Chapter 7: System Timing

by Charles L. Seitz,

Dept. of Computer Science, California Institute of Technology

Sections:

The Third Dimension

The successful design of large scale integrated systems requires careful management not only of the two-dimensional silicon area, but also of the operation of the system in the time dimension. Although time is physically different than the spatial dimensions, the general strategies already introduced for carrying the spatial design from conception to layout apply to system timing as well. These are the usual strategies for containing complexity: use of abstraction and structured design.

Much of the functional design of the spatial aspect of a system is done with the help of block diagrams, logic diagrams, circuit diagrams, and stick diagrams, in a metric-free topological domain. These representations are helpful because they allow designers to suppress detail, so that they can think about system behavior at a level of abstraction which is effective for the task at hand. One specific abstraction employed in these diagrammatic representations is the suppression of geometrical detail, while focussing on the topological structure of the circuit or system. Topology is sufficient to specify information flow between functional parts, so diagrammatic representations are a useful abstraction to the functional or logical structure of a system.

The third dimension, time, may also be regarded as having features analogous to geometry and topology. The definition of a sequential process -- whether represented by a program, flowchart, state diagram, or in plain English -- specifies only the ordering, or partial ordering, of the individual steps that compose it. Thus it is the metric-free "topological" concept of *sequence*, rather than the physical concept of a *time metric*, that is most useful for the functional design of a system.

As was pointed out in section 3-[Relating Different Levels of Abstraction], it is important that the levels of abstraction be related to each other and to physical concepts. The *sequence domain* is a self-

consistent abstraction that applies across several levels of system design -- programming, organization, logic. However, flowcharts and state diagrams do not say anything explicit about space, time, and other physical characteristics of a system, just as logic diagrams do not. The value of abstraction to the design process is that it permits one to defer certain bindings to physical form. The hazard is that one can become so isolated from physics and economics as to produce elegant schemes that are unworkable in practice. Thus, an important goal of any study of timing is to devise and explain methods by which sequence and time can be related.

At some level in the mechanization of a sequential process, one may no longer ignore the time metric. The electrical behavior of devices and wires is governed by physical laws which are expressed as partial differential equations in time. Devices and wires also take space, and their temporal behavior depends on these geometrical aspects of their construction. It is not generally possible, therefore, to separate the spatial and temporal aspects of system design. Failure to account for physical delays in the implementation of systems frequently results in unreliable operation, poor performance, or both.

Time is also important to people. We believe that the process of design starts with a conception of the functional operation of a system, together with a set of requirements -- or desires -- expressed in metrical units of space and time. These requirements are determined not by physics, but by human needs, expectations, and desires. The "interactive" text system that requires several seconds to respond to a keystroke is misnamed.

Unfortunately, the world is full of examples of digital systems which -- even when functionally correct -- have disappointed their designers and users as being unreliable or too slow. Why is it that so many systems have "timing problems", or fail to achieve performance objectives?

As is the case with the spatial dimensions, the design problems in the third dimension result not from a lack of possible forms, but rather from an overabundance. If one is to build a large scale integrated system with any hope of correct operation, it is necessary to restrict oneself to a consistent style of design. The canonical forms in the time dimension are signaling conventions which are adhered to throughout the system, and serve the function of establishing between all parts engaged in a communication an interval or sequence of intervals of time for this communication. If such a scheme is to be regarded as a discipline, it must be possible to state precisely the requirements that the signaling convention places on system interconnections and element timing.

Alternative disciplines of design in this dimension can be characterized by the way in which they connect sequence and time. Let us take as an example the synchronous discipline of design, which has been used in a form with a two-phase clock in the designs presented previously in this book. Here, sequence and time are connected by means of the clock. The term "synchronous" comes from Greek: syn, or sym = same, or together + chronos = time; and the discipline is well named, since it requires all parts of the system to operate together in time. Since synchronous systems are by far the best known and most widely used, we take them as the starting point for the body of this chapter.

Synchronous systems possess some serious limitations, which are made even worse as λ is scaled down, and as systems become larger. One problem is efficiency. It usually happens that most of the combinational paths are short, and the system clock period is determined by one or a few seldom used slow paths. One particularly difficult situation with slow paths occurs when synchronous signals must be driven off-chip. The time required to drive a signal off-chip is today a substantial fraction of the minimal clock period of about 100τ. As λ is scaled down, the τ-relative delay off-chip gets larger, indeed may exceed the minimal clock period. So, it appears that synchronous communication across chip boundaries will become less and less attractive. Synchronous communication within a chip appears to be at least possible down to the fundamental limit of about 0.25μ channel lengths, but it would be very difficult to manage a synchronous design of the number of parts implied by this scaling while achieving reasonable efficiency.

The same considerations of managing the design of very large integrated systems which provide a motivation for dividing a system into modular parts argue that the parts be independently timed. If the parts are each synchronous systems with *independent* clocks, information communicated from one part to another must be synchronized to the receiver's clock. Unfortunately, as we show in a later section, this synchronization cannot be accomplished with complete reliability. The reason for this problem is that synchronizing elements are bistable, and have a metastable or balanced condition that occurs under the conditions in which synchronizers must operate. As was discussed in Chapter 1, there is no bound for the time the bistable element may remain in this metastable condition. There are many methods to reduce the probablity that such a fault would crash a system, but all cost time and so reduce efficiency.

The limitations imposed by the synchronous discipline suggest that other disciplines be tried. The final sections of this chapter are devoted to an outline of an alternative called *self-timed logic.*

This term refers to a discipline of digital system design in which the timing aspect of the design is confined to the interior of *elements*. Elements can be designed in a number of ways, for example, as synchronous systems with an internal clock which can be stopped synchronously and restarted asynchronously. This type of clock allows synchronous elements to communicate reliably, because their clocks are partly dependent, i.e. not independent. Elements may also be designed as speed-independent asynchronous circuits. In any case, the terminal behavior of a self-timed element must satisfy a *sequence domain* representation, which assures that correct sequential operation of a self-timed system is insensitive to element and wiring delays. There are no clocks or global time references in a self-timed system. Instead, initiation of a given computational step depends on completion signals produced by its sequential predecessors. Thus self-timed systems operate at a rate determined *locally* by element and wiring delays, a rate which tends to reflect typical rather than worst case delays.

The subject of self-timed logic has two principal facets, the design of elements and the design of systems of interconnected elements. Along the seam between these subjects are conventions for delay-insensitive signaling. This bifurcation of the discipline is deliberate. The design of elements is difficult because it is here that logic, physics, and timing come together. However, the element designer can work within a domain in which physical and logical scale are both restricted to be small enough to make the design managable. The design of systems is difficult because of the combinatorics of scale. However, the system designer can work within a domain similar to that of a programmer, in which many of the details of the underlying physical system have been suppressed and replaced by an abstraction which is free of hidden rules -- namely, the sequence domain abstraction.

Synchronous Systems

In the synchronous discipline of design, sequence and time are connected through the use of a system-wide *clock* signal. The clock signal serves two purposes -- or one might say it serves two masters. The clock is a global sequence reference, and is also a global time reference. As a sequence reference, its *transitions* serve the *logical* purpose of defining successive instants at which system state changes may occur. As a time reference, the *period* or interval, either fixed or variable, between clock transitions serves the *physical* purpose of accounting for element and wiring delays in paths from the output to input of clocked elements.

The ability of the clock signal to serve two masters, logic and physics, has a certain compact elegance, and conforms to an obsolete tradition of parsimony in the use of active elements. However, the dual role of the clock binds the system sequencing and timing so closely that "timing" is the source of numerous difficulties in the design, maintenance, modification, and reliability of synchronous systems.

The logical model which synchronous systems resemble is the finite-state machine, a model that has been described in detail in section 3-[Finite-state Machines] and in chapter 6. As illustrated in Figure 1, any such system must satisfy a *topological requirement* that every closed signal path pass through a *clocked storage element.* Closed paths which do not pass through clocked storage elements are excluded as they may create non-deterministic behavior either through oscillation or through asynchronous latching. There are several important consequences of this topological constraint on the logical design. First, it assures deterministic behavior if the physical aspects of the design are also correct. Second, it relieves the designer of any requirement that the combinational logic be free of transients -- static or dynamic hazards -- on its outputs. The only dynamic characteristic of a combinational net that matters is its propagation delay time. Finally, the storage or history dependence of the system resides entirely within the clocked storage elements, a fact which simplifies the design process and often also the maintenance and testing of a system.

The clocked storage elements in a synchronous system may take any of a variety of forms, discussed below, depending on physical requirements such as speed, economy, or static operation. While these elements are distinguished as being the only recipients of clock signals, in practice there may be a number of timing signals derived as different phases or submultiple frequencies of the clock. In these cases it may be difficult to see the correspondence between the circuit and the

finite-state model. Circuits such as shift registers are finite-state machines, but already possess such a natural and regular structure that it would be pointless and awkward to describe their behavior with state diagrams. Control elements, such as the finite-state machine stoplight controller described in section 3-[Finite State Machines], are a case of imposed structure, in which the combinational logic and clocked storage elements can be patterned on the silicon in a form that mimics the usual block diagram of a finite-state machine.

Clocked storage elements may be clocked by different schemes, but all are binary storage devices. The sort of physical device that has the property of storing information -- also called memory, or history dependence -- are those which store energy, or are those such as film or punched cards in which energy is required to change some detectable condition of the medium.

For semiconductor integrated circuits the energy represented by charge stored on circuit capacitances is the only practical mechanism for storing information. Inductance plays the same role in superconducting circuits. MOS circuits employ this mechanism very directly in the dynamic register introduced in Chapter 3 and used in designs throughout this book. In the dynamic register illustrated in Figure 2, the output stored data follows the input as long as the enable input to the transfer gate is high. When the enable signal goes low, the charge stored on the node is very well isolated, and so maintains the same voltage.

Unfortunately, this is not the whole story. While the charge is very well isolated, it is not perfectly isolated. Charge escapes by two different mechanisms, which scale differently. The principal leakage path for 1978 MOS technology is the reverse leakage current of the drain junction of the pass transistor. The time constant of this decay is in the order of a few seconds at room temperature, but decreases exponentially with temperature to a millisecond or so at 70°C. This leakage path is a current per unit area, and because scaling down the circuit dimensions increases the capacitance per unit area due to decreased oxide thickness, the time constant of charge decay increases with reduced circuit dimensions. The scaling is largely masked by the exponential temperature dependance, however, so the time constant of junction leakage is reasonably regarded as constant for values of λ over the recent past and future. Subthreshold currents are expected to become the limiting factor in holding charge on a node as soon as threshold voltages are reduced to much below 1 Volt. This effect of scaling down the dimensions of MOS circuits was discussed in section 1-[Effects of Scaling Down the Dimensions of MOS Circuits and Systems]. We refer to the time a node will reliably hold a high level as the *refresh period.*

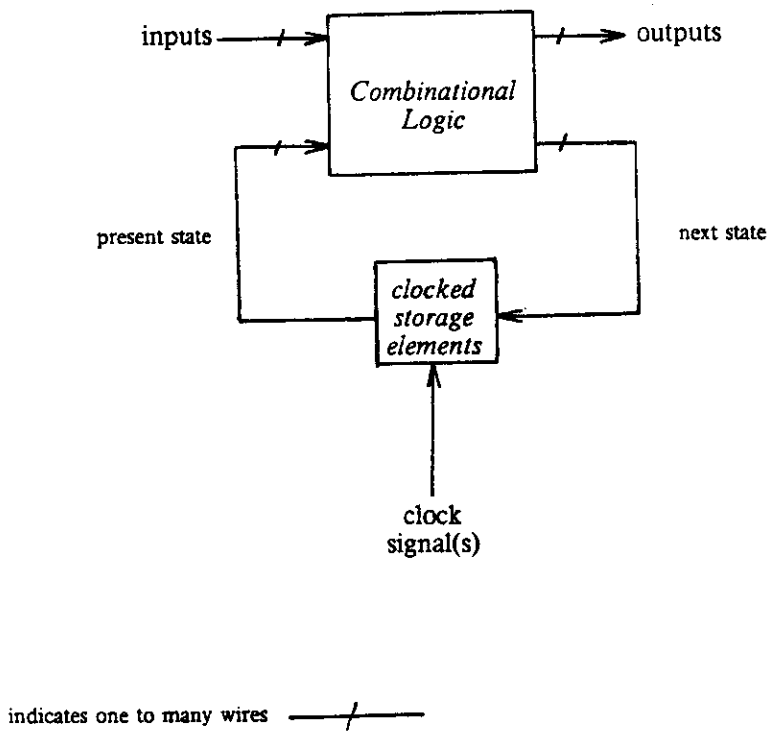


Figure 1: Finite-state model

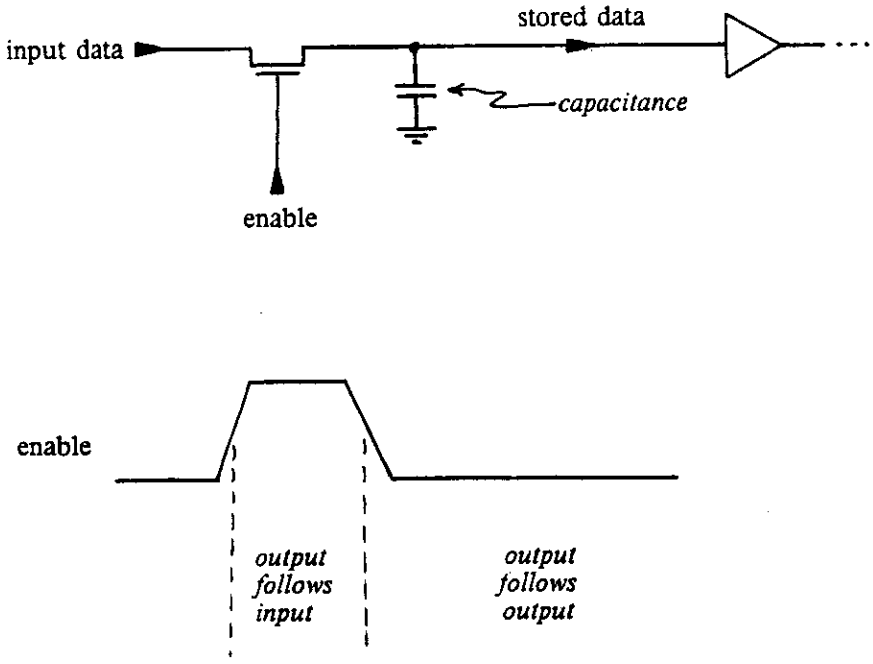


Figure 2: Dynamic register (1 bit)

The decay of charge on a dynamic node is no problem so long as the charge is sensed and refreshed frequently enough, for example in every clock period as is done in the OM design. In some cases this is not possible, and so-called *static registers* are used instead. An assortment of static registers is shown in Figure 3. The general idea evident in the first two circuits is to amplify and feed back the stored data so as to counteract the decay of charge. The first circuit does this through a resistance which must be much larger than the effective R_{on} of the pass transistor, so that the storage node can be driven to the value desired. For voltages close to the switching threshold this circuit amounts to a negative resistance termination to V_{inv}, where the resistance is -(large R)/(voltage gain of the pair of inverters). This circuit can be used advantageously as a termination for buses to assure static operation. The third diagram is a storage circuit typical of bipolar families, in which low impedances make dynamic storage unattractive, so that static storage is the rule rather than the exception.

These static circuits are logically equivalent to the dynamic storage circuit, except that the stored information will remain indefinitely. The use of extra circuitry for each node to accomplish this continuous refreshing is usually unnecessary, and it will be omitted in the following discussion and figures with the understanding that it could in all cases be included where required.

As an aside, the reader may wonder about the statement above that capacitance provides the mechanism for information storage. Is this true of the static and cross-coupled storage circuits as well? Many references on switching circuits leave the impression that the existence of two logically consistent stable states in these cross-coupled circuits is sufficient to insure that the circuit will store a bit. Some references mention also that the circuit must have more than unity gain around the loop, which is indeed a necessary but not sufficient condition. Consider the consequence if the circuit capacitances were all taken to zero. The circuit would then be able to respond instantly to an excitation, which means that a current pulse of arbitrarily short duration could change the state of the circuit. If this arbitrarily small amount of energy could change the state of the circuit, one could not reasonably expect the circuit to remain in either state. Without capacitance, it cannot store information. This physical aspect of storage devices is discussed in detail in Chapter 9.

With this background on storage elements, let us proceed to clocking schemes. As we pointed out in the opening section of this chapter, it should be possible to state precisely the requirements the timing form places on system interconnections and on element timing. For synchronous systems, the requirement on interconnections is the topological requirement that all closed paths

pass through a clocked storage element. The requirements on element timing depend on the clocking scheme.

The storage devices described above, all logically equivalent to the dynamic register, may be used in a cheap, fast and *risky* clocking scheme illustrated in figure 4. We know of no example of this scheme being used in LSI circuits, but it was common in many of the early "transistorized" computers, with the gated, cross-coupled storage element shown in figure 3. This scheme might best be termed "narrow pulse clocking," because it requires that the clock pulse be narrow compared to the delay of the combinational logic. The present state information changes a short time, about $R_{on}C_{in}$, after the leading edge of the clock. The delay through the combinational logic must be greater than the clock width, or else the change in the present state information will propagate through the combinational logic to change the next state information before the trailing edge of the clock.

Once a clock period and width are established, the combinational logic must be designed to satisfy a *two-sided bound* on its delay time -- greater than the clock width and less than the clock period. As indicated above the clock width is also bounded on two sides -- below by the time required to transfer charge to the present state inputs of the combination logic, and above by the minimum delay of the combinational logic. The clock period also has a two-sided bound, unless static registers are used. The relations which must be satisfied are summarized in figure 4. They are relations which apply in a worst-case sense, in spite of variation in temperature, power supply voltage, aging, and manufacturing.

This "narrow pulse clocking" scheme was abandoned because of the difficulty of satisfying so many two-sided bounds simultaneously under so many conditions of variation. Also, the economies achieved due to the simplicity of the clocked elements were partly offset by necessity to "pad out" the delay in many of the combinational nets. This clocking scheme is quite feasible for certain LSI systems, since it is inherent in their manufacture and operation that most of the variables will track if the clock signal is generated on-chip.

Two-sided bounds on timing create enough difficulties in the design and maintenance of a system that it is generally worthwhile to use more logic to make the timing bounds one-sided. Elimination of the two-sided bound on clock width, or the complementary bound on combinational delay, requires the use of at least two clock phases. This minimum form occurs for much the same reason that ship canal locks require at least two watertight gates.

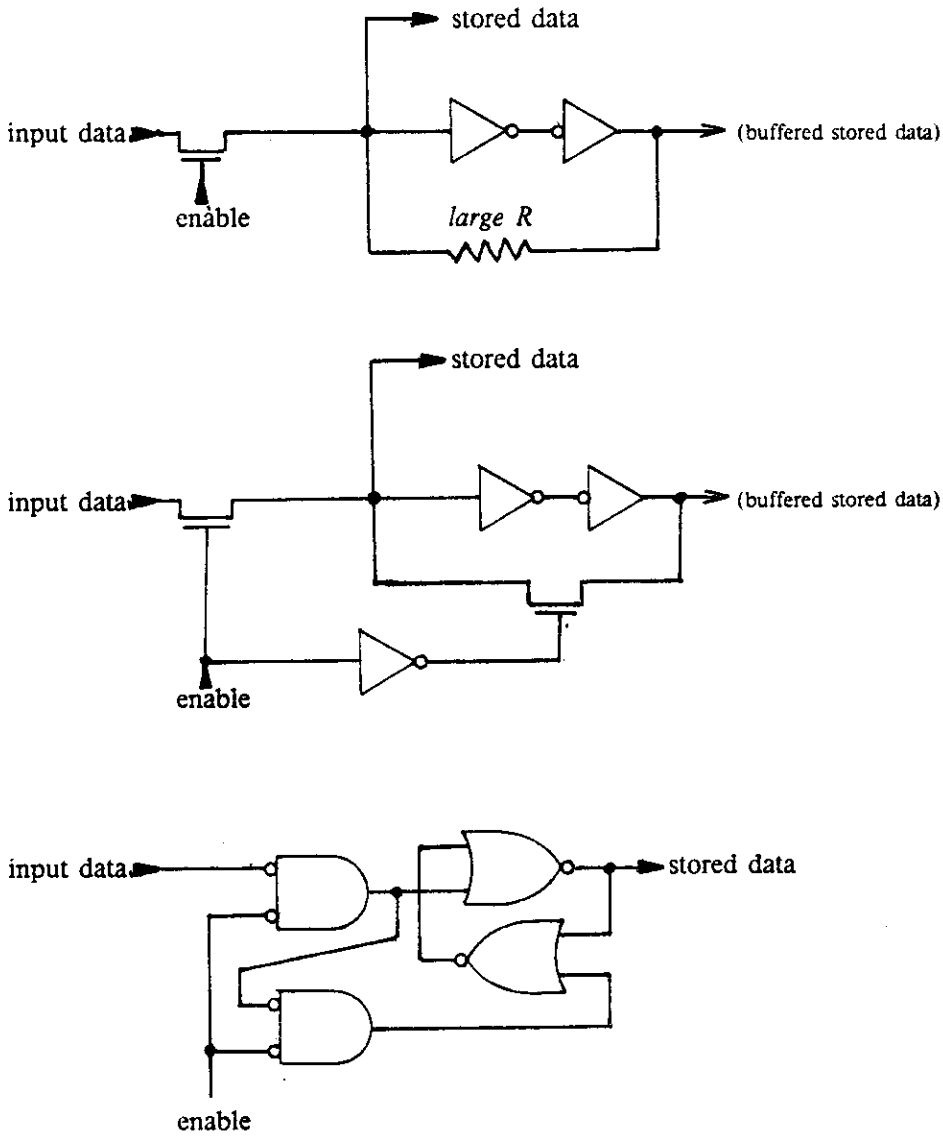
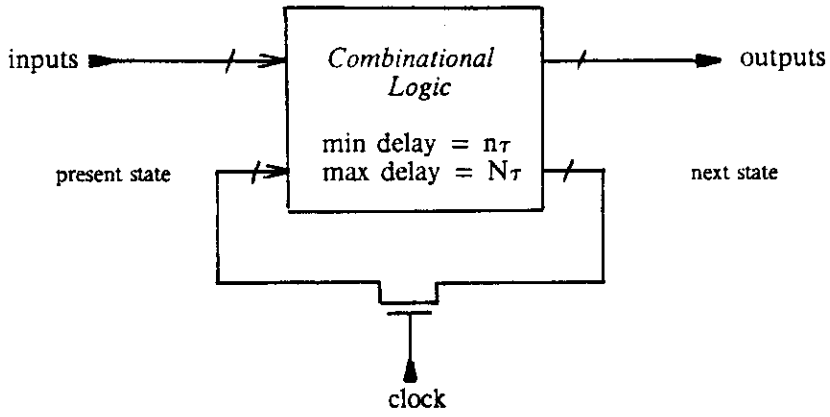
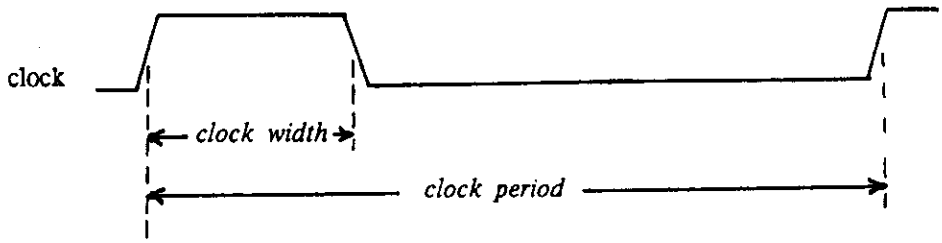


Figure 3: Static Registers





$$(R_{on}C_{in})_{max} < \text{clock width} < n\tau + (R_{on}C_{in})_{min}$$
$$N\tau < \text{clock period} < \text{refresh period}$$

where R_{on} is the effective on resistance of the pass transistor, and
C_{in} is the input capacitance of the combinational logic

Figure 4: Cheap, fast, and Risky Clocking Scheme

The two-phase clocking scheme illustrated in figure 5 includes four sequentially repeated epochs. During $\varphi 1$, previously stored information is applied to the present state inputs of the system's combinational logic. The $\varphi 1$ signal must remain high long enough to charge the present state input nodes, a process which incurs some delay on the order of $R_{on}C_{in}$, and which is called the *delay time* of the clocked storage element. Following this delay, if inputs are also available, the combinational logic starts setting up the outputs and next states, independent of when $\varphi 1$ may transit from high to low. This epoch is analogous to the operation of a canal lock releasing a ship. The gate must open before the ship leaves, and the ship must clear the gate before it is again closed. If the lock master chooses to leave the gate open for a while after the ship leaves, it does not slow down the ship.

What the lock master must never, never do is to open both gates at once! The epoch labeled in figure 5 as t_{12} is an interval produced by the non-overlapping phases of the clock. By analogy with the narrow pulse clocking scheme, it is clear that overlap less than the minimum delay of the combinational logic is harmless to correct operation. However, as the minimum delay of the combinational logic is ordinarily legislated to be zero -- most people would agree it could not be less, and for circuits such as shift registers the delay does approach zero --, the overlap period t_{12} must be greater than zero. In practical cases, because t_{12} does not represent "dead time", but time during which the combinational logic is working, this time is made as short as is convenient but not necessarily as short as is possible.

In the epoch during which $\varphi 2$ is high, the clocked element samples its input. The combinational outputs must be stable slightly before the trailing edge of $\varphi 2$, an interval called the *preset time* of the clocked storage element. After this point in time, changes at the input of the pass transistor clocked by $\varphi 2$ will not be fully passed to its output. Of course, $\varphi 2$ must be wider than the preset time. This $\varphi 2$ epoch is analogous to the entry of a ship into a lock. The ship cannot enter the lock until the gate is opened, and the gate should not be closed until the ship is completely inside.

Following $\varphi 2$ is another period of non-overlap, t_{21}, during which the system is idle. The minimum clock period for correct operation is the maximum combinational delay, plus the maximum delay time and preset time, plus t_{21}. It is important therefore to make t_{21} as small as possible if one is designing for performance. As we show in the following section, t_{21} does serve a useful purpose of accommodating clock skew, a variation in the arrival time of the clock to

different clocked storage elements, and so in some systems t_{21} can be reduced only as much as a clock skew allows.

The net result of this two-phase clocking scheme is that the clock period and its constituent epochs are, with static storage devices, bounded only below. A region of correct operation can always be found by making periods larger. With dynamic storage devices, the upper bound on the period is so large compared with the lower bound, at least for present values of threshold voltages, that the region of correct operation is always adequate. The complementary requirement on the propagation delay of the combinational logic is a simple upper bound.

Many variations on this basic scheme are found in different kinds of digital systems, more schemes than we could hope to describe individually. Many processors and storage systems are advantageously designed with more than two clock phases. In processors these phases -- usually four or eight, and sometimes a variable number -- delineate minor cycles which subdivide the major cycle. The general rules here are quite simple. No path may lead from a register or storage element output to an input clocked on the same phase. The maximum propagation delay time of a path from a register clocked on some phase, say $\varphi 3$, terminating at the input of a register clocked on some other phase, say $\varphi 6$, cannot exceed the minimum time between the leading edge of $\varphi 3$ to the trailing edge of $\varphi 6$, less delay and preset times. Multiple phases are also commonly used in systems which employ precharged pullup, and other charge transport techniques such as CCD storage. Magnetic bubbles are made to move in response to a rotating magnetic field, a two-phase clocking scheme when viewed as two orthogonal fields with sinusoidal oscillation 90^{O} out of phase with each other.

Some mention of variant forms with respect to inputs and outputs is also required. Inputs to synchronous systems must appear in synchrony with the clock, a requirement which is readily satisfied when the inputs are outputs of another system which shares the same clock. If an input does not assume its correct value until after the leading edge of $\varphi 1$, its worst-case delay relative to the leading edge of $\varphi 1$ is accounted for in the same way as the delay of the clocked storage element. The general procedure for checking compliance with timing bounds consists of marking nodes starting with clocked element outputs and system inputs with the latest time relative to the beginning of $\varphi 1$ that the signal will become stable. Programs to accomplish such checking are not trivial, as they may at first appear, because the program should account for different delays in different states and inputs. Otherwise, the large differences between delays for positive and negative transitions cannot be accounted for; nor can circumstances such as time of output

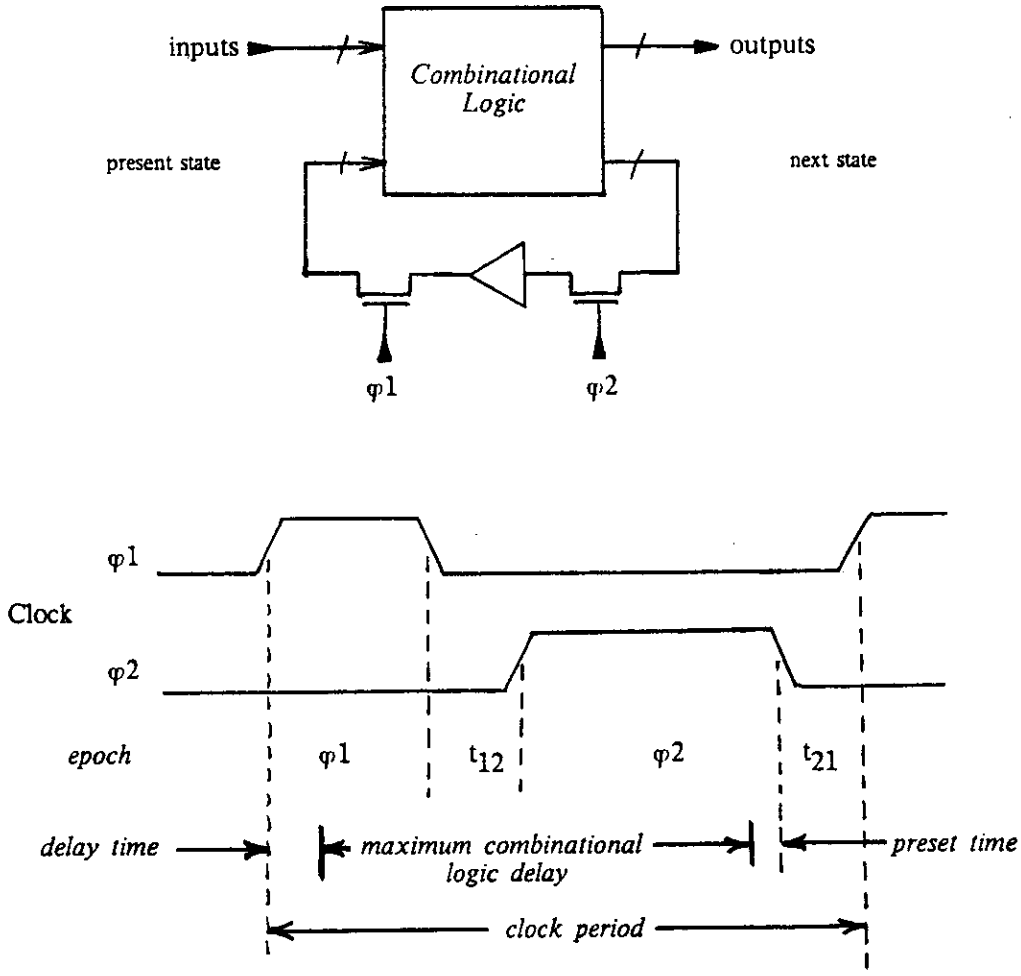


Figure 5: Two-phase Clocking

determination in simple AND and OR circuits be dealt with simply, because these times are data dependent. Unfortunately, this checking problem is about as difficult as exhaustive simulation.

The form of finite-state machine model of synchronous systems used in the descriptions above of clocking schemes is the transition output (Mealy) machine. It is more general than the state output (Moore) machine in that the outputs are functions both of the input and present state, while for the state output machine the output is a function only of the present state[9]. Transition output machines tend to be used in cascade arrangements in which economy is the principal design goal, while state output machines are characteristic of pipeline architectures in which high performance is sought.

Networks of transition output machines must be acyclic, the same requirement as for the combinational paths in a single machine. Combinational delays may accumulate on paths through many machines, so the general checking procedure described above must be used. Networks of state output machines may be connected cyclically, and checking is confined to communicating pairs of machines. Checking can be made entirely local to each finite-state machine if communication between machines is performed in pipeline fashion through clocked elements. The finite-state machine described in section 3-[Finite State Machines, (figure 14)] is of this sort.

In a two-phase clocking scheme, $\varphi 1$ and $\varphi 2$ are symmetrical in the sequence sense, and perhaps also in time. This symmetry suggests a reversal of roles between $\varphi 1$ and $\varphi 2$ is possible. For example, there is no reason in the finite-state machine structure shown in Figure 5 not to replace the simple amplifier or double inverter with combinational logic. The general structure allowed in a two-phase clocking scheme is any composition of basic elements consisting of registers followed by combinational logic, in which outputs of elements clocked by $\varphi 1$ drive only inputs to elements clocked by $\varphi 2$, and vice-versa. Please note that combinational delays must be checked across communicating pairs of machines. This scheme is similar to that used in the OM2 and its controller, and is well illustrated in Chapters 5 and 6.

Clock Distribution

Readers who have been designing systems with catalog parts are now invited to stand up and object: "What is with all of these clock phases? My systems use a single-phase clock." That is a good question.

The clock supplied externally to catalog parts such as registers, counters, shift registers, and microprocessors is most often a single-phase clock because this approach is convenient and the clock then uses only a single package pin. Internally, a two-phase clock or its functional equivalent is derived from the single-phase clock, either as part of each clocked element with the single-phase clock distributed through the chip, or once for the chip with the derived two-phase clock signals distributed as required.

Figure 6a shows a relationship between a single-phase clock and a two-phase clock derived from it, and figure 6b is a circuit which performs this function. While other conventions of relating single-phase and two-phase clocks are possible, in what follows this form will be taken as canonical. The single-phase clock is used in a trailing-edge triggering discipline, so-called because system state changes occur following the trailing edge of the clock pulse. This mode of operation has many advantages over leading-edge triggering. For example, because the state variables are stable during the clock pulse, one can perform logical operations between the clock and state variables in order to derive gated clock signals for selective register loading.

The preset and delay time relations to the derived two-phase clock can be translated, as shown in figure 6a, to be referred to the trailing edge of the single-phase clock. A study of this figure shows that, because of delays in deriving $\varphi 1$ and $\varphi 2$, preset times referred to the single-phase clock may be negative. Over a set of clocked storage elements the preset times will have maximum and minimum values, just as delay times will. The maximum preset time is called the *setup time*, and the minimum preset time, with its sign reversed, is called the *hold time*. In single phase clocking arrangements, the input to a clocked storage element should become stable by the setup time before the clock edge, and hold this value until at least the hold time after the clock edge. In the two phase clocking arrangements most often used in MOS LSI, the preset time is always positive since it is referenced to the trailing edge of $\varphi 2$, so only its maximum value or setup time is critical to correct circuit operation.

It is an essential feature of clocking schemes with a one-sided bound that there is a critical period

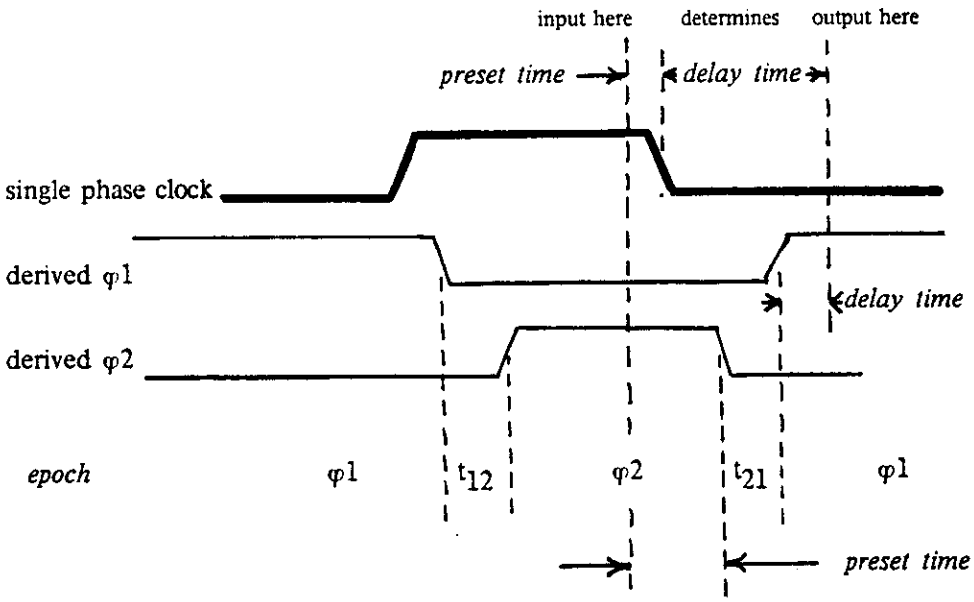


Figure 6a: Relation between a single phase clock and a two-phase clock

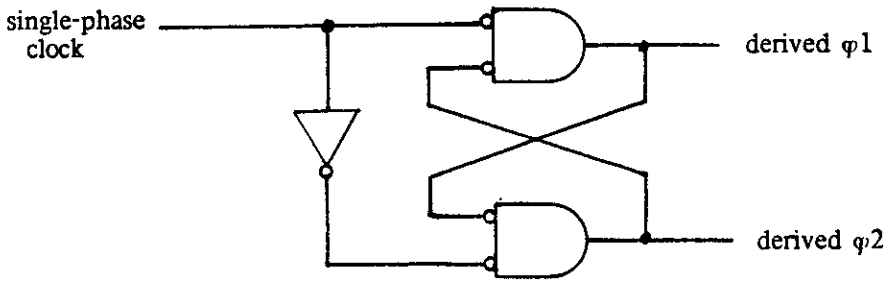


Figure 6b: Logic Circuit to derive two-phase *non-overlapping* clock from a single-phase clock

during which the clocked storage element is actually storing two bits of information. For either single- or two-phase clocking this critical period begins at the preset time and ends at the delay time. One of the stored bits is the input value presented before the preset time; the other is the bit presented at the clocked storage element output. This critical period provides a built-in tolerance for clock skew. This term refers to a variation in the effective arrival time of the clock at different clocked elements. These variations may be due to a combination of several effects: different threshold voltages, signal propagation delays on wires, or variation in element delays such as is found when gated clock signals are used to control register loading.

Clock skew even within a chip can be a problem. As was pointed out in section 2-[Electrical Parameters], propagation of signals on poly lines, as they are of fairly high resistance, is a diffusion process for which the delays are not negligible. Given the distances and need for short delay in clock distribution, use of the diffusion layer for carrying the clock more than short distances is not recommended either. For example, the delay in a line 6μ wide and 6 mm. long is calculated by the method presented in section 1-[Delays in Another Form of Logic Circuitry], and using the typical 1978 MOS electrical parameters given in Table 1, section 2-[Electrical Parameters], to be about 100 nsec. in $70\Omega/\square$ poly and about 30 nsec. in diffusion. Propagation over the same distance on a 9μ metal line requires only 0.1 nsec. Other sources of clock skew are little problem. Threshold voltages of elements built as monolithic companions tend to track. Skew in signals produced by gating a clock is present but consistent, and can be controlled by a method presented later in this section.

Clock skew is a much more serious and common problem in systems built from "families" of SSI and MSI circuits. The origin of the problem is not entirely clock distribution, but the manufacturers' inattention to preset and delay time bounds. It is easy to find examples in popular families of catalog parts certain pairs of clocked elements in which the hold time (minimum negative preset time) specified for one part is greater than the minimum delay time of another part. Although the system may work if delays are "typical," some fraction of a production run can be expected not to work, or worse, may fail intermittently in service. Designers of families of circuits intended and advertised to work together should strive to make all preset and delay time characteristics identical within a family, as variations from one part to another decrease the tolerance of the system to clock skew.

Let us return now to the typical MOS integrated system with a two-phase clock distributed as the signals $\varphi 1$ and $\varphi 2$. These are generally "popular" signals, second only to VDD and ground, and

the capacitance accumulated in their distribution and gate connections is considerable. It is possible, of course, to drive $\varphi 1$ and $\varphi 2$ directly onto the chip from an external driver. Systems operating at the extremes of performance of the MOS technology benefit from this approach. The external clock driver can be made to switch fast, and to a higher voltage than VDD, so that transmission gate outputs can then transit all the way to VDD. Because the saturation current varies as $(V_{gs}-V_{th})^2$, there is a large speed advantage on the chip of the higher voltage clock drive.

The performance benefits from driving $\varphi 1$ and $\varphi 2$ onto the chip from fast off-chip drivers are achieved at the expense of external components. Although on-chip clock drivers may limit the performance of a system slightly, they are generally used in the interests of economy. An integrated system which is large by today's standards may have a clock load on the order of $10^4 C_g$, where C_g is the gate capacitance of a minimum dimension transistor. The techniques developed in section 1.[Driving Large Capacitive Loads] for driving large capacitive loads are applicable to clock driving. The total delay in an exponential driving structure with the optimum fanout of e, starting from a minimum energy signal to drive $10^4 C_g$, is only 25τ. However, the last stage of the driver would be impractically large; the gate width would be $2{\cdot}10^4\lambda/e$. The tradeoffs between performance and area may dictate a large fanout for the final stage of the clock driver, say about 40, with smaller fanouts for the drivers preceeding it. Most of the delay in this driver structure would be in the final stage.

A clock driver of this sort is illustrated schematically in figure 7a, with size ratios indicated by the delay times. Waveforms expected for this circuit are shown in figure 7b. The clock is assumed to originate from somewhere off the chip. There is no reason to use two pins for this function where one would serve, so the canonical scheme for production of a two-phase clock from a single-phase clock is used. The capacitance of the pin and package are so large compared to C_g that there is no point is starting from a minimum energy signal. The structure shown presents a load of about $30C_g$. If the clock source were an on-chip clock generator, more stages would be used. The slight asymmetry between $\varphi 1$ and $\varphi 2$ is of course due to the inverter at the input.

The reader should take careful note of an important characteristic of this clock driver; namely, that non-overlap of the clock phases is assured independent of clock loading. This desirable characteristic is the reason that it is the clock phases, rather than the NOR gate outputs, that are fed back in a cross-coupled fashion. The non-overlap periods can be reduced somewhat at the expense of silicon area by cross-coupling at every stage. This same circuit trick can be extended

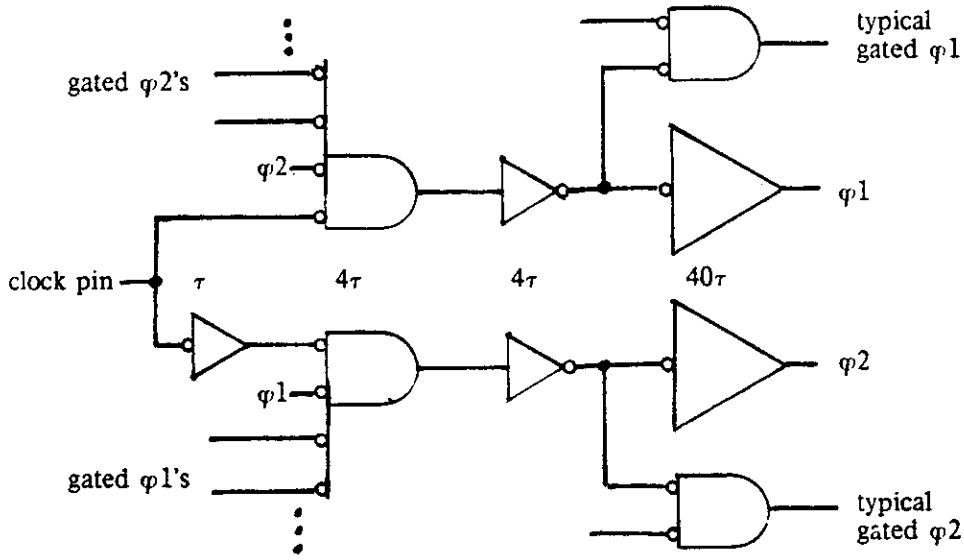


Figure 7a: Clock Driver

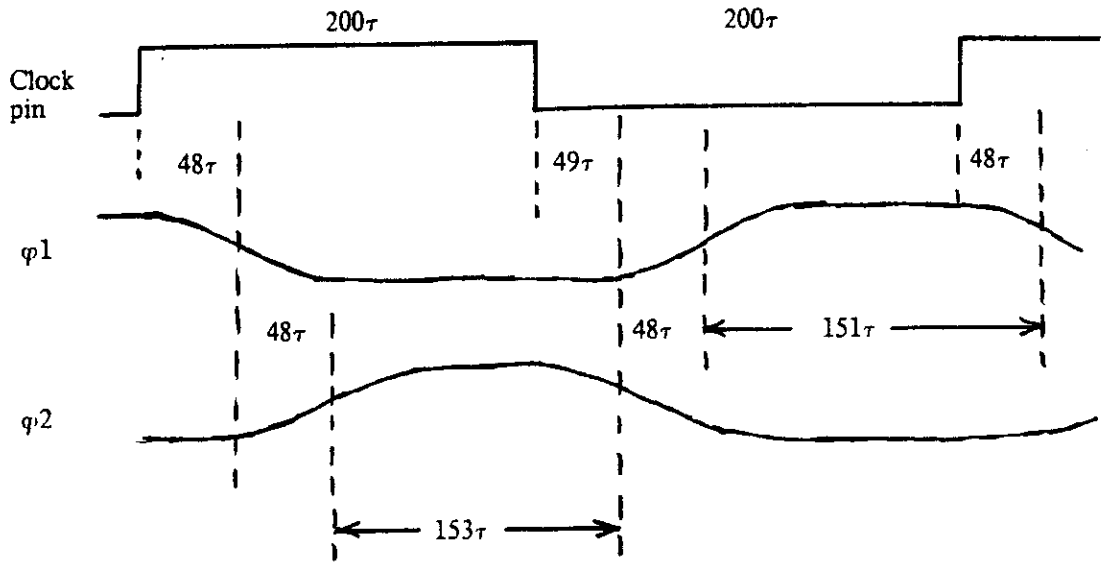


Figure 7b: Clock Waveforms

as shown to assure non-overlap independent of clock loading for gated clocks as well. The techniques used in these clock driver circuits to assure the existence of the non-overlap period independent of the effects of loading on element timing are much in the spirit of those techniques used in speed-independent and self-timed systems, in that the circuit adapts its temporal behavior to conform to a sequencing constraint.

A question is sometimes raised as to whether clocks for different sections of a chip should be buffered separately, as is often done at a circuit board level for systems built from collections of circuit boards. On a physical basis, the answer is no. It is best for two reasons that the clock lines be common throughout the chip. First, this approach minimizes clock skew. Second, since one must pay the same transistor area whether the driver is lumped or distributed, optimum driver design locates the *stray* capacitance of the clock distribution wires where the largest signal energy is available, which is *after* the final driver. There may be organizational reasons for distributing the final driver stage to locations close to sections being driven when some logical operations are performed on the clock signal.

Clock Generation

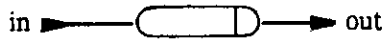
Where great clock accuracy is required, or where a system clock is distributed to many circuits, the clock generation task is external to the chip and is not the direct concern of its designers. The clock will ordinarily originate from an electronic oscillator circuit, the period of which is controlled by a crystal or some resonant network. Process variation in integrated circuit fabrication does not allow accurate resonant networks to be fabricated by usual means, but it is perfectly feasible, indeed essential for self-contained VLSI systems, to generate clock signals on the chip. It is best in approaching this subject to forget about electronic oscillator circuits, and instead to take a more basic approach originating with an understanding of what clocks are for.

As we have mentioned before -- and this is a principle that bears repeating on every opportunity --, the role of the clock in a synchronous system is to connect sequence and time. The interval between clock transitions, whether these transitions are on one or distributed over several wires, must be such as to permit enough time for the activities planned for that interval. When viewed in this way, a clock is more like a set of timers than like an oscillator. A *model* of the temporal behavior of the systems being clocked is built into the clock generator in the choice of times for the various timers.

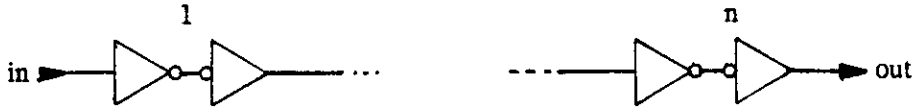
The easiest way to build these timers is as chains of inverters. The propagation delay time of such a chain will of course vary with τ, according to the way in which the fabrication process, aging, temperature, and power voltage affect τ. However, these variations only make the inverter chain a better model of the system being clocked than a fixed timer would be, since on the same piece of silicon these variable factors are nearly the same for the clock and for the system.

It is helpful to distinguish between the two kinds of timers shown in figure 8. The first is a *symmetrical delay*, so-called because the propagation delay for positive and negative transitions at the input is about the same. The second is a logic network designed to produce as asymmetrical a delay as possible. A negative transition propagates through the delay in about 5τ independent of length. A complementary form of *asymmetrical delay* is also possible, but to simplify the figures and symbology in what follows, we shall use only the form in which a low input resets the delay and a positive transition propagates slowly. The symbols shown in the figure allow for *taps* at various points along the delay.

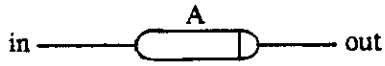
Clocks which employ these delays as timers are all elaborations of the *ring oscillator* circuit shown



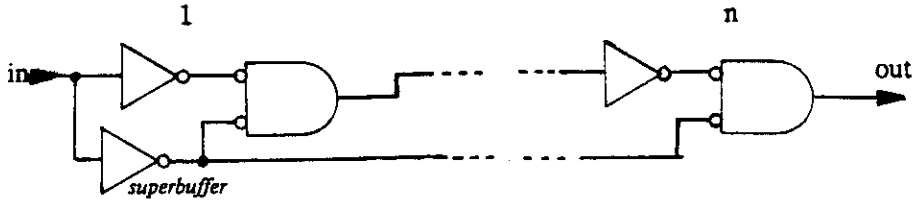
(symbol)



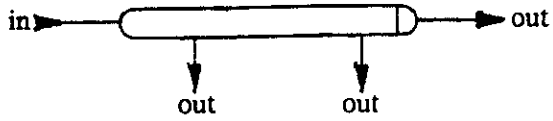
symmetrical delay, delay $\approx 5n\tau$



(symbol)



asymmetrical delay,
positive transition delay $\approx 5n\tau$
negative transition delay $\approx 5\tau$



symbol for a delay with taps

Figure 8: Delays

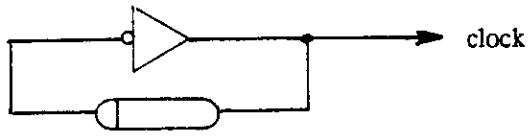


Figure 9a: Ring Oscillator

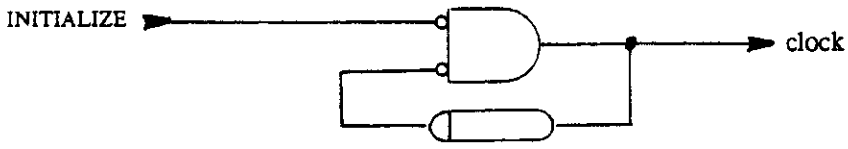


Figure 9b: Symmetrical Clock

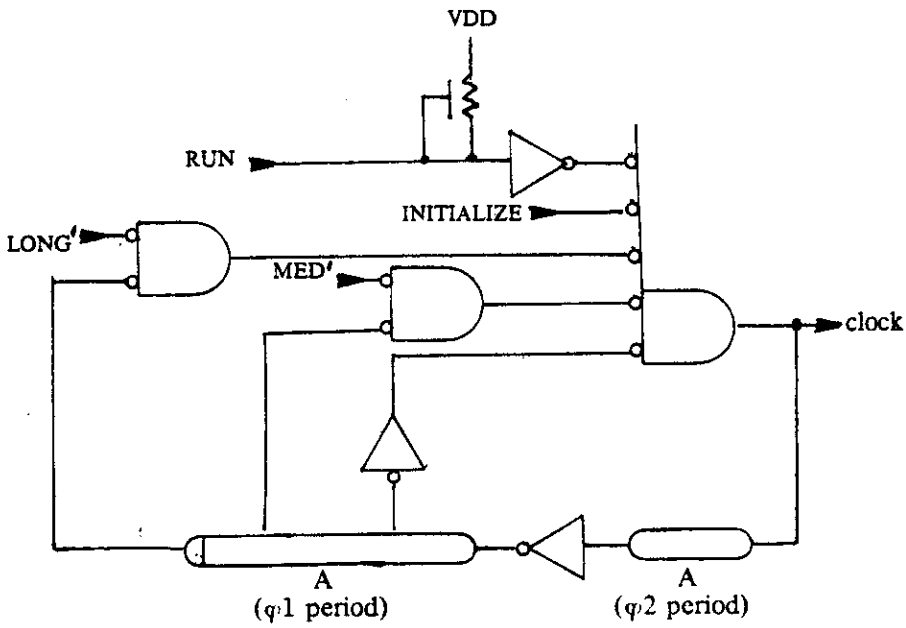


Figure 9c: Universal Clock

in figure 9a. Rings of an odd number of inversions have no stable condition, and will oscillate with a period which is some multiple of the delay time twice around the ring. The oscillation of the largest period will eventually predominate following bringing power on, but the erratic clock signals produced during power-up could leave the system in a peculiar state. It is much better to produce an initialization signal which is held high during power-up, and to use it to initialize the state *and the clock.* In the modification of the ring oscillator shown in figure 9b, clock signals are suppressed during initialization, and will start immediately following the negative transition of the INITIALIZE signal. This circuit produces a symmetrical single-phase clock which can be converted to a two-phase clock by the circuit shown in figure 7a.

Although the clock circuit shown in figure 9b would be adequate for many applications, many elaborations can be included which are shown together in figure 9c. The width of the single-phase clock pulse, related to the $\varphi 2$ period, is determined by one *asymmetrical* delay, while the interval between clock pulses, related to the $\varphi 1$ period, is determined by an independent *asymmetrical* delay. Another feature of this universal clock is that it allows the system being clocked to select between a variety of periods, which can be changed on a cycle-by-cycle basis according to the combinational delay of the operation performed on that cycle. In order to visualize how this works, note that following the trailing edge (negative transition) of the clock signal, any high input to the 5-input NOR circuit has the effect of preventing the occurrence of the next clock pulse. The usual default case is with the LONG', MED', and RUN high, and INITIALIZE low, resulting in a short period determined by the first tap on the period delay. If a decoding of the state indicates that a longer period is required for that cycle, the MED' or LONG' lines must be driven low before the short default period is elapsed. If for example the LONG' line were low, the period before the next clock pulse is stretched to that determined by the full delay. Of course, this scheme may be generalized to any number of delay taps. Signals such as MED' or LONG' can be derived from function coding of the combinational sections whose modeled delay they match, or as microcode bits.

The RUN line is a bus intended to generalize this cycle stretching feature so that any part of the system being clocked may stop the clock synchronously, and then permit it to restart asynchronously. If this cycle is ever to be stretched to more than the refresh time, static storage elements must be used (at least for the $\varphi 1$ part of the cycle for two-phase clocking). This technique of control over the clock is the basic mechanism exploited later in this chapter to allow asynchronous communication between synchronous systems.

Synchronization Failure

Jean Buridan (1295 - ? after 1366), a 14th century French philosopher often cited as a precursor of Issac Newton for his priority in giving a technical definition of kinematic terms such as inertia and force, posed in his commentary on Aristotle's *De caelo* a paradox; that a dog could starve if placed midway between two equal amounts of food. The unfortunate creature placed in this position would be equally attracted to each source of food, and in a position of equilibrium. One 20th century explanation of this paradox, if indeed it is a paradox at all, is that the structure consisting of the dog and the two sources of food is and behaves just as any other structure which can store a bit of information.

The analysis of the electrical behavior of cross-coupled circuits presented in section 1-[Properties of Cross-Coupled Circuits], and developed in physical terms in section 9-[Energetics of the flip-flop], applies also to the situation which Buridan described. The equilibrium condition either for the dog or for the cross-coupled circuit is unstable, as any displacement from equilibrium brings about forces which tend to destroy rather than restore the equilibrium condition. An unstable equilibrium of this sort is called a *metastable condition.* Buridan was correct in believing that the dog could starve, as it is characteristic of a metastable condition that it *may* persist indefinitely. A functional definition of metastability applied to cross-coupled circuits is the occurrence under undriven conditions of an output voltage in a *range* around V_{inv} which cannot reliably be interpreted as either high or low.

A bistable element in a self-contained synchronous system never has the opportunity to reach a metastable condition, since satisfaction of the timing constraints assures that the output is driven to a voltage outside of the metastable range. But is any system really self-contained? A system such as a microprocessor may be entirely synchronous internally, but cannot extend this synchrony indefinitely to encompass all of the external world with which it may interact. If asynchronous signals of external origin are allowed to enter a synchronous system as ordinary inputs, the timing constraints required to assure correct operation cannot be satisfied, since there is no known relationship between the timing of the asynchronous inputs and the clock.

Figure 10 illustrates in a small fragment of a larger synchronous system the consequence of ignoring synchronization altogether. Even if one employs a model of the clocked storage elements as having perfectly discrete outputs, the unequal delay in the paths from the asynchronous input X to the clocked storage elements allows the inputs to the clocked storage

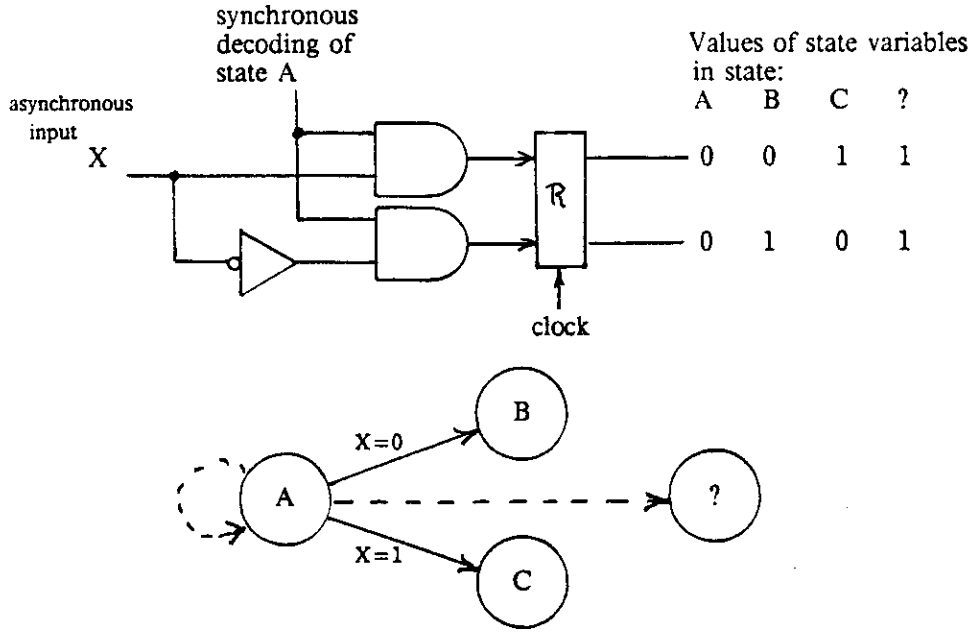


Figure 10: Fragment of a logic circuit illustrating the consequence of ignoring synchronization

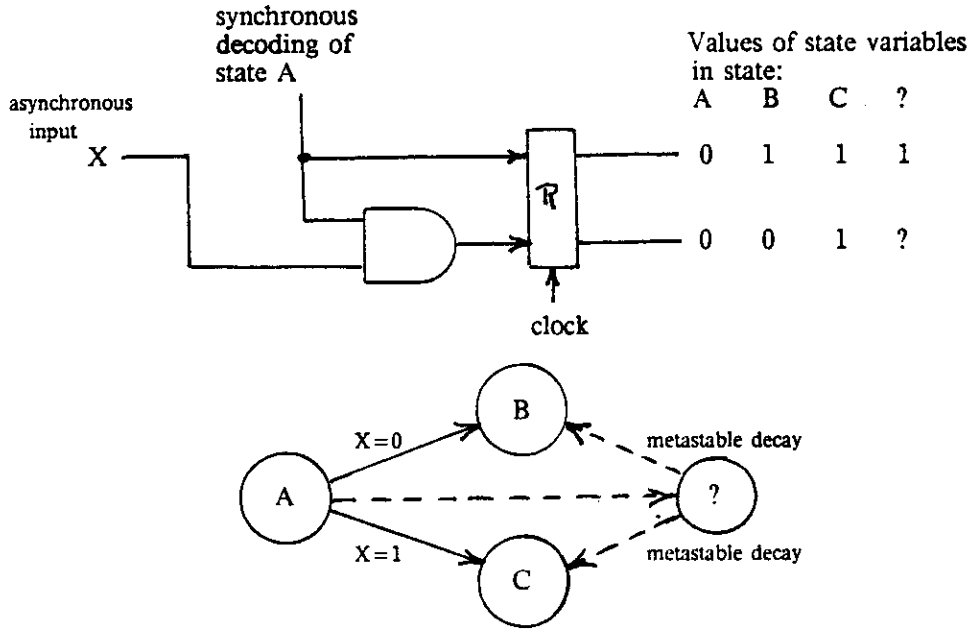


Figure 11: Fragment of a logic circuit illustrating a synchronizer which would work perfectly if perfectly discrete bistable devices were possible

elements during state A to represent an illegal successor state for a period following a transition of the asynchronous input. If the clock happens to capture the inputs during this transitory period, one of the illegal state transitions shown in dashed lines on the state diagram will result.

So, a slightly smarter thing to do is to assure that only one clocked storage element is affected by a given asynchronous input. A clocked storage element that is used in this way is called a *synchronizer,* since it is intended to produce an output signal which is in synchrony with the clock. Figure 11 shows a redesigned version of the previous fragment of a circuit modified in its state coding so that the asynchronous input X affects the input to only one clocked storage element. If it were only possible to build perfectly discrete bistable devices -- indeed, if perfect discreteness exists in nature, for which, see chapter 9 --, this scheme would be perfectly reliable. Unfortunately, there is some probability of *synchronizer failure,* since a transition of the asynchronous input in certain times relative to the clock will leave the synchronizer in a metastable condition, and the time required for the clocked storage element to get out of a metastable condition is unbounded. During the period in which the synchronizer output remains in a metastable condition, the logic cannot discriminate between states B and C, and if the condition persists for too long, an illegal or incorrect successor state can result.

It is part of the interesting history of the subject of synchronization that even long after the necessity to synchronize asynchronous inputs was recognized as a standard part of good engineering practice, the faith of logic designers in the discreteness of the outputs of clocked storage elements was so great that the very existence of synchronization failure was widely denied. Another curious aspect of the sociology of the problem is the many schemes proposed to "solve" the problem, but which only move it to another location in a system or reduce its probability. Synchronization failure was discovered independently by numerous researchers, designers, and engineers in the 1960's, some of whom published reports of their analyses or observations[1,2,3,4,5,6]. The work done at the Computer Systems Laboratory of Washington University by Thomas J. Chaney and Charles E. Molnar[7], and by Marc Hurtado[8] has provided convincing demonstrations of the existence and fundamental nature of the problem.

It is fairly easy to estimate the probability of a synchronization failure with a simple mathematical model. If one observes that a bistable device is in a metastable condition at some time t, what is the probability that it will have left this condition by time $t+\delta$, in the limit as δ approaches zero? The only answer to this question that seems reasonable is that the probability is proportional to δ; let it be $\rho\delta$. This *assumption* produces a simple model in which the exit from a metatstable

condition is a Poisson process of rate ρ, and the probability that a clocked storage element will remain in the metastable condition, once in it, for a period D or longer is $e^{-\rho D}$. The prediction of this simple model has been verified experimentally and is consistent with analyses based on circuit models[8], including the analysis presented in section 1-[Properties of Cross-Coupled Circuits]. The parameter ρ depends on circuit characteristics. A dynamic storage element is not an acceptable synchronizer, as the time evolution of its output in undriven conditions is possibly even *toward* the metastable region (See section 9-[Energetics of the Flip-flop]). One can identify ρ in the analysis in section 1-[Properties of Cross-Coupled Circuits] with $1/\tau_0$. The time evolution of the voltage output of the cross-coupled circuit has the effect of transforming a uniform probability distribution of initial conditions to an exponential or Poisson distribution of exit events. For ratio logic with a ratio of r, τ_0 is about the pair delay, $(r+1)\tau$.

In order to estimate the probability of a fault in a synchronous system due to the non-vanishing probability that a synchronization will take longer than some bounded time, one must also calculate the probability that a synchronization event will put the synchronizer into a metastable condition. For most synchronizations, the asynchronous level to be synchronized will transit sufficiently far away from the time at which it is sampled that the clocked storage element will be overdriven in the usual way. Only over a rather narrow time aperture, denoted here as Δ, does the occurrence of a transition result in the synchronizer taking more than the usual delay time of the clocked storage element. The boundaries of this aperture are not sharp, but may be treated as such, so that for a particular frequency of transitions of the asynchronous signal, f, the *probability* that a metastable condition will be produced in a single synchronization event is $f\Delta$. One may take this relation as the definition of Δ.

The overall probability of a system failure at each synchronization event, $f\Delta e^{-\rho D}$, depends on ρ and Δ, which are parameters of the clocked storage element used as a synchronizer; on D, which is a parameter of the synchronous system in which the synchronizer is used; and on f, which is a parameter of the asynchronous input signal. D is the time allowed in the synchronous system for the *decay* of the probability of metastability, and is effectively like a delay. It corresponds to the *excess* delay allowed from clocked storage element outputs to inputs (see figure 5), and even for zero combinational delay cannot exceed the clock period less delay and preset times.

In order to get some feeling for the failure rates involved, consider a synchronous processor which is accepting data from, or sending data to, a disc storage unit at a 1MHz rate. An asynchronous signal alerts the processor to the presence of, or need for, a new data item, but the processor is

able to clear this signal synchronously. We can assume that ρ is about $1/5\tau$ for ratio logic with r=4. If almost all of a fairly short 100τ clock period were available for the decay of the probability of metastability, D would be about 80τ. It is interesting that when D is expressed as a multiple of τ, the exponent in the formula is then independent of τ, and so is independent of scaling circuit dimensions. The scaled down version of this system would allow less time for the decay of the probability of metastability, but the synchronizer would exhibit a proportionately higher metastable exit rate.

We are not aware of any experimental determinations of Δ for nMOS circuits, but by analogy with experiments performed with several bipolar circuit families, we believe that Δ is a small fraction of τ, say about $\tau/10$. This estimate agrees with the notion that Δ is time aperture corresponding to that time required for a signal to transit through a small voltage range around the switching threshold, a time which is proportional to τ. For present values of τ, Δ is then approximately 30 picoseconds, and the probability of a system failure for a single synchronization event would be about $(10^6)(30\cdot10^{-12})e^{-16}$ or about $3\cdot10^{-12}$. So, about one in each $3\cdot10^{11}$ items transferred across this interface would be in some fashion mistreated.

Failure *rate* depends on the frequency at which the system samples asynchronous inputs. This frequency cannot be greater than the clock frequency, and as is clear from a careful study of figure 11, this frequency may be as low as the frequency of the states whose choice of successor depends on the asynchronous input. This figure is intended to suggest how synchronizers can be *sheltered* from needless synchronization events, as the synchronizer is here sheltered by ANDing the asynchronous input with a signal which indicates that the system is in state A. Synchronizers that are used directly on asynchronous input signals and whose outputs enter PLA or ROM structures may cause failures even when the system is in a state in which the successor does not depend on the asynchronous input.

For the example above, the synchronous processor must sample every transition of the asynchronous input in order to transfer every data item. If this processor were engaged in transferring data at a 1 MHz. rate about one third of the time, synchronization failures would occur at a (Poisson) rate of once each 10^6 seconds, or about every 10 days. It is worth noting that the exponential relation in ρD makes the failure rate remarkably sensitive to ρD. This dependence may be particularly noticable if the clock signal originates off-chip so that ρD depends on τ. A chip with a slightly larger than typical τ may exhibit a drastically higher than typical failure rate.

The probabilistic character of synchronization failures makes them exceedingly difficult to trace. Designers of synchronous systems who wish to avoid the curses and plagues that are the just reward for those that build secret flaws into human tools and enterprises should cultivate a rational conservatism toward this problem. The worst-case failure rate for a design should be calculated. If the failure rate is higher than some criterion, it can be reduced by techniques which increase D. Use of cascaded synchronizers is one technique for increasing D which does not require increasing the system clock period. Criteria for acceptable failure rates depend on many of the same economic and social factors that influence other aspects of system reliability.

One conservative failure rate criterion that can be supported by a physical argument is that the rate of synchronization failures should be on the order of the rate at which the bistable synchronizer will change state due to the random thermal motions of the electrons. This rate is shown in section 9-[Thermal Limit] to be $(1/\tau)e^{-(E_{sw}/kT)}$. If s is the frequency of synchronization events -- often s = f --, this physical criterion is:

$$sf\Delta e^{-\rho D} = sf\Delta e^{-D/(r+1)\tau} < (1/\tau)e^{-(E_{sw}/kT)}.$$

Solving for D yields: $D > [E_{sw}/kT + \ln(sf\Delta\tau)](r+1)\tau.$

If one takes Δ as about $\tau/10$ and s and f in the order of $1/100\tau$, it is clear that the second term in the brackets can be ignored. The switching energy for today's circuits is about 10^8kT, and bistable devices are consequently so reliable that the time required to allow the probability of metastability to decay to achieve the same reliability is very large -- about $5{\cdot}10^8\tau$, or 0.15 seconds! However, this criterion scales in a remarkable way. The ratio D/τ, which represents the number of transit times the criterion provides for the metastable exit, scales with the switching energy, which goes down as α^3. This scaling should not be interpreted as meaning that smaller devices have a higher probability of metastable exit per transit time. Rather, smaller transistors result in less reliable storage devices, which makes it possible to lower one's standards. Ultimately small transisitors with channel lengths of about 0.25μ would allow circuits with a switching energy of about 10^4kT. Because of the significant subthreshold currents at the low threshold voltages implied by this scaling, CMOS circuits with r = 1 would have to be used. At these minimum dimensions, this criterion implies $D > 2{\cdot}10^4\tau$, and taking τ = 0.02 nanoseconds, D > 400 nanoseconds. So, at this ultimate limit of MOS technology, one cannot disregard synchronization failure, but one would not expect it to limit designs for synchronization rates up to 1 MHz. or so. Since this criterion represents the most conservative position that can still be rationally defended on physical grounds, it is known as the *Mead Criterion.*

Self-timed Systems

The operation of a synchronous system is reminiscent of soldiers marching to the commands of a drill sergeant. The temporal control over a collective activity is centralized in a single authority, and the soldiers respond to known commands that are synchronous with the marching cadence. Lockstep control results in a particularly simple form of organized behavior which people often associate with the relentless efficiency of machines. However, lockstep control is certainly not the only way to coordinate the collective activity of many participants, nor is it efficient unless the tasks of the participants are very well matched. *Self-timed systems* are patterned on quite a different image of organized activity, one in which the temporal control is delegated to the participants. If one were to try to construct a mental image of self-timed behavior, it would be one in which the airplane could not leave until after all the passengers scheduled for the flight had gotten on board. One tries to assure that everything occurs in proper *sequence*, but nothing ever has to occur at a particular *time*.

In a self-timed system, the metrical notion of time is confined strictly to the interior of parts called *elements*. Time and sequence are related *inside* these elements in such a way that the terminal behavior of an element obeys a set of sequence domain *relations* on the occurrence of signal transitions. The only kind of relations possible are $x \leq y$, x *preceeds* y, for which one might also say either that x is *before* y or that y is *after* x, and $x \# y$, x *is concurrent with* (not ordered with) y. The relation $\leq$ is a partial ordering on the set of occurrences of signal transitions; it is reflexive, $x \leq x$; antisymmetric, $x \leq y \cap y \leq x \Rightarrow x = y$; and transitive, $x \leq y \cap y \leq z \Rightarrow x \leq z$. The relation $x = y$ in the definition above means that x and y are *identical*, not simultaneous. The notion of *simultaneity* is specifically regarded as meaningless and disallowed by the antisymmetric property of $\leq$.

According to the special theory of relativity, it would not be possible to have any relations between occurrences of signal transitions at different points in space, as the order might be interpreted differently for observers in different locations. The problem with the relativistic environment is that one knows so little about time that systems are either impossible to make, or more complicated than is justified by the actual situation. Over volumes that are sufficiently small, volumes that are here called *equipotential regions*, one is justified in applying the approximations that (1) a signal is *identical* at all points on a wire, and (2) a relation which holds

anywhere in the region holds everywhere in the region. Of course, an element must be contained entirely within an equipotential region or else the set of relations which define its terminal behavior would be meaningless. A system may contain many equipotential regions. Each of them is conformal to a point in a relativistic space, and each keeps time to itself.

In a medium such as free space, events that originate at one point in a certain order are observed to occur in the same order at any other point. However, in a medium in which signals are carried on wires which may follow different routes from one equipotential region to another, order is not in general preserved. Some self-timed systems may use a multi-wire structure called a *bundle*, which can convey a set of signals from one equipotential region or element to another while preserving specified ordering relations. For example, a two-wire bundle with inputs (a,b) and outputs (c,d) may satisfy the constraint that $a \leq b \Rightarrow c \leq d$, but would not also be required to satisfy $b \leq a \Rightarrow d \leq c$, as the pair of requirements would imply that the bundle satisfy a two-sided timing bound.

The time required for an electromagnetic wave to traverse a large chip is less than 0.1 nsec, and for 1978 MOS circuits the signal transition times on-chip are greater than about 0.3 nsec. Accordingly, the equipotential environment is very well approximated on a single chip. For typical transition times on pins of many nanoseconds, a circuit board up to about a foot square is a good approximation of an equipotential region. However, communication delays in pin driving structures are so long compared with internal transition times that two chips may not be regarded as being in the same equipotential region.

<remainder of this section in preparation>
discussion of the formation of systems from elements based on the recursive definition: A self-timed system is either (1) a self-timed element, or (2) a legal interconnection of self-timed systems. Correctness proofs. Fundamental motivation for self-timed systems: that correctly functioning elements assure a correctly functioning system.

Signaling Conventions

<in preparation>

two-cycle (transition) and four-cycle (Muller) signaling -- data validity -- the MOSbus convention -- ternary and data-driven signaling.

Synchronous Elements

<in preparation>

illustration of the use of the stoppable clock to implement self-timed elements as synchronous systems.

Asynchronous Elements

<in preparation>

self-timed elements implemented without clocks -- delay modeled timing -- speed-independent timing -- inherent error detection and correction.

Arbitration

<in preparation>

asynchronous arbiters -- application to shared self-timed systems.

Acknowledgements

<in preparation>

A. W. Holt, W. A. Clark, C. E. Molnar, Carver, Lynn, Ivan, Sproull, Dick, Wayne.

References

1. Gray, H. J., *Digital Computer Engineering*, Prentice-Hall, Englewood Cliffs, New Jersey, 1963 (pp. 198-201).

2. Catt, Ivor, "Time Loss Through Gating of Asynchronous Logic Signal Pulses," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 1, February 1966, pp. 108-111.

3. Littlefield, W. M., and T. J. Chaney, "The Glitch Phenomenon," Technical Memorandum #10, Washington University Computer Systems Laboratory, St. Louis, Missouri, December 1966.

4. Couranz, G. R., "An Analysis of Binary Circuits Under Marginal Triggering Conditions," Technical Report # 15, Washington University Computer Systems Laboratory, St. Louis, Missouri, November 1969.

5. Seitz, C. L., "Graph Representations of Logical Machines," MIT Ph.D. thesis, January 1971, preface.

6. Chaney, T. J., S. M. Ornstein, and W. M. Littlefield, "Beware the Synchronizer," COMPCON-72 IEEE Computer Society Conference, San Francisco, California, September 1972.

7. Chaney, T. J. and C. E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits," *IEEE Transactions on Computers*, Vol. C-22, No. 4, April 1973, pp. 421-422.

8. Hurtado, M., "Dynamic Structure and Performance of Asymptotically Bistable Systems," Washington University D.Sc. Dissertation, 1975.

9. Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill, 1970, Chapter 9.