A formal model for MOS clocking disciplines

Kevin Karplus

Abstract

This paper presents a formalization of clocking disciplines used to prevent race conditions in VLSI circuits. A signal-labeling scheme for the two-phase clocking discipline informally described in Mead and Conway [MC] is presented. The correct labeling of a circuit consisting of combinatorial logic and memory elements can be checked mechanically. The signal-labeling conventions are based in part on those of Noice, Mathews, and Newkirk [NMN].

A formal basis is presented for constructing signal-labeling schemes for multi-phase clocks (both overlapping and non-overlapping) from a definiton of the master timing signals. Both two-phase non-overlapping and four-phase overlapping clocks are used to illustrate the formalism.

Introduction

Timing errors are a major source of design flaws in integrated circuits. *Race conditions* (situations where the operation of a circuit depends on the relative speeds of multiple paths through the circuit) are difficult to diagnose and eliminate. Timing simulation is not an adequate tool for detecting race conditions, as there are far too many paths through the circuitry on VLSI chips to simulate even a small fraction of them. Furthermore, the delays in MOS circuitry vary enormously with small changes in the fabrication process, so simulations would have to be done repeatedly with different parameters.

Clocking disciplines attempt to eliminate race conditions by making the times at which events occur discrete. A master clock provides a periodic signal defining the discrete time events. If the order in which two signals arrive at some subcircuit is important, then they must be associated with different events in the timing system. As long as the master clock is slow enough, no race conditions can occur. Timing analysis can be used to determine how fast the master clock can run, and what paths in the circuit are responsible for limiting the speed. TV [J] and Crystal [01, 02] are examples of timing analysis programs based on two-phase clocking.

This paper discusses synchronous systems, in which time can be viewed as periodic. The formalism is not intended to cover asynchronous or self-timed systems.

Draft of Technical Report 84-632, Department of Computer Science, Cornell University, August 1984.

Discrete, periodic time

Although physicists and circuit designers usually view time as continuous, it is convenient when designing digital systems to view time as a sequence of distinct events.

Different levels of design call for different types of events. A scheduler for an operating system may view execution of a complete program as a single event; a state-machine controller takes each input as a separate event; a flip-flop may take each clock edge as a separate event. This paper examines time at the finer grain of the flip-flop or state machine, not the coarser grain of an operating system or communications protocol.

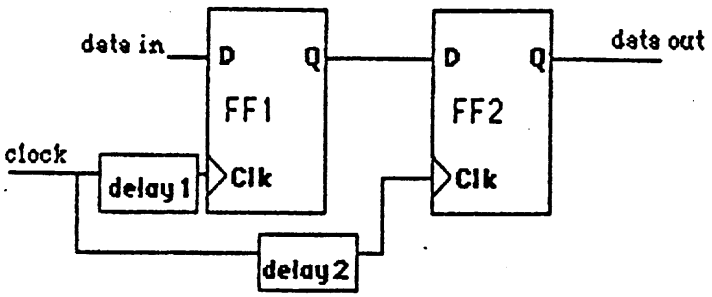
To describe an infinite sequence of events in finite space, it is necessary to impose some structure on the events. The simplest such structure is a periodic one, in which a short sequence of events is repeated over and over.

Single-phase clocking

A square-wave or pulse train is easily described as a periodic repetition of the two events (**high** and **low**). Note that the events correspond to intervals of time, not single points. This simple description of a square wave does not cover all of time. In any real system, some time is required to go from the high state to the low state and vice versa. A better description of a square wave is as a periodic repetition of four events (**rise**, **high**, **fall**, **low**). During the **high** and **low** events, the signal does not change.

Single-phase clock schemes generally use the rise and fall events to provide timing information for circuits. Many circuits use only one of the two events for timing. This edge-triggering is very popular with TTL designers, but is rarely seen in nMOS or cMOS design.

There are two problems with edge-triggering. First, the rise and fall events must be glitch-free to avoid false triggering. If the signal rings too much, an edge-triggered circuit may see multiple rise and fall events where only one was intended. Second, the widely varying delays of MOS circuits make rise and fall events arrive at different parts of the circuit at unpredictable times. With edge-triggering, it is hard to avoid situations in which the order of arrival of signals over different paths affects the operation of the circuit.



The figure shows a simple race condition. Two D flip-flops are connected in series to act as a shift register. When the clock line rises, data is supposed to be transfered from FF1 to FF2, and from the input to FF1. If the clock pulse arrives at FF2 sufficiently after it arrives at FF1, the input will be transfered to both FF1 and FF2. The difference in arrival time of supposedly simultaneous events is called *clock skew*.

Two-phase clocking

The standard solution to the problems of edge-triggering in MOS is to use only level-sensitive, not edge-sensitive, circuitry. To a level-sensitive circuit, the periodic repetition of a single event appears the same as the continuation of the event. At least two distinct event types are needed to distinguish one event from the next.

With a single clock line and standard 2-level logic, the possible events to use are **high** and **low**. A simple pulse train alternates between the two events. Unless the rise and fall times are large, a small clock skew can present **high** to one part of the circuit, while presenting **low** to another. Large rise and fall times are undesirable, because the intermediate signal levels can propagate through a circuit resulting in non-digital behavior of supposedly digital circuits.

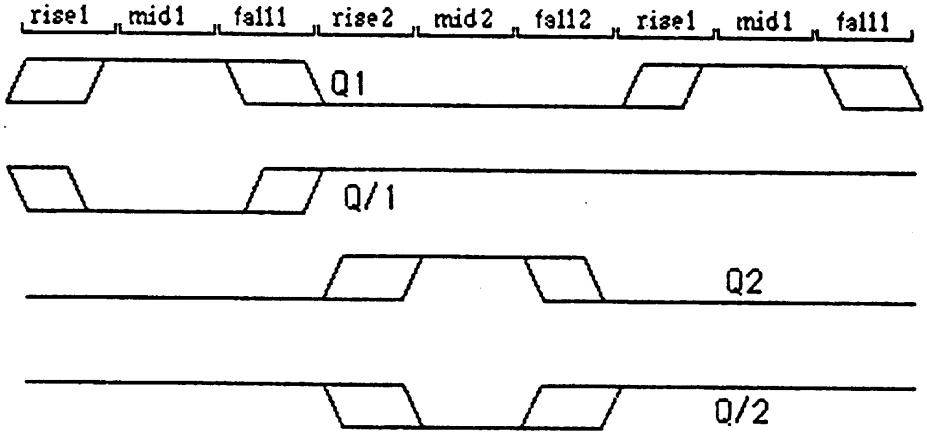
To separate events, we may either depart from two level-logic, or have at least two signals for timing. In some special cases (such as RAM design), multiple logic levels are appropriate, despite the more complex circuit design. In general, multiple signal lines are easier to design with.

Mead and Conway's two-phase non-overlapping clock scheme uses two lines which are high alternately. The periodic sequence is (rise1, mid1, fall1, rise2, mid2, fall2). The mid1 and mid2 events are used for timing. All phase 1 timing signals rise sometime during rise1 and fall sometime during fall1. By making the rise and fall events long enough, clock skew can be hidden in them. The mid events need only be long enough for actions that are to occur during them.

Derived clock signals

A complex circuit uses many different signals for timing functions. Because these control signals are usually derived from a master clock, they are often called derived clock signals or qualified clock signals. They are divided into four disjoint classes (Q1, Q/1 (pronounced Q-bar-1), Q2 and Q/2). Q1 and Q2 are active high, Q/1 and Q/2 are active low. Q1 and Q/1 are active only during phase 1, Q2 and Q/2 are active only during phase 2. (Phase 1 refers to the three events rise1, mid1, and fall1, and phase 2 refers to events rise2, mid2, and fall2.)

A signal is Q1 if it is always low during phase 2, does not change value during mid1, and if low during mid1 is low throughout phase 1. Notice that a Q1 signal must be low during phase 2, but may be either high or low during phase 1, and is carrying stable information only during the mid1 event. A signal is Q/1 if it is high throughout phase 2, doesn't change in mid1, and if high in mid1 is high throughout phase1. (Q2 and Q/2 are similarly defined). Glitches can occur at the rising and falling edges of a Q-labeled signal, but no glitches are allowed when the signal is supposed to be inactive. The timing diagram below gives examples of each of the signal types.



If the timing signals of a chip are labeled by class, the intervals for the six events can be found by simulation. Rise1 begins when the first Q1 signal rises (or Q/1 signal falls), mid1 begins when all the Q1 and Q/1 signals stabilize, and fall1 begins when the first Q1 signal falls or Q/1 rises. (Similarly for phase 2 signals.) The main speed limitation is that mid1 be long enough for all relevant data signals to propagate.

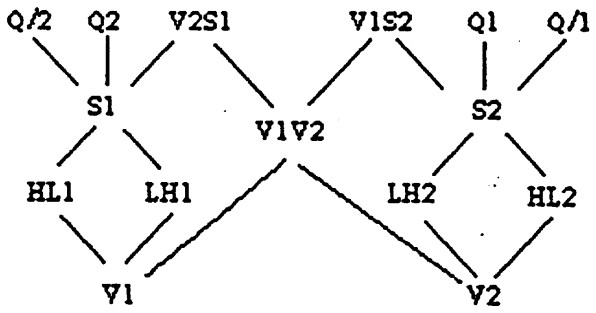
Two-phase labeling for data signals

Not all signals in a system are timing signals. The majority are used for storing or transmitting data. For these signals a different labeling scheme is needed, to identify the times when the signals have valid data. The following table gives the signal labels for data signals in a two-phase system:

label	read	means
V1	valid-1	signal doesn't change during fall1
S1	stable-1	signal doesn't change during rise1, mid1, and fall1
V1S2	v-1-s-2	signal doesn't change during fall1, rise2, mid2, and fall2
LH1	low-high-1	signal is constant low during rise1, mid1, and fall1, or is constant high during fall1
HL1	high-low-1	signal is constant high during rise1, mid1, and fall1, or is constant low during fall1
V1V2	v-1-v-2	signal is both V1 and V2.

(Labels V2, S2, V2S1, LH2, and HL2 are defined similarly.)

Some labels are *stronger* than other labels. For example, any signal that is S1 doesn't change during fall1, so is also V1, and can be used where a V1 signal is needed. The labels can be arranged in a lattice as follows:



When the input to a circuit requires a signal labeled *x*, a signal with any label above *x* along the lines of the diagram may be used. However, note that Q2 and Q/2 carry no information in phase 1, and are S1 only because they are guaranteed inactive throughout phase 1. Using a Q1 signal where an S2 or V2 signal would normally be used is probably an error, since no data is carried during phase two.

As a general rule, input signals are labeled with the weakest condition necessary for the circuit to work (usually V1 or V2) and output signals are

labeled with the strongest statement that can be made about the signal (usually S1, S2, V1S2, or V2S1). The LH1 and HL1 labels are useful for precharged circuits, particularly cMOS domino logic.

The results of boolean operations on labeled signals can be derived from the definitions. Most labels are unchanged by negation, since negating a constant signal results in a constant signal. Only the Q, HL, and LH signals say anything about the logic level of a signal.

A	*not A*	*A*	*not A*
Q1	Q/1	Q2	Q/2
V1	V1	V2	V2
S1	S1	S2	S2
V1S2	V1S2	V2S1	V2S1
LH1	HL1	LH2	HL2
HL1	LH1	HL2	LH2
V1V2	V1V2		

A AND B

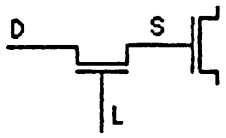
A B-	Q/1	Q1	V1	LH1	HL1	S1	V2S1	V1V2	V1S2	S2	HL2	LH2	V2	Q2	Q/2
Q/1	Q/1														
Q1		Q1				Q1	Q1								
V1			V1	V1	V1	V1	V1	V1	V1						
LH1			V1	LH1	V1	LH1	LH1	V1	V1						
HL1			V1	V1	HL1	HL1	HL1	V1	V1						
S1		Q1	V1	LH1	HL1	S1	S1	V1	V1						
V2S1		Q1	V1	LH1	HL1	S1	V2S1	V1V2	V1V2	V2	V2	V2	V2		
V1V2			V1	V1	V1	V1	V1V2	V1V2	V1V2	V2	V2	V2	V2		
V1S2			V1	V1	V1	V1	V1V2	V1V2	V1S2	S2	HL2	LH2	V2	Q2	
S2							V2	V2	S2	S2	HL2	LH2	V2	Q2	
HL2							V2	V2	HL2	HL2	HL2	V2	V2		
LH2							V2	V2	LH2	LH2	V2	LH2	V2		
V2							V2	V2	V2	V2	V2	V2	V2		
Q2									Q2	Q2				Q2	
Q/2															Q/2

Blanks in the table indicate illegal operations.

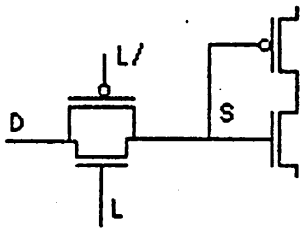
The tables for other boolean operations can be derived from the above tables. For example, OR looks the same as AND, with Q1 and Q/1 swapped, and Q2 and Q/2 swapped.

Memory elements in a two-phase system

Boolean operations only provide ways to move down the lattice of timing labels. A strictly combinatorial circuit cannot change a V1 input to an S1 or S2 output. To do any movement up the lattice, storage elements are needed.

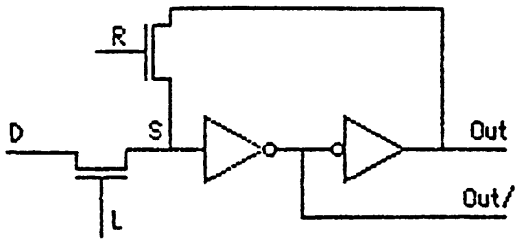


The figure shows a standard dynamic storage element for nMOS. The data input is D, latch control is L, and the storage node is S. A similar circuit can be used for dynamic storage in cMOS:

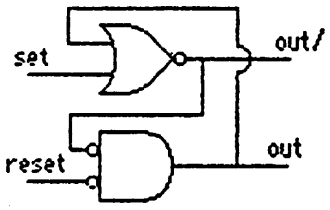


Here is the table for determining the labeling of S from D and L. The same table applies to both the nMOS and cMOS circuits. In fact, any memory element will exhibit this behavior. Note that the latch signal is always a Q-labeled timing signal, and the data must be stable during the event in which the latch signal turns off. The blanks in the following table are the cases in which the data is not stable at the right time.

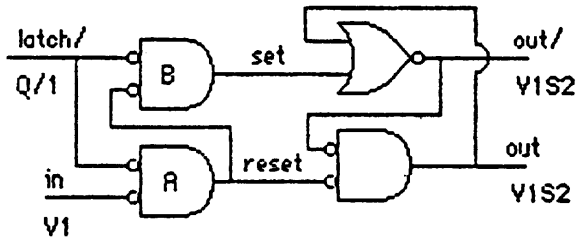
D=	L=	Q1	Q2
Q/1			constant-1
Q1			constant-0
V1		V1S2	
LH1		V1S2	
HL1		V1S2	
S1		V1S2	
V2S1		V1S2	V2S1
V1V2		V1S2	V2S1
V1S2		V1S2	V2S1
S2			V2S1
HL2			V2S1
LH2			V2S1
V2			V2S1
Q2		constant-0	
Q/2		constant-1	



This figure shows a standard nMOS pseduo-static transparent latch. The labeling for S is the same as in the dynamic latch. New data is latched when L is high, and the old data is refreshed when R is high. Both L and R should have the same Q label (both Q1 or both Q2), but the signals are mutually exclusive, that is, L and R must never be high at the same time.



This figure shows a static SET-RESET latch. SET and RESET must be mutually exclusive (otherwise both OUT and OUT/ are low). If SET and RESET are both Q1 signals, then the outputs are V1S2. If SET and RESET have differing Q labels, then the outputs are only V1V2. Using a SET-RESET latch with non-Q-labeled signals will almost certainly cause race conditions.

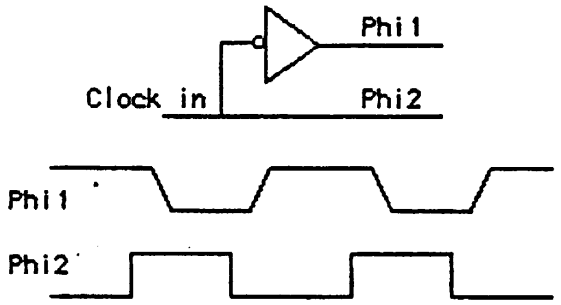


Above is a static D-latch based on the SET-RESET latch of the previous figure. The **latch/** signal is labeled Q/1, so the data input must be V1, and the outputs are V1S2. The labels for **reset** and **set** can't be assigned, leading us to suspect that the circuit has a race condition. Indeed, if the path from **latch/** through A and B to **set** is faster than the path from **latch/** through B to **set**, then the flip-flop could get set instead of reset with an input of 0. Luckily, in almost any design for the circuit, this race is never lost. The behavior of the D latch as a unit can be described in the clocking discipline, even though the internals of the circuit do not follow the discipline.

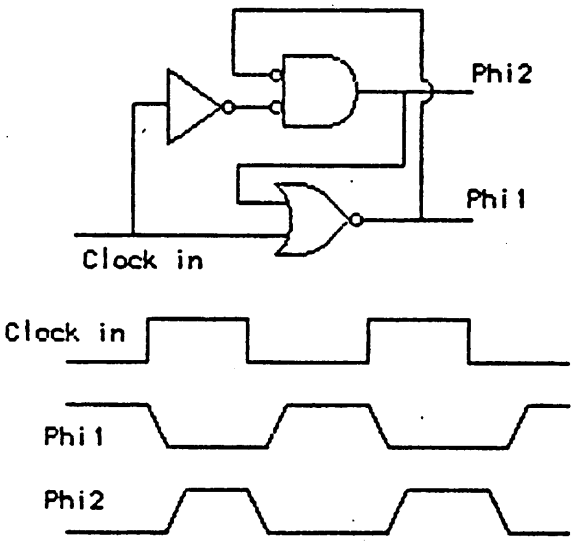
Generating two-phase clocks from single phase

Although most nMOS circuits use two-phase clocking internally, systems designers are reluctant to dedicate two pins to clock signals. Many designs are limited by the number of pins available in a package, and off-chip wiring is fairly expensive. Designers commonly use an external single phase clocking scheme, and convert it internally to a two-phase system.

The simplest scheme for converting single-phase to two-phase clocks uses the input clock as one of the master clocks, and inverts it to form the other. Unfortunately, such a system is almost guaranteed to have some overlap between the master clock signals, so is unusable.



Adding a pair of cross-coupled NOR gates to the inverter yields a non-overlapping pair of clock signals. However, the timing signals derived from the clock signals will be delayed by varying amounts, and may overlap with timing signals from the other phase.

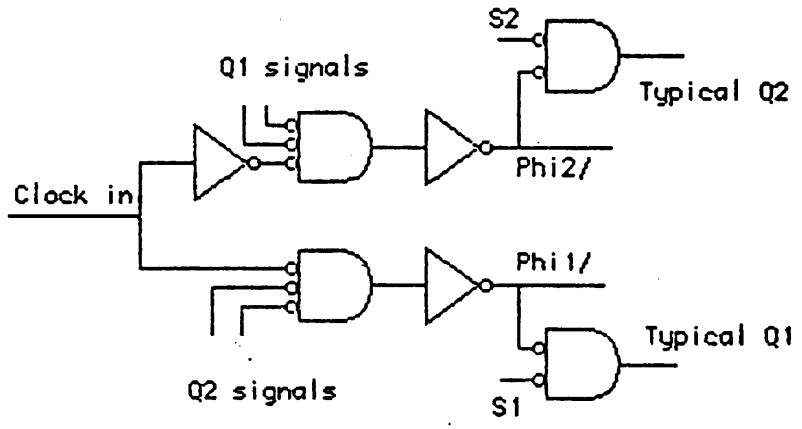


The non-overlap period must be long enough that all timing signals derived from the master clocks properly follow the two-phase discipline. There are several methods for ensuring this. The simplest is to make the

capacitive load on the Phi1 and Phi2 lines large enough that the delay in the clock generator dominates the delay in other parts of the circuit. This method is used in the clock generator pad in [NM]. Unfortunately, it produces timing signals with slowly rising and falling edges. Since the speed of MOS circuits is dependent on the slope of the rising and falling edges, a system with this simple clock generator is slower than the same circuitry with a sharper clock.

Another simple technique for clock generation is to introduce a sufficiently large delay in the feedback paths of the cross-coupled NOR gates. The clock signals can be kept crisp, and still not overlap. The delay should be larger than the delay of the slowest signal for each clock phase. Choosing the appropriate delay time requires timing analysis of the rest of the circuit. Too small a delay can cause failures from clocks overlapping, too large slows the system down unnecessarily.

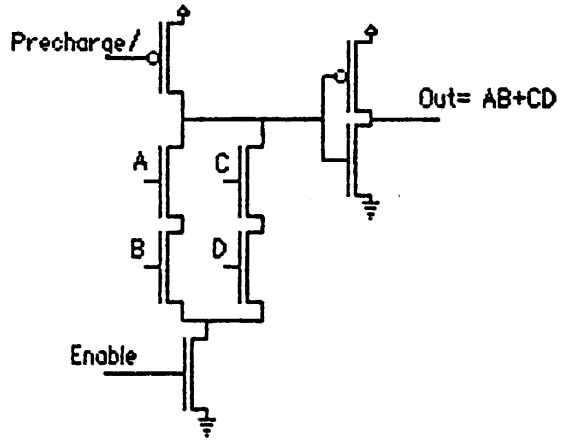
A more complex, but more reliable, method feeds back all the Q-labeled signals. A typical Q1 signal is generated with a NOR gate from the Phi2/ master clock and an S2 control signal (usually from a ROM or PLA state machine). All the Q1 (and negations of the Q/1) signals are fed back to a giant NOR gate in the clock generator. No timing analysis is needed to ensure correct operation, since all Q1 are guaranteed not to overlap Q2 signals. By breaking one metal line, the clock generator pad in [NM] can be used for this generation scheme.



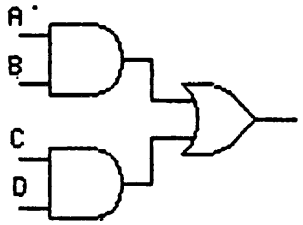
For prototypes and student projects, external two-phase clock generation is usually better than converting single-phase internally. The cost of an extra pin is irrelevant, and external clock generation adds testing flexibility. Errors in timing analysis can be more easily detected and compensated for with external clocks.

Domino Logic

Domino logic is a cMOS logic family that uses precharging to reduce the size and complexity of cMOS circuits [KLL]. Here is a representative circuit



implementing the following function:



The intermediate node is charged during one clock phase and evaluated during the other. For example, if the circuit is to be active during phase 1, *Enable* should be Q1 and *Precharge/* should be Q/2. The inputs and output are all LH1. During clock phase 2, the intermediate point is pre-charged high and is either left high or pulled low during phase 1. This behavior is best described with the HL1 signal label, resulting in the LH1 label for the output of the inverter stage. If the inputs are high at any point during phase 1, the intermediate point could be discharged. Therefore, if the inputs are to be correctly interpreted as low values, they must remain low throughout phase 1. This puts the LH1 restriction on the inputs.

Enable and *Precharge* are usually connected directly to the master clock, so the gate evaluates its output on every clock cycle. The main requirement on them is that they must not be simultaneously active (to avoid shorting power and ground). In actual implementations, both are usually connected to the same clock signal. With only one clock, domino logic is easy to route, and can be interfaced with circuits following almost any timing discipline.

Clocking domino logic with a single clock raises the possibility of a glitch, due to a difference in switching speed of the transistors. Glitches could occur at the beginning or end of phase 1. At the beginning of phase 1, there could be a brief connection between power and ground. Although this would raise power consumption slightly, it would not change any of the signals. At the end of phase 1, the intermediate node could get pulled high when it should remain low. Since the two control signals are normally tied together with a short metal connection, the skewing is slight and the glitch is short or non-existent. With normal circuit design, the capacitance of the intermediate node is large enough that the glitch is filtered out.

The main limitation of Domino logic is that the gates are non-inverting (they can't compute a function that is high when the inputs are low). An inverting gate can easily be created, but would have an HL1 output, which is not acceptable as input to another Domino logic gate.

Domino logic is full compatible with standard CMOS, since any S1 signal may be used as an input (S1 => LH1) and any gate that accepts V1 may be connected to the outputs (LH1=>V1). Of course, if a V1 signal is sufficient for the output, the inverter stage can be omitted, and the precharged signal used directly (HL1=>V1).

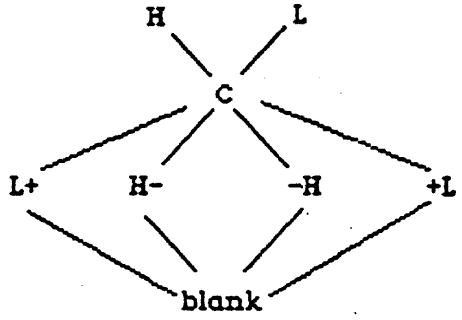
Multi-phase clocking schemes

To discuss a given multi-phase clocking scheme, a set of signal labels is needed like the ones presented for two-phase clocking. Rather than cobbling together a unique set of labels for each new clocking discipline, we can use the following method to generate the labels of any multi-phase clocking scheme.

An n-phase clocking scheme has a periodic time scheme consisting of $3n$ events, a rise_k, mid_k, and fall_k for each phase k. A signal can be labeled as a vector of $3n$ symbols, one for each event. The event label symbols are:

symbol	**behavior of signal during event**
L	signal is low
H	signal is high
C	signal is constant
L+	signal is constant low or ends high
H-	signal is constant high or ends low
+L	signal is constant low or starts high
-H	signal is constant high or starts low
blank	behavior unspecified in this event

We can put a partial order on the event labels :



We say that a label is *stronger* than another if it is above it in the partial order. A vector of labels is *stronger* than another vector, if the labels for all events are the same as or stronger than the corresponding labels in the other vector.

Since the discrete event labels are a model for a continuous process, the signal cannot change instantaneously between events. Thus, vectors in which H and L are adjacent do not correspond to realizable signals. There must be an intervening event in which the signal can change (blank, L+, H-, -H, or +L). For example, H, H-, L is a legal sequence, but H,-H, L is

not, since in the second event the signal doesn't start low, it must be constant high through the second event and change instantaneously to constant low for the third event.

For the same reasons, a C label adjacent to an L [or H] can be replaced by an L [or H]. When two labels have equivalent meaning for an event, the stronger is generally used. For completeness, here is a table of the strongest equivalent labels for pairs of adjacent labels:

first label \ second label	H	L	C	L+	H-	+L	-H	blank
H		*	H,H				H,H	
L	*		L,L			L,L		
C	H,H	L,L						
L+		L,L						
H-	H,H							
+L								
-H								
blank								

Empty entries mean that no equivalent labeling stronger than the initial pair of labels exists. The * indicates an unrealizable signal. Some extra information can be deduced in the blank cases (for example, the pair HL+ will not be constant low for the second event, so must end high), but no more concise labeling has been created for these cases.

The results of boolean operations on event labels are easily stated. ORing anything with H results in H, and ANDing with L results in L. In all other AND and OR operations, the result is the highest label that is as low or lower than the input labels. Negation swaps H with L and + with -, leaving all other labels unchanged. Each component of a multi-event vector of labels can be computed independently. To illustrate the boolean operations, here is the table for the NAND operation:

NAND operation on event labels

	H	L	C	L+	H-	+L	-H	blank
H	L	H	C	H-	L+	-H	+L	
L	H	H	H	H	H	H	H	H
C	C	H	C	H-	L+	-H	+L	
L+	H-	H	H-	H-				
H-	L+	H	L+		L+			
+L	-H	H	-H			-H		
-H	+L	H	+L				+L	
blank		H						

Several of the positions that evaluate to *blank* actually have some information known about them. For example, H- NAND -H is either constant low, or both starts and ends high. We could introduce a label (+L+) for this, and another label (-H-) for a signal which is constant high or starts and ends low. However, there seems to be no need for these extra labels.

The vectors of labels for each event make a convenient algebra for computing, but are a little unwieldy to write. For labeling circuit diagrams, we need a more easily written set of labels. Ideally, it should convert the vectors of length six to the labels already chosen for two-phase clocking. There are 8^6=262,144 vectors of length six. Even after removing the illegal vectors, and the ones for which a stronger equivalent vector exists, there are far too many possible signal labels.

Let's examine the vectors for the signal labels of two-phase clocking. The vectors are given in the order ($rise_1$, mid_1, $fall_1$, $rise_2$, mid_2, $fall_2$).

Q1	(L+,C,+L,L,L,L)	Q2	(L,L,L,L+,C,+L)
Q/1	(H-,C,-H,H,H,H)	Q/2	(H,H,H,H-,C,-H)
V1	(,,C,,,)	V2	(,,,,,C)
S1	(C,C,C,,,)	S2	(,,,C,C,C)
LH1	(L+,L+,C,,,)	LH2	(,,,L+,L+,C)
HL1	(H-,H-,C,,,)	HL2	(,,,H-,H-,C)
V1S2	(,,C,C,C,C)	V2S1	(C,C,C,,,C)
V1V2	(,,C,,,C)		

Ideally, all the vectors should be derived from the basic ones defining the clock (Q1 and Q2). The vector for a basic timing signal consists of an L+, followed by one or more C's, followed by +L, with the rest of the events labeled L. For the clock to be useful, there must be at least one event labeled L. We can complete the set of timing signals by adding the negations of the basic ones (Q/1 and Q/2).

If a timing signal is used to control a storage element, the data input to the storage element must be stable every time the timing signal makes a transition from active to inactive. Thus we need a vector with a C in each place where the timing signal has a +L [or -H]. This creates the V1 and V2 labels.

The output of a storage element does not change from the time its control goes inactive to the time it goes active again. Such a signal can be labeled with a vector that has a C everywhere the timing signal has an L or +L [or H or -H]. This creates the V1S2 and V2S1 labels.

We want to control timing signals, turning them on and off. We are most interested in control signals that are not strongly synchronized with the timing signals. What vectors can be ANDed with a basic timing vector which would still result in a basic timing vector? Each event label must be as strong or stronger than the ones in the timing vector, except where the timing vector has an L. Vectors with an L+ or +L imply fairly strong synchronization with the timing signal. The only interesting vectors left for control signals are those that have a C wherever the timing signal has L+, C, or +L. This creates the S1 and S2 labels. Looking at all boolean operations combining a control signal with a timing signal does not reveal any other useful vectors for control signals.

Precharged circuits are similar to memory elements, except that two timing signals are usually involved. One timing signal stores a constant value, and the other modifies it. For definiteness, assume that the two timing signals are active high. The vectors then look like this:

precharge timing	L+	C	+L	L...	L	L	L	L...
evaluate timing	L	L	L	L...	L+	C	+L	L...
precharged high			H	H...	H-	H-	C	
precharged low			L	L...	L+	L+	C	

In a multi-phase clock system, a pair of precharge vectors could be generated for each pair of timing signals. The two-phase system does not include these vectors. Instead, the precharge timing signal is ignored and only the part of the vector relevant to the evaluation is included. These are the HL1, LH1, HL2, and LH2 signal labels.

After constructing the basic data vectors from the basic clock vectors, add any vectors that result from boolean operations on vectors that already exist. Doing this until no more new vectors are needed is the *boolean closure* of the data vectors. The label V1V2 was created by ANDing V1S2 and V2S1.

The simplification of the precharge labels does more than reduce their number from $2n^2$ to $2n$. When we take the boolean closure of the event vectors for data signals, the simplified set generates no new vectors, but the complete set of vectors generates many useless signal labels.

Major and minor cycles

Many systems perform actions that take longer than one cycle of the two-phase clock. If each possible action takes the same number of cycles, designers often talk about *major* and *minor* cycles. A minor cycle is the two-phase cycle we've just discussed. A major cycle is the sequence of minor cycles that allows one action to be done.

The two-phase clocking labels are easily extended to a major/minor cycle scheme. If there are *n* minor cycles in a major cycle, label the basic timing signals Q1a, Q1b, Q2a, Q2b, . . . , Q*n*a, and Q*n*b. Allow ORing the timing labels, as long as they are not from adjacent phases. For example, Q1a2a is acceptable, but Q1b2a is not, and Q1a2b is only acceptable if *n* is three or more. From this expanded set of basic timing signals, generate the full set of labels. It will include labels like V1bS2a2b3a and V1bV3a.

Some of the labels are quite long (particularly if a signal is stable for many minor cycles in a row). To shorten the labels, use a range indication instead of listing sequences of adjacent phases. Also, merge the *stable* phases after a *valid* phase into the V sequence. For example, V1bS2a2b3a could be abbreviated to V1b-3a, and S2a2b3a3b to S2a-3b. Only the V labels can be merged with subsequent S labels, as they are the only labels that refer to single events, rather than sequences of events. A label like LH1a-2b means (L+,L+,L+, L+,L+,L+, L+L+,L+, L+,L+,C), which is not the same as LH1aS1b-2b = (L+,L+,C, C,C,C, C,C,C, C,C,C).

Timing Analysis

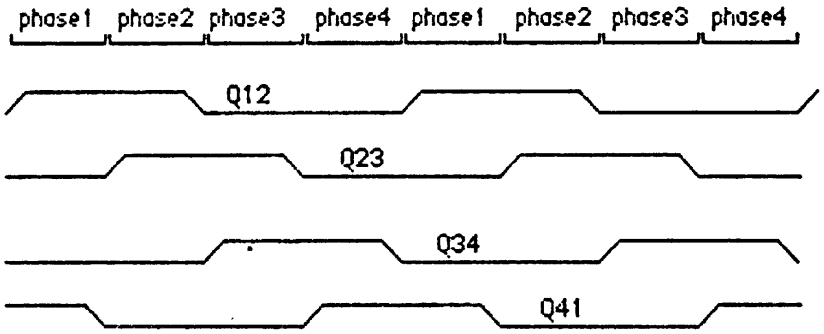
The techniques of worst-case timing analysis used in TV [J] and Crystal [O1 and O2] are easily applied to the event-vector model of clocking disciplines. Each signal must stabilize in less time than the sum of the durations of the events between the inputs becoming stable and the signal needing to be stable. The time taken for any particular signal to stabilize can be computed with any of the standard methods [J, O2, N]. Each signal results in an inequality of the form

$$t_s < e_1 + e_2 + \ldots + e_n$$

where t_s is the time needed for the signal to stabilize, and $e_1, \ldots, e_n$ are the durations of the events during which the signal is computed. The minimum event durations can be found by standard techniques from numerical analysis.

Four phase overlapping clocks

As an example of the multi-phase clocking scheme, let's consider a scheme that involves four overlapping clock signals, rather than two non-overlapping ones. This scheme has been used with cMOS circuits. The events defining periodic time are the standard ones for multi-phase clocking (rise$_k$, mid$_k$, fall$_k$ for k=1,2,3,4). The four basic timing signals are labeled Q12, Q23, Q34, and Q41. Each is active for two adjacent phases. Here is a timing diagram for them:



Note that Q12 and Q34 together form a non-overlapping two phase system, as do Q23 and Q41.

The vectors for the basic timing signals are:

Q12	(L+,C,C,	C,C,+L,	L,L,L,	L,L,L)
Q23	(L,L,L,	L+,C,C,	C,C,+L,	L,L,L)
Q34	(L,L,L,	L,L,L,	L+,C,C,	C,C,+L)
Q41	(C,C,+L,	L,L,L	L,L,L,	L+,C,C)

By negation we generate:

Q/12	(H-,C,C,	C,C,-H,	H,H,H,	H,H,H)
Q/23	(H,H,H,	H-,C,C,	C,C,-H,	H,H,H)
Q/34	(H,H,H,	H,H,H,	H-,C,C,	C,C,-H)
Q/41	(C,C,-H,	H,H,H	H,H,H,	H-,C,C)

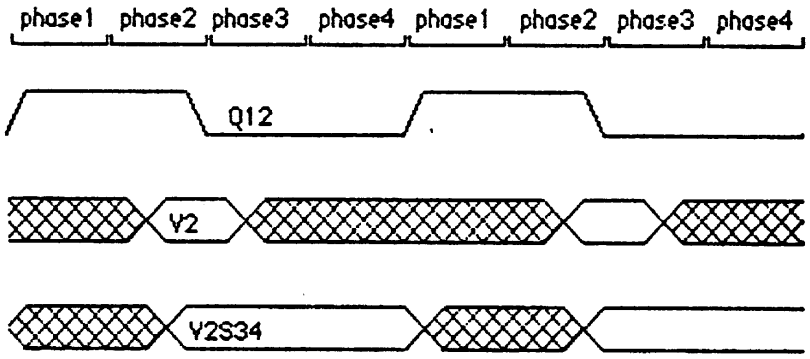
From the inputs to storage elements we generate:

V2	(„	„C,	„	„)
V3	(„	„	„C,	„)
V4	(„	„	„	„C)
V1	(„C,	„	„	„)

From the outputs of storage elements we generate:

V2S34	(,,,	,,C	C,C,C	C,C,C)
V3S41	(C,C,C,	,,,	,,C	C,C,C)
V4S12	(C,C,C,	C,C,C	,,,	,,C)
V1S23	(,,C,	C,C,C	C,C,C	,,)

As an illustration, here is a timing diagram for a storage element clocked with a Q12 signal:



The control signals needed for changing timing signals yield:

S12	(C,C,C,	C,C,C,	,,,	,,)
S23	(,,,	C,C,C,	C,C,C,	,,)
S34	(,,,	,,,	C,C,C,	C,C,C)
S41	(C,C,C,	,,,	,,,	C,C,C)

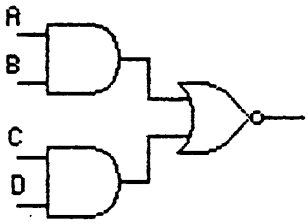
Taking the boolean closure of the data vectors adds:

V1S2	(,,C,	C,C,C	,,,	,,)
V2S3	(,,,	,,C	C,C,C	,,)
V3S4	(,,,	,,,	,,C	C,C,C)
V4S1	(C,C,C,	,,,	,,,	,,C)
V1V3	(,,C,	,,,	,,C,	,,)
V2V4	(,,,	,,C,	,,,	,,C)
S1	(C,C,C,	,,,	,,,	,,)
S2	(,,,	C,C,C,	,,,	,,)
S3	(,,,	,,,	C,C,C,	,,)
S4	(,,,	,,,	,,,	C,C,C)

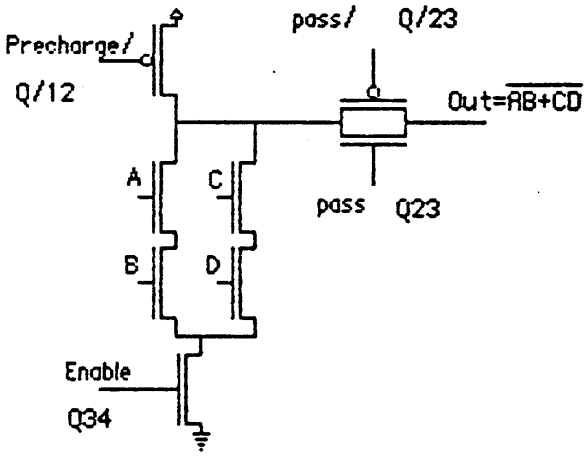
For the two-phase labeling scheme, more labels were introduced for pre-charged nodes. These HL labels are very useful for domino logic, but are generally not needed in four-phase systems. Although precharging is frequently used, the inputs to a precharged gate are always stable when they are being evaluated.

Examples of four phase circuits

Now we're ready to look at some examples of four-phase circuits. The following function:



is implemented with this pre-charged circuit:



The intermediate node is precharged high on phases 1 and 2, and the output node is precharged high on phase 2. The inputs are evaluated during phase 3, after which the output holds its value until the next phase 2. The output may be labeled V3S41. The inputs must be S3, so the gate is called a *phase-3* gate. The identical circuitry with different labeling can make a phase-1, 2, 3 or 4 gate. The output of a phase-*n* gate can only be used by a phase-*n+1* or *n+2* gate, so each gate introduces one or two phases of delay.

The precharge, evaluate, and pass signals are usually connected directly to the master clock. Dynamic and pseudo-static memory elements can be constructed by using a latch control signal for the pass signal (instead of the master clock). Ordinary cMOS (not precharged) gates must be used for generating timing signals from the master clocks, since the precharged circuits cannot have a Q labeled output.

As in the domino logic examples, the precharge signal is often connected directly to the enable signal and labeled Q34. The potential race conditions are not dangerous. Problems will only arise when both switches are on.

which can only occur between phases 2 and 3 or between 4 and 1. With both switches on between phases 2 and 3, a momentary short of power and ground occurs, raising power consumption, but not affecting the signals. Between phases 4 and 1, having both precharge and evaluate on may affect the intermediate node, but not the output. Therefore, the substitution of a Q34 signal for Q/12 on precharge is even less likely to cause problems in four-phase clocking than it is in domino logic. Unfortunately, the same is not true of the signals on the transmission gate, so both pass and pass/ signals are needed.

REFERENCES

[J] Norman P. Jouppi. "Timing Analysis for nMOS VLSI" *20th Design Automation Conference.* Miami Beach, Florida. June 1983. pp. 411--418.

[KLL] R.H. Krambeck, Charles M. Lee, and Hung-Fai Stephen Law. "Hi-Speed Compact Circuits with CMOS" *IEEE Journal of Solid-State Circuits.* SC-17(3). June 1982 (614--618).

[MC] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[N] L. S. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* ERL Memo ERL-M520. University of California, Berkeley, May 1975.

[NM] John Newkirk and Rob Mathews. *The VLSI Designer's Library.* Addison-Wesley (reading, Massachusetts) 1983.

[NMN] David Noice, Rob Mathews, and John Newkirk. "A Clocking Discipline for Two-Phase Digital Systems" *IEEE International Conferences on Circuits and Computers* September 1982. pp. 108--111.

[O1] John K. Ousterhout. "Crystal: a Timing Analyzer for nMOS VLSI Circuits" *Proceedings of the 3rd Caltech Conference on VLSI.* Bryant (editor), Computer Science Press, 1983. pp. 57-70.

[O2] John K. Ousterhout. "Switch-level Delay Models for Digital MOS VLSI" *21st Design Automation Conference.* Albuquerque, New Mexico June 1984. pp. 542--548.