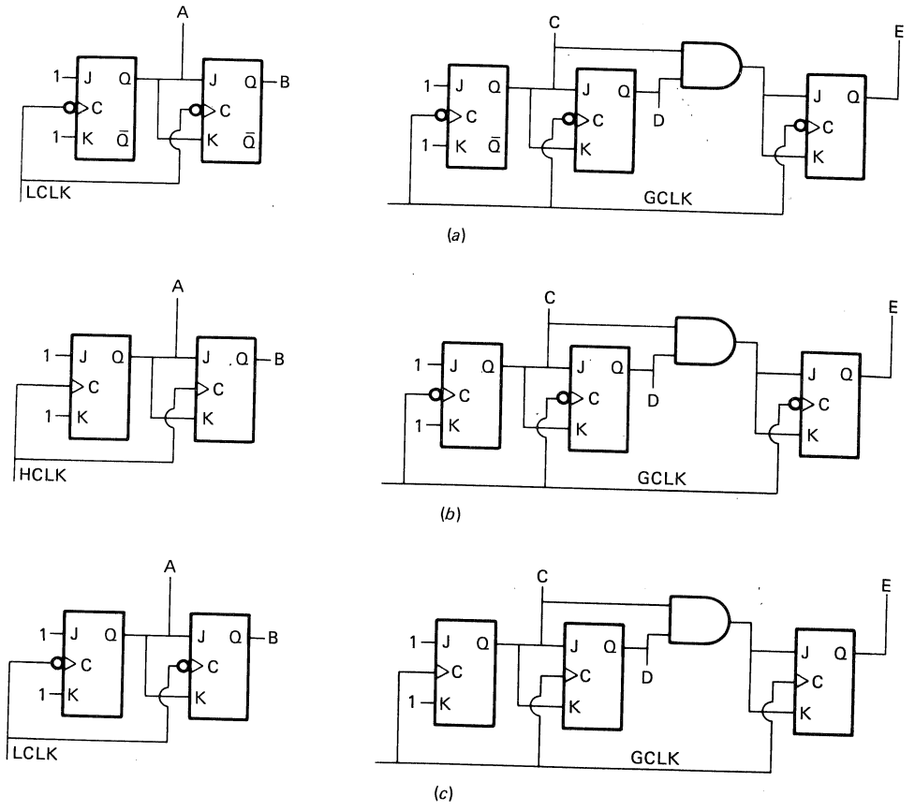


Figure P4-1 Gated clock configurations.

4-6 Design a six-state counter, using JK flip-flops for the ASM chart of Fig. 4-17.

4-7 Consider the counter of Prob. 4-6. It is called a Moebius counter and has several useful properties.

(*a*) Design a decoder of each of the six states. What do these decoders have in common? Can you find a simple rule for decoding a state without having to resort to K maps?

(*b*) What additional gating would be required to stop the counter when it reached a particular state?

Hint: It is a single gate and does not require decoding states.

(*c*) If the three flip-flops had different delays, would output races occur in the decoding of part (*a*)? Explain your answer.

CHAPTER

FIVE

CLOCKS AND SIGNALS

The clock signal of any sequential machine is very important. Up to now we have been ignoring the details of the clock signal, assuming that it had any attributes necessary for proper operation of the ASM. Actually, generating and distributing clock signals in an ASM is a complex subject.

Clock Skew

Since digital systems are often very large, the situation depicted in Fig. 5-1 usually occurs. Here, the number of flip-flops is so large that the clock circuit cannot drive all of them directly. In this example, two inverters are used to drive the clock lines. Some of the flip-flop clock inputs are connected to each inverter. The signal propagation delays through these two inverters are probably different. Usually only maximum delay times are specified for logic circuits. Any individual circuit may have a delay considerably less than the maximum. The delay through any logic circuit is dependent on the number of other circuits connected to it. If one inverter drove 10 clock inputs and the other drove 5 inputs, their propagation delays would be different, even if both inverters were electrically identical. Also, as seen in the last chapter, gating the clock also causes unequal clock delays.

Does the fact that the clocks of two flip-flops occur at different times cause problems? In the simple 2-bit shift register of Fig. 5-2, clock inputs are driven by two different sources. The clock of the first flip-flop X occurs later than the clock of the second flip-flop Z. The timing diagram in Fig. 5-2 shows data shifted through the register correctly. Because *CLKA* is later than *CLKB*, data are shifted properly from flip-flop X to flip-flop Z. In fact, whenever information is transferred from a later clock to an earlier clock, the information transfer will take place correctly. However, if information is transferred from an earlier to a later clock, the transfer could occur

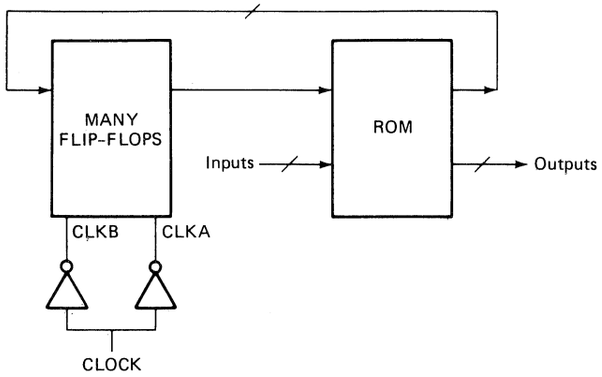


Figure 5-1 A large sequential machine.

incorrectly. In Fig. 5-3, flip-flop *X* is clocked by the earlier clock. Flip-flop *X* changes to a 1 so early that the later clock *CLKB* stores this 1 into flip-flop *Z* on the current clock transition and not on the next clock transition! Both flip-flops are loaded with the same datum simultaneously. The circuit fails to function as a shift register.

Circuits can seldom be arranged to always transfer information from a later to an earlier clock. Even in the simple counter of Fig. 5-4, data transfers must occur in both directions. Data are transferred from a later to an earlier clock, and vice-versa.

Before we can calculate the maximum delay permitted for *CLKB* of Fig. 5-3 without causing a malfunction, we need the definitions of the timing specifications of a flip-flop. The following definitions are illustrated in Fig. 5-5 for a type D flip-flop.

Setup time, t_S. The setup time is the time that an input must have valid data *before* a clock transition if the flip-flop is to recognize it reliably.

Hold time, t_H. The hold time is the time that an input must remain correct *after* a clock transition if the flip-flop is to recognize it reliably.

Propagation delay, t_P. The propagation delay is the time that occurs between the clock transition and an output change caused by that clock transition.

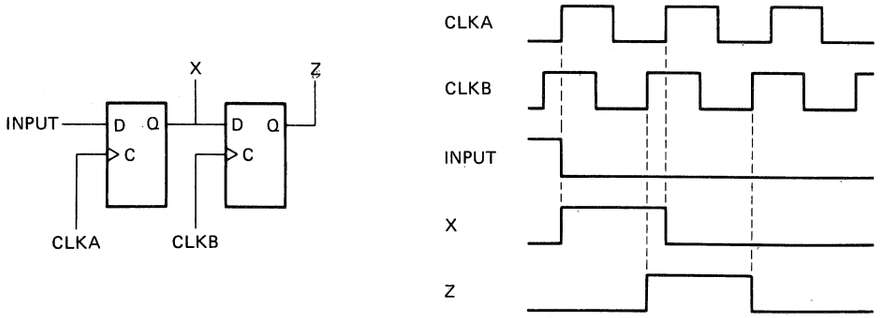


Figure 5-2 Correct data transfer from a later to an earlier clock.

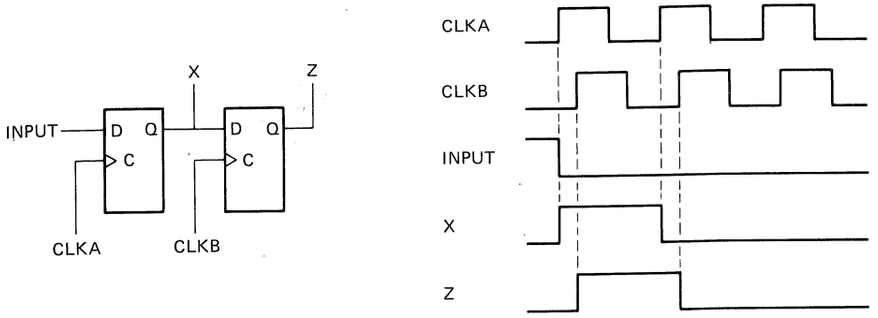


Figure 5-3 Incorrect data transfer from an earlier to later clock.

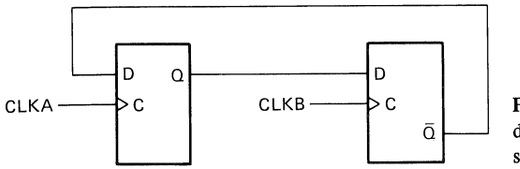


Figure 5-4 Simple counter that requires data transfers in both directions between skewed clocks.

Propagation delay is sometimes dependent on the direction of the output transition. For example, if the *Q* output makes a transition from 1 to 0, the propagation delay for a high-to-low transition t_{PHL} may be 25 ns. However, the propagation delay, if the output is making a low-to-high transition t_{PLH} may be 40 ns. Normally, maximum propagation delays are specified by the manufacturer. Also, typical propagation delays are specified. Typical delays are simply specifications averaged for a large number of circuits.

The difference in time between two clock signals is called *clock skew* and is computed as shown in Fig. 5-6. Here, a datum is being transferred from an earlier to a later clock, and the circuit could malfunction. To prevent the same datum from being stored in both flip-flops simultaneously, output *X* must not change until after the hold

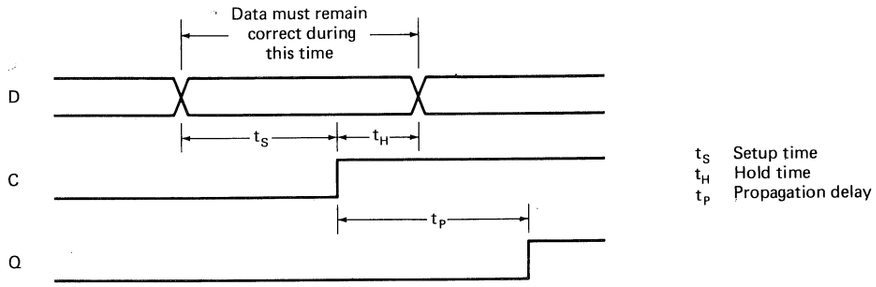


Figure 5-5 Timing specification for a D flip-flop.

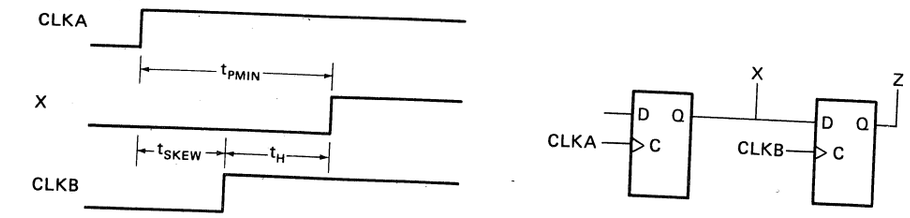


Figure 5-6 Calculation of clock skew.

time of flip-flop *Z*. The maximum allowable clock skew is easily read from the timing diagram as the *minimum propagation delay minus the hold time.* This calculation is correct for the worst-case situation of flip-flops connected directly to each other. If we add any gates between the flip-flops, the maximum allowable clock skew increases, as shown in Fig. 5-7. Often the hold time is guaranteed to be zero (i.e., the input may change simultaneously with the clock); so the maximum clock skew is equal to the minimum propagation delay. If the shortest (quickest) path in the circuit from one flip-flop to another also contains gates, you may add the *minimum* propagation delay of these gates to the clock-skew calculation. This will increase the allowable clock skew. *Minimum* gate propagation delays are seldom specified. You could estimate a minimum from the maximum and typical specifications. It is safest to assume a minimum of zero if none is specified.

What can be done if the calculated maximum allowable clock skew is exceeded by the difference in delays expected from the clock driving circuits? Several alternatives are possible. Clock driving circuits with shorter maximum delays will decrease the

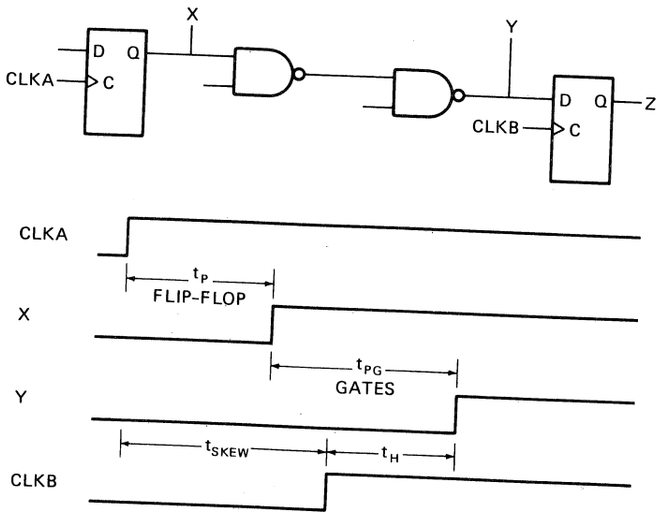


Figure 5-7 Clock skew with gate delays.

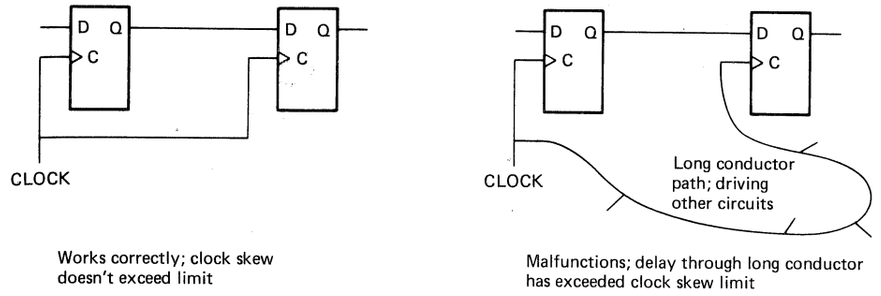


Figure 5-8 Logic signal delays through long conductors.

absolute delays and, more importantly, decrease the maximum *difference* in delays. This is because the maximum difference occurs if one clock driver operates with the guaranteed maximum delay while another has zero delay. The difference is simply the guaranteed maximum. Using gates or inverters fabricated on the same integrated circuit as clock drivers will reduce delay variations. Equalizing the number of inputs connected to each driving circuit output will also tend to equalize delays. Of course, you can add additional delay (e.g., two inverters) between flip-flops to increase the minimum propagation delay and increase allowable clock skew. As will be seen in the next section, this could decrease the speed at which your circuit will operate. Adding extra components will certainly increase cost and decrease circuit reliability. Finally, in very-high-speed digital circuits, even the delay in the wires themselves could cause the circuit to malfunction, as shown in Fig. 5-8. Interconnection delays vary, but a common rule of thumb is 6 ns/m for printed-circuit wiring.

Clock Rate

Every ASM implementation has a maximum clock rate, above which the circuit will not operate reliably. This maximum clock rate is determined by the time required for a change in flip-flop outputs to be converted to the next state and appear at the flip-flop inputs. The maximum-rate calculation for the simple ROM/register ASM, shown in Fig. 5-9, proceeds as follows. A clock pulse transition causes the flip-flop outputs to

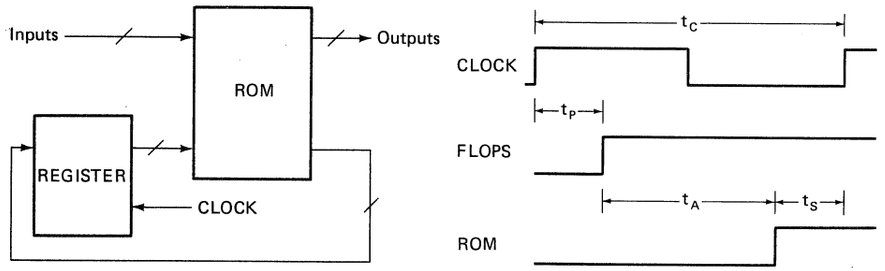


Figure 5-9 Maximum-operating-frequency calculation.

change after a propagation delay t_P. These flip-flop output changes are address inputs to the ROM. The outputs of the ROM change to their new values after the propagation delay or access time t_A of the ROM. These output changes are the next state and must appear at the flip-flop inputs no later than the worst-case setup time before the next clock transition. Thus, the time between clock transitions has a minimum value for proper operation.

$$t_C \geq t_P + t_A + t_S$$

The maximum clock rate or frequency is the reciprocal of the minimum period:

$$f_{MAX} = \frac{1}{t_C}$$

The calculation for maximum clock rate is performed identically if gates are used instead of a ROM. The longest delay path through the gating structure must be used to calculate the maximum propagation delay expected through the gates t_{PG}. The minimum clock period is obtained by using the gate propagation delay instead of the ROM access time.

$$t_C \geq t_P + t_{PG} + t_S$$

No matter how complex the digital circuit, you can always find the minimum clock period by looking for the *maximum delay* for a clock-induced flip-flop change to appear at a flip-flop input in time to be properly recognized at the next clock transition. This maximum delay may not be easily found in a complex circuit.

Clock Rate for Asynchronously Coupled Circuitry

In the simple counter of Fig. 5-10, each flip-flop is asynchronously coupled to the previous flip-flop. Since flip-flop A must change before B changes and B must change

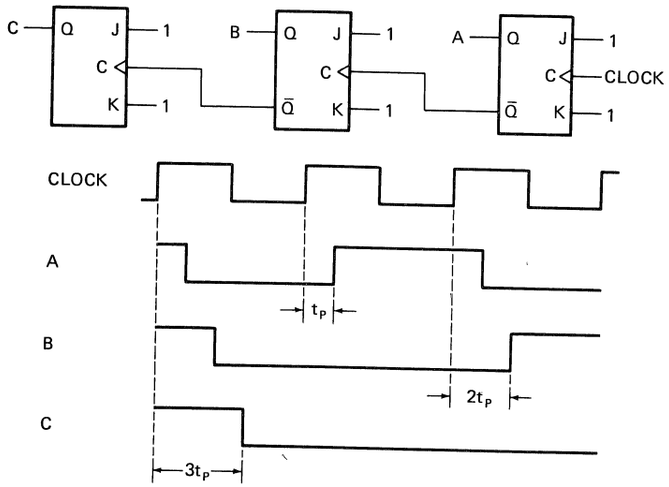


Figure 5-10 Propagation delays in an asynchronous counter.

before C can change, three flip-flop propagation delays will be required between the clock transition and the change in the output of flip-flop C. Flip-flop C behaves as if it has a propagation delay of $3t_P$. It is this longer, apparent propagation delay that must be used in maximum-clock-rate calculations for this circuit. Similarly, flip-flop B will have an apparent propagation delay of $2t_P$. However, since all three flip-flops do not change at each clock pulse, the time required to reach a new state varies, as shown in Fig. 5-10. Even so, we shall almost always have to consider the maximum flip-flop propagation delay to be $3t_P$ whenever using this counter in an ASM.

Clock Rate for a Gated Clock

Consider the ASM of Fig. 5-11. Both a normal positive transition clock *HCLK* and a gated clock *GCLK* are generated. Signals that gate the clock are created by combining flip-flop outputs with gates or a ROM. All these signals that enable the clock gate must have reached their correct state by the time *LCLK* becomes a 1 and enables the clock gate. Since these gating signals are generated by the ASM, we can easily calculate the minimum time t_{C1} that *LCLK* must remain low. In Fig. 5-11, *HCLK* clocks the ASM's flip-flops after a delay through the inverter t_I. The flip-flop outputs will change after a propagation delay t_P, and the gating signals will change after a gate delay t_{PG} (or t_A if using a ROM). The clock gating signals must appear before *LCLK* becomes a 1. Writing this statement as an inequality gives us the constraint on t_{C1}.

$$t_{C1} \geq t_I + t_P + t_{PG}$$

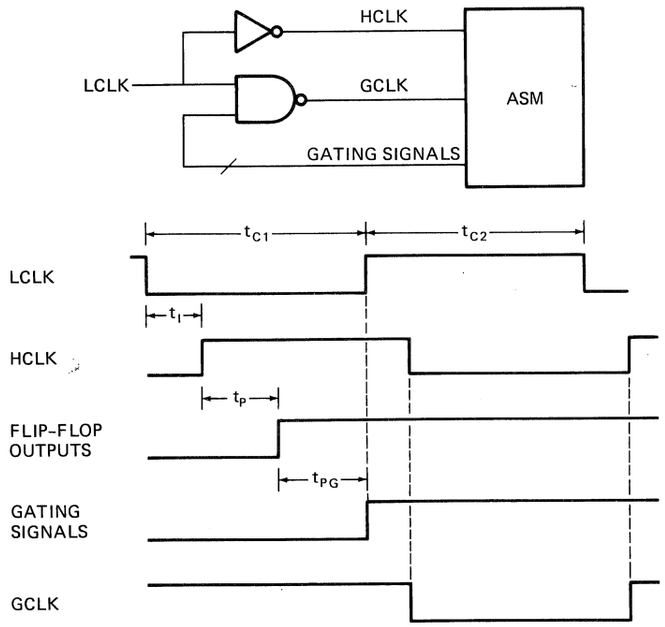


Figure 5-11 Maximum clock rate with gated clock.

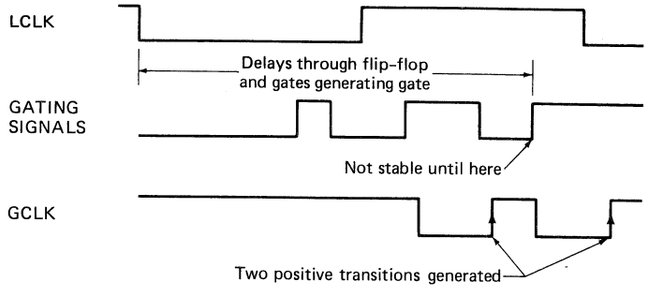


Figure 5-12 Gated clock malfunction caused by exceeding maximum rate.

If we are using a symmetric clock signal (i.e., $t_{C1} = t_{C2}$), then the minimum clock period $t_C = 2t_{C1}$.

$$t_C \geqslant 2(t_I + t_P + t_{PG})$$

This additional factor of 2 can potentially halve the maximum operating frequency of the circuit! If gated clocks must be used and operating speed is important, two approaches may be used to prevent the maximum operating speed from being halved. First, any delay associated with generating the gating signals should be as small as possible. The clock inverter and those flip-flops and gates that generate the clock gating signals should be faster than the other logic circuits in the system. Second, since our timing restriction really only applied to t_{C1}, when *LCLK* is low, we can reduce t_{C2} to increase the operating speed of the circuit without violating the inequality for t_{C1}. However, t_{C2} cannot be made arbitrarily small for two reasons. First, the flip-flops in the ASM have a specified minimum clock-pulse width, below which proper operation is not guaranteed. Second, very narrow signal pulses are difficult to generate and distribute for electrical reasons.

If we were to violate the minimum-time restriction for t_{C1}, the situation of Fig. 5-12 might arise. The gating signals do not become completely stable until well past the point at which *LCLK* enables the clock gate. As a result, two positive transitions occur on the gated clock, where only one is desired.

Gate-Implemented ASM Timing Example

A very simple ASM is shown in Fig. 5-13. The timing specifications for the parts used are tabulated in Table 5-1. We'd like to calculate maximum allowable clock skew and the maximum operating frequency for this circuit. The maximum clock skew is the smallest propagation delay between any two flip-flops minus the hold time. In this simple example, the shortest path is easily found. The smallest delay will obviously occur through gate *C*. We have the minimum propagation delay specified for the flip-flops, but, unfortunately, not for the gates. A conservative design could assume no delay in gate *C* so that

$$t_{SKEW} \leqslant t_{PMIN} - t_H$$
$$\leqslant 10 - 5 = 5 \text{ ns}$$

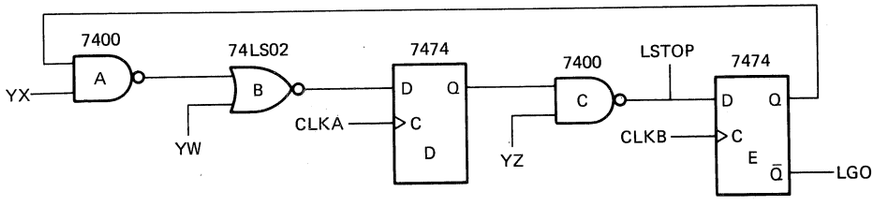


Fig. 5-13 ASM for timing example.

To calculate the maximum operating frequency, we must calculate the maximum delay through the longest path. This path is easily spotted as the connection through gates *A* and *B*. We know that the output of gate *A* will move in the opposite direction from the outputs of flip-flop *E* and gate *B* since *A* and *B* are inverting gates. If flip-flop *E* makes a 0-to-1 transition, the circuit delay will be:

$$t_{PLHE} + t_{PHLA} + t_{PLHB}$$
$$25 + 15 + 20$$
$$60 \text{ ns}$$

If flip-flop *E* makes a 1-to-0 transition, then the delay will be:

$$t_{PHLE} + t_{PLHA} + t_{PHLB}$$
$$40 + 22 + 20$$
$$82 \text{ ns}$$

The longest delay will be 82 ns, so that the clock period will be:

$$t_C \geqslant 82 \text{ ns} + t_S$$
$$\geqslant 82 + 20 \text{ ns}$$
$$\geqslant 102 \text{ ns}$$

The maximum clock frequency is:

$$f_{MAX} = \frac{1}{102 \text{ ns}} = 9.8 \text{ MHz}$$

Table 5-1 Component timing specifications

	t_{PLH}			t_{PHL}		
Part	Min	Typ	Max	Min	Typ	Max
7400		11	22		7	15
74LS02		10	20		10	20
7474	10	14	25	10	20	40
7474: Setup Time $t_S = 20$						
Hold Time $t_H = 5$						

Note: All times are in nanoseconds.

Other Calculations for the ASM

The ASM in Fig. 5-13 has inputs and outputs. The timing constraints for these inputs and outputs must be calculated so that other engineers desiring to connect or *interface* to this circuit will be able to do so. For example, if this circuit were implemented as an MSI circuit, it would be drawn as shown in Fig. 5-14 and timing specifications would be given for all inputs and outputs.

Each input will require both setup and hold specifications. The setup time of an input will be the setup time of the flip-flop affected by that input plus the maximum gate delay between the input and the flip-flop. If an input affects more than one flip-flop, use the longest setup time of all those calculated. Setup time will now be dependent on the transition direction of the input because the gates used have delays that depend on the transition direction.

The setup time for a low-to-high transition on input YX is given by

$$t_{SLHX} = t_{SD} + t_{PLHB} + t_{PHLA}$$
$$= 20 + 20 + 15$$
$$= 55 \text{ ns}$$

The setup time for a high-to-low input transition is:

$$t_{SHLX} = t_{SD} + t_{PHLB} + t_{PLHA}$$
$$= 20 + 20 + 22$$
$$= 62 \text{ ns}$$

If one setup time t_{SX} is to be specified for convenience, it should be the larger of the two times calculated.

$$t_{SX} = 62 \text{ ns}$$

The hold time on an input will be equal to the hold time of the flip-flop affected by that input *minus* the minimum propagation delays of any gates between the input and the flip-flop.

To calculate the hold time for input YW, we must assume a minimum propagation delay for gate B. If we're conservative and choose to assume 0 ns for this delay, the hold time of this input is just equal to the hold time of the flip-flop.

$$t_{HW} = t_H = 5 \text{ ns}$$

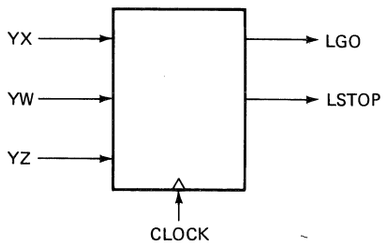


Figure 5-14 ASM of Fig. 5-13 as an MSI circuit.

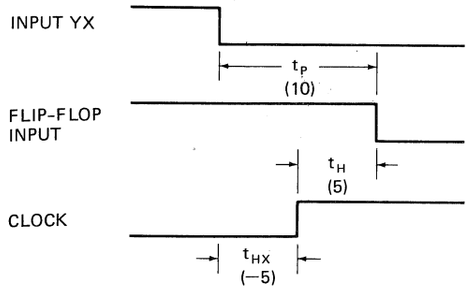


Figure 5-15 Negative hold time.

We can assume the minimum delay for gate B to be 2 ns. Now the hold time could be specified as:

$$t_{HW} = 5 - 2 = 3 \text{ ns}$$

If many gates are connected between an input and a flip-flop, the hold time could become negative. Assume that gates A and B in Fig. 5-13 each have a minimum propagation delay of 10 ns. The hold time of input YW will be:

$$t_{HW} = t_{HD} - t_{PG}$$
$$= 5 - 10 = -5 \text{ ns}$$

This negative hold time is shown in the timing diagram of Fig. 5-15. Input YW may actually change 5 ns *before* the clock without a circuit malfunction. Since at least 10 ns are required to propagate this change through gates A and B, the change will arrive at the D input of flip-flop D no earlier than 5 ns *after* the clock.

Finally, we must specify the propagation delays for the outputs. These delays will be equal to the maximum flip-flop delays plus any gate delays.

$$t_{PLHSTOP} = t_{PLHC} + t_{PHLD}$$
$$= 22 + 40 = 62 \text{ ns}$$
$$t_{PHLSTOP} = t_{PHLC} + t_{PLHD}$$
$$= 15 + 25 = 40 \text{ ns}$$

The timing specifications for the circuit of Fig. 5-13 are summarized in Table 5-2.

READ-WRITE-MEMORY TIMING

Read-write memory is most easily interfaced to an ASM as shown in Fig. 5-16. Since the chip-select input is activated in the last half of the clock cycle, RWM timing is somewhat similar to a gated clock. RWM timing is divided into *read-cycle timing* and *write-cycle timing*. We'll first discuss all the timing specifications of an RWM. Then we'll show how they are related to the ASM interface of Fig. 5-16. Finally, we will

Table 5-2 Timing specifications for the ASM of Fig. 5-13

	Inputs		
Setup times	*X*	*W*	*Z*
t_{SLH}	55	40	35
t_{SHL}	62	40	42
Hold times			
t_H	5*		

	Outputs			
	LSTOP		LGO	
Propagation delays	Max	Min	Max	Min
t_{PLH}	62	10*	25	10
t_{PHL}	40	10*	40	10

Note: All times in nanoseconds

*These calculations conservatively assume 0 ns as the minimum gate propagation delay.

give a numerical example for a real RWM circuit. Since memory nomenclature is not standardized, we will use Intel Corporation's nomenclature for a 2101A RWM.

Read-Cycle Timing

Six timing specifications are required for read-cycle timing.

Read-cycle time, T_{RC}. This time is specified as a minimum and is the shortest time possible to complete a read cycle. It is measured between changes of address inputs. Cycle time is a composite of other timing parameters and is a measure of memory performance.

Access time, T_A. This time is specified as a maximum. It is the time required for valid data to appear on the memory's data outputs after an address change.

Chip-enable to output time, T_{CO}. This time is specified as a maximum. It is the time

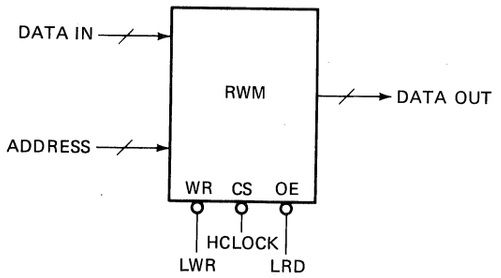


Figure 5-16 Read-write memory interface.

required for the three-state data outputs to be enabled after the chip-select (also called chip-enable) input is asserted.

Output enable time, T_{OE}. This time is specified as a maximum. It is the time required for the three-state data outputs to be enabled after the output enable input is asserted.

Data-false time, T_{DF}. This time is specified as a maximum. It is the time required for the three-state outputs to be disabled. It is measured from whichever input, chip-select or output enable, becomes false first.

Data-valid-after-address-change time, T_{OH}. This time is specified as a minimum. It is the time data will remain valid after an address change.

The timing diagram for the read cycle of the RWM of Fig. 5-16 is shown in Fig. 5-17. Uppercase *T* represents timing parameters associated with the RWM itself. Lowercase *t* represents timing parameters of the ASM interfaced to this memory. These latter parameters are:

t_{PA}. The propagation delay from the clock edge until the address arrives at the memory.

t_{PRD}. The propagation delay from the clock edge until the read signal arrives at the memory.

t_S. The setup time of the flip-flops that will save or test the data output from memory.

t_H. The hold time of the flip-flops that will save or test the data output from memory.

Several timing constraints may be seen in Fig. 5-17. The address must arrive in sufficient time to allow output data to reach the ASM's data flip-flops at or before their setup time:

$$t_{PA} + T_A + t_S \leqslant t_{CLK}$$

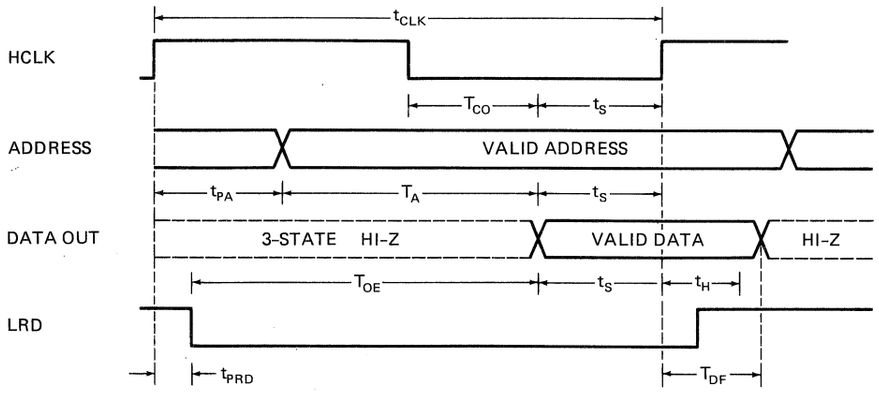


Figure 5-17 Read cycle timing.

The read signal must reach the output enable input soon enough to enable the tristate outputs before the setup time of the ASM's data storage flip-flops:

$$t_{PRD} + T_{OE} + t_S \leqslant t_{CLK}$$

The chip-select input is asserted by the clock. Chip select must occur soon enough to enable the tristate outputs also:

$$T_{CO} + t_S \leqslant \frac{t_{CLK}}{2}$$

The three inequalities above must all be satisfied. Thus, the inequality requiring the largest value of t_{CLK} will constrain the speed of operation of the RWM.

Data output from the memory must remain correct until after the hold time of the flip-flops that are to read those data:

$$T_{DF} \geqslant t_H$$

This inequality must be satisfied for proper operation.

Write-Cycle Timing

Seven parameters are required to specify write-cycle timing

Write-cycle time, T_{WC}. This time is specified as a minimum. It is the smallest time in which a write cycle can be completed. It is analogous to read-cycle time and is a measure of memory performance.

Address to write time, T_{AW}. This time is specified as a minimum. It is the shortest time after an address change that the memory's write circuitry can be enabled. If the write circuitry is enabled earlier, the memory may write into the wrong address. The write circuitry is enabled by the last signal asserted of either the chip-select or the write signal.

Chip-select to write time, T_{CW}. This time is specified as a minimum. This memory is *not* edge-triggered. It remembers the data present on its inputs when the write signal becomes false. This time is the minimum time that chip select must be asserted before the write signal becomes false.

Write pulse time, T_{WP}. This time is specified as a minimum. It is the minimum time that the write signal must be asserted.

The above definitions for T_{CW} and T_{WP} assume that the write signal will become false before the chip select. Since the write circuitry is activated by the AND function of the write and chip-select signals, their timing specifications may be interchanged. Thus, T_{CW} could be the time that the write signal must be asserted before the chip select becomes false. T_{WP} would become the minimum time that the chip select must be asserted. This latter interpretation is used in the ASM interface of Fig. 5-18.

Write recovery time, T_{WR}. This time is specified as a minimum. It is the time the address must be held correct after the write signal becomes false (or chip-select signal if that becomes false first).

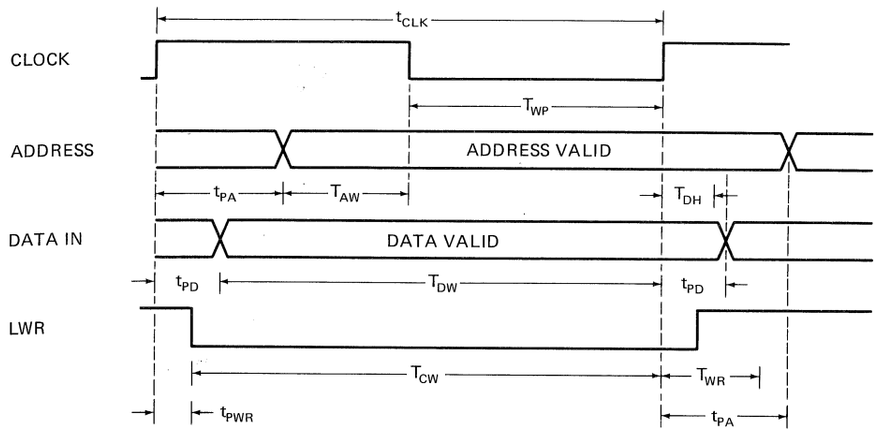


Figure 5-18 Write cycle timing.

Data setup time, T_{DW}. This time is specified as a minimum. It is the time the data must be correct before the write signal becomes false (or chip-select signal if it becomes false first).

Data hold time, T_{DH}. This time is specified as a minimum. It is the time the data must be correct after the write signal becomes false (or chip-select signal if it becomes false first).

The memory write cycle for the RWM of Fig. 5-16 is shown in Fig. 5-18. Two additional ASM parameters will be required for the write cycle.

t_{PD}. The propagation delay from the clock edge until data are correct at the memory data inputs.

t_{PWR}. The propagation delay from the clock edge until the write signal arrives at the memory.

Several timing constraints may be read from Fig. 5-18. The address must reach the memory sufficiently soon to prevent data from being written into the wrong memory location:

$$t_{PA} + T_{AW} \leqslant \frac{t_{CLK}}{2}$$

The write signal must reach the memory at least T_{CW} before the chip-select signal becomes false.

$$t_{PWR} + T_{CW} \leqslant t_{CLK}$$

The chip-select input must be at least a write-pulse width long.

$$T_{WP} \leqslant \frac{t_{CLK}}{2}$$

The data must arrive sufficiently soon to be properly recognized by the RWM:

$$t_{PD} + T_{DW} \leqslant t_{CLK}$$

Thus, we have four inequalities. All must be satisfied; so one inequality will limit the minimum clock time allowed.

Additionally, the hold time requirement of the memory must be satisfied:

$$t_{PD} \geqslant T_{DH}$$

Finally, the address must remain stable after the chip-select signal becomes false:

$$T_{WR} \leqslant t_{PA}$$

Memory Timing Example

Table 5-3 lists the timing parameters of an Intel 2101A-2 RWM. Also listed are timing parameters of the ASM controlling the RWM. The read clock cycle is governed by three inequalities given previously:

Address delay

$$t_{PA} + T_A + t_S \leqslant t_{CLK}$$
$$40 + 250 + 20 \leqslant$$
$$310 \text{ ns} \leqslant$$

Read signal delay

$$t_{PRD} + T_{OE} + t_S \leqslant t_{CLK}$$
$$40 + 130 + 20 \leqslant$$
$$190 \text{ ns} \leqslant$$

Chip-select delay

$$T_{CO} + t_S \leqslant \frac{t_{CLK}}{2}$$
$$180 + 20 \leqslant$$
$$400 \text{ ns} \leqslant t_{CLK}$$

Since all three inequalities must be satisfied, we must limit the clock period to be:

$$t_{CLK} \geqslant 400 \text{ ns}$$

Finally, we should check the hold-time inequality for the read cycle:

$$T_{DF} \geqslant t_H$$
$$180 \geqslant 5 \text{ ns}$$

The hold time is easily satisfied.

Table 5-3 Memory timing example parameters

2101A-2 read-cycle parameters	
T_{RC}	250 ns
T_A	250 ns
T_{CO}	180 ns
T_{OE}	130 ns
T_{DF}	180 ns
T_{OH}	40 ns
2101A-2 write-cycle parameters	
T_{WC}	170 ns
T_{AW}	20 ns
T_{CW}	150 ns
T_{WP}	150 ns
T_{WR}	0 ns
T_{DW}	150 ns
T_{DH}	0 ns
ASM parameters	
t_{PA}	40 ns
t_{PRD}	40 ns
t_S	20 ns
t_H	5 ns
t_{PD}	40 ns
t_{PWR}	40 ns

Four inequalities control the write clock cycle.

Address delay

$$t_{PA} + T_{AW} \leqslant \frac{t_{CLK}}{2}$$
$$40 + 20 \leqslant$$
$$120 \text{ ns} \leqslant t_{CLK}$$

Write signal delay

$$t_{PWR} + T_{CW} \leqslant t_{CLK}$$
$$40 + 150 \leqslant$$
$$190 \text{ ns} \leqslant$$

Chip-select width

$$T_{WP} \leqslant \frac{t_{CLK}}{2}$$
$$150 \leqslant$$
$$300 \text{ ns} \leqslant t_{CLK}$$

Data delay

$$t_{PD} + T_{DW} \leqslant t_{CLK}$$
$$40 + 150 \leqslant$$
$$190 \text{ ns} \leqslant$$

The write clock cycle must be greater than or equal to 300 ns to satisfy all these inequalities. The data hold time inequality must be checked:

$$t_{PD} \geqslant T_{DH}$$
$$40 \geqslant 0 \text{ ns}$$

The address hold inequality must also be satisfied:

$$T_{WR} \leqslant t_{PA}$$
$$0 \leqslant 40 \text{ ns}$$

Both inequalities are easily satisfied.

Memory Performance Improvement

Since both read and write timing inequalities must be satisfied, the system in the previous example must operate with a 400 ns or longer clock cycle. This is considerably slower than T_{RC} of 250 ns or T_{WC} of 170 ns. The RWM is operating below its capability because of the simple interface we've chosen. Since the second half of the read cycle limits the timing, we could expect to improve system speed by using an asymmetric clock signal. The chip-select delay inequality of the read cycle constrained the *low part* of the clock cycle to be not less than 200 ns. Let's tabulate our timing restrictions in Table 5-4. We can easily see from this table that we can shorten the entire clock cycle to 310 ns. We must also be sure that the high part of the cycle is greater than 60 ns, while the low part must be greater than 200 ns. Since it is easier to generate asymmetric clocks in which each part is related by a common multiple, we'll choose 103.3 ns for the high part of the clock cycle and 206.7 ns for the low part, as shown in Fig. 5-19. The entire clock cycle is 310 ns, while each part of the cycle meets the timing restrictions of Table 5-4. The system's performance has been increased by almost 25 percent.

Table 5-4 Memory performance restrictions

	Clock cycle		
Inequality	High part	Low part	Entire
Read cycle			
Address delay			310
Read signal delay			190
Chip-select delay		200	
Write cycle			
Address delay	60		
Write signal delay			190
Chip-select width		150	
Data delay			190

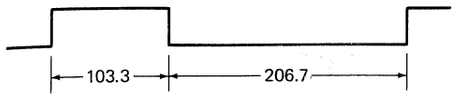


Figure 5-19 Maximum performance clock signal for memory interface.

NOISE IN CIRCUIT IMPLEMENTATIONS

All the digital signals we have used in this book have been ideal. They have always caused the action desired. Real digital signals are far from ideal. They are corrupted by interference from many sources. All this interference is collectively called *noise* and can cause particularly vexing circuit malfunctions. A noise malfunction may not be continuous. It may occur only occasionally. The circuit may malfunction every hour, every day, or a few times each year. It may malfunction only with particular data inputs or only on alternate Thursdays. Immunity to these malfunctions is accomplished by careful logical, electric, and physical design. We will discuss noise problems and their solutions without much electric engineering. For those who desire a more technical treatment, an excellent synopsis of digital noise problems and their solutions is found in Blakeslee.[1]

Noise Margin

The manufacturers of digital ICs guarantee *worst-case* voltages for both 0 and 1 logic levels. For example, let's consider the logic family called *TTL* (transistor-transistor logic). The output of a TTL circuit is guaranteed to be greater than 2.4 V if a logic 1 is being represented. However, an input of a TTL circuit will recognize any voltage greater than 2.0 V as a logic 1. The difference between the *output level* of 2.4 and the *input threshold* of 2.0 V is the *noise margin* of 0.4 V. A TTL 0 output is guaranteed to be less than 0.4 V, while a TTL input recognizes any voltage less than 0.8 V as a 0. The 0-level noise margin for TTL is also 0.4 V. Any TTL system will tolerate up to 0.4 V of interference on its signals before that system malfunctions. This noise margin is very important for a great amount of noise is present in most complex digital systems. Without noise margins, it would be difficult or impossible to construct reliable digital systems.

The worst-case guaranteed output levels are specified for *rated* fan out. A TTL gate with a maximum fan out of 10 is guaranteed to have outputs greater than 2.4 and less than 0.4 V only when connected to no more than 10 inputs. Most digital circuits

[1] Thomas R. Blakeslee, Digital Design with Standard MSI and LSI, John Wiley & Sons, New York, 1975, chap. 10.

will work better than their worst-case specification. A TTL circuit might have actual output levels of 0.2 and 3.6 V. Such a circuit might work even if its fan-out rating is exceeded or if more than 0.4 V of noise is present on its data lines. For this reason, *successful operation of a circuit does not mean the circuit has been correctly designed.* Many poorly designed and poorly constructed circuits will work correctly. Unfortunately, such a circuit may work correctly *only* with a particular "better-than-normal" IC or *only* with a specific power supply voltage or *only* at a particular temperature or *only* when Saturn is in conjunction with Mars! Manufacturers' worst-case ratings are *carefully* specified to include effects of temperature, fan out, power supply voltage, etc. *Always design to the worst-case specifications.* Never violate fan-out or propagation delay specifications. *Measure* the noise to make sure it is less than the worst-case noise margin. Even though these extra steps require time to be performed, the overall cost of your design will be reduced because fewer problems will occur in production or use of your circuit.

Transmission Lines

Ideally, a wire should carry a digital signal from one circuit to another without changing that signal in any way. Unfortunately, real interconnections do change digital signals considerably. A wire carrying a fast digital signal acts like a *transmission line.* It will be impossible to completely explain transmission lines without advanced electric engineering. Instead, we will try to explain their operation intuitively.

Digital signals travel along transmission lines at a finite speed. This *propagation velocity* is about 0.169 m/ns for the printed-circuit wiring used for most digital circuits. Propagation velocity is almost always specified by giving its reciprocal or propagation time of 5.9 ns/m (1.8 ns/ft). Transmission lines also have a *characteristic impedance* Z_0. This impedance is the ratio of the voltage to the current of the propagating signal and is about 50 to 200 Ω for printed-circuit connections. When a signal propagating down a transmission line reaches the end of that line, several things may happen. If the transmission line has a resistor equal to Z_0 connected to its receiving end, the signal will simply stop propagating. This situation is as close as we can come to ideal signal transmission. All we must consider is the 5.9-ns/m propagation time of the signal.

Most digital signal connections are terminated by gate inputs. Not only are gate input impedances not equal to Z_0, but gate inputs are nonlinear. That is, the gate input impedance is a function of the voltage applied to that input. When a digital signal reaches a gate input, part of the signal is reflected backwards toward the source. Since the source is usually a gate output which also does not have an impedance of Z_0, the reflected signal will be reflected again when it reaches the source. The signal will be reflected back and forth between the source and receiving ends. The amplitude of the reflected signal is reduced at each reflection. Eventually, the reflections will no longer be visible, and both ends of the wire will have the same stable logic level. Transmission-line reflections in a digital system are called *ringing* and appear as shown in Fig. 5-20 if viewed on an oscilloscope.

Everything we've said about transmission lines has been simplified. Different kinds of gate inputs and outputs will have different kinds of nonlinear impedances. Each

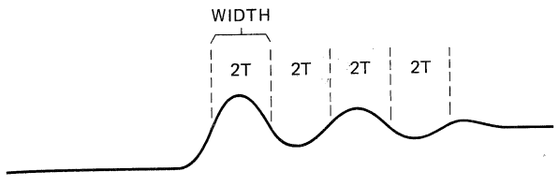


Figure 5-20 Ringing.

physical discontinuity in the transmission line will be a discontinuity in impedance that will produce a reflection. These discontinuities include multiple paths, logic circuit inputs connected along the wire, and connectors through which the interconnection passes. This myriad of reflections will produce a signal much more complex than Fig. 5-20.

Effect of Reflections

Reflections occur both on clock lines and other signal lines. Reflections on clock lines can cause a flip-flop to be clocked twice. Assume the ringing of Fig. 5-20 appeared on the clock input of a flip-flop. That flip-flop would be clocked twice if the amplitude of the ringing exceeded the clock input threshold and the width of the ringing exceeded the minimum pulse width of the flip-flop's clock input. How could we prevent the flip-flop from being clocked twice? One technique we could use would be to reduce the width of the ringing so that it would be less than the minimum pulse width required to clock the flip-flop. The ringing width is equal to the time required for a *round trip* of the signal on the transmission line. Making the transmission line shorter will reduce the width of the ringing. To ensure that the ringing is less than the minimum clock-pulse width, we can make all our interconnections shorter than

$$\text{maximum length} = \frac{\text{minimum clock-pulse width}}{2\ (\text{propagation time})}$$

For the 7474 flip-flop used in a previous example, the minimum clock-pulse width is 30 ns:

$$\text{maximum length} = \frac{30\ \text{ns}}{2\ (5.9\ \text{ns/m})}$$

$$= 2.54\ \text{m}$$

If no clock line exceeds 2.54 m, the width of the ringing will not exceed 30 ns and will not clock the flip-flop incorrectly.

Especially for fast-logic families, the restriction on clock line length may be intolerably short. The only solution possible when using very fast logic may be reducing the amplitude of the reflections. This can be done by modifying the impedance of gate outputs and inputs to more closely match the Z_0 of the transmission line. Often, this technique involves inserting a resistor in series with the driver output and another resistor in parallel with the receiving end of the transmission line. Selection of values

of these resistors is beyond the scope of this book. Again, the interested reader is advised to start by looking at Blakeslee.[1]

The same sort of problems occur with signal lines as with clock lines. However, an additional solution is available for curing noise on signal lines. Ringing occurs right after a signal transition and will eventually die away. We can delay the next clock transition sufficiently so that the amplitude of any ringing on signal lines will be small and not cross the logic threshold voltage. If possible, we should always design circuits to operate below their maximum possible speed. More problems will occur as a circuit operates closer to its limit of correct operation. We can use an *effective propagation velocity* of digital signals that includes reflection effects. The effective propagation velocity will depend on output impedances, input impedances, and other impedance discontinuities. Since signal wires usually are shorter and have fewer discontinuities than clock wires, we can approximate reflection effects by considering only output and input impedances. This means that we can specify effective propagation velocities for logic families. For example, *Schottky TTL* (STTL) signals will be sufficiently close to correct levels when they first arrive at the end of a transmission line. Thus, the effective propagation time of STTL signals is:

$$T_{\text{STTL}} = 5.9 \text{ ns/m}$$

On the other hand, standard TTL signals will not reach correct values at the receiving end until three trips over the transmission line (from source to receiver to source to receiver). The effective propagation time of TTL signals will be 3 times slower:

$$T_{\text{TTL}} = 17.7 \text{ ns/m}$$

A logic system built with TTL would operate more slowly than a STTL system for *two* reasons. First, STTL internal propagation delays are less than TTL delays. Second, the wire propagation delays of STTL will appear to be one-third those of the TTL system. This is because the amplitude of a TTL signal at the end of a transmission line does not become correct until the signal makes three trips on the interconnection.

Other Noise Sources

Each TTL circuit has both a power and ground connection. These two connections complete the electrical circuit for logic signals between ICs. For example, if a flip-flop output goes from 1 to 0, current will flow *into* the flip-flop output. This current comes from the inputs of gates connected to this flip-flop's output. Every electrical circuit requires two wires. The ground wire of the flip-flop returns current to the gates which supplied the current going into the flip-flop output. If the ground wire was constructed identically to the signal wire, they would both have the same impedance. Since they have the same current flowing in them, any voltage change in the flip-flop's output would be divided equally between the two wires, as shown in Fig. 5-21. Until reflections subsided, the ground wire would become more positive by half the output

[1] Blakeslee, Digital Design with Standard MSI and LSI, pp. 262–266.

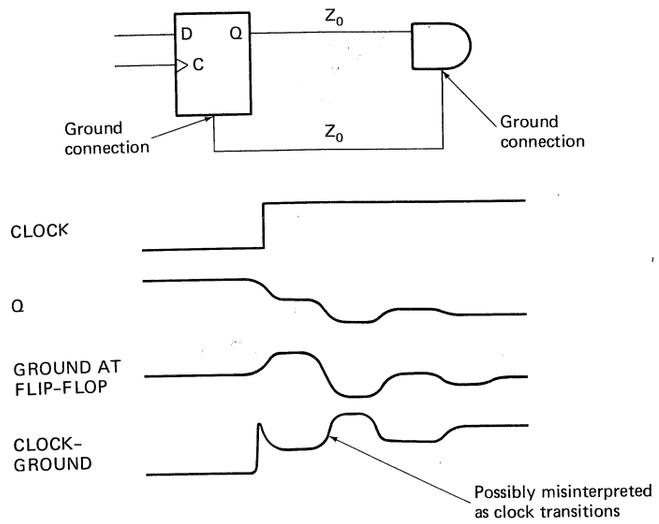
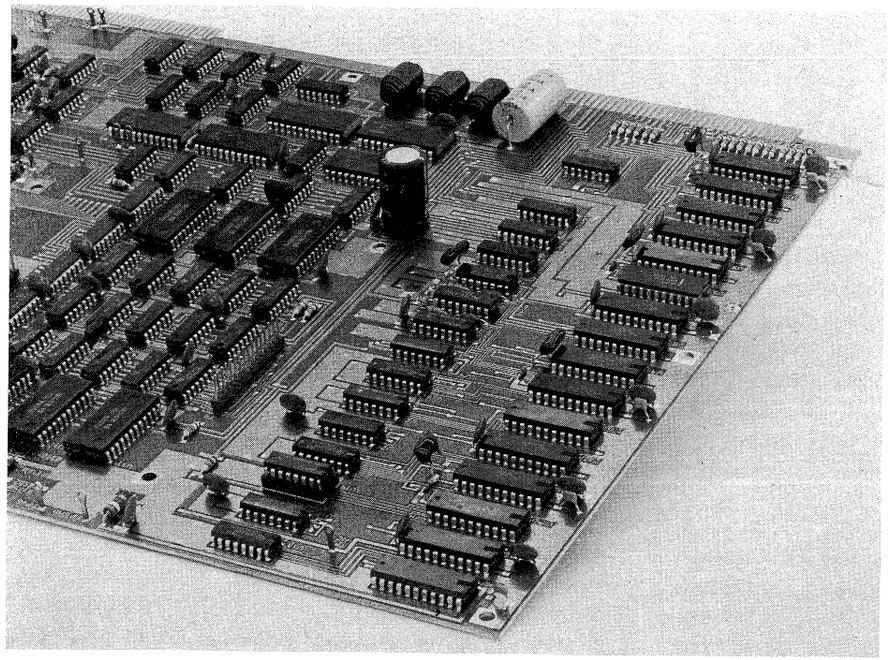


Figure 5-21 Ground noise.

voltage change and the signal wire would only change halfway from 1 to 0. Moving the ground wire more positive has exactly the same effect as moving all other flip-flop signal wires more negative. Figure 5-21 also shows the *difference* between the clock and ground signals. Even though the clock driver is not changing, the flip-flop could be clocked again because its clock input *is* changing with respect to the flip-flop's ground! Actually, the situation is worse than this. While the flip-flop output is changing, a *large* momentary current surge passes through the flip-flop from power supply to ground. This current surge adds to the return current in the ground wire and raises the ground potential even more positive. Worse yet, this same current surge acts to reduce the power supply voltage to the flip-flop. This could cause the flip-flop to forget its state! For any of these malfunctions to occur, the round-trip propagation delay on the power and ground connections must be longer than the flip-flop's minimum clock-pulse width. Unfortunately, this requirement is easily satisfied for TTL and faster-logic families. The power and ground wires are often the longest in the system.

The solution to this problem of power and ground noise is reduction of the impedance of the power and ground connections. The voltage change on the flip-flop's output divides between signal and ground wires in proportion to their impedance. Reducing the impedance of the ground wire will reduce the voltage change appearing on it. Reducing the impedance of the power and ground connections is accomplished in two ways. First, impedance-lowering *capacitors* are added between power and ground connections. Often, as many as one capacitor for each two or three ICs are used. Second, the geometry of the power and ground conductors is made different from that of the signal wires. Many wide conductors are used to distribute power and



A printed circuit board showing wide power and ground conductors and bypass capacitors. (*Hewlett-Packard Co.*)

ground in a grid. Wide power conductors will have a lower impedance than narrower signal conductors. Paralleling many conductors in a grid parallels and reduces their impedance. In extreme cases of very-high-speed logic and signals, the power and ground connections are almost solid sheets of copper called power and ground planes. Four-layer printed circuits, shown in Fig. 5-22, are needed to provide these additional planes.

One other important noise source is common to all digital circuits. Conductors act

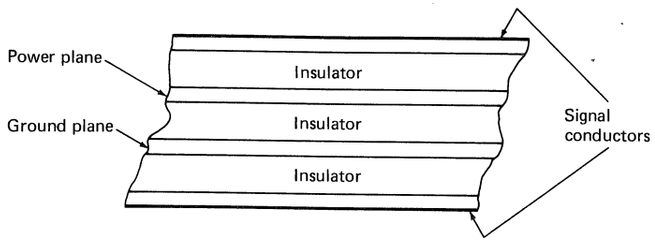
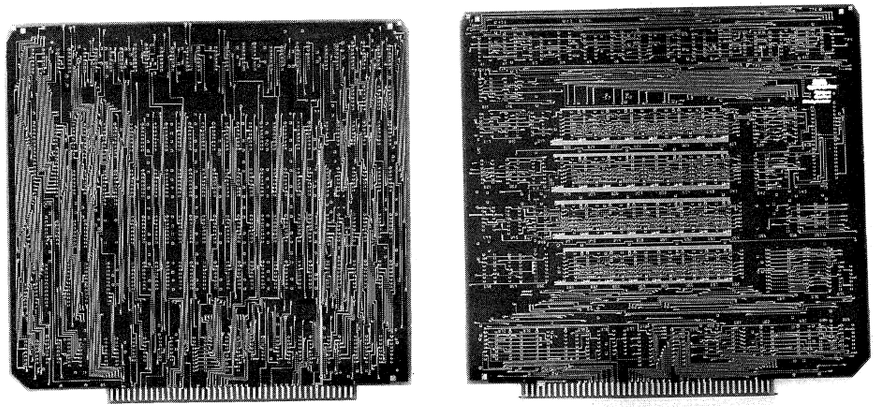


Figure 5-22 Side view of four-layer printed circuit.



Two sides of a printed circuit board showing right angle conductors. (*Hewlett-Packard Co.*)

as both transmitting and receiving antennae. Signal changes can travel between adjacent conductors, even if no physical connection exists! This is called *crosstalk*. Power and ground planes minimize crosstalk. Lowering the impedance of signal wires also minimizes crosstalk. Multilayer printed circuit boards are expensive. Conductor width is limited by available space on the printed circuit. Fortunately, two-sided printed circuits can be designed to have little crosstalk because signal wires that are not changing can act as a ground plane for those that are changing. Conductors at right angles have minimum crosstalk. Conductors on opposite sides of a printed circuit are placed at right angles. Not only is crosstalk between sides minimized, but the conductors on one side act as a ground plane for those on the other side. Long parallel conductors have the worst crosstalk. Ground conductors placed between long parallel conductors will act as a shield to minimize crosstalk.

Since crosstalk is the transmission of signal *changes* between conductors, it must be avoided on clock lines at all cost. Crosstalk on signal wires appears immediately after a clock transition. If the clock speed is sufficiently slow, the crosstalk will have disappeared by the next clock transition. This is another reason we want to operate a circuit at a speed less than the maximum speed.

Other Logic Families

Each logic family has noise problems peculiar to it. Faster logic like *ECL* (emitter-coupled logic) will aggravate the problems already mentioned. Slower-logic families like *CMOS* (complementary metal oxide silicon) have fewer problems. In fact, CMOS is so slow that none of these noise problems usually occurs. The transmission lines in a CMOS circuit may be modeled as capacitors. The effective propagation velocity is slowed as a gate output has to charge up more capacitance from increased wire length or additional gate inputs.

Conservative Design

Many noise calculations must be approximate. For this reason, it is best to design digital circuits conservatively. Conservative noise design may be summarized as follows:

1. Use the slowest-logic family possible.
2. Design to worst-case specifications.
3. Do not operate logic at the limit of its speed.
4. Use short conductors.
5. Use low-impedance signal conductors (as wide as possible).
6. Use even-lower-impedance power and ground conductors (capacitors and wide conductors in a grid).
7. Use ground and power planes if 5 and 6 are inadequate.
8. *Measure* the noise to be sure it is less than the worst-case noise margin.

PROBLEMS

5-1 The ASM shown in Fig. P5-1 uses the components specified in Table P5-1.
(*a*) Calculate the maximum clock skew.
(*b*) Calculate the maximum operating frequency.
(*c*) Calculate the setup and hold time specification for input *YON*.
(*d*) Calculate the propagation delay specifications for outputs *LX* and *HY*.

Table P5-1 Component parameters for problems

	t_{PLH}		t_{PHL}	
	Min	Max	Min	Max
7400	2	22	3	15
7404		22		15
7408		27		19
7410		22		15
7420		22		15
7474	10	25	10	40
74109	4	16	9	28
74LS139		27		38
74163 CO	23	35	23	35
74163 Q	17	25	19	29

	t_S	t_H
7474	20	5
74109	10	6
74163 Data	15	0
74163 Enable	20	0
74163 Load	25	0
74163 Clear	20	0

Note: All times in nanoseconds

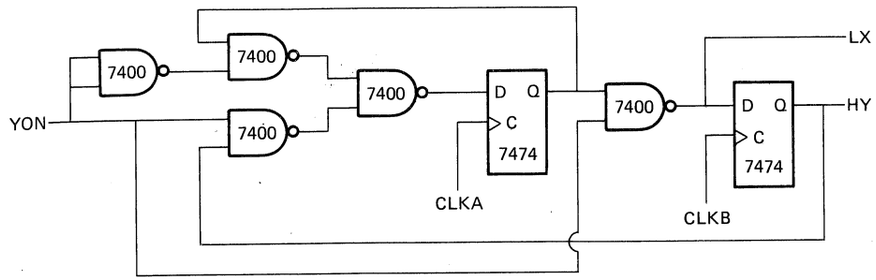


Figure P5-1 ASM for Prob. 5-1.

5-2 The ASM of Fig. P5-2 uses a ROM with a maximum access time of 75 ns and a minimum access time of 25 ns.
(*a*) Calculate the maximum allowable clock skew.
(*b*) Calculate the maximum operating frequency.
(*c*) Calculate the setup and hold times of the inputs.
(*d*) Calculate the propagation delay of the outputs.

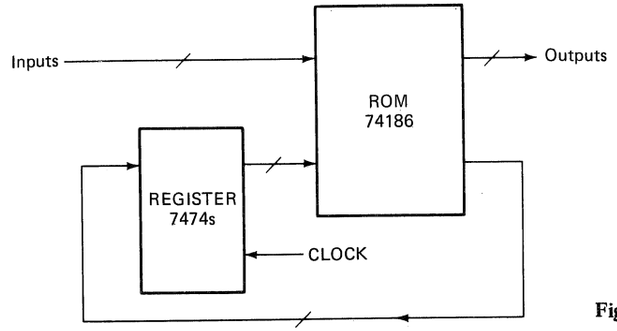


Figure P5-2 ASM for Prob. 5-2.

5-3 The ASM of Fig. P5-3 is the same as the ASM of Fig. P5-2 except that it includes a *synchronizing register*. This register ensures that inputs presented to the ROM will change only at each clock transition, even if these inputs come from sources not synchronized with the system clock. Calculate the setup and hold times for this circuit's inputs.

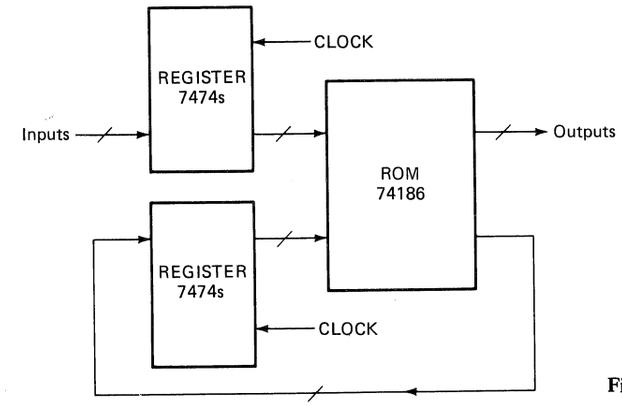


Figure P5-3 ASM for Prob. 5-2.

5-4 Consider the ASM in Fig. P5-4. Its state register is a 3-bit asynchronous counter. The counter stops counting when it reaches 111. It can be started only by external input *LSTRT*. What is the maximum clock frequency at which this circuit will correctly operate? Use the component specifications in Table P5-1.

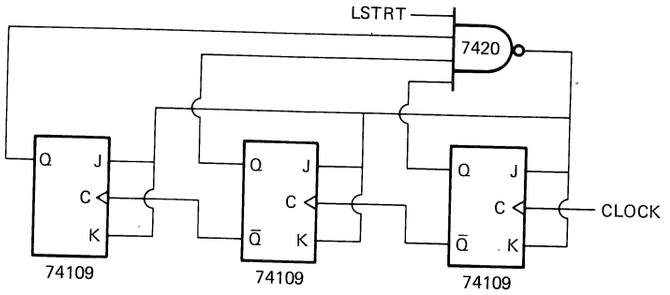


Figure P5-4 Asynchronous counter ASM.

5-5 Consider the grated-clock ASM of Fig. 4-14. Will clock skew between *HCLK* and *GCLK* cause faulty operation? What is the allowable clock skew? What is the maximum skew attributable to the clock drivers? What is the maximum clock rate of this circuit? Assume the NAND gate is a 7410, the AND gate is a 7408, the inverter is a 7404, and the flip-flops are 74109s. The specifications for these components are shown in Table P5-1.

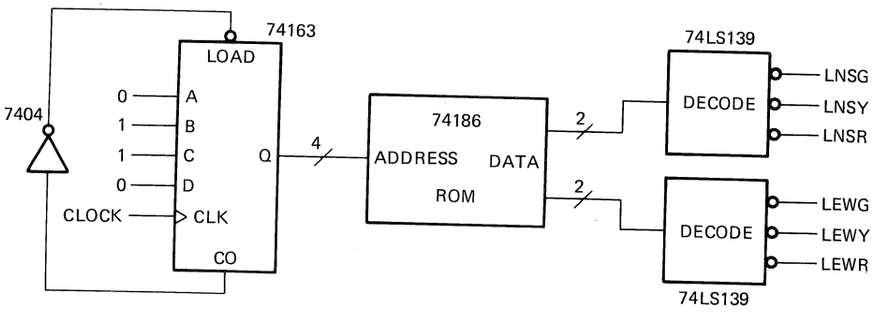


Figure P5-5 Traffic-light controller.

5-6 The circuit of Fig. P5-5 is the simplest traffic light controller. It uses a counter for state register and has no branches in its ASM chart. What is the maximum clock rate of this circuit? Use component parameters in Table P5-1.

5-7 The RWM interface example in this chapter did not use the memory to its full speed capability. Even using an asymmetric clock, the memory write cycle was longer than it might be. Worse yet, the entire ASM was constrained to a 310 ns clock. Design an interface from ASM to RWM that will shorten the write cycle.

Hint: Try dividing the memory cycle into more than one ASM cycle.

CHAPTER

SIX

ASYNCHRONOUS INPUTS AND ASMS

In previous chapters, we have always assumed that external inputs to the ASM will change synchronously with the clock. Inputs were assumed to change just after the clock transition (along with the state variables), which allows sufficient time for the effect of input changes to arrive at state flip-flop inputs before the next clock transition. Inputs will be synchronous if they are outputs of another ASM using the same clock signal. Many times, synchronism will not exist. For example, an input could be generated by another ASM which is timed by a different clock signal. Often, inputs are generated by pushbuttons or other physical events which have no time relationship to any clock signal. An input not synchronized with the ASM clock is called an *asynchronous input*. What problems arise when an asynchronous input is connected to a synchronous ASM? One obvious problem is that any output of the ASM that is a function of *both* the current state *and* an asynchronous input will, itself, be asynchronous. An asynchronous input change will simply propagate through the output ROM or gating. If asynchronous outputs (that change at times other than clock transitions) must be avoided, the ASM chart can have no conditional outputs that depend on asynchronous inputs.

Figure 6-1 shows a conditional output *IHLD* which depends on an asynchronous input *YX* while in state 12. You can see that *IHLD* follows the asynchronous changes of *YX*. *IHLD* is an immediate output; so any changes between clock transitions will cause the *IHLD* function to occur. In the timing diagram of Fig. 6-1, the *IHLD* function would occur twice. The next state would be state 13. The ASM chart of Fig. 6-1 indicates that *IHLD* should occur once and always should be followed by state 14!

Transition Races

Asynchronous inputs can cause a much more serious problem: erroneous next states. A simple ASM is shown in Fig. 6-2. The single input *YI* is an asynchronous input.