the count. (A shift register, when used as a counter, is sometimes called a Johnson counter.)

TIMING ANALYSIS

Over the years, the following notation has evolved as conventional when writing timing expressions:

D : Delays through delay elements.

d : Stray delays through combinational logic.

s : Set-up times for flip-flops.

f : Propagation delays through flip-flops.

Subscripts M and m are used to represent maximum and minimum values respectively. This notation will be used throughout the timing analysis that follows.

One important specification for any asynchronous multiple input change machine is a time interval during which several input signals may change. The machine is to consider these input changes to be simultaneous. That is, these input signal changes are to be considered as only one input state change. Given this specification and the required machine behavior, a circuit is designed to realize the machine. One result from the design is the determination of a second time interval. The inputs must remain stable during this second interval while the machine perambulates from one state to the next state. If the inputs do not remain stable, unpredictable behavior will result.

Definition: $\underline{\delta_1}$ is the time interval following the first input change in which the other input signals are permitted to change. The machine is to behave as if all input changes occurring during this interval are simultaneous.

Definition: $\underline{\delta_2}$ is the time interval following δ_1 that the inputs must remain stable for the machine to properly change to the final stable state.

δ_2 starts with the end of δ_1 and separates groups of input changes. The minimum time between input state changes is the sum of the two intervals, $\delta_1+\delta_2$.

It is unfortunate that the traditional symbol for the next-state transition function, δ, is also the traditional symbol for the two time intervals. The subscript and the context should provide the key as to whether an interval or a mapping function is being referenced.

Huffman-Moore MIC Machine Analysis

The Huffman-Moore model was shown in Figure 1. As stated earlier, careful analysis of the transition function is required when an asynchronous machine is built based on this model using level-sensitive logic. This model has served as a basis for most timing analysis (Unger, 1969) and

has been the only basis for all previous unbounded input change design (Unger, 1971).

It has been shown (Unger, 1969) that, for proper MIC operation, changes to the present-state variables induced by the first input signal change must not reach the inputs of the combinational logic before all the changes induced by the last input change reach the combinational logic outputs. The earliest a change can reach the logic inputs is D_m+d_m while the latest a change reaches a logic output is δ_1+d_M. Thus the inequality,

$$D_m+d_m \geq \delta_1+d_M.$$

This results in a minimum delay element value of

$$D_m \geq \delta_1+(d_M-d_m).$$

When any machine generates multiple output states, it does so by perambulating through intermediate total states generating output states. If the inputs do not remain stable until the final stable state is reached, the fundamental mode assumption is violated. (Lift the fundamental mode restriction and the machine is in UIC mode.) The time between successive intermediate states (and thus successive output states) is determined by the propagation delay through the combinational logic block and the delay element. The time for one intermediate state transition (D+d) is bounded by a minimum of D_m+d_m and a maximum of D_M+d_M.

The last changes caused by the final input change of an input state, including any state variable change, must reach

the combinational logic outputs before the first change of the next input state. If n is the number of intermediate internal state transitions required to produce all the output states, then

$$\delta_2+d_m \geq d_M+n(D_M+d_M).$$

Thus the time between input states must satisfy the inequality

$$\delta_1+\delta_2 \geq \delta_1+n(D_M+d_M)+(d_M-d_m).$$

The term for the maximum time between intermediate states in the expression above (D_M+d_M) can be reduced by D_m if transient spurious pulses on the outputs can be tolerated. Transient spurious next-state outputs of duration less than D_m can be filtered by the delay elements and the proper next state will still be reached. Intentionally designing a machine with transient spurious outputs is not in the spirit of the work presented herein.

If the machine is designed to operate in single output change mode, then n=1. If the transition function has no essential hazards, then state assignments exist (Tracey, 1966) that can result in a delay-free realization ($D_M=0$). For any level-sensitive Huffman-Moore machine, a proper state assignment must be found. This assignment is customized, based on the transition function, using the techniques developed by Liu and others.

Self-Synchronized SOC Machine

While the Huffman-Moore model could be used to describe a self-synchronized machine, it is better to augment the model slightly as shown in Figure 6.

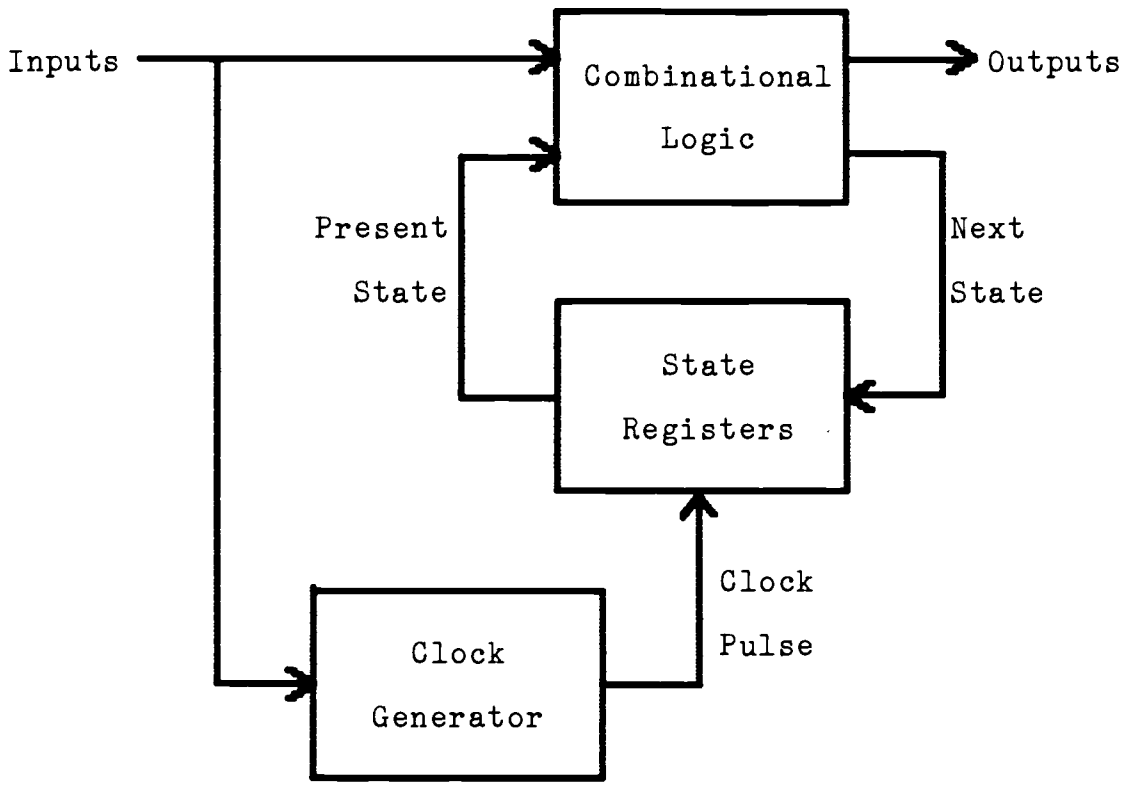


Figure 6

Self-Synchronized Asynchronous State Machine

The generalized delay elements have been replaced by edge-triggered flip-flops organized as state registers and a clock generator block has been added. With proper clock generator design, only one delay element is required (inside

the clock generator). This delay element is used to properly time the pulse edge that clocks the flip-flops. The first self-synchronized machine was built in just this fashion (Bredeson and Hulina, 1971). It operated only in normal fundamental mode. This model can realize a multiple input change machine, but is not suitable for a multiple output change machine since there is no way for the clock generator to determine when the final stable state has been reached.

For any self-synchronized machine, the pulse edge that clocks the flip-flops must not affect the flip-flops before the input changes propagate through the transition function combinational logic and set up the flip-flops. The minimum delay through the combinational logic of the clock generator and delay element is $D_m+d'_m$, where d' refers to the delay from the input through the combinational logic to the clock generator delay element. The maximum delay through the combinational logic and flip-flop set-up time is d_M+s. This results in the restriction

$$D_m+d'_m \geq \delta_1+d_M+s,$$

and a minimum clock generator delay value of

$$D_m \geq \delta_1+d_M+s-d'_m.$$

The clock pulse caused by the first input change of the next input state must not reach the flip-flops before the last state variable changes caused by the previous input state reach and set up the flip-flops. Thus the inequality,

$$\delta_2+D_m+d'_m \geq D_M+d'_M+f_M+d_M+s,$$

and the minimum time between input states for a SOC self-synchronized machine is given by

$$\delta_1+\delta_2 \geq \delta_1+f_M+d_M+s+((D_M+d'_M)-(D_m+d'_m)).$$

It is now possible to compare the speeds of the self-synchronized machine and the level-sensitive Huffman-Moore SOC machine (n=1 state transition). Each expression can be decomposed into two parts: a minimum combinational logic delay term plus a propagation delay uncertainty term. Assuming equivalent technologies, the combinational logic delays (d_M) should be equal. The uncertainty term is simply the difference between the fastest and slowest state variable change (d_M-d_m) or clock pulses ($(D_M+d'_M)-(D_m+d'_m)$). The magnitude of the two uncertainty terms should also be nearly equal for equivalent technologies. The two machines operate at the same speed when the right hand side of the input state timing inequalities are equal. Equating the two right hand sides and canceling these approximate equalities results in

$$D_M=f_M+s.$$

Examining this expression leads to the following conclusions. The Huffman-Moore machine will always be faster if the machine does not have an essential hazard since there exists a delay-free ($D_M=0$) state assignment (Tracey, 1966). The flip-flop set-up time (s) and propagation delay (f_M) for the self-synchronized machine are technology dependent constants, but D_M for the Huffman-Moore machine

increases with δ_1. Conclusion: the greater δ_1, the greater the advantage for self-synchronization if the machine has an essential hazard.

It should be noted that f_{M+s} can be very small. Typical set up time values for common commercially available parts are 4 nanoseconds for a 74F374 or 1.4 nanoseconds for a 10H131. Typical values for the maximum propagation delay are 10 nanoseconds for a 74F374 and 2.1 nanoseconds for a 10H131.

Self-Synchronized MOC Machine

The model given in Figure 6 must be modified if the machine is to produce multiple output changes in response to a single input state change. The clock generator must have additional inputs to be able to determine if additional clock pulses are required. Without additional inputs, the clock generator can never determine when the final stable state is reached. The first of two methods for solving this problem is shown in Figure 7 on the next page.

The architecture of Figure 7 has appeared in the literature (Huertas and Acha, 1976), but no timing analysis or implementation method was given. This scheme has simplicity as a benefit. If the clock generator is provided with present-state and next-state information, it can determine if another state is to follow.

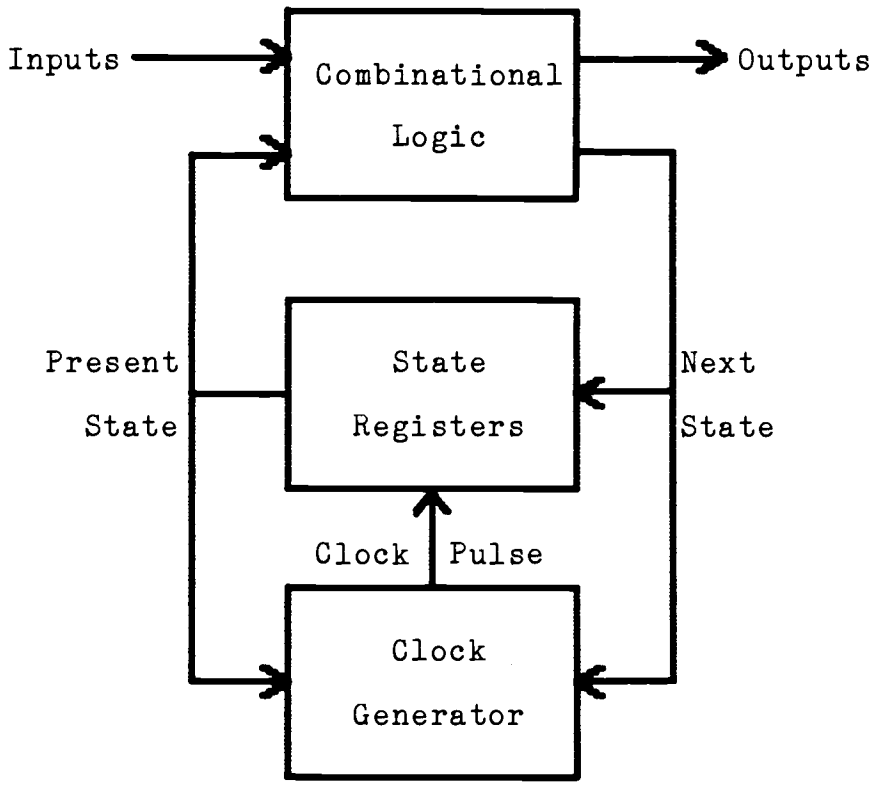


Figure 7

MOC Clock Generator - Present and Next State

Clock pulses must be generated until the final stable state is reached. This occurs when the transition function maps the total state to the present internal state. As in the previous case, the pulse edge that clocks the flip-flops must not reach the flip-flops before the input changes have propagated through the transition function combinational logic and reach and set up the flip-flops. The total delay from input change to clock generator delay element is d'_m. This includes the delay through the transition function combinational logic (d_m). As in the previous cases,

$$D_m+d'_m \geq \delta_1+d_M+s,$$

and the minimum clock generator delay value is

$$D_m \geq \delta_1+d_M+s-d'_m.$$

The maximum state transition time is the time for a signal to propagate through the flip-flops (f_M), the clock generator combinational logic (d'_M) and delay element (D_M). The clock pulse generated for the next input state must not reach the flip-flops before the last changes caused by the previous input state reach and set up the flip-flops. Thus the restriction,

$$\delta_2+D_m+d'_m \geq n(D_M+d'_M+f_M)+d_M+s,$$

and the minimum time between input states for this configuration is

$$\delta_1+\delta_2 \geq \delta_1+n(D_M+d'_M+f_M)+d_M+s-(D_m+d'_m).$$

However, this architecture must also meet an additional restriction which forces it to operate at less than the ultimate speed. The clock generator must be able to detect when one intermediate state transition is complete. Again, a clock pulse is generated (after a suitable delay) only when the next state changes to being different from the present state. Before another clock pulse can be issued, the logic must reach the condition of next-state and present-state equal. This condition may be very temporary for an intermediate state transition during a multiple output change perambulation.

The necessary condition that present-state equals next-

state can cause real problems if not properly met for a MOC machine. Consider what might happen if the minimum delay through the next-state and clock generator combinational logic is less than the maximum delay through the clock generator combinational logic. In that case, at time f_M after the clock pulse changes state s_i to s_j, state s_j appears at the inputs of both the transition and clock generator combinational logic. If the next state in the sequence, s_k, comes out of the transition combinational logic and penetrates to the clock generator delay element before it can reset, the clock generator may never detect present-state equals next-state for s_j. This will cause the machine to lock up in intermediate state s_j and even subsequent input changes may not be able to dislodge it.

This clock generator scheme could use either a monostable multivibrator or an inertial delay for the delay element. Suppose a monostable multivibrator is used in the clock generator. As outlined above, the maximum propagation delay from the state register through the clock generator logic to the monostable multivibrator must be less than the minimum propagation delay through the transition function combinational logic and the clock generator logic to the same point. Since the monostable multivibrator is triggered by a change from equal to different, when this condition is not met, there will be no trigger for the next clock pulse. Thus a monostable multivibrator imposes the restriction

$$f_M+(d'_M-d_M)<f_m+d'_m.$$

When an inertial delay is used, the equality of next and present states described above must persist for an time D_M before the inertial delay output can go low. This imposes the restraint

$$f_M+(d'_M-d_M)+D_M<f_m+d'_m.$$

(As noted above, d'_M includes d_M as one component.)

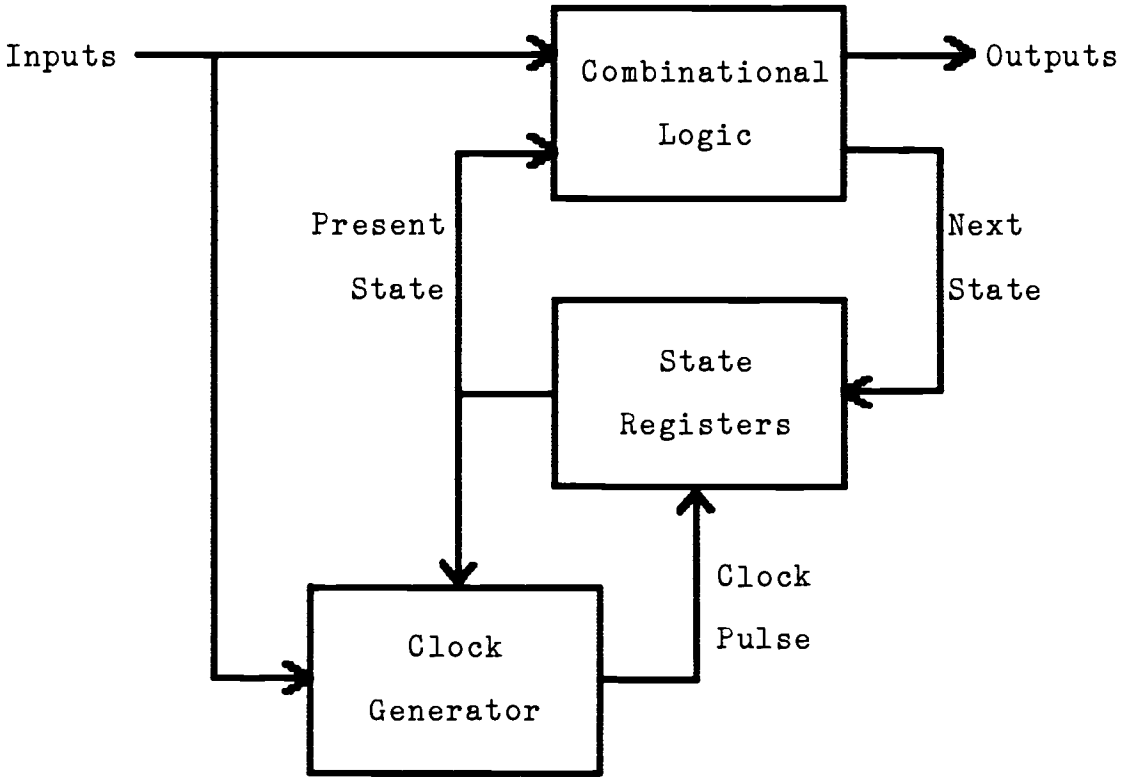


Figure 8

MOC Clock Generator - Input and Present State

The second method of MOC clock synthesis is illustrated in Figure 8. The advantage of this structure is that a

"standard" clock generator can be designed without using any information about the behavior of the machine. This universal approach reduces the design effort without any sacrifice in performance. Clock generators for this structure monitor the inputs and state variables to produce a clock pulse any time an input or state variable changes. However, all previous clock generators for this structure produce one extra clock pulse as the final stable state is entered. The clock generator does not "know" that it is done. This structure was used by Rey and Vaucher (Rey and Vaucher, 1974).

As in all the previous cases, the first clock pulse edge must not reach the flip-flops before the input-generated changes have gone through the combinational logic. Here again,

$$D_m+d'_m \geq \delta_1+d_M+s,$$

$$D_m \geq \delta_1+d_M+s-d'_m.$$

The state transition time is the sum of the delay through the flip-flops, the combinational logic, and the set-up time for the flip-flops (f_M+d_M+s). Because there must be a clock pulse generated after every state change, n state transitions will generate n+1 clock pulses. Thus,

$$\delta_2+D_m+d'_m \geq (n+1)(f_M+d_M+s)+D_M+d'_M,$$

and input state changes are separated by

$$\delta_1+\delta_2 \geq \delta_1+(n+1)(f_M+d_M+s)+((D_M+d'_M)-(D_m+d'_m)).$$

The time between input states is proportional to n+1, but an

optimum machine would have a delay proportional to n.

The clock generator logic could also be customized to the behavior of the machine (Chuang and Das, 1973). It could compute the next state and generate a clock pulse if required. This form of clock generation is then functionally identical to the previous case (Figure 7) and the same timing restrictions apply. Closer inspection shows that a clock generator based on present-state equals next-state (Figure 7) is only a special case of the more general form shown in Figure 8. Since the clock generator has available to it all the information that is available to the combinational logic, it can duplicate any required calculations and operate in exactly the same mode as Figure 7. This is exactly the mode of operation in the Chuang and Das machine.

SELF-SYNCHRONIZING CLOCK GENERATORS

It has been known since at least 1962 (Unger, 1977), that the many advantages of synchronous design could be realized in an asynchronous machine by generating a pulse each time an input changed. The problem has been to develop a clock generator that works reliably and does not compromise the inherent speed advantage of an asynchronous machine.

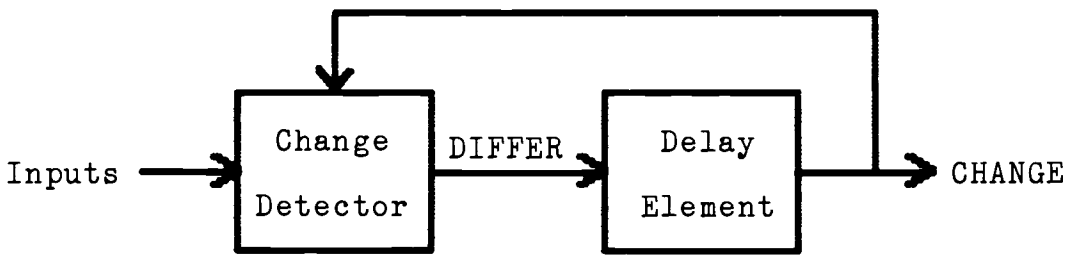


Figure 9

Clock Generator Expanded

The structure of the clock generator is shown in Figure 9. There are two parts: change detector circuitry to determine when a clock pulse is needed and a delay element to generate the clock pulse. As discussed earlier, the change detector could be customized to the machine behavior and generate a clock pulse when required. This approach

introduces additional timing restrictions which reduce the speed of the machine. A better (faster) solution is to use a generalized change detector.

The purpose of the change detector is to output the signal DIFFER when a change in the input signals is detected. After DIFFER has propagated through the delay element, it emerges a predictable time later as the signal CHANGE. This signal may be fed back to the change detector which turns off DIFFER. A short while later, DIFFER off propagates through the delay element and CHANGE goes off. Alternately, if a monostable multivibrator is used as the delay element, this feedback path is not needed. In either case, it is usually the final transition on CHANGE that clocks the state register flip-flops. If, for some special reason, the flip-flops are clocked from the leading edge of CHANGE then the previous timing analyses must be modified to take into account the interval in which CHANGE is high after the flip-flops have been clocked.

Single Input Change Mode Clock Generation

A normal fundamental mode machine was the first machine for which a practical clock pulse generator was developed (Bredeson and Hulina, 1971). Since only a single input is permitted to change, the modulo 2 sum of the input vector components changes for each input state. Thus a modulo 2

adder can be used for the change detector and the problem is solved. Notice in this case, feeding CHANGE back into the change detector (as shown in Figure 9) is not required.

This clock generation method is not suitable for a multiple output change machine unless only a single state variable changes for each state change (required for the modulo 2 adder to work as a change detector). State encodings with this property do exist. The best known encoding with this property is based on a Hamming code (Unger, 1969), but special codes can be designed for specific machines. These encodings are, in general, not unicode state assignments - one code per machine state. If a unicode state assignment is desirable, another change detector method must be found. One reason for a self-synchronized machine is to simplify the state assignment. Constraining the state encoding erases this advantage.

Multiple Input Change Mode Clock Generation

Two clock pulse generation methods have been proposed for MIC machines. Rather than a generalized change detector, the first approach uses a combinational logic network customized for the required machine behavior (Chuang and Das, 1973). The combinational logic uses the inputs and state variables to determine when and if a clock pulse is needed. This approach requires an inertial delay to filter

spurious outputs from the combinational logic, requires a new design for each machine, and has been shown (Unger, 1977) to be slower than a generalized change detector.

The second approach (Rey and Vaucher, 1974) used monostable multivibrators on the inputs to convert changes in the level-sensitive inputs into pulse-mode inputs. These pulse-mode inputs are then combined in an OR gate which triggers another monostable multivibrator to form the clock pulse. This kind of change detector is fundamentally sound, but any implementation may have practical problems. Since each input has an edge-to-pulse converter, one multivibrator is required for each input signal. High operating speeds require short clock pulses. It is difficult to build accurate multivibrators for narrow pulse widths.

An improved version (Unger, 1977) of this second approach is shown in Figure 10 on the next page. This network solves the problems present in a clock generator built with monostable multivibrators. This kind of network is called a digital differentiator since it converts a change in input level into a level. To understand its operation, assume that the present input state is stored in the latch and the enable (CHANGE) is off. When one or more inputs change, the appropriate EXCLUSIVE-OR gate outputs a high and DIFFER goes high. After a time determined by a delay element (not shown), CHANGE goes high and the latch is opened. When the latch outputs match the input state,

DIFFER, and eventually CHANGE, again go low. The change detector is now ready to accept the next input state.

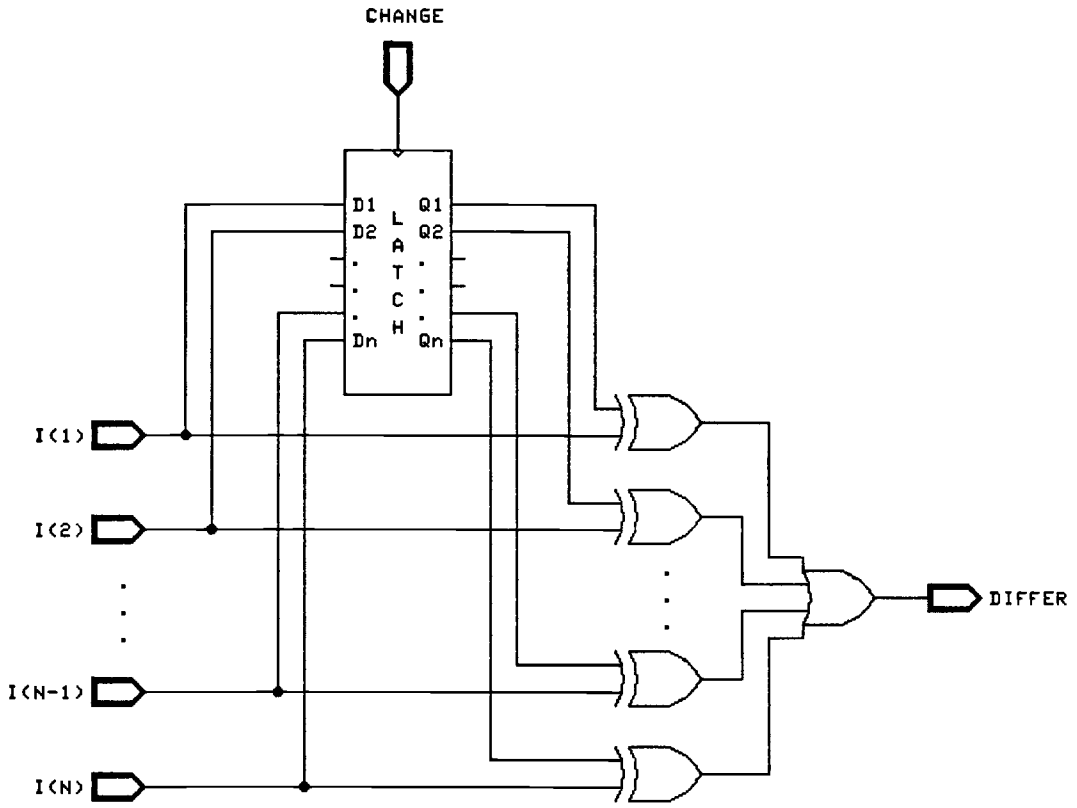


Figure 10

Digital Differentiator

Figure 11, on the next page, shows the timing relationships for this change detector. The first input change (I_1) starts the cycle while I_n is the last change to be part of this input state change. This form of clock generator is particularly economical to realize. For example, a clock generator that accepts up to eight inputs

can be built with only two parts: one 74F373 eight bit latch and one 74F521 eight bit equality comparator. The min/max propagation delay from input to DIFFER is 3/11 nanoseconds and from CHANGE to DIFFER is 8/24 nanoseconds. Thus d'_m is 11 nanoseconds and d'_M is 35 nanoseconds.

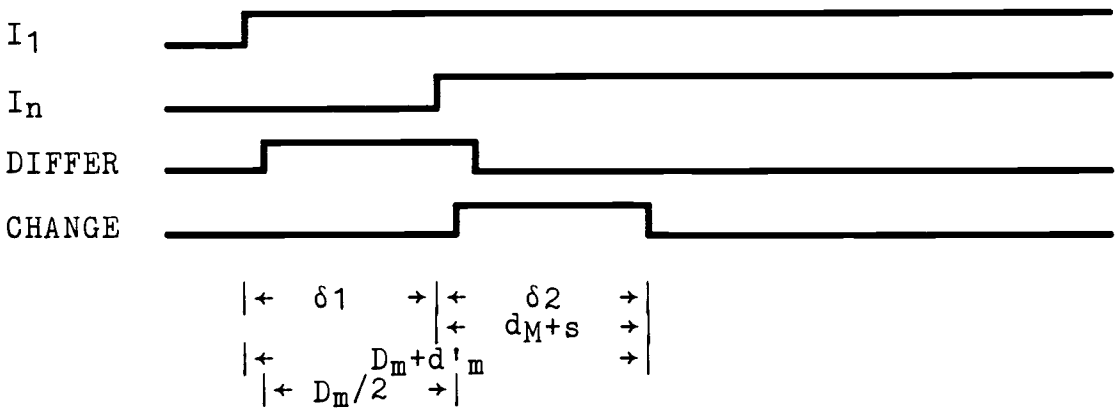


Figure 11

Symmetrical Delay Element MIC Timing Diagram

If a self-synchronized machine with a symmetrical delay element uses this clock generator, an additional timing restriction forces the machine to operate at less than ultimate speed. The trailing edge of CHANGE, which clocks the flip-flops, is delayed by D_m. However, the delay element itself only delays a signal by $D_m/2$. If the delay element is a pure delay, then DIFFER had better not go low while the inputs are still permitted to change. If DIFFER does go low, any input signal changing near the end of the period will drive DIFFER high again and generate additional clock pulses which will pass through a pure delay. Thus the restriction

$$D_m/2 \geq \delta_1,$$

$$\delta_1+\delta_2 \geq 2\delta_1.$$

DIFFER can be permitted to change $D_m/2$ before the end of δ_1 if an inertial delay is used. Any pulse shorter than $D_m/2$ will not pass through the delay element. But since the delay is inertial, any input change that occurs at the end of δ_1 will extend the trailing edge of the clock pulse by an additional $D_m/2$. Thus, if an inertial delay element is used, there are the additional restrictions

$$\delta_2 \geq D_m/2,$$

$$\delta_1+\delta_2 \geq (3/2)\delta_1.$$

These delay induced restrictions have always prevented a self-synchronized machine from operating at optimum speed. The new asymmetrical delay element was designed to avoid this delay induced restriction.

An Optimum Clock Generator

Three problems caused by the structure of the clock generator must be overcome if ultimate operating speed is to be achieved: the symmetrical delay element limitations must be overcome, the extra clock pulse generated (when the final stable state is entered) must be eliminated, and the time between intermediate internal state transitions (due to a multiple output change perambulation) must be optimized. In an optimum solution, the ultimate speed will be limited only

by the fundamental considerations of implementation technology and the machine behavior.

The first problem to solve is the limitation imposed by the symmetrical delay element. It would be desirable if DIFFER were high for the entire input change period. With a symmetrical delay element, the length of time DIFFER is high is also the minimum time that DIFFER must remain low. By separating these two times, each can be optimized. In fact, the optimum low time is zero.

The answer is to delay the rising and falling edge of CHANGE by different amounts. The asymmetrical delay element presented earlier has a rising edge delay that is set by design while the falling edge delay is limited only by the technology. As such, it represents an optimum choice. Figure 12 is the timing diagram for a MIC machine using such an asymmetrical delay element.

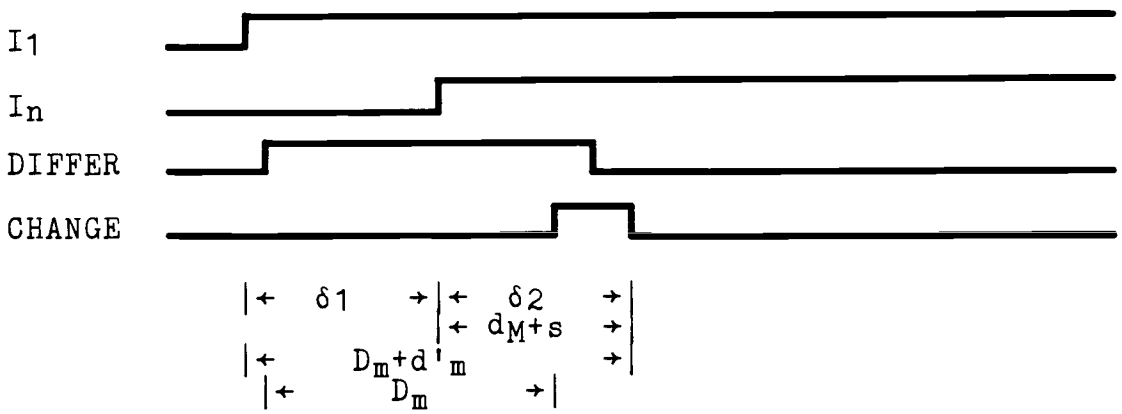


Figure 12

Asymmetrical Delay Element MIC Timing Diagram

The beauty of this delay element is that D_m can be chosen based on the problem specification and technology. The asymmetrical delay element can easily be designed so Differ remains high throughout the time the inputs are permitted to change without forcing an extension of the overall period between input state changes. A monostable multivibrator that is not retriggerable is the only other delay with this property, but it is difficult to build a monostable multivibrator when the pulse width required is less than 50 nanoseconds. It is easy to build an asymmetrical delay element to provide this delay.

The second problem to solve is the extra clock pulse that occurs at the end of a MOC internal state change. An early indication that the next state is stable is needed. A transition into a stable state is easy to determine from the transition function. If the present internal state and the next internal state are equal then the total state is stable; if not equal, then more states follow.

As a first step, an output called MORE has been added to the combinational logic block. The resulting machine architecture is illustrated in Figure 13 on the next page. If there are additional transitions required, MORE will be high, otherwise MORE will be low. This "early finish" indication needs to be incorporated into the change detector with minimum impact. This is very easy. MORE is connected to the T input of a T flip-flop which is clocked by the

trailing edge of CHANGE. When the machine is clocked with MORE high, the flip-flop output changes. But a change on a signal is exactly what the change detector is designed to process. This flip-flop output change causes another clock pulse. If a clock pulse occurs with MORE low, then the T flip-flop does not change and the sequence ends.

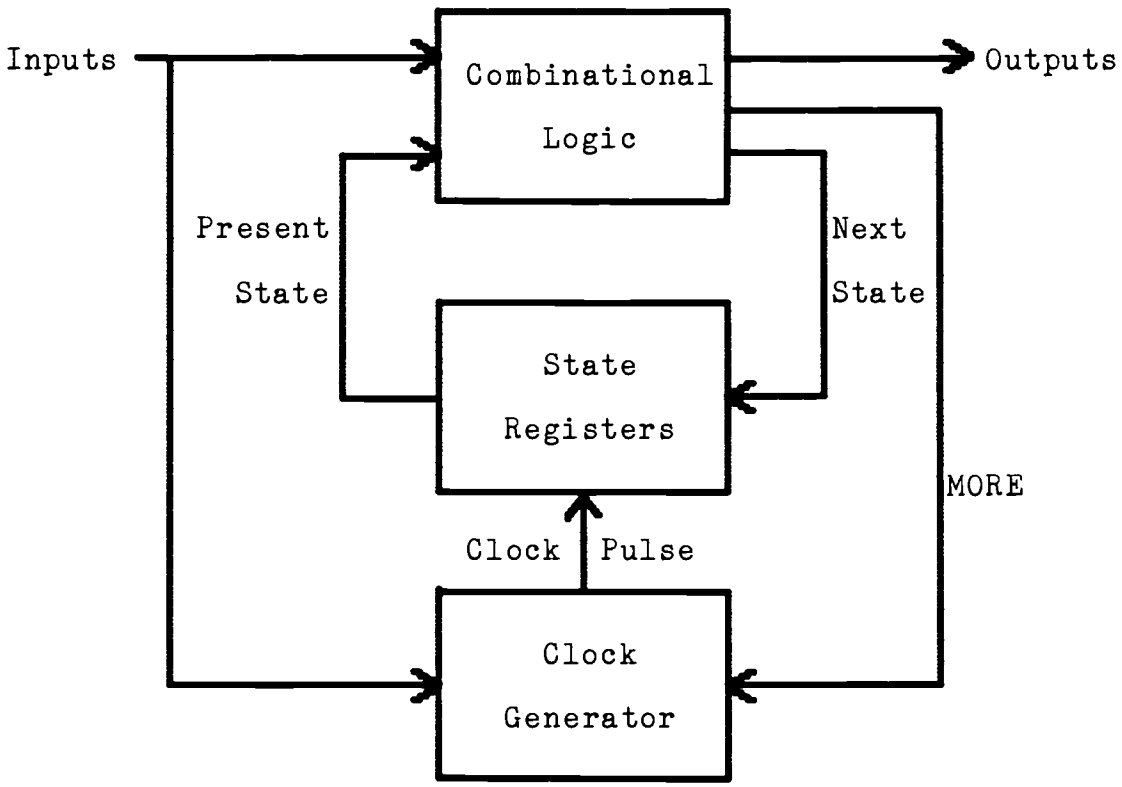


Figure 13

MOC Machine With Early Final State Indication

It is essential that the T flip-flop be clocked on the trailing edge of CHANGE to guarantee the clock generator is ready to accept the next T flip-flop output change, if there

is one. The following timing diagram illustrates one state transition caused by an input state change and two additional state transitions caused by the T flip-flop. The output of the T flip-flop is named NEXT.

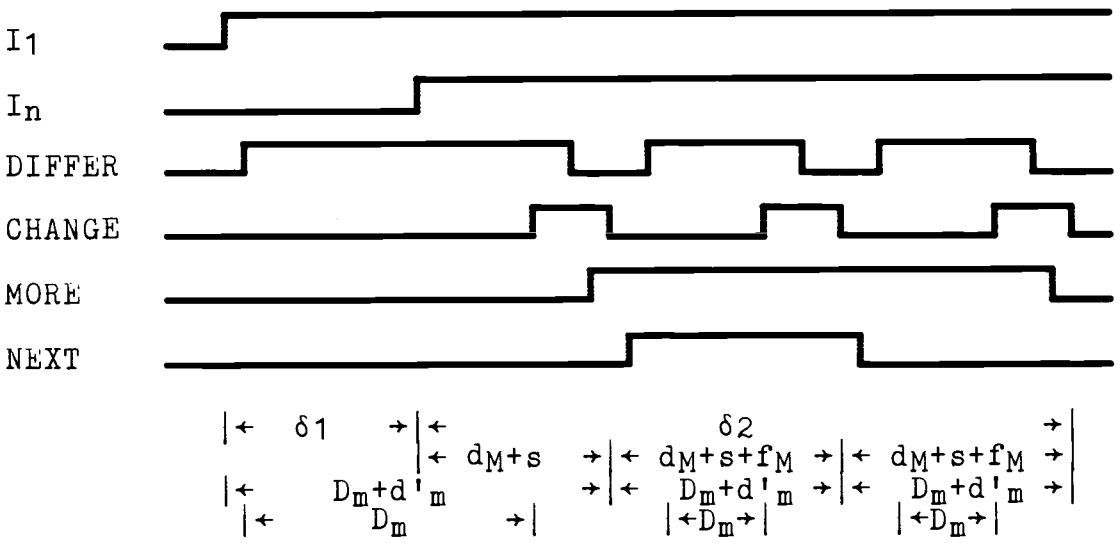


Figure 14

Early Final State Indication Timing Diagram

Figures 13 and 14 also illustrate the solution to the third problem - to optimize the internal state transitions during a multiple output change perambulation. The delay element time for a state transition is defined to be D_m, but notice it is not a constant. The first D_m value was selected so the first state transition would not malfunction. All total state transitions except the first are under complete control of the machine designer. Since the input state must remain stable, every state transition except the first appears as a single input change transition to the clock

generator; NEXT is the only clock generator input changing. For the combinational logic block, only the state variables change, and since they come from the state register clocked by CHANGE, they arrive at the combinational logic block simultaneously. Thus, for all state transitions after the first, D_m can be reduced.

This approach and the programmable asymmetrical delay element of Figure 5 also gives the designer a new freedom he has never had in the past. The designer may be privy to special information about the machine being designed. For example, he may know when state s_i is reached, only one input will change to cause the next state transition, but in state s_j, the next state transition will be caused by a multiple input change. In the past, he had to design the machine for the worst case (s_j) and the less demanding case would take longer than necessary. There was no way to use this auxiliary information. Suppose an additional set of output signals, called DELAY, are produced just as are the state variables. DELAY is a function of the states and inputs just as are the state variables. As shown in Figure 5, these additional outputs are used to control the delay time D_m. The delay time leaving each state can be customized, on a state by state basis, based on the required machine behavior. The number of different D_m times needed determines the number of bits in DELAY. The timing diagram of Figure 14 shows two values for D_m, a long time for the

first state transition following an input state change and a short time for all subsequent state transitions, but each state transition could have had different delay times.

Caution! Since DELAY and NEXT change at the same time, the maximum propagation delay for DELAY to the multiplexer in the programmable asymmetrical delay element of Figure 5 must be less than the minimum propagation delay for NEXT through the change detector to DIFFER and eventually the multiplexer. This is required so the multiplexer in the delay element is properly set up. Otherwise, the select inputs to the multiplexer would be changing at the same time the data inputs are changing. This restriction is not a problem in practice since the NEXT-DIFFER path is considerably longer.

EXTENDING SELF-CLOCKED MACHINES

The self-synchronized MIC machine presented can be easily extended to operate in unrestricted input change (UIC) mode, pulse mode, or speed independent mode.

Unrestricted Input Change Mode

Almost all asynchronous designs assume the machine will operate in fundamental mode - once an input state change is perceived by the machine, the machine will reach a final stable state before the next input state change is permitted. When a machine is operating in UIC mode the fundamental mode assumption is violated. The problem is to describe what is a satisfactory outcome for a state transition.

As a first step in defining a satisfactory outcome, the concept of an n-cube and spanning must be introduced. There are 2^n binary vectors with n components. This set of all 2^n vectors is said to form an n-cube. A subset of 2^m n-dimensional vectors having the same value in n-m locations is said to form a subcube.

Definition: Given a set of n-dimensional binary vectors, V, the set of vectors spanned by V is the smallest subcube containing every member of V. This set of vectors spanned is written T(V).

The concept of spanning is best illustrated by example. If

$$V=\{01001,01100\},$$

then T(V) would be

$$T(V)=\{01000,01001,01100,01101\},$$
$$=\{01-0-\}.$$

Notice Unger's definition of spanning differs from the definition of spanning used in linear algebra.

$T(i_a,i_b)$ would be all the input states that might be passed through as an input vector changed from i_a to i_b. The machine will not necessarily respond to all input states in the set of states spanned. For example, with T as above, the machine may respond as if any of these input sequences occur: (01001, 01000, 01100), (01001, 01101, 01100), or (01001, 01100). But not the sequence (01001, 01000, 01101, 01100) since once an input changes, it will not change back. If more than one input changes, some input states in the set of states spanned must be skipped. However, observe the states in the input sequence are not necessarily all single input changes (Hamming distance one). The following definition, first presented by Unger, describes a satisfactory outcome (Unger, 1971).

Definition: A satisfactory outcome of an input change from i_1 to i_n with initial state A is any stable state (s,i_n) which could have been reached by a sequence of input changes $i_1, i_2, \ldots i_n$ where, for j=2 to n, i_j is a member of $T(i_{j-1},i_n)$.

The extension of the MIC machine to UIC mode is straight-forward. All the inputs are to pass through a transparent latch before they are presented to the machine. While the machine is in a stable state, the gate signal for the latch is true and the latch inputs pass through to the machine. While the machine is in transition to a final stable state, the gate signal is false and the latch outputs are frozen. Since the machine is busy if DIFFER, CHANGE, or MORE are true, the latch gate signal is the complement of (DIFFER OR CHANGE OR MORE). When an input changes, the gate is turned off, freezing a member of $T(i_a,i_b)$ in the latch. This input state is processed and the latch is opened to capture another input state. As far as the machine is concerned, it sees only multiple input changes and operates in fundamental mode. Thus, the input latch groups the input changes into a sequence of batches for the machine to process, and any such sequence eventually leaves the machine in a satisfactory outcome.

It should be noted that the UIC latch may exhibit metastable behavior since the set-up or hold time

specification may be violated. (Metastable behavior is an increase in propagation delay due to violation of set-up or hold time.) To compensate for this metastable behavior, δ_1 must be extended.

Pulse Mode

Pulse-mode circuits have a separate input terminal for each input state. The pulse widths on these inputs, as well as the spacing between pulses, are bounded both above and below. Pulse-mode circuits are generally designed using synchronous techniques. Suppose however, the asynchronous design includes some pulse-mode inputs and some level inputs. The problem is then to change the pulse-mode inputs into suitable level signals.

Pulse-mode input durations are, in general, narrower than state transition times. A pulse can be converted to a suitable level by presenting the pulse to the clock input of a T flip-flop (T input held true). Every pulse will change the state of the flip-flop. The output of the T flip-flop is exactly the kind of input the change detector needs for proper operation. Every pulse changes the T flip-flop output (which is a change detector input). In the traditional definition of pulse mode, only one input is allowed to pulse. Once the pulses are converted to levels, the remaining machine operates in single input change mode.

While T flip-flops provide an easy way to convert a pulse into a level, pulse-mode inputs also provide an opportunity to implement a different kind of change detector, illustrated in Figure 15.

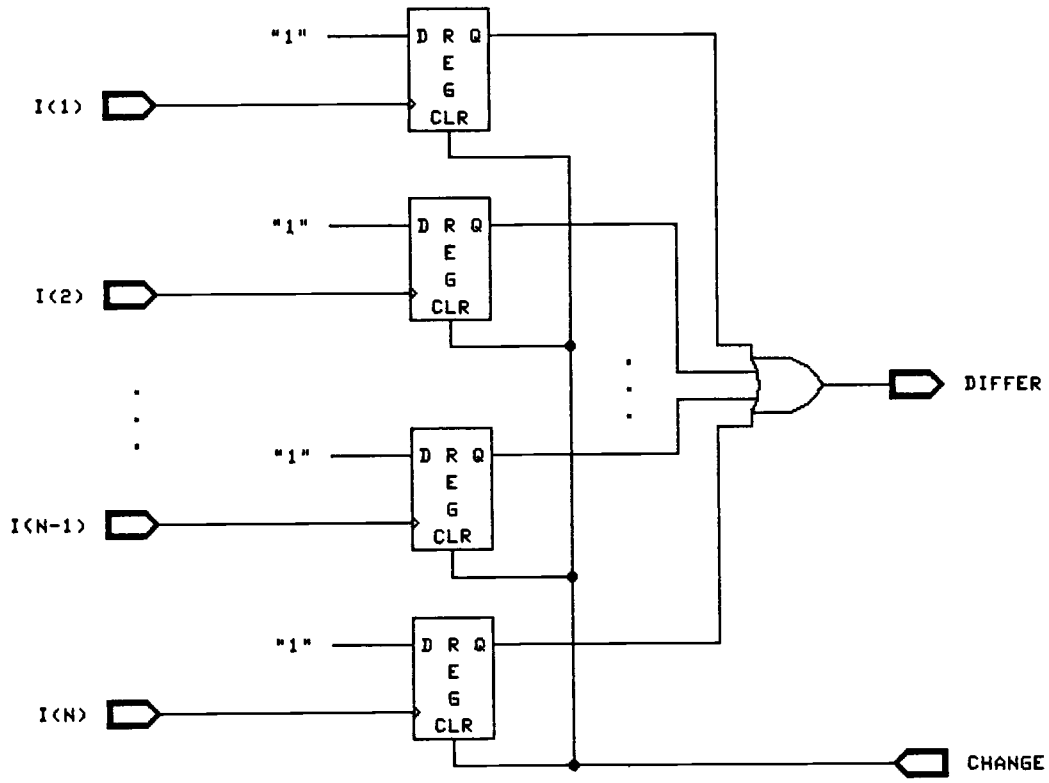


Figure 15

Pulse Mode Alternate Change Detector

This change detector has the block diagram shown in Figure 9. If the pulse-mode inputs drive the clock input on a D flip-flop with the D input fixed high, the output of the flip-flop will go high when the input pulse occurs. Whether the flip-flop clocks on the positive or negative transition is an implementation detail - the change detector operates

the same in either case. By connecting the reset input on the flip-flops to CHANGE, the EXCLUSIVE-OR gates shown in Figure 10 can be eliminated and the flip-flop outputs can be connected directly to the OR gate. When CHANGE goes high, the flip-flop will be reset and the cycle can be repeated.

Speed Independent Mode

Speed independent designs are characterized by a completion handshake. Such a design is based on a chain of individual sub-circuits, each one sending completion signals to its predecessor and responding to completion signals from its successor. In addition to the special completion signals, the input states are separated by a special input state called a spacer.

Each sub-circuit cycles continuously through a sequence: output data, request spacer, output spacer, request data. When a sub-circuit has responded to the input data from its predecessor, and its output data has been accepted by its successor, the sub-circuit sends a completion signal (traditionally called S) to its predecessor requesting a spacer. A sub-circuit outputs the spacer when it is requested by its successor and its predecessor has output a spacer. When a sub-circuit has a spacer input and the successor is requesting data, the sub-circuit requests data by sending a signal (traditionally

called D) to its predecessor.

The operation of a speed independent sub-circuit can be clarified by examining the asymmetrical delay element of Figure 5. It is really a sub-circuit operating in the speed independent mode. The predecessor is the change detector, but there is no speed independent successor. The special input state, spacer, occurs when DIFFER is low, while DIFFER high is the data input state. The request for spacer (S) is indicated by CHANGE true and the request for data is indicated by CHANGE false. Notice how the machine continuously cycles through request data, data, request spacer, spacer. This cycle can be clearly seen in the timing diagram of Figure 14. Had there been a speed-independent successor sub-circuit, there would have been an S/D input from this successor into the asymmetrical delay element.

A self-synchronized machine itself is not speed independent, but it can be made to operate in a chain of speed independent sub-circuits. For a self-synchronized machine to operate in speed independent mode, the S and D completion signals must be added to send to the predecessor sub-circuit and the machine must respond to completion S and D signals from the successor sub-circuit. There is no timing relationship between sub-circuits, so the self-synchronized sub-machine must operate in UIC mode.

Suppose a self-synchronized machine is built, using the structure depicted in Figure 13, to operate in a chain

of speed independent sub-circuits. Request for data and spacer from the successor would be normal inputs to the sub-circuit and the S/D output to the predecessor would be a normal output. One input state and one output state would be designated as the special spacer. When the successor requests data, the sub-circuit requests data from its predecessor. When the data is supplied, the sub-circuit performs the task it was designed to do and passes its data to the requesting successor. The successor, after some period of time, will send to the sub-circuit a request for spacer. At that time, the sub-circuit requests a spacer from its predecessor. When the spacer appears at the sub-circuit's inputs, it then sends a spacer to the requesting successor and the cycle repeats. The key is to realize that since the sub-circuits are time-independent, each must operate in UIC mode.

TWO DESIGN EXAMPLES

This chapter presents two examples of the application of the ideas presented in this dissertation. The first example demonstrates the simplicity of the design process when a self-synchronization scheme is used. The second design example is a practical application of interfacing two machines operating with different clocks. This second example uses nearly all the ideas presented herein in one coherent design.

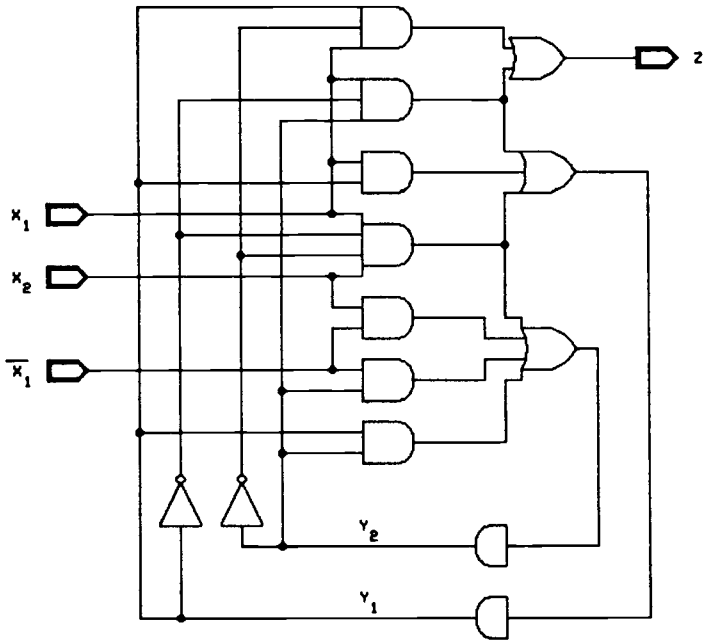
The Crumb Road Traffic Control Machine

There is a rather well known example of all the things that can go wrong in asynchronous design. This hypothetical problem involves the design of a sequential machine to control the traffic at the intersection of Crumb Road and Route 1. (For a complete description of the problem, see Unger, 1969.) Beginning with a verbal description of the problem, Unger developed the flow matrix of Figure 16 as a first attempt to solve the problem. From this flow matrix, he derived logic equations and developed the sequential circuit of Figure 17. As a demonstration, this first design

	x_1 x_2				y_1 y_2
	0 0	0 1	1 1	0 0	
1	1,0	2,0	4,0	1,0	0 0
2	2,0	2,0	3,0	3,1	0 1
3	1,0	2,0	3,1	3,1	1 0
4	2,0	2,0	4,0	4,0	1 1

Figure 16

Crumb Road Problem Flow Matrix



$$z = x_1\bar{y}_1y_2 + x_1y_1\bar{y}_2$$

$$Y_1 = x_1x_2\bar{y}_1\bar{y}_2 + x_1\bar{y}_1y_2 + x_1y_1$$

$$Y_2 = \bar{x}_1x_2 + y_1y_2 + \bar{x}_1y_2 + x_1x_2\bar{y}_1\bar{y}_2$$

Figure 17

Crumb Road Problem Sequential Machine

attempt intentionally ignored the design issues of unrestricted input change mode, critical races, essential hazards, and logic hazards. Unger then demonstrated how proper operation of this sequential circuit depends upon the relative magnitude of the stray delays.

However, if this circuit did operate correctly, what would be the speed of operation? To answer this question assume the gates and inverters have a min/max propagation delay of 3/7 nanoseconds and the delay element maximum delay is 1.5 times the minimum delay. These values fairly represent the real times for 74Fxx series parts. Since the two input variables, x_1 and x_2, can change at any time, the machine operates in unrestricted input change mode and δ_1 is 0. Using these values and the previously developed timing analysis, the minimum delay element time is now calculated as

$$D_m \geq \delta_1 + d_M - d_m = 0 + (7+7) - (3+3) = 8 \text{ ns},$$

and the time between input states is calculated as

$$\delta_1 + \delta_2 \geq \delta_1 + n(d_M + D_M) + (d_M - d_m),$$

$$\geq 0 + 1(21 + 1.5 \times 8) + (21 - 6) = 48 \text{ ns}.$$

As a demonstration of the power and simplicity of the self-synchronized design style, this abysmal failure of an asynchronous machine (Unger, 1969) will be converted into a practical design by simply adding a change detector, UIC latch, and state registers. The resulting design is shown in Figure 18.

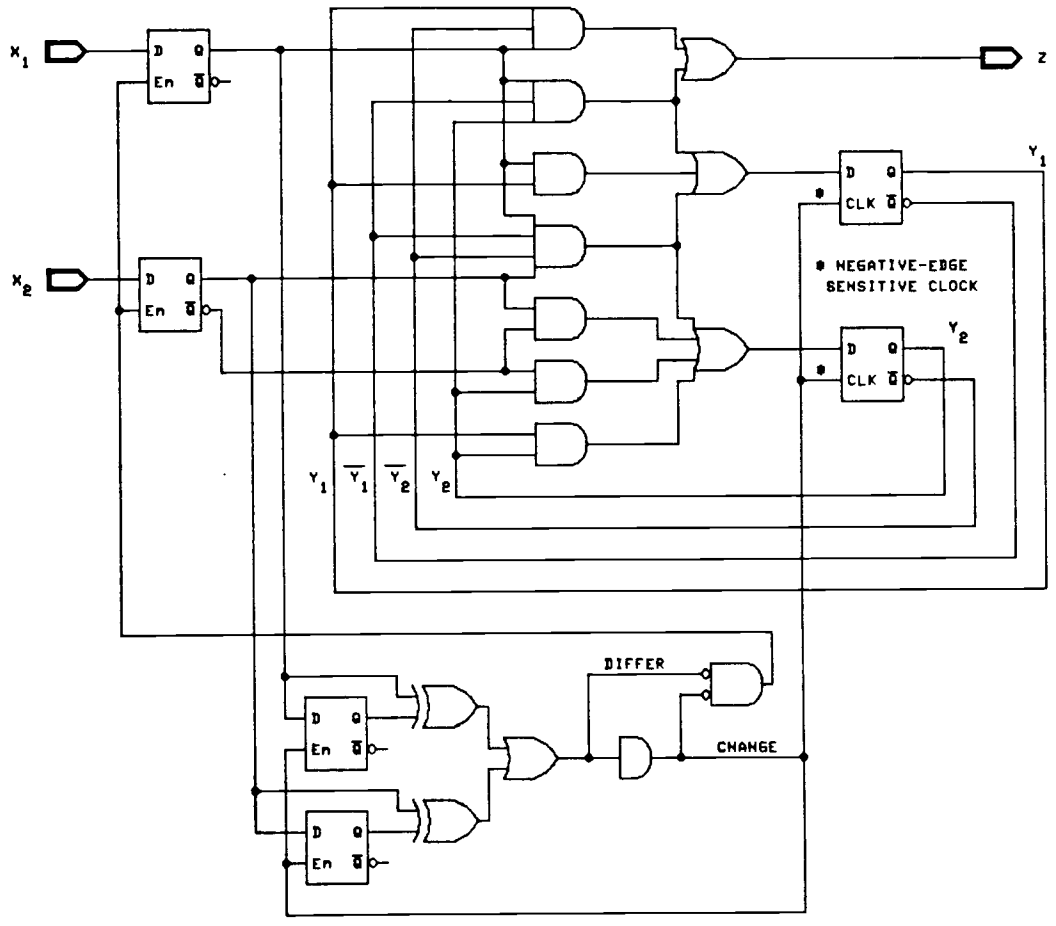


Figure 18

Crumb Road Problem Self-Synchronized Machine

Again, assuming the flip-flops have a set-up time of 2 nanoseconds and a maximum propagation delay of 10 nanoseconds, with the gate delays the same as the previous case, the delay element time is found to be

$$D_m \geq \delta_1 + d_M + s - d'_m = 4 \times 10 + (7+7) + 2 - (3+3) = 50 \text{ ns.}$$

δ_1 now is assigned the value of 4 times the flip-flop propagation delay to compensate for the potential metastable

condition in the UIC latch. This condition occurs when the latch inputs are changing at a time that violates the manufacturer's specified set-up or hold time. Thus the minimum time between input states is

$$\delta_1+\delta_2 \geq \delta_1+f_M+d_M+s+((D_M+d'_M)-(D_m+d'_m)),$$
$$\geq 40+10+(14)+2+((75+14)-(50+6))=99 \text{ ns.}$$

If the probability of simultaneous input changes is considered so improbable the UIC latch is unnecessary (reasonable in this design problem), the minimum delay element time can be reduced to 10 nanoseconds. This also reduces the time between input states to 39 nanoseconds. The resulting self-synchronized design will then be essentially the same speed as the Huffman-Moore machine without self-synchronization. But even more importantly, it will work.

A Practical Design Example

Consider the practical design problem of interfacing two digital machines, each with its own independent clock. Assume the purpose of one machine is to monitor the behavior of the other. This assumption greatly simplifies the interface; now the information flow across the interface is one direction only. The monitoring machine must gather information about the machine under test, but ideally, the monitored machine should not be affected by the monitoring machine.

This scenario is common. A test equipment manufacturer builds such equipment to help his customers diagnose, test, and evaluate their designs. One critical component of such test equipment is the sub-circuit that monitors the customer's signals and generates suitable clock signals for the test equipment.

Consider the design issues relevant to this sub-circuit. Besides the obvious design goals of high speed, reliable operation, and reasonable cost, this circuit must have a high degree of flexibility. It must interface with a wide variety of customer's circuits, each with its own set of signals - each circuit different in signal numbers, polarity, waveform, timing relationships, and so on. The essential behavioral characteristics of this machine must be programmable. That is, bit patterns written into registers and memory locations to program this circuit will customize this machine to a customer's specific requirement. These parameters are subject to change under control of a microprocessor in this test equipment.

Machine Block Diagram and Overview

A block diagram of such an interface machine is shown in Figure 19. The overall structure of the interface is identical to Figure 13, but two new blocks have been included. The transition function combinational logic and

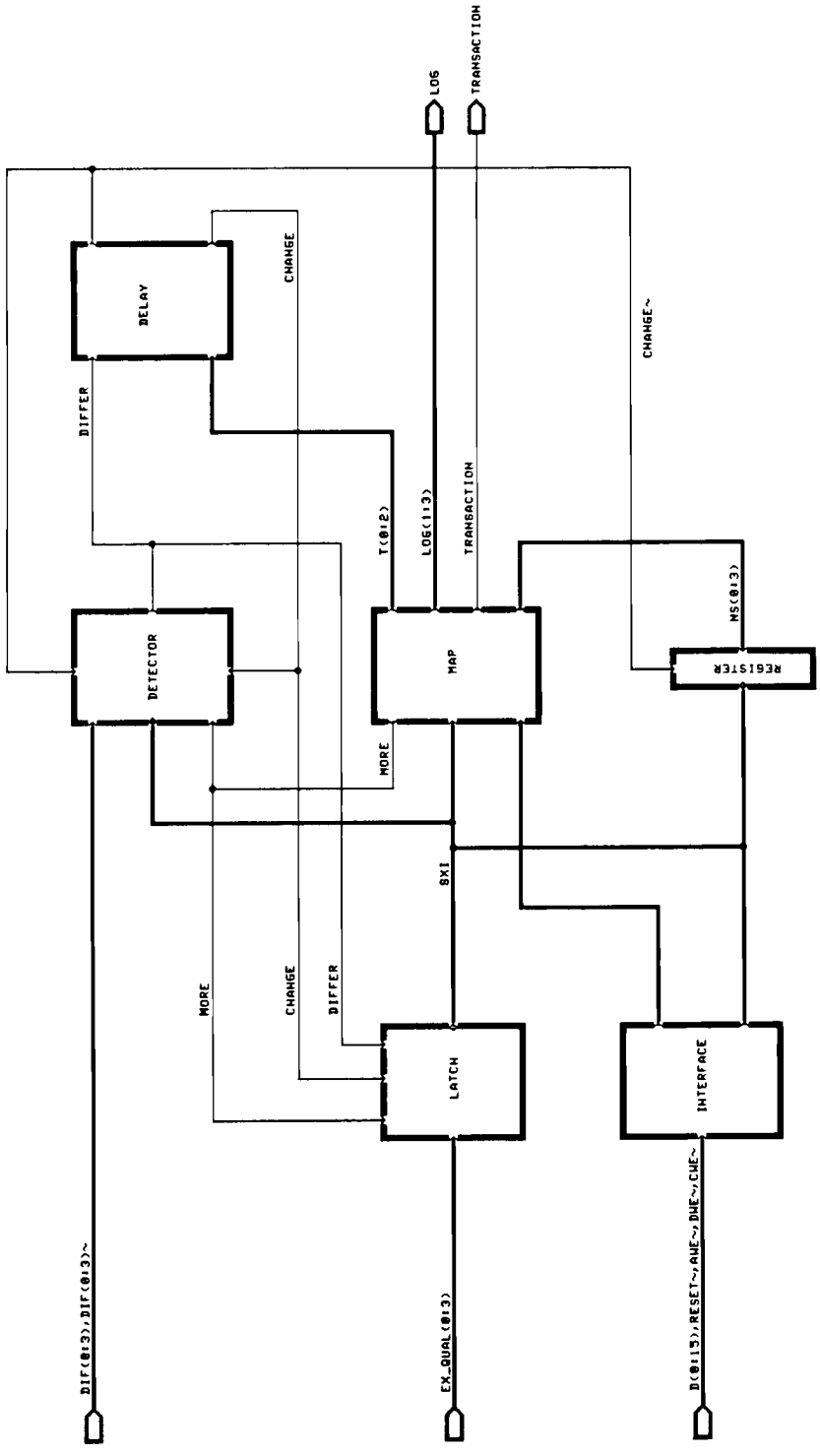


Figure 19

Practical Example - Block Diagram

state register blocks of Figure 13 have been labeled Map and Register in Figure 19. The clock generator block of Figure 13 has been expanded, using the form shown in Figure 9, into the change Detector and programmable asymmetrical Delay blocks. One new block, the unbounded input change Latch, has been added to the scheme of Figure 13 to extend the machine to UIC mode. The other new block, the microprocessor programming Interface, is essential to customize the interface machine to a customer's requirements.

The purpose of this chapter is to demonstrate a practical implementation of the ideas presented in this dissertation. Knowledge of the actual machine behavior, as determined by the transition function and output function, is not required to understand this example. For a similar reason, the nature of the programming signals from the test equipment microprocessor need not be detailed. The mnemonics chosen for these microprocessor signals suggest the actions performed, but timing and other details are not necessary. However, implementation details will be given as required to aid in understanding circuit operation.

The actual customer's circuit is at the end of a probe. Static protection circuits and buffers to protect the test equipment from damage are in the probe. By the time the customer's signals reach this interface, they have been converted from his levels to differential ECL signal levels. These inputs are labeled DIF(0:3) and EX_QUAL(0:3) in the

block diagram, change detector diagram, and UIC diagram.

To meet the design issue of high speed, this machine is implemented in the fastest logic family, emitter-coupled logic (ECL). The output of an ECL device is an emitter follower. This type of output structure permits the construction of the OR function by simply connecting outputs together. This is called WIRE-OR or "emitter dotting." Using an emitter follower output structure also requires every node to have an external pull-down resistor. Since the emitter follower can only pull high, the external resistor pulls the node low when all emitter followers are off.

The Change Detector

The waveforms, timing relationships, polarities, or even the number of signals the test equipment must monitor to generate the clocks it requires are not known at the time of manufacture. These design issues dictate that the change detector be implemented using the alternate pulse-mode method of Figure 15 because this method can accept either pulse-mode inputs or level-sensitive inputs.

The complete change detector, shown in Figure 20, can process both pulse mode inputs and level-sensitive inputs. This is accomplished by having two flip-flops (10H131's) for each input signal - one for the positive edge and one for the negative edge. That way, either polarity pulse can be

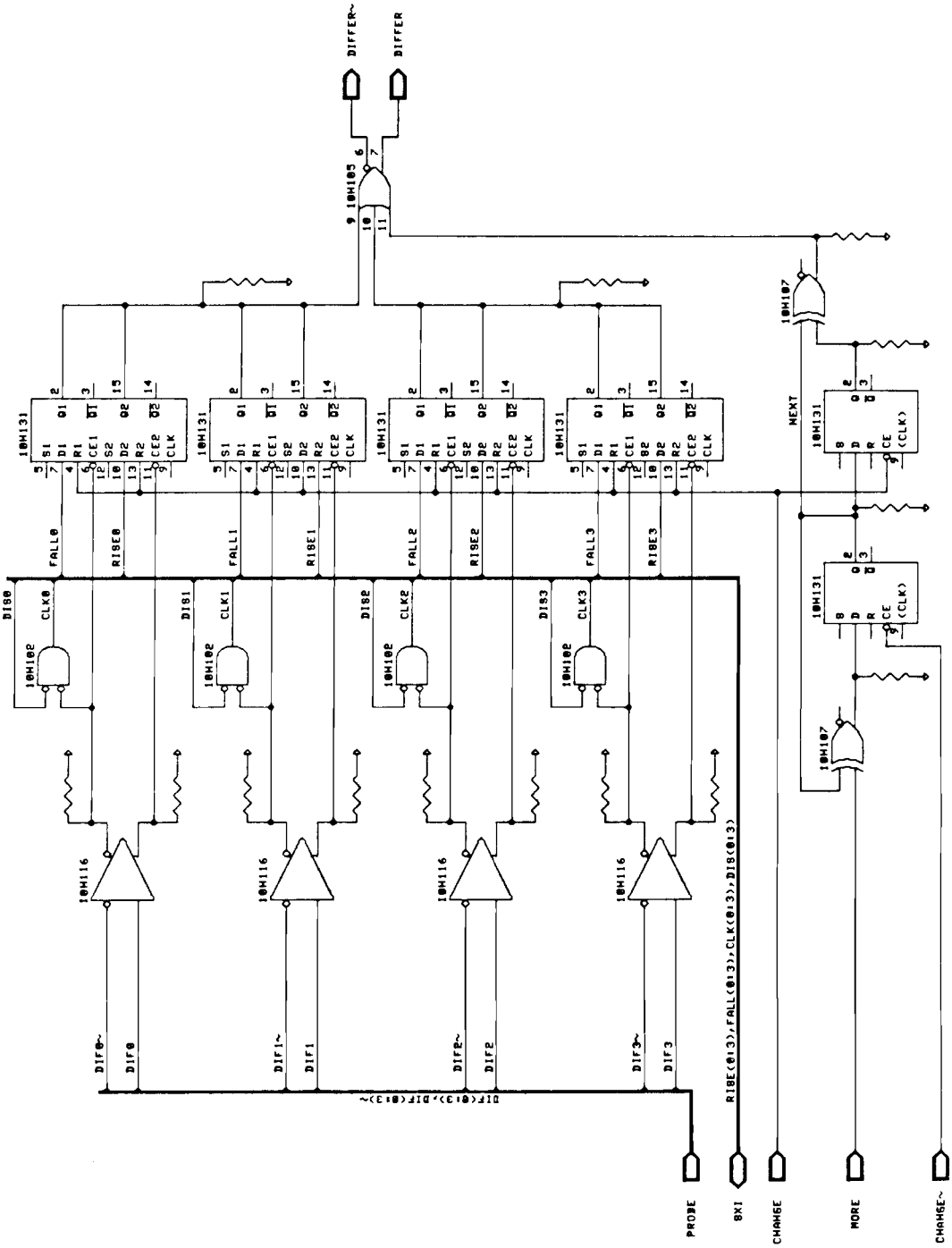


Figure 20

Practical Example - Change Detector

accommodated and, by enabling both flip-flops, a level-sensitive input can be accommodated. The choice is made by programming the signals RISE(0:3) and FALL(0:3). A high on the appropriate signal enables a clock pulse while a low inhibits a clock pulse on that edge.

The cost of an ECL circuit can be reduced by careful use of WIRE-OR's. Notice the individual outputs from each flip-flop are combined in a two stage process. First, four outputs are WIRE-ORed. Next, these nodes are ORed using a standard 10H105 OR gate. This configuration results in the best compromise of speed and parts count.

Consistent with the design goal of high speed, Figure 20 also shows the MORE-NEXT early final state indication implementation. Since the ECL family does not have T flip-flops, the D flip-flop equivalent is used (the 10H107 and 10H131). The output of this "T flip-flop" provides the input for a one-bit version of the standard digital differentiator built as shown in Figure 10; the other inputs are processed using the alternate pulse-mode change detector. The output from this digital differentiator is then ORed with the pulse mode change detector outputs to make DIFFER.

The Delay Element

For high speed, an asymmetrical delay element should be used. But δ_1 is not known, so a programmable version must be used. The temperature drift associated with the scheme of Figure 4 makes the improved asymmetrical delay of Figure 5 the best choice. As the customer will be specifying the delay times, they must be fixed and well defined. Again, the gated oscillator used in the scheme of Figure 5 satisfies this design issue. The complete delay element is shown in Figure 21 on the next page.

The gated oscillator (shown as a block in Figure 4) has been implemented using the SP9685 comparator and the 10H105 OR gate. This oscillator operates at 100 Megahertz, so each tap on the shift register (the 10H186's) is 10 nanoseconds away from its nearest neighbor. The propagation delays through the 10H105, the first two 10H186 stages, and the 10H164 are such that the minimum delay is approximately 20 nanoseconds. This first tap is the one selected for all internal state transitions when MOC perambulations are required. The customer has a choice of eight D_m's since one of eight different delay times is selected by the three bit code on the signals T(0:2). The delay time for the next state transition is clocked into a register (10H176) at every state transition.

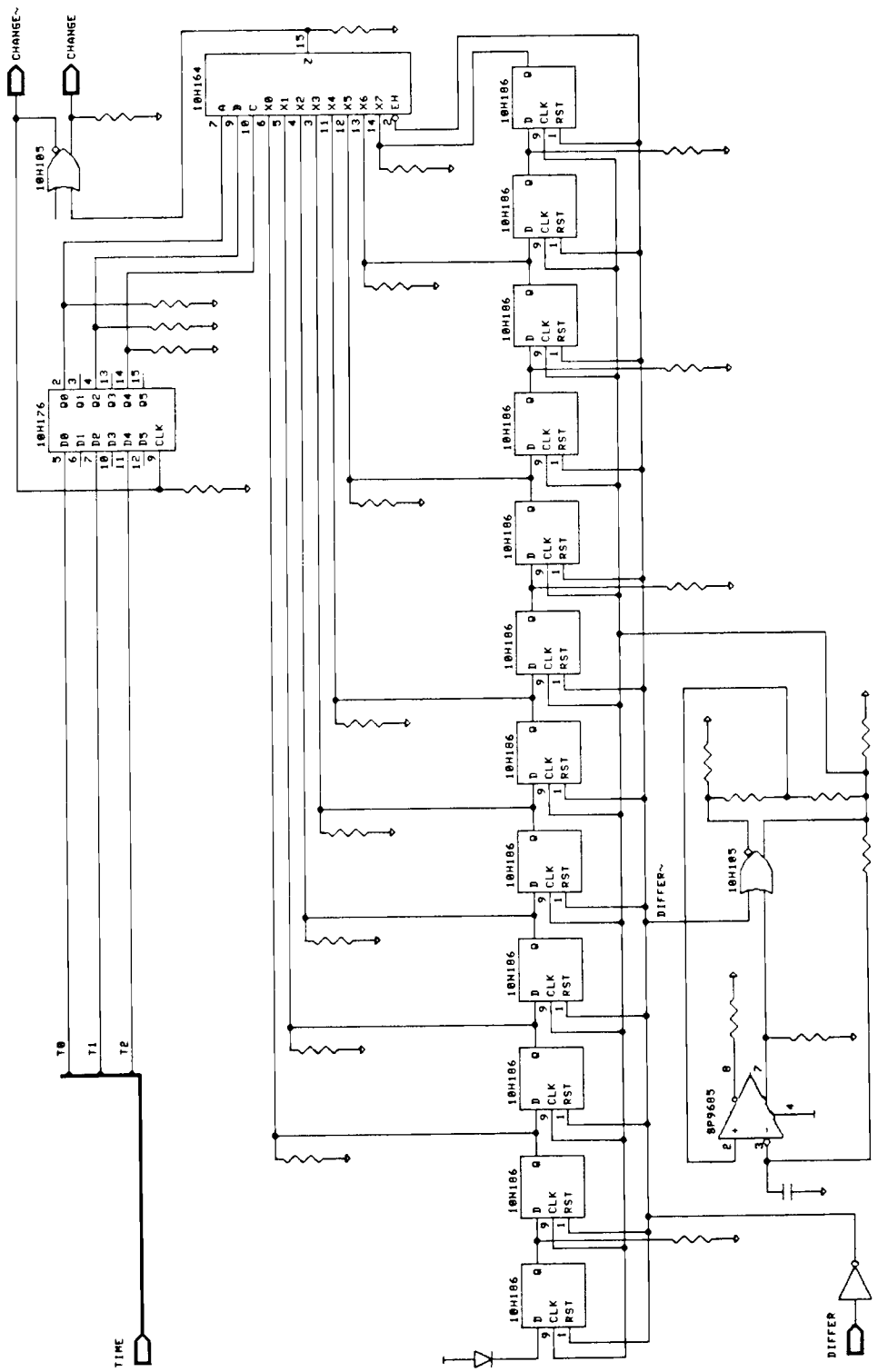


Figure 21

Practical Example - Asymmetrical Delay

State Variable Register

Since this is a self-synchronized design, the present state of the machine must be stored in an edge-sensitive register. The state variable register is nothing more than a package of D flip-flops (10H186). When the clock goes high, the next state code, NS(0:3), presented to the D inputs, is transferred to the outputs and becomes the new present state, PS(0:3). The complete state register is shown in Figure 22.

Every sequential machine needs an orderly method of starting. The LOAD signal connected to the reset input (RST) on the flip-flops provides this method. When LOAD is true, the flip-flops are cleared. When LOAD goes false, the sequential machine begins processing the customer's signals. Thus, the all zeros state is always the start state for this machine.

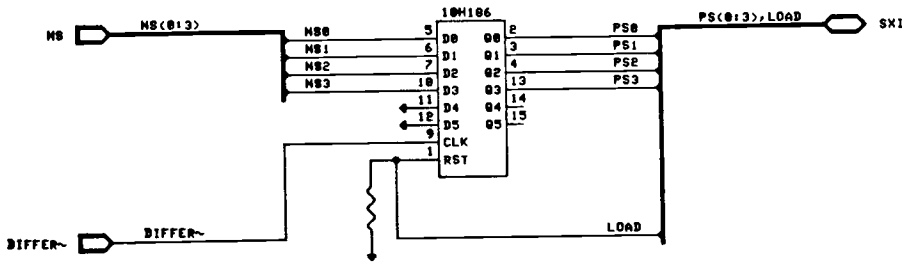


Figure 22

Practical Example - State Register

Transition Function Map Array

The timing relationships between customers' signals are as diverse as the customers themselves. While a transition function implemented with dedicated AND-OR-INVERT gates would be fastest (smaller d_M), only a programmable transition function can meet these diverse requirements. A random access memory, made possible because of the self-synchronized architecture, can provide this high degree of flexibility. The transition function is stored in a mapping array RAM. Outputs from this mapping array include the next state variables NS(0:3), early final state indicator MORE, test instrument specific outputs LOG(1:3) and TRANSITION, and the delay time selector T(0:2). The complete transition function map array is shown in Figure 23 on the next page.

UIC Latch

In addition to clock inputs, the customer may require inputs to affect the behavior of the machine, but changes in these inputs do not generate state transitions. Special inputs, such as these, are called <u>qualifiers</u> - they qualify or affect the behavior of the machine, but never cause a self-synchronizing clock pulse.

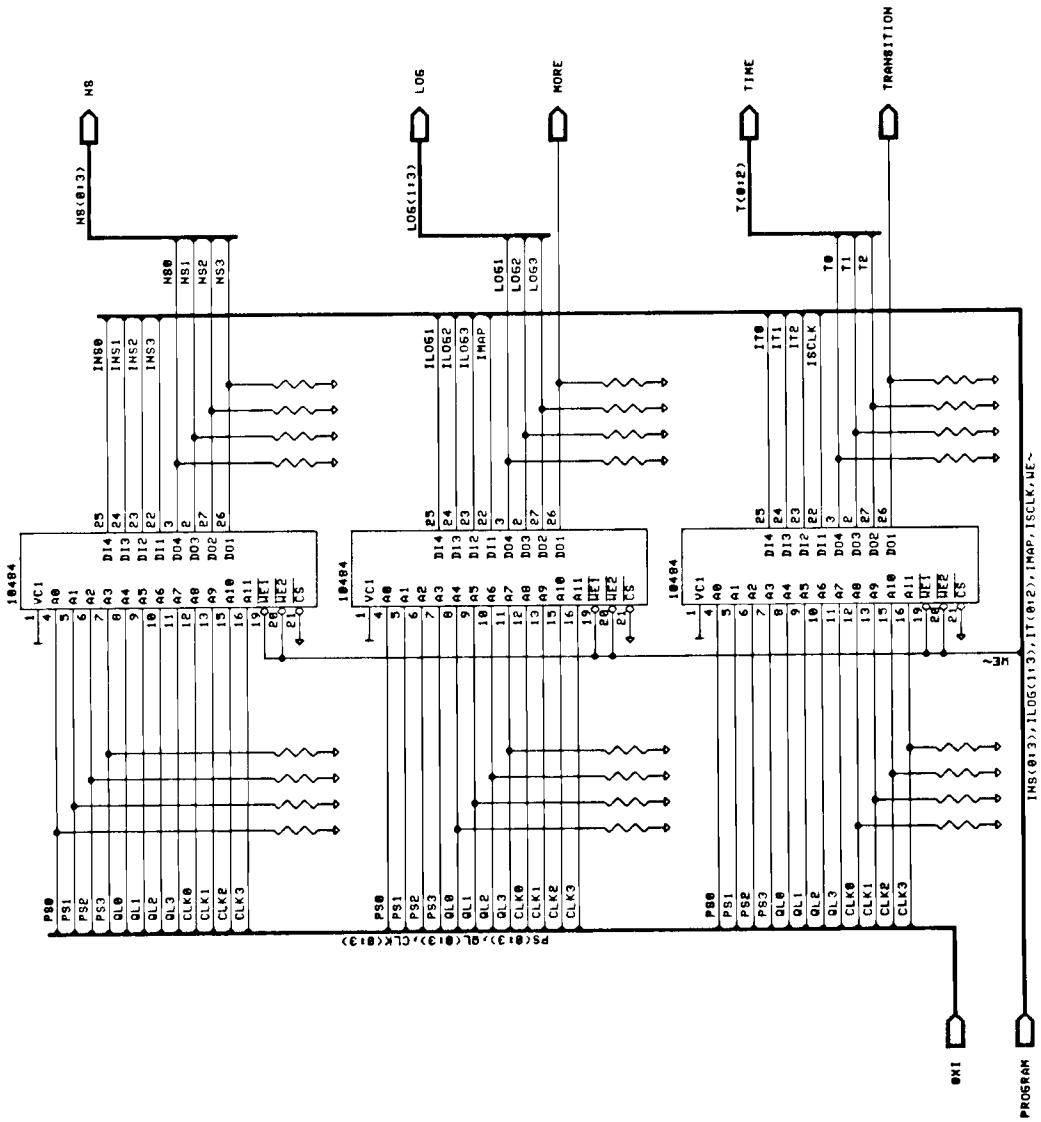


Figure 23

Practical Example - Transition Function

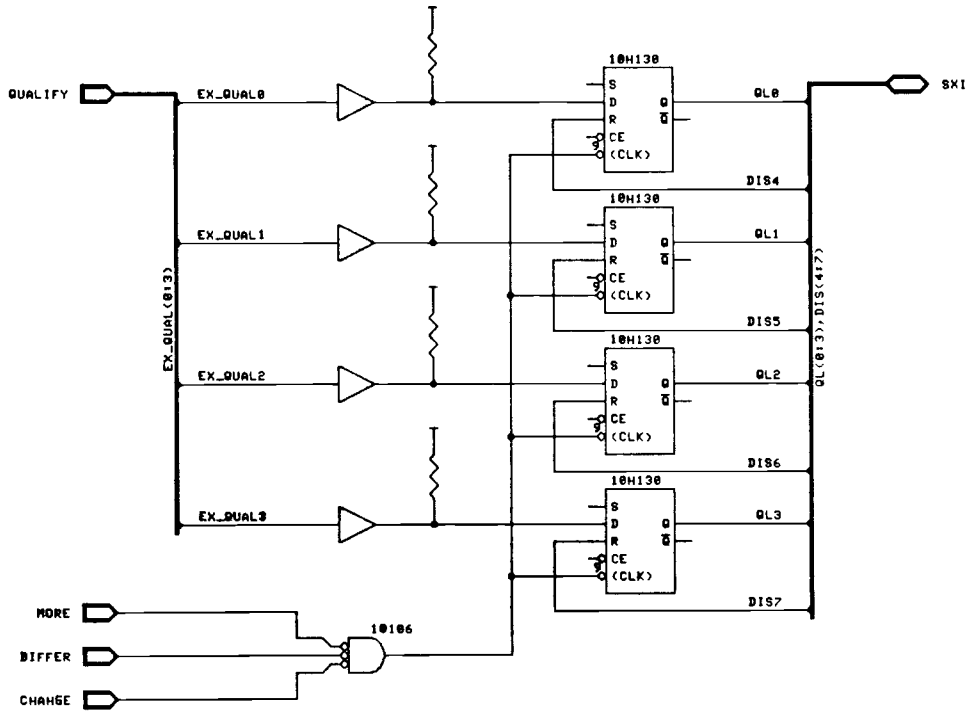


Figure 24

Practical Example - UIC Latch

When a customer uses these qualifier inputs, labeled EX_QUAL(0:3) in Figures 19 and 24, he is supposed to maintain these qualifiers at a steady state during a state change. Since there is no guarantee the customer will observe this restriction, an unbounded input change latch has been added so if this restriction is violated, the test equipment will operate in the manner programmed in the transition function. If these qualifier inputs are not stable, the test equipment may not do what the customer expected, but it will not fail in an unpredictable way.

If the customer chooses not to use a qualifier input, the reset line on the appropriate UIC latch is programmed to be asserted. This holds the latch in the reset condition and disables the input.

Microprocessor Interface

This interface machine must have great flexibility; that is, the ability to be programmed to behave as a customer requires. The purpose of the microprocessor interface, shown in Figure 25 on the next page, is to convert the microprocessor output levels to ECL levels and to program the sequential machine. The signals on the bus labeled PROGRAM are from the microprocessor.

The design issue of low cost requires making one part do double duty, if possible. This interface was designed so the level conversion process, from microprocessor levels to ECL levels, also forms a vital part of the programming process. During the level conversion process, the programming signals RISE(0:3), FALL(0:3), and DIS(0:7) are latched or saved so the sequential machine will be properly configured during its execution.

An important issue is the efficient loading of the transition function into the RAM. Since there are 4096 locations, a twelve bit counter (the 74HC193's) has been added to aid in loading the transition function. The

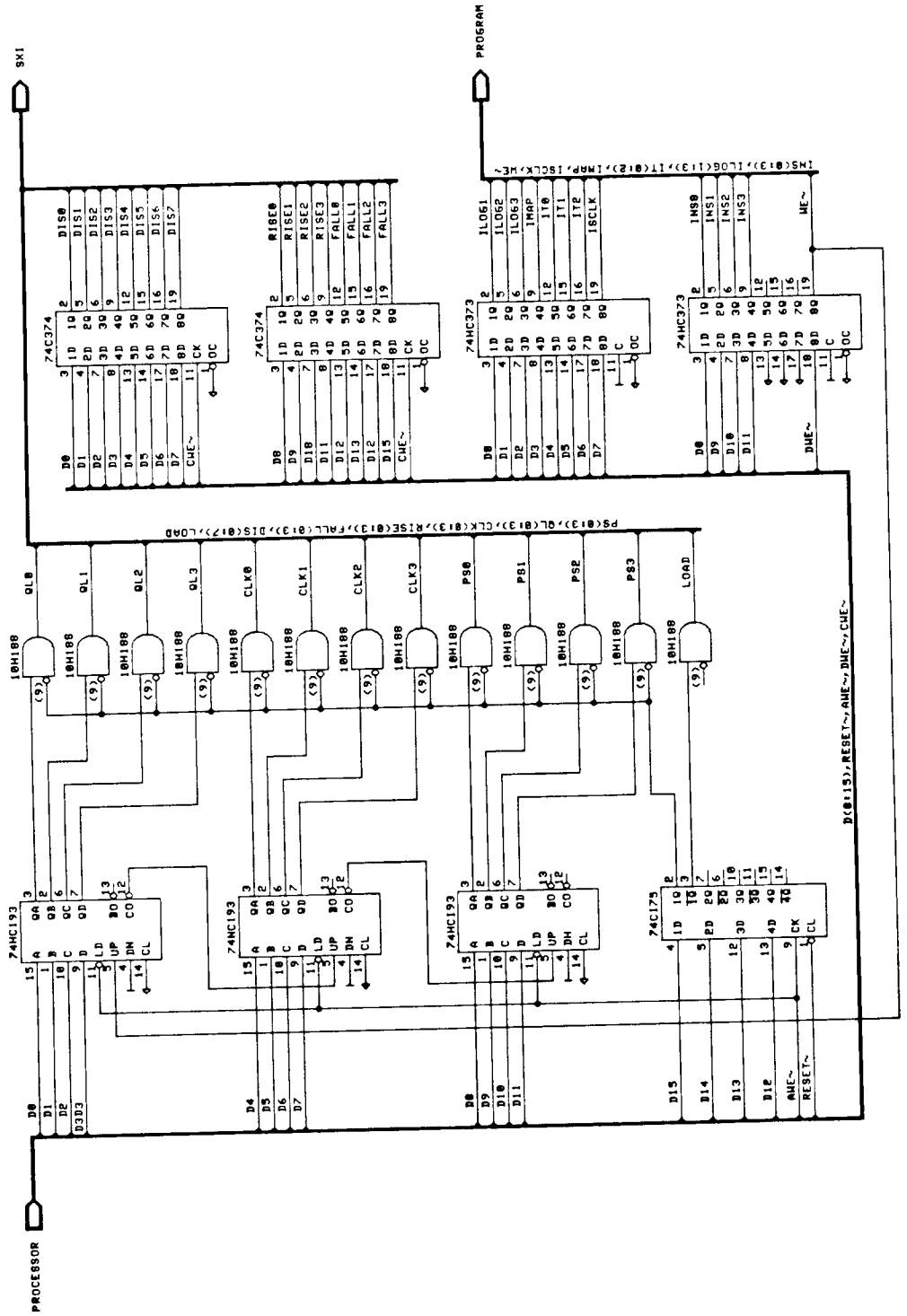


Figure 25

Practical Example - Programming Interface

microprocessor programs the first RAM address into the counter, then writes data into the RAM. The address counter is automatically incremented after each write. This method simplifies and speeds the transition function programming.

Readers who are familiar with ECL components may have noticed that one unusual aspect of this design is the power supply. ECL components usually use the power supply voltages of 0 and -5.2 volts. To ease the burden of interfacing the programming microprocessor to the ECL, this machine operates all the ECL parts level-shifted to +5 and 0 volts. This permits other positive logic families, such as CMOS, to drive the ECL directly and removes the need for a separate -5.2 volt supply.

Notice the addresses to the transition function RAM must come from one of two places, either the customer's signals (when the machine is running) or the 4096 bit address counter (when the transition function is being loaded). The ECL WIRE-OR is put to good use here. Both signal sources are WIRE-ORed together and connected to the RAM address pins. The signals not being used are forced to a low, so the other signals will control the address. When the RAM is being loaded, the DIS(0:7) lines and the LOAD signal are asserted, disconnecting the customer's signals and the PS(0:3) signals. When the machine is running, the 4096 bit counter is cleared, allowing the customer's signals to control the address.

SUMMARY AND CONCLUSIONS

This dissertation has presented a design style for a multiple input change self-synchronized machine that achieves an optimum speed solution. This design style is applicable to both SOC and MOC behavior. While the design style presented herein is not faster than the standard Huffman-Moore solution without self-synchronization when no essential hazard is present (no machine is), the greater δ_1, the greater the advantage of this solution.

This optimum solution was made possible by these key concepts introduced in this dissertation:

* the asymmetrical delay element
* the early final state indicator
* customizing the delay time to the state and input of the machine
* extending the self-synchronized machine to UIC mode

It is the asymmetrical delay, the early final state indicator, and the customized delay time that permit the design of a self-synchronized machine to processes input states at a speed determined only by the required machine behavior and the technology of implementation.

It seems incredible that a change detector customized to the required behavior of the machine should process input changes at a rate slower than a generalized change detector - custom-built is supposed to be better. The reason is now clear. After the inputs become stable, a customized change detector decides what to do and then does it - two distinct operations performed serially. A generalized change detector assumes that something is to be done. The task of producing a clock pulse proceeds in parallel with determining what is to be done, a distinctly faster solution.

While previously published self-synchronized machines have been analyzed by others for possible use in UIC mode (Unger, 1971), they have been shown to be unsuitable. This dissertation is the first to demonstrate a self-synchronized machine to operate properly in UIC mode.

Self-synchronized machines are important because they achieve the speed of a standard Huffman-Moore machine, yet free the design process from the difficult task of finding a race-free assignment and designing hazard-free logic. The final design using only the Huffman-Moore machine, without self-synchronization, could easily require more circuitry than a self-synchronized machine equivalent. The clock generator circuitry can be very small (two integrated circuits) compared to significant circuitry for hazard suppression, additional state variables (to suppress races), and multiple delay elements (one per state variable).

Unfinished work in this field includes investigation into self-synchronized unbounded output change machines. A closely related topic is the effect of lifting the traditional restriction of only one single pulse-mode input change per input state. This dissertation has assumed the flow table for the machine being designed was a design specification. There is still much work to be done in the area of machine equivalence and flow table reduction. This is especially true in asynchronous flow table reduction where the fundamental mode assumption, time dependent assumption, or time independent assumption dramatically affect the reduction process (Unger, 1969).

BIBLIOGRAPHY

Bredeson, J. G. and Hulina, P. T., "Generation of a clock pulse for asynchronous sequential machines to eliminate critical races," IEEE Trans. Comput., vol. C-20, pp. 225-226, Feb. 1971.

Chiesa, G. L., "Asynchronous processor control," IBM Tech. Disclosure Bull., vol. 22, pp. 2375-2376, Nov. 1979.

Chuang, H. Y. H., and Das, S., "Synthesis of multiple-input change asynchronous machines using controlled excitation and flip-flops," IEEE Trans. Comput., vol. C-22, pp 1103-1109, Dec. 1973.

Clare, C. R., Designing Logic Systems Using State Machines, McGraw-Hill, New York, 1973.

Fletcher, W. I., An Engineering Approach to Digital Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

Friedman, A. D., and Menon, P. R., "Synthesis of asynchronous sequential circuits with multiple-input changes," IEEE Trans. Electron. Comput., vol. C-17, pp. 559-566, June 1968.

Hartmanis, J., and Stearns, R. E., Algebraic Structure Theory of Sequential Machines, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

Hayes, A. B., "Stored state asynchronous sequential circuits," IEEE Trans. Comput., vol. C-30, pp. 596-600, Aug. 1981.

Hill, F. J., and Peterson, G. R., Switching Theory & Logical Design, 3rd Edition, Wiley, New York, 1981.

Huertas, J. L., and Acha, J. I., "Self-synchronization of asynchronous sequential circuits employing a general clock function," IEEE Trans. Comput., vol. C-25, pp. 297-300, Mar. 1976.

Huffman, D. A., "The synthesis of sequential switching circuits," J. Franklin Inst., vol. 257, pp 151-190, 275-303, March and April 1954.

Kohavi, Z., Switching and Finite Automata Theory, 2nd Edition, McGraw-Hill, New York, 1978.

Liu, C. N., "A state variable assignment method for asynchronous sequential switching circuits," J. ACM, vol. 10, pp. 209-216, 1963.

Mealy, G. H., "A method for synthesizing sequential circuits," BSTJ, vol. 34, pp. 1045-1079, Sept. 1955.

Miller, R. E., Switching Theory, Krieger, New York, 1979.

Moore, E. F., "Gedanken-experiments on sequential machines," Automata Studies, Annals of Mathematics Studies No. 34, pp. 129-153, Princeton University Press, New Jersey, 1956.

Muller, D. E., "The general synthesis problem for asynchronous digital networks," IEEE Conference Record of Eighth Ann. Symp. Switching, Automata Theory, pp. 71-82, Oct. 1967.

----, "Asynchronous logics and application to information technology," Proc. of a Symp. on the Application of Switching Theory in Space Technology, Stanford University Press, Mar. 1962.

Muller, D. E., and Bartky, W. S., "A theory of asynchronous circuits I," Report No. 75, University of Illinois, Digital Computer Laboratory, Nov. 1956.

----, "A theory of asynchronous circuits II," Report No. 78, University of Illinois, Digital Computer Laboratory, Mar. 1957.

----, "A theory of asynchronous circuits," Proc. of an Int. Symp. on the Theory of Switching, vol. 29, Annals of the Computation Laboratory of Harvard University, Harvard University Press, pp. 204-243, 1959.

Rey, C. A., and Vaucher, J., "Self-synchronized asynchronous sequential machines," IEEE Trans. Comput., vol. C-23, pp. 1306-1311, Dec. 1974.

Sholl, H. A., and Yang, S. C., "Design of asynchronous sequential networks using read-only memories," IEEE Trans. Comput., vol. C-24, pp. 195-206, Feb. 1975.

Stone, H. S., Discrete Mathematical Structures and Their Applications, Science Research Associates, Chicago, 1973.

Tan, C. J., "State assignments for asynchronous sequential machines," IEEE Trans. Comput., vol. C-20, pp. 382-391, April 1971.

Thomas, B., and Chandrasekharan, P. C., "Economical realization of asynchronous sequential circuits using random-access memories," IEE Proc., vol. 128, pp. 129-132, May 1981.

Tracey, J. H., "Internal state assignments for asynchronous sequential machines," IEEE Trans. Electronic Computers, vol. EC-15, pp. 551-560, Aug. 1966.

Unger, S. H., "Hazards and delays in asynchronous sequential switching circuits," IRE Trans. Circuit Theory, vol. CT-6, pp. 12-25, Mar. 1959.

----, Asynchronous Sequential Switching Circuits, Wiley Interscience, New York, 1969.

----, "Asynchronous sequential switching circuits with unrestricted input changes," IEEE Trans. Comput., vol. C-20, pp. 1437-1444, Dec. 1971.

----, "Self-synchronizing circuits and nonfundamental mode operation," IEEE Trans. Comput., vol. C-26, pp. 278-281, Mar. 1977.