Since the decade counter is a very important and useful configuration, many of the "basic 4-bit counters are internally connected to provide a modified count of 10—a mod-10 or decade counter. For instance, the 54/74160 and the 54/74162 are synchronous decade counters that operate in the count-up mode. The 54/74190 and the 54/74192 are also synchronous decade counters but they can operate in either a count-up or count-down mode.

The counters mentioned above are all TTL MSI circuits, and as such we have little control over the internal logic used to implement each counter. Our concern is directed at how each unit can be used in a digital system. Thus we consider each of these counters as a logic block, and our efforts are concentrated on inputs, outputs, and control signals. Even so, the logic block diagram is given for each counter, since a knowledge of the internal logic gives a depth of understanding that is invaluable in practical applications.

Synchronous Up Counters

The pinout and logic diagram for a 54/74163 synchronous 4-bit counter are given in Fig. 10.25. The pinout contains a logic block diagram for this unit. The power requirements are $+V_{CC}$ and GROUND on pins 16 and 8, respectively. The "clock" is applied on pin 2, and you will notice from the diagram that the outputs change states on positive clock transitions (PTs).

The four flip-flop outputs are Q_A, Q_B, Q_C, and Q_D, while the CARRY output on pin 15 can be used to enable successive counter stages (e.g. in a units, tens, hundreds application).

The two ENABLE inputs (P on pin 7 and T on pin 10) are used to control the counter. If either ENABLE input is low, the counter will cease to advance; both of these inputs must be high for the counter to count.

A low level on the $\overline{\text{CLEAR}}$ input will reset all flip-flop outputs low at the very next clock transition, regardless at the levels on the ENABLE inputs. This is called a *synchronous reset* since it occurs at a positive clock transition. On the other hand, note that the 54/74161 has an asynchronous clear, since it occurs immediately when the $\overline{\text{CLEAR}}$ input goes low, regardless of the levels on the CLOCK, ENABLE, or $\overline{\text{LOAD}}$ inputs.

When a low level is applied to the $\overline{\text{LOAD}}$ input, the counter is disabled, and the very next positive clock transition will set the flip-flops to agree with the levels present on the four data inputs (D, C, B, and A). For instance, suppose that the data inputs are $DCBA = 1101$, and the $\overline{\text{LOAD}}$ input is taken low. The very next positive clock transition will load these data into the counter and the outputs will become $Q_DQ_CQ_BQ_A = 1101$. This is a very useful function when it is desired to have the counter begin counting from a predetermined count.

For the counter to count upward in its normal binary count sequence, it is necessary to hold the ENABLE inputs (P and T), the $\overline{\text{LOAD}}$ input, and the CLEAR input all high. Under these conditions, the counter will advance one count for each positive clock transition, progressing from count 0000 up to count 1111 and then repeating the sequence. Since the flip-flops are clocked synchronously, the outputs change states simultaneously and there are no counting spikes or glitches associated with the counter outputs. The state diagram given in Fig. 10.26a shows the normal count sequence, where each box corresponds to one count (or state) and the arrows show how the counter progresses from one state to the next.

The count length can be very easily modified by making use of the synchronous $\overline{\text{CLEAR}}$ input. It is a simple matter to use a NAND gate to decode the maximum count desired, and use the output of this NAND gate to clear the counter synchronously to count 0000. The counter will then count from 0000 up to the maximum desired count and then clear back to 0000. This is the technique that can be used to construct a counter that has any desired modulus.

SN54163, SN74163 Synchronous binary counters

SN54161, SN74161 Synchronous binary counters are similar; however, the CLEAR is asynchronous as shown for the SN54160, 74160 decade counters at left.

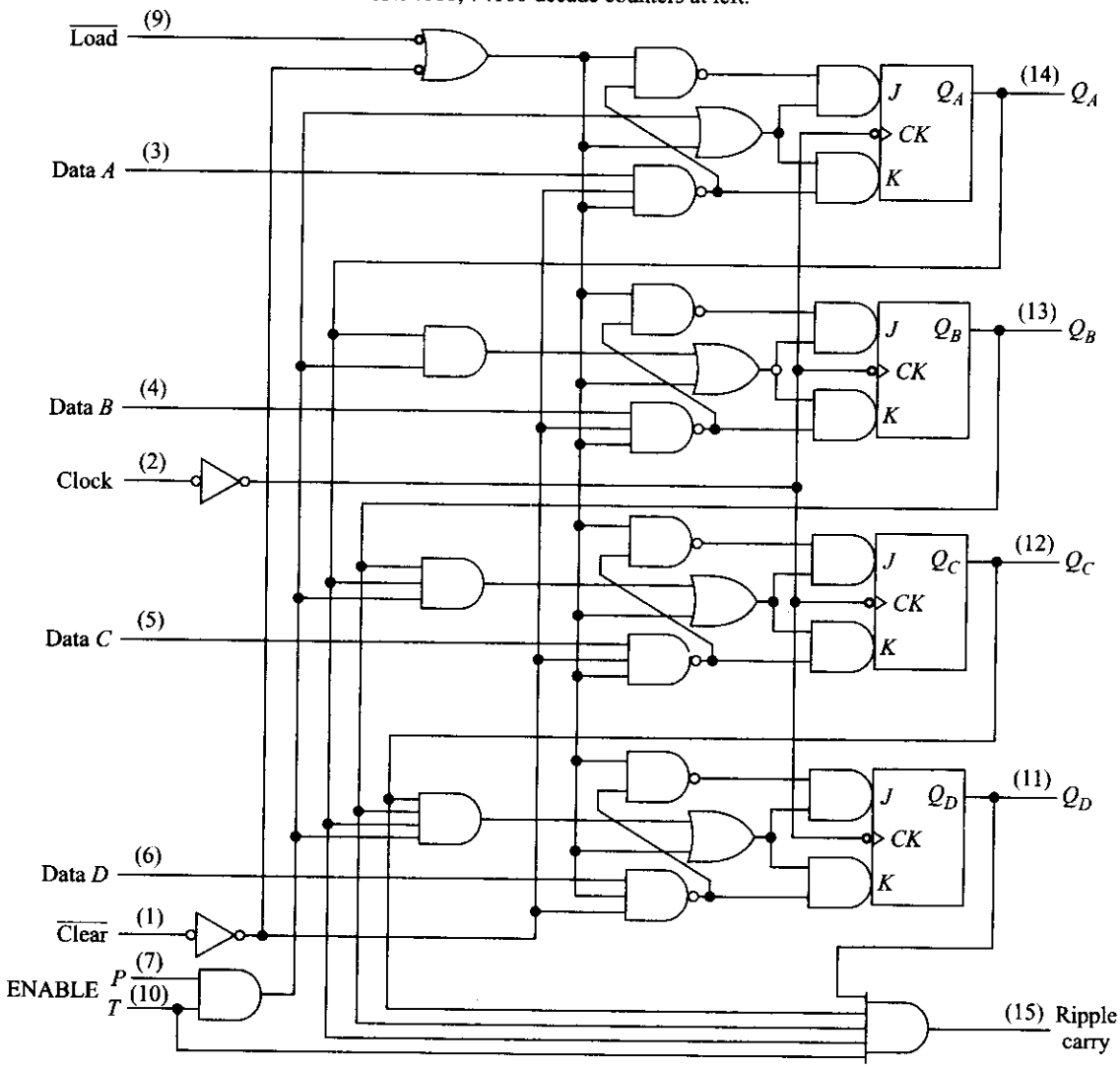
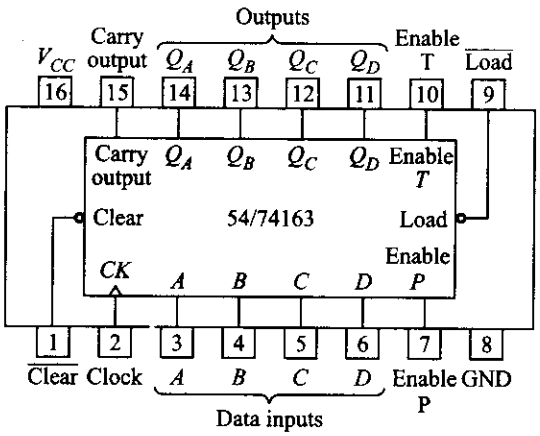


Fig. 10.25 54/74161 and 54/74163

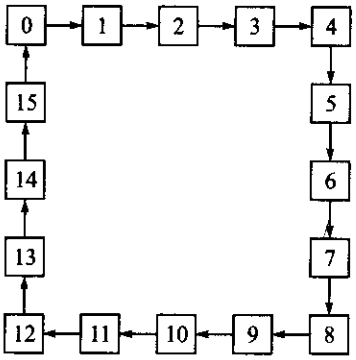
For instance, if a maximum count of 9 is desired, we connect the inputs of the NAND gate to decode count 9 = $DCBA$ = 1001. We then have a mod-10 counter, since the count sequence is from 0000 up to 1001. The NAND gate used to decode count 9 along with the modified state diagram are shown in Fig. 10.26b and c, respectively. Notice that it was necessary to use two inverters to obtain $Q_{\overline{B}}$ and $Q_{\overline{C}}$. The modified

J or *N* dual-in-line or W flat package (Top view)

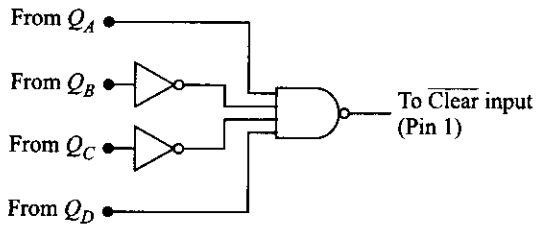


Positive logic: See description.

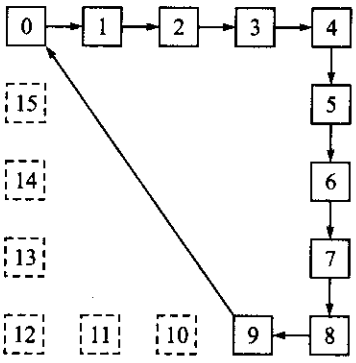
Fig. 10.25 (Continued)



(a) Mod-16 counter state diagram



(b) Gate to decode count 9 (1001)



(c) Modified state diagram for Mod-10 counter

Fig. 10.26

state diagram has solid boxes for states in the modified, mod-10 counter, and dashed boxes for omitted states.

Example 10.12 What are the NAND-gate inputs in Fig. 10.26b if this figure is to be used to construct a mod-12 counter?

Solution The counter must progress from 0000 up to 1011 (decimal 11); the NAND-gate inputs must then be Q_D, Q_C, Q_B, and Q_A.

A set of typical waveforms showing the clear, preset, count, and inhibit operations for a 54/74163 (and 54/74161) is given in Fig. 10.27. You should take time to study them carefully until you understand exactly how these four operations are controlled.

SN54161, SN54163, SN74161, SN74163 Synchronous binary counters

Typical clear, preset, count, and inhibit sequences

Illustrated below is the following sequence.

1. Clear outputs to zero.
2. Preset to binary twelve.
3. Count to thirteen, fourteen, fifteen, zero, one, and two.
4. Inhibit.

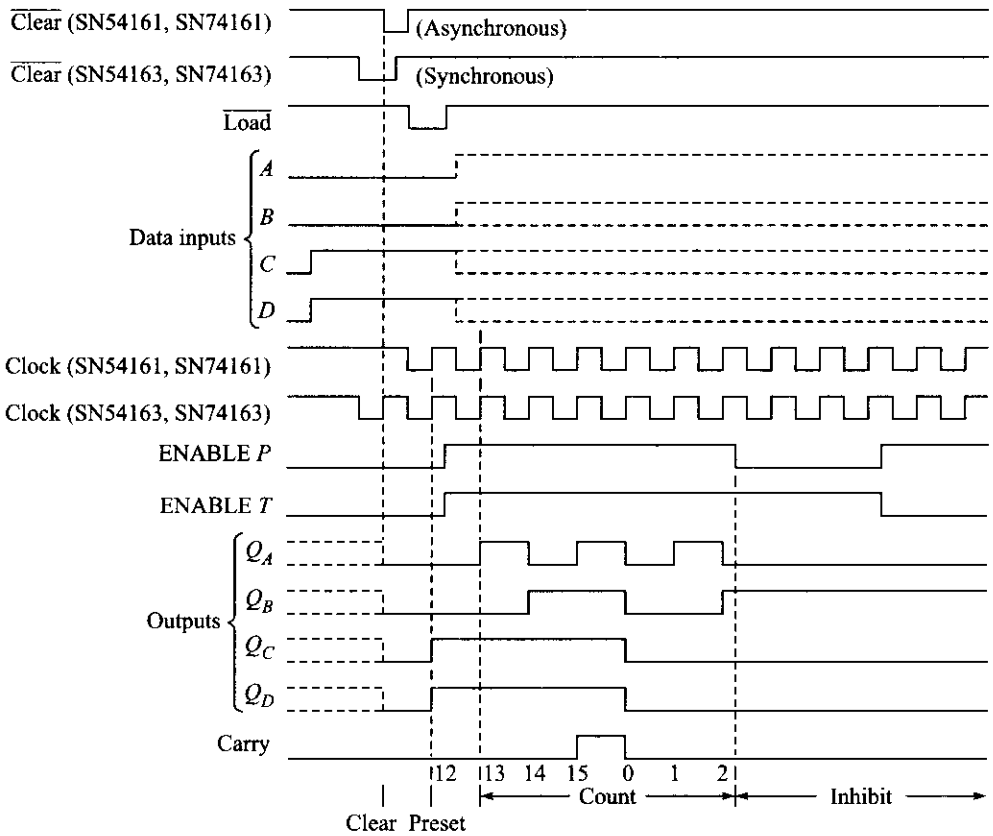
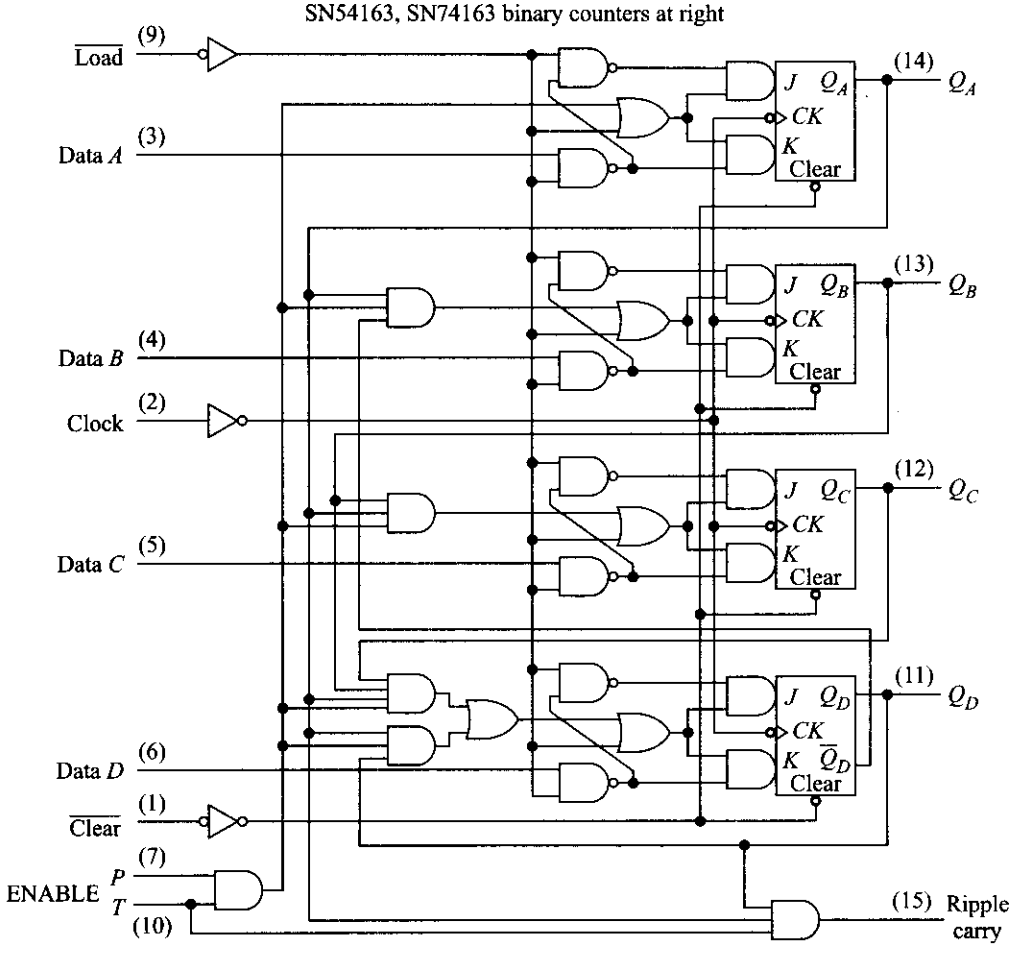


Fig. 10.27

The logic diagram and a typical set of waveforms for the 54/74160 and the 54/74162 are given in Fig. 10.28. (The pinout is identical for the previously given 54/74163.) These two counters have been modified internally and are decade counters. Other than that, the input, output, and control lines for these two counters are identical with the previously discussed 54/74163 and 54/74161. These counters advance one count with each positive clock transition, progressing from 0000 to 1001 and back to 0000. The state diagram for these two units would appear exactly as shown in Fig. 10.26c; this is the state diagram for a mod-10 or decade counts.

SN54160, SN74160 Synchronous decade counters

SN54162, SN74162 Synchronous decade counters are similar; however, the clear is synchronous as shown for the SN54163, SN74163 binary counters at right



(a)

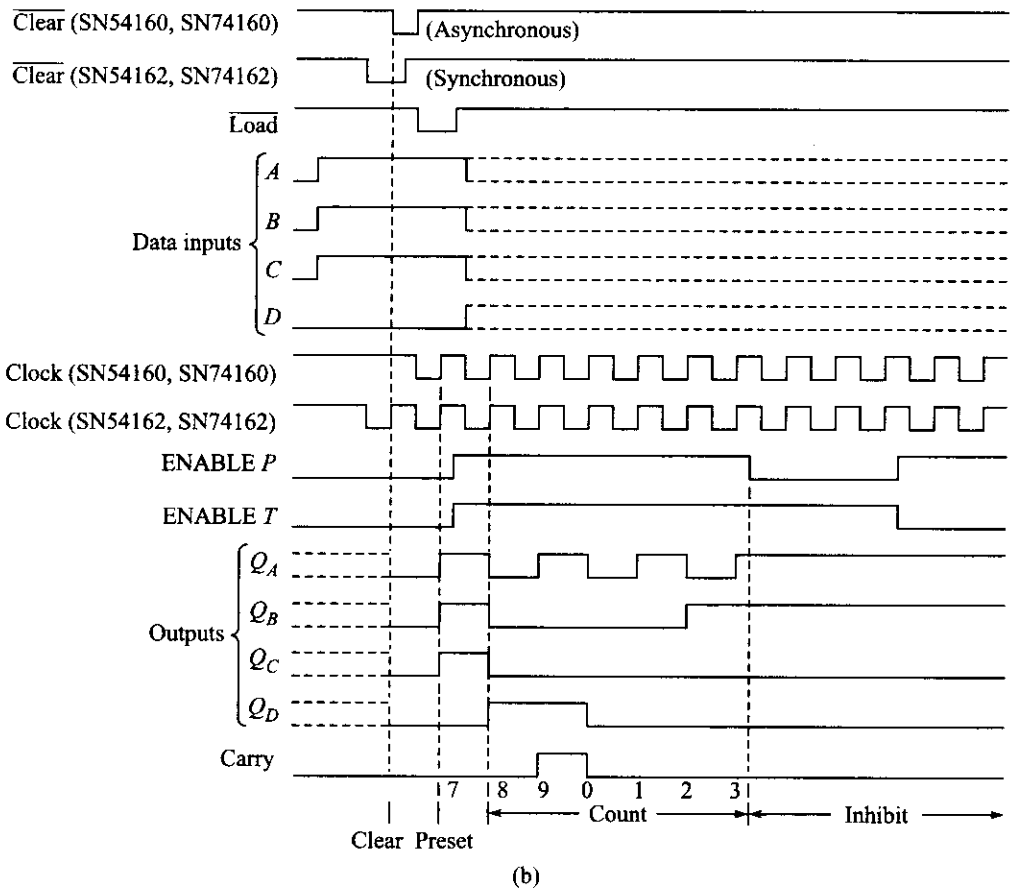
Fig. 10.28 54/74160 (continued on next page)

SN54160, SN54162, SN74160, SN74162 Synchronous binary counters

Typical clear, preset, count, and inhibit sequences

Illustrated below is the following sequence.

1. Clear outputs to zero.
2. Preset to BCD seven.
3. Count to eight, nine zero, one, two, and three.
4. Inhibit.



(b)

Fig. 10.28 (Continued)

Synchronous Up-Down Counters

The 54/74193 is a 4-bit synchronous up-down binary counter. It has a master reset input and can be reset to any desired count with the parallel load inputs. The logic symbol for this TTL MSI is shown in Fig. 10.29a. Pin $\overline{PL}$ is a control input for loading data into pins P_A, P_B, P_C, and P_D. When the device is used as a counter, these four pins are left open and $\overline{PL}$ must be held high. Pin *MR* is the master reset, and it is normally held low. (A high level on *MR* will reset all flip-flops.)

Outputs TC_U and TC_D are to be used to drive the following units, such as in a cascade arrangement. The clock inputs are CP_U and CP_D. Placing the clock on CP_U will cause the counter to count up, and placing the clock on CP_D will cause the counter to count down. Notice that the clock should be connected to either CP_U or CP_D, but not both, and the unused input should be held high. The outputs of the counter are Q_A, Q_B, Q_C and Q_D.

A state diagram is a simple drawing which shows the stable states of the counter, as well as how the counter progresses from one count to the next. The state diagram for the 54/74193 is shown in Fig. 10.29b. Each box represents a stable state, and the arrows indicate the count sequence for both count-up and count-down operations. This is a 4-bit counter, and clearly there are 16 stable states, numbered 0, 1, 2, ..., 15.

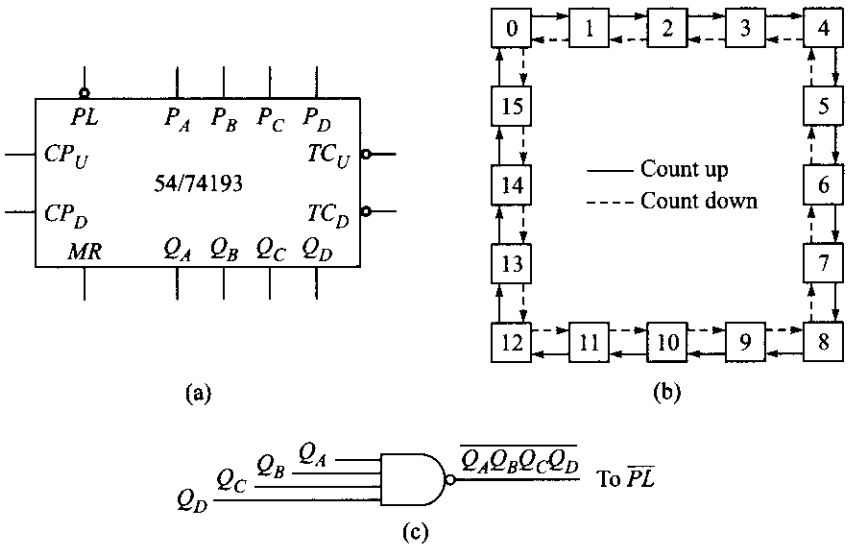


Fig. 10.29 4-bit binary counter (presettable)

The 54/74193 has a parallel-data-entry capability which permits the counter to be preset to the number present on the parallel-data-entry inputs (P_A, P_B, P_C, and P_D). Whenever the parallel load input ($\overline{PL}$) is low, the data present at these four inputs is shifted into the counter; that is, the counter is preset to the number held by $P_DP_CP_BP_A$.

Now, here is another technique for modifying the count. Simply use a NAND gate to detect any of the stable states, say, state 15 (1111), and use this gate output to take $\overline{PL}$ low. The only time $\overline{PL}$ will be low is when Q_D, Q_C, Q_B, and Q_A are all high, or state 15(1111). At this time, the counter will be preset to the data $P_DP_CP_BP_A$.

For example, suppose that $P_DP_CP_BP_A = 1001$ (the number 9). When the clock is applied, the counter will progress naturally to count 15(1111). At this time, $\overline{PL}$ will go low and the number 9 (1001) will be shifted into the counter. The counter will then progress through states 9, 10, 11, 12, 13, and 14, and at count 15 it will again be preset to 9.

The count sequence is easily shown by the state diagram in Fig. 10.30 on the next page. Notice that count 15 (1111) is no longer a stable state; it is the short time during which the counter is preset. The stable states in this example are 9, 10, 11, 12, 13, and 14. This is, then, a mod-6 counter. Notice that this technique is

asynchronous since the preset action is not in synchronism with the clock. Therefore, you should be aware that counting spikes or glitches may be associated with the outputs of this presetting arrangement.

Example 10.13 Suppose that the counter just discussed is still preset to 1001 (the number 9) but the clock is applied to count down rather than count up. What are the counting states? What is the modulus?

Solution The counter will count down to 15, then preset back to 9, and repeat. The resulting state diagram is given in Fig. 10.31. The modulus is clearly 10.

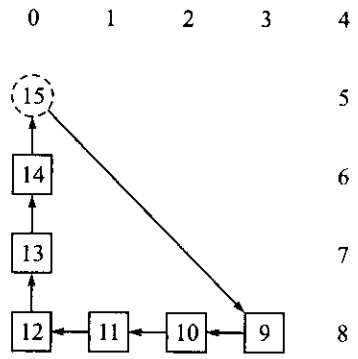


Fig. 10.30

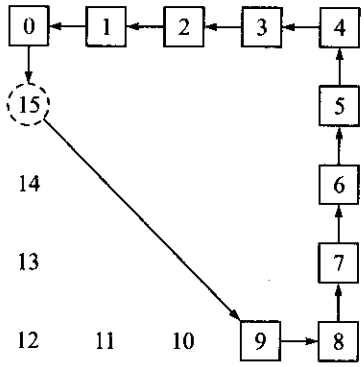


Fig. 10.31

SELF-TEST

12. Name two popular synchronous binary counters.
13. What is the difference between the 74161 binary counter and the 74191 binary counter?
14. What is the modulus of the 74160 counter?
15. Can a 74160 counter be used to count down?

10.7 COUNTER DESIGN AS A SYNTHESIS PROBLEM

Section 8.11 of Chapter 8 presents a systematic approach towards sequential logic circuit design using FSM concept. In this section, we consider counter as a state machine and discuss counter design steps through an example.

Let us try to design a modulo-6 counter, the counting states (memory values) of which are shown in state transition diagram of Fig. 10.32. We need three memory elements or flip-flops for this as with n flip-flop we can get at most 2^n number of different counting states.

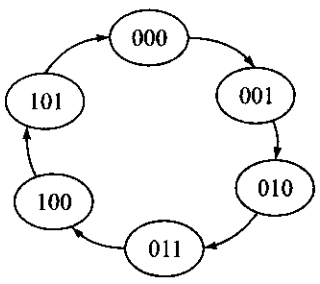


Fig. 10.32 **State sequence of a modulo-6 counter**

Now with three flip-flop, 8 different states are possible but in our design states 110 and 111 are not used in the counting

sequence. To start with we shall assume the counter is always initialized with one of the valid states and not 110 or 111. We decide to use three *JK* flip-flops labeled *A*, *B* and *C* as memory element for this design.

The next step to be taken is to form a state synthesis table as shown in Table 10.1. In this, the first column represents current state of the counter and second column, as shown in the next state of the counter state transition diagram. We fill up next three columns using excitation table of *JK* flip-flop given in Fig. 8.34 of Chapter 8. Excitation table gives inputs need to be present when clock triggers a certain $Q_n \rightarrow Q_{n+1}$ transition of the flip-flop. In the first row, we see both *C* and *B* make transition $0 \rightarrow 0$ and hence corresponding *JK* inputs should be 0× from excitation table. For flip-flop *A*, transition is $0 \rightarrow 1$ and input should be 1× . This is continued to fill up other five rows of input columns for three flip-flops.

Table 10.1 **State Table for Design of Modulo-6 Counter Given in Fig. 10.32**

C_n	B_n	A_n	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	0	0	1	×	1	0	×	×	1

Our next objective is to get logic equation for each flip-flop input as a function of present state of the counter. We use Karnaugh Map for this as shown in Fig. 10.33. Note that values corresponding to unused states 110 and 111 appear as don't care '×'. We have not shown Karnaugh Map for J_A and K_A as it is obvious from Table 10.1 that $J_A = K_A = 1$.

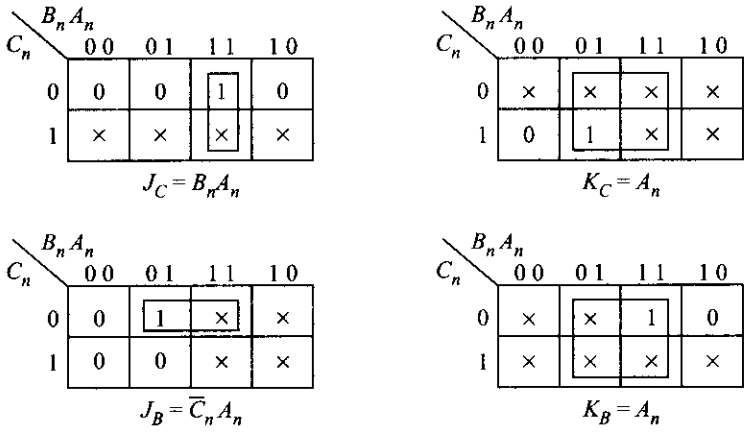


Fig. 10.33 **Derivation of design equations from Karnaugh Map**

The final step is to draw the circuit diagram from these design equations, which is shown in Fig. 10.34. The decoding output is obtained from a three input AND gate which goes high every time the counter goes to a valid state *CBA* = 000 and that occurs in every 6th clock cycle.

Note that the method we have explained is a general one and can be used to design counter of any modulo number and that can follow any given counting sequence. An *irregular counter* is the one which

does not follow any regular binary sequence but has N number of distinct states and thus qualifies as a modulo-N counter. In Example 10.15, we present a modulo-4 irregular counter.

One question can be raised at this point for the above circuit. What happens if the circuit for any reason goes to one of the unused state? Does it come back to any of the valid counting state or in the worst case gets locked as shown in Fig. 10.35a? Initializing the designed circuit with 110 or 111 unused state we find that they get back to counting sequence as shown in Fig. 10.35b. However, a designer may not leave unused states to chances and want them to follow certain course if the circuit accidentally enters into one of them. Example 10.14 shows how to handle unused states in a counter design problem.

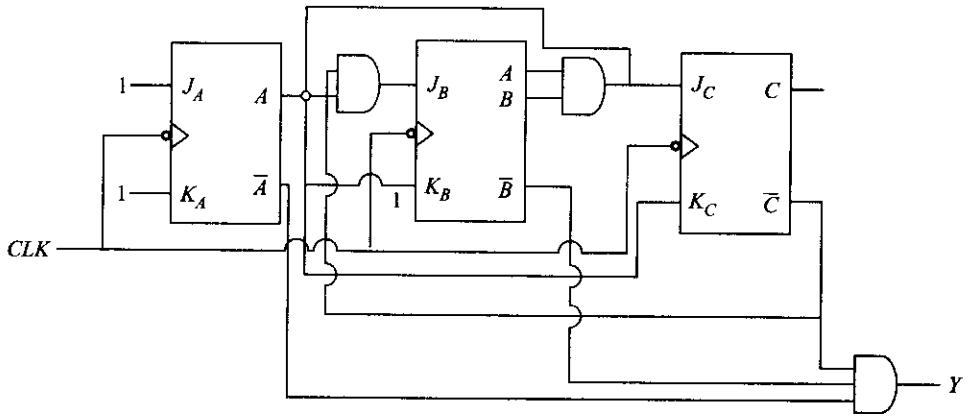


Fig. 10.34 **Circuit diagram of modulo-6 synchronous counter described in Fig. 10.32**

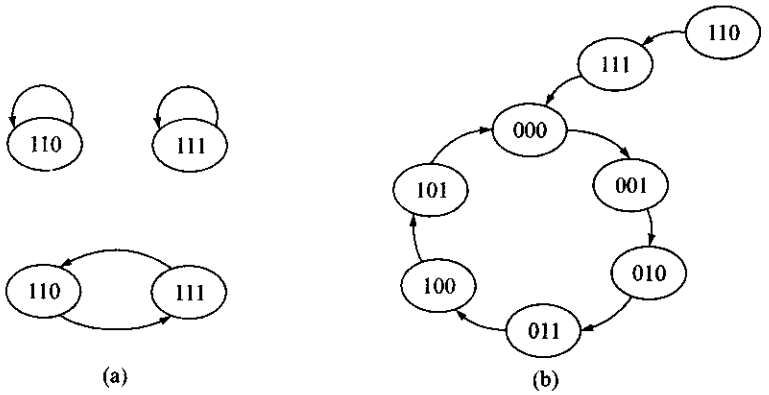


Fig. 10.35 **(a) Lock-in conditions, (b) Full state transition diagram for circuit in Fig. 10.34**

Example 10.14 Design a *self-correcting* modulo-6 counter as described in Fig. 10.32 in which all the unused state leads to state $CBA = 000$.

Solution For this we have to add two more rows as given next for two unused states in state Table 10.1.

C_n	B_n	A_n	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_C	K_C
1	1	0	0	0	0	×	1	×	1	0	×
1	1	1	0	0	0	×	1	×	1	×	1

Accordingly, Karnaugh Map giving design equations changes to as given in Fig. 10.36.

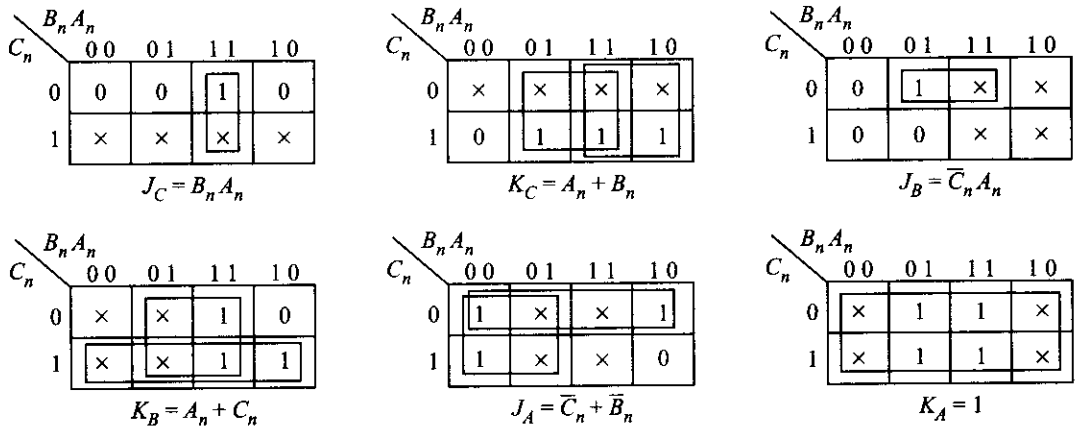


Fig. 10.36 Design equations for Example 10.14

Note the difference between Fig. 10.33 and 10.36. Unused states 110 and 111 can no longer be considered as don't care. This type of design is called self-correcting as the circuit comes out on its own from an invalid state to a valid counting state sequence. The final circuit diagram from design equations are shown in Fig. 10.37.

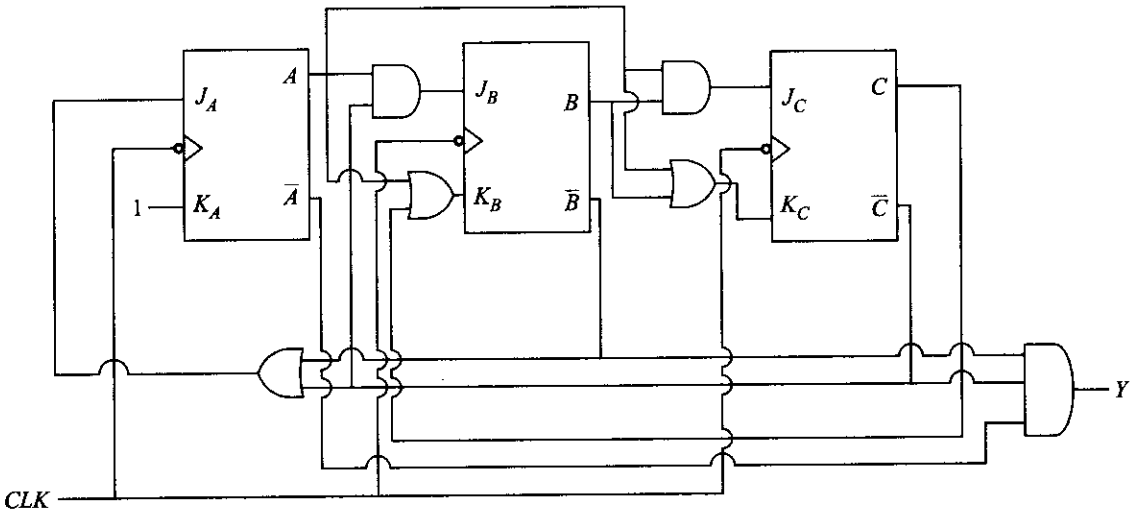
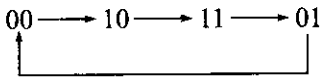


Fig. 10.37 Circuit diagram for Example 10.14

Example 10.15 Design a modulo-4 irregular counter with following counting sequence using *D* flip-flop.



Solution Using state excitation table of *D* flip-flop (Fig. 8.34), the state table can be formed as shown in Table 10.2.

Table 10.2 State Table for Design of Irregular Counter

B_n	A_n	B_{n+1}	A_{n+1}	D_B	D_A
0	0	1	0	1	0
0	1	0	0	0	0
1	0	1	1	1	1
1	1	0	1	0	1

Design equations from Karnaugh Map can be derived as shown in Fig. 10.38(a), and corresponding logic circuit is shown in Fig. 10.38(b).

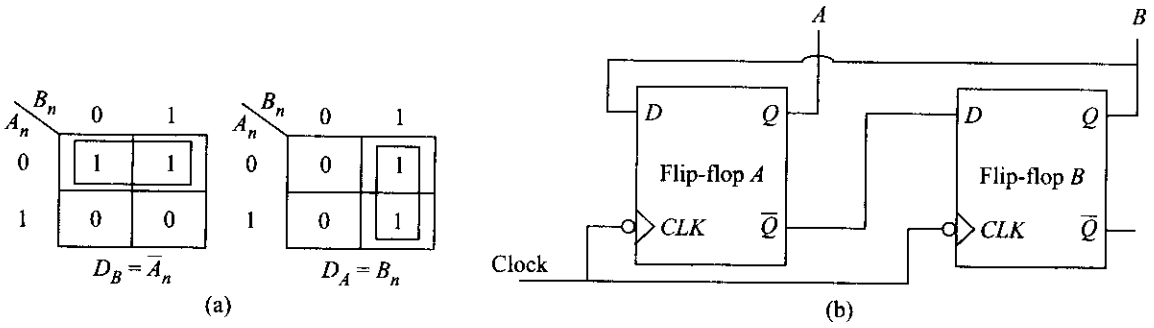


Fig. 10.38 (a) Deriving design equations for Example 10.15, (b) Circuit diagram

Example 10.16 Show how a modulo-4 counter designed with two flip-flops can generate a repetitive sequence of binary word '1101' with minimum number of memory elements?

Solution Let the counting sequence of two flip-flops B and A be 00 → 01 → 10 → 11 → 00 ..., i.e. a modulo-4 synchronous up counter. The corresponding output is 1 → 1 → 0 → 1 → 1.... As shown in Fig. 10.39(a) the sequence '1101' will be generated repetitively by Y. Figure 10.39(b) gives Karnaugh Map representation of Y and we get $Y = A + B'$. A standard modulo-4 up counter and an 2-input OR gate connected as shown in Fig. 10.39(c) generates the given sequence.

Note that for N-bit sequence generator we need modulo-N counter. Modulo-N synchronous counter requires m number of flip-flops where m is the lowest integer for which $2^m \geq N$. The design procedure remains the same as discussed in Example 10.16. Output Y now is a function of m state variables representing m memory elements.

Compare this design with shift register based sequence generator design discussed in Chapter 9 that requires N number of memory elements for N-bit sequence generator. Though shift register based design does not require any combinatorial circuit to generate output logic the overall hardware cost is more and it is more pronounced for large N.

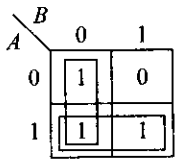
A similar design for sequence detector circuit with minimum number of flip-flops is discussed in Chapter 11.

SELF-TEST

16. What is lock-out of a counter?
17. For 48-bit sequence generator what is the minimum number of memory elements required?

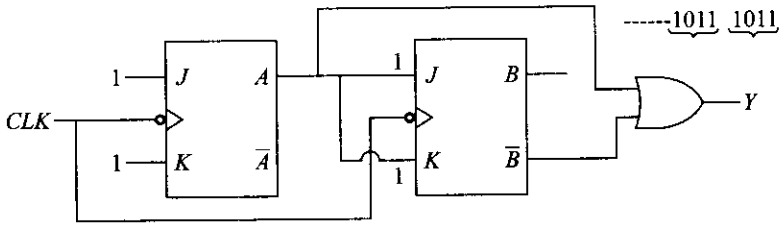
A	B	Y
0	0	1
0	1	1
1	0	0
1	1	1

(a)



$Y = A + \overline{B}$

(b)



(c)

Fig. 10.39 **Sequence generator circuit using synchronous counter, (a) State Table, (b) Output equation, (c) Circuit diagram**

10.8 A DIGITAL CLOCK

A very interesting application of counters and decoding arises in the design of a digital clock. Suppose that we want to construct an ordinary clock which will display hours, minutes, and seconds. The power supply for this system is the usual 60-Hz 120-Vac commercial power. Since the 60-Hz frequency of most power systems is very closely controlled, it is possible to use this signal as the basic clock frequency for our system. Note that in several countries commercial power supply is 50-Hz and not 60-Hz. There one can use standard variable frequency signal generator, set at 60-Hz, as input.

In order to obtain pulses occurring at a rate of one each second, it is necessary to divide the 60-Hz power source by 60. If the resulting 1-Hz waveform is again divided by 60, a one-per-minute waveform is the result. Dividing this signal by 60 then provides a one-per-hour waveform. This, then, is the basic idea to be used in forming a digital clock.

A block diagram showing the functions to be performed is given in Fig. 10.40. The first divide-by-60 counter simply divides the 60-Hz power signal down to a 1-Hz square wave. The second divide-by-60 counter changes state once each second and has 60 discrete states. It can, therefore, be decoded to provide signals to display seconds. This counter is then referred to as the seconds counter.

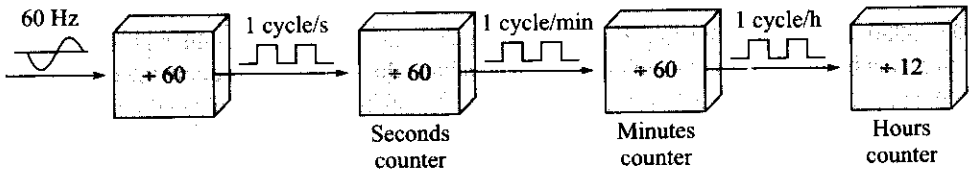


Fig. 10.40 **Block diagram of digital clock**

The third divide-by-60 counter changes state once each minute and has 60 discrete states. It can thus be decoded to provide the necessary signals to display minutes. This counter is then the minutes counter.

The last counter changes state once each 60 minutes (once each hour). Thus, if it is a divide-by-12 counter, it will have 12 states that can be decoded to provide signals to display the correct hour. This, then, is the hours counter.

As you know, there are a number of ways to implement a counter. What is desired here is to design the counters in such a way as to minimize the hardware required. The first counter must divide by 60, and it need not be decoded. Therefore, it should be constructed in the easiest manner with the minimum number of flip-flops.

For instance, the divide-by-60 counter could be implemented by cascading counters ($12 \times 5 = 60$, or $10 \times 6 = 60$, etc.). The TTL MSI 7490 decade counter can be used as a divide-by-10 counter, and the TTL MSI 7492 can be used as a divide-by-6 counter. Cascading these two will provide a divide-by-60 counter as shown in Fig. 10.41. The amplifier at the input provides a 60-Hz square wave of the proper amplitude to drive the 7490. The 7492 is connected as a divide-by-12 counter, but only outputs Q_A, Q_B, and Q_C are used. In this fashion, the 7492 operates essentially as a divide-by-6 counter.

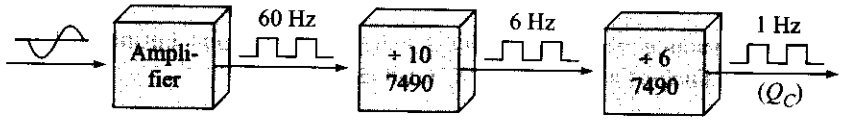


Fig. 10.41 **Divide-by-60 counter**

The seconds counter in the system also divides by 60 and could be implemented in the same way. However, the seconds counter must be decoded. We are interested in decoding this counter to represent each of the 60 s in 1 min. This can most easily be accomplished by constructing a mod-10 counter in series with a mod-6 counter for the divide-by-60 counter. The mod-10 counter can then be decoded to represent the units digit of seconds, and the mod-6 counter can be decoded to represent the tens digits of seconds.

Since both the 7490 and the 7492 count in straight 8421 binary, a 7447 decoder-driver can be used with each to drive two 7-segment indicators, as shown in Fig. 10.42. Notice that the 7492 is connected as a divide-by-12 counter, but only outputs Q_A, Q_B, and Q_C are used to drive the 7447 decoder-driver.

The minutes counter is exactly the same as the seconds counter, except that it is driven by the one-per-minute square wave from the output of the seconds counter, and its output is a one-per-hour square wave, as shown in Fig. 10.42.

The divide-by-12 hours counter must be decoded into 12 states to display hours. This can be accomplished by connecting a mod-10 (54/74160) decade counter in series with a single flip-flop E as shown in Fig. 10.43. This forms a divide-by-20 ($10 \times 2 = 20$) counter. Feedback is then used to form a mod-12 counter.

The hours counter must count through states 00, 01, 02, ... , 11, and then back to 00. The NAND gate in Fig. 10.43 will go low as the counter progresses from count 11 to count 12, and this will immediately clear the 74160 to 0000 and reset the flip-flop E to 0. The counter actually skips from count 11 to count 00 omitting the eight counts in between. This is the mod-12 hours counter; the 74160 will provide the units of hours while the flip-flop will provide the tens of hours. Notice that the 74160 is reset asynchronously and there might then be glitches at the outputs of the decoding gates. However, this is one case where these glitches will have no effect, since they are too narrow to cause a visible indication on the light emitting diodes (LEDs).

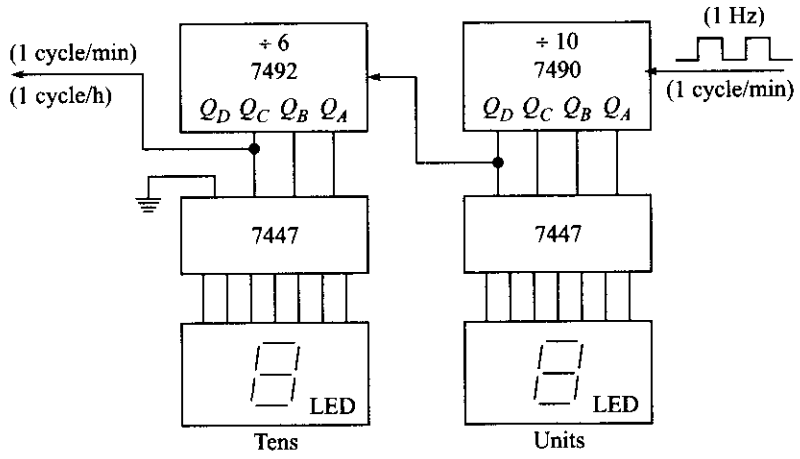


Fig. 10.42 **A 10 × 6 mod-60 counter with units and tens decoding**

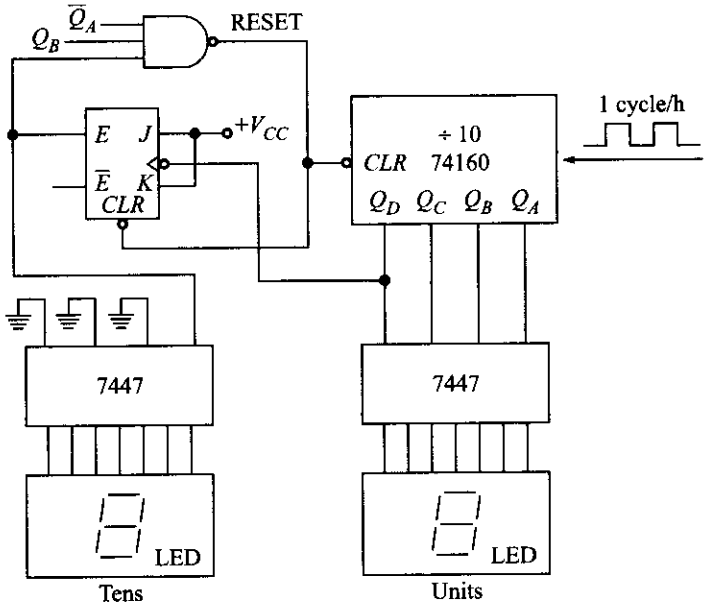


Fig. 10.43 **Mod-12 hours counter**

Finally, some means must be found to set the clock because the flip-flops will assume random states when the power is turned off and then turned back on again. Setting the clock can be quite easily accomplished by means of the SET push-buttons shown in Fig. 10.44. Depressing the SET HOURS button causes the hours counter to advance at a one-count-per-second rate, and thus this counter can be set to the desired hour. The minutes counter can be similarly set by depression of the SET MINUTES button.

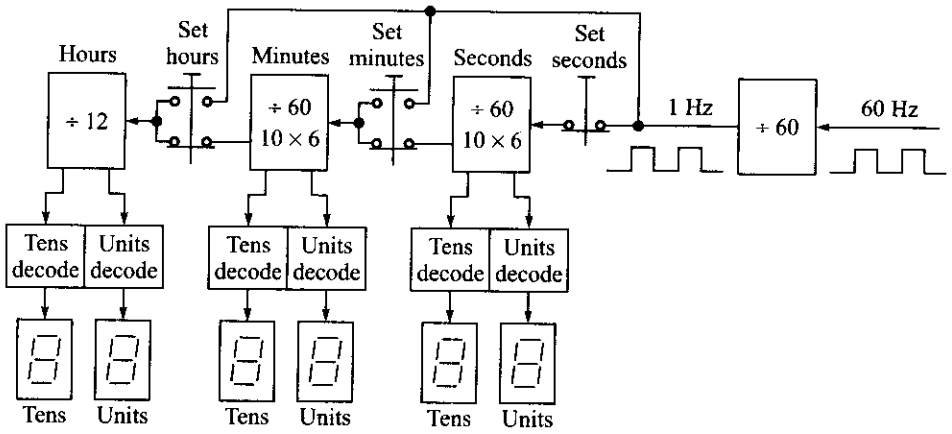


Fig. 10.44 **Digital clock**

Depression of the SET SECONDS button removes the signal from the seconds counter, and the clock can thus be brought into synchronization.

By means of large-scale integration (LSI), it is possible to construct a digital clock entirely on one semiconductor chip. Such units are commercially available, and they perform essentially the function shown in the logic diagram in Fig. 10.43 (the seven-segment indicators are, of course, separate). The National Semiconductor 5318 is one such commercially available LSI digital clock. It is available in a 24-pin dual in-line (DIP) package measuring 0.54 × 1.25 in.

10.9 COUNTER DESIGN USING HDL

Counter design in HDL is straight forward if one uses arithmetic operator + and – that corresponds to binary addition and subtraction respectively. We show a modulo-8 up counter design in the example given in first column. It is left to the compiler to decide which flip-flop is to be used. If one wants to ensure use of a particular type of flip-flop say, *JK* then the code should be written in a manner shown in second column for modulo-3 up counter shown in Fig. 10.16a.

```
module UC(Clock, Reset,Q);
input Clock, Reset;
output [2:0] Q;
  //modulo 8 requires 3 flip-flop
reg [2:0] Q;
always @ (negedge Clock
   or negedge Reset)
  if (~Reset) Q=3'b0;
  else Q = Q+1;
endmodule
```

```
module UCJK(A,B,Clock,Reset);
input Clock,Reset;
output A,B; //modulo-3 requires 2 flip-flop
wire JA,JB,KA,KB;
assign JA=~B;

assign KA=1'b1;
assign JB=A;
assign KB=1'b1;
JKFF JK1(A,JA,KA,Clock,Reset); //instantiates JKFF
JKFF JK2(B,JB,KB,Clock,Reset); //instantiates JKFF
endmodule
```

```
module JKFF(Q,J,K,Clock,Reset);
input J,K,Clock,Reset;
 output Q;
 reg Q;
always @ (negedge Clock or negedge Reset)
   if(~Reset) Q=1'b0;
   else Q <=  (J&~Q) | (~K&Q);
endmodule
```

Example 10.17 Design a modulo-8 up down counter which counts in upward direction if input MODE = 0, else counts in downward direction. It should also have a parallel load facility. When *PL* = 1, a 3-bit number *D* is asynchronously loaded to the counter. The counter counts at the negative edge of CLOCK and its output is represented by *Q*.

Solution The Verilog HDL code for the problem is given below. We have used a new keyword integer to hold a value temporarily. This helps us in writing both up and down count in one single statement that responds to clock within always block.

```
module UDCPL(CLOCK,PL,MODE,D,Q);    //Up Down Counter
input CLOCK,PL,MODE;                 //with parallel load
input [2:0] D;
output  [2:0] Q; //modulo 8 requires 3 flip-flop
reg [2:0] Q;
integer updown; //updown will be +1 or -1 depending on MODE
always @ (negedge CLOCK)
    begin
      if (MODE) updown=-1; //If MODE=1, counts downward
      else updown=1;
      if (PL) Q=D;  //If PL=1, parallel loading takes place
      else Q=Q+updown;  //Last else statement responds to clock
    end
Endmodule
```

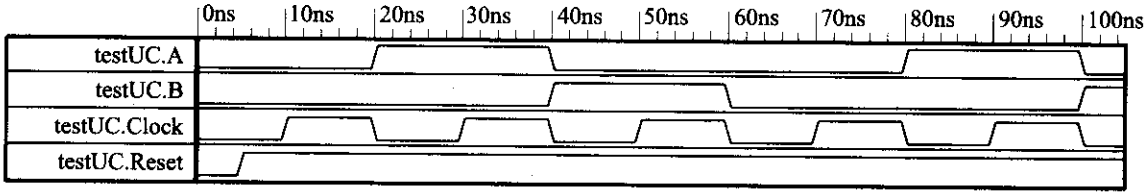
Example 10.18 The code in the first column when executed with modulo-3 *JK* counter, UCJK described in this section generates monitor output as shown in column 2 and timing diagram as shown below. Show how the test bench verifies Verilog code UCJK is that of a modulo-3 counter.

```
module testUC();
 reg Clock,Reset;
 wire A,B;
 UCJK M3(A,B,Clock,Reset); /*instantiates
   modulo-3 JK FF  counter*/
 initial
     begin
```

```
 0 Clock= 0, A=0 B=0
10 Clock= 1, A=0 B=0
20 Clock= 0, A=1 B=0
30 Clock= 1, A=1 B=0
40 Clock= 0, A=0 B=1
50 Clock= 1, A=0 B=1
```

```
    Clock = 0; // initial value of clock
    Reset = 0; //initial value of Reset=0
    #5 Reset =1; //Reset becomes 1 after 5 second
    #100 $finish; // Terminate simulation after 100ns
  end
 always
    begin
    #10 Clock = ~Clock;  //Clock toggles every 10 ns
    end
 initial
   begin
   $monitor($time, " Clock= %b,A= %b  B=%b\n",
     Clock,A,B);
   end
endmodule
```

```
 60 Clock= 0, A=0 B=0
 70 Clock= 1, A=0 B=0
 80 Clock= 0, A=1 B=0
 90 Clock= 1, A=1 B=0
100 Clock= 0, A=0 B=1
```



Solution The test bench, given in the code above runs the simulation for 5 + 100 = 105ns duration. At every 10 ns clock toggles giving a 10 + 10 = 20ns clock cycle time. So we have 5 negative edges of the clock (1 to 0 transition) from start at 20, 40, 60, 80 and 100ns. If we look at the timing diagram and monitor output we see *JK* flip flop output changes value at negative edges as *BA* = 00 (initially reset by 'Reset'), 01, 10, 00, 01, 10 etc. These show that three counting states 00, 01, 10 get repeated. Hence, the code of module UCJK behaves like a modulo-3 counter.

PROBLEM SOLVING WITH MULTIPLE METHODS

Problem Design a self correcting modulo-3 down counter.

Solution We need 2 flip-flops, say *B* and *A* for this purpose which has 4 states. Let the down counter count like 10 → 01 → 00 → 10... and undesired state 11 corrects itself to 10. The excitation table of Fig. 8.35 is used for the design purpose.

In Method-1, we use *SR* flip-flop for design purpose. Figure 10.45a shows the state table and in the second column, necessary inputs for the two *SR* flip-flops are given. Figure 10.45b shows the use of Karnaugh Map to get the design equations.

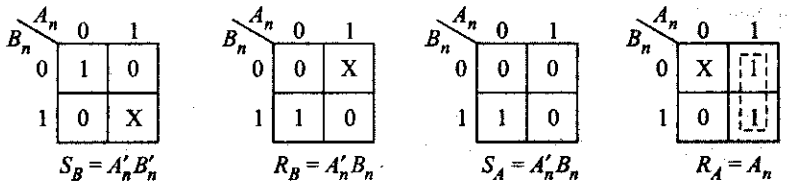
In Method-2, we use *JK* flip-flop for design purpose. The fourth column shows necessary inputs for the two *JK* flip-flops. Figure 10.45c shows the use of Karnaugh Map to get the design equations.

In Method-3, we use *D* flip-flop for design purpose. The third column shows necessary inputs for the two *D* flip-flops. Fig. 10.45d shows the use of Karnaugh Map to get the design equations.

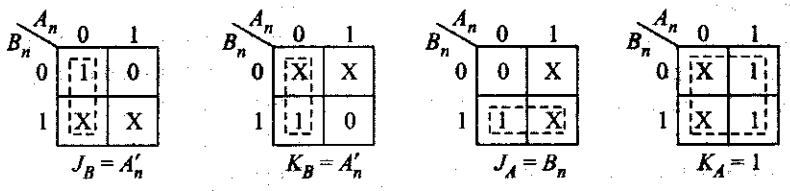
In Method-4, we use T flip-flop for design purpose. The last column shows necessary inputs for the two D flip-flops. Figure 10.45e shows the use of Karnaugh Map to get the design equations.

Present State B_nA_n	Next State $B_{n+1}A_{n+1}$	S_BR_B S_AR_A	D_B D_A	J_BK_B J_AK_A	T_B T_A
0 0	1 0	1 0 0 X	1 0	1 X 0 X	1 0
0 1	0 0	0 X 0 1	0 0	0 X X 1	0 1
1 0	0 1	0 1 1 0	0 1	X 1 1 X	1 1
1 1	1 0	X 0 0 1	1 0	X 0 X 1	0 1

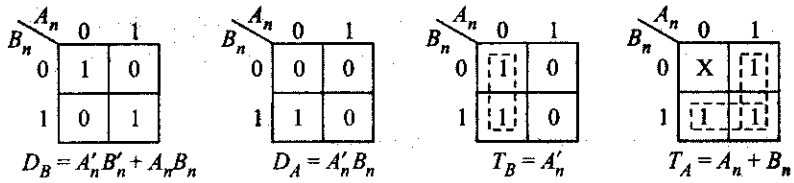
(a)



(b)



(c)



(d) (e)

Fig. 10.45 **(a) State table for the self correcting modulo-3 counter and required inputs, (b) Design with SR flip-flops, (c) Design with *JK* flip-flops, (d) Design with *D* flip-flops, (e) Design with *T* flip-flops**

SUMMARY

A counter has a natural count of 2^n, where n is the number of flip-flops in the counter. Counters of any modulus can be constructed by incorporating logic which causes certain states to be skipped over or omitted. One technique for skipping counts is to steer the clock pulses to certain flip-flops at the proper

time—this is called steering logic. A second technique is to precondition the logic inputs to each flip-flop in order to omit certain states. This is called look-ahead logic.

Logic can be included such that the counter can operate in either a count-up or count-down mode. Furthermore, logic gates can be designed to uniquely decode each state of a counter.

Higher-modulus counters can be easily constructed by using combinations of lower-modulus counters. Such configurations represent a compromise between speed and hardware count.

The digital clock is an interesting application that illustrates some of the methods employing counters and decoders.

GLOSSARY

- ***decoding gate*** A logic gate whose output is high (or low) only during one of the unique states of a counter.
- ***glitch*** An undesired positive or negative pulse appearing at the output of a logic gate.
- ***lock out of a counter*** Counter getting locked into unused states.
- ***modulus*** Defines the number of states through which a counter can progress.
- ***natural count*** The maximum number of states through which a counter can progress. Given by 2^n, where n is the number of flip-flops in the counter.
- ***parallel counter*** A synchronous counter in which all flip-flops change states simultaneously since all clock inputs are driven by the same clock.
- ***presettable counter*** A counter incorporating logic such that it can be preset to any desired state.
- ***ripple counter*** An asynchronous counter in which each flip-flop is triggered by the output of the previous flip-flop.
- ***sequence generator*** Generates a binary data sequence.
- ***up-down counter*** A basic counter, synchronous or asynchronous, that is capable of counting in either an upward or a downward direction.

PROBLEMS

Section 10.1

10.1 Draw the logic diagram, truth table, and waveforms for a two-flip-flop ripple counter similar to that in Fig. 10.1.

10.2 Draw the logic diagram, truth table, and waveforms for a three-flip-flop ripple counter that uses *JK* flip-flops sensitive to a clock PT.

10.3 What is the clock frequency if the period of *B* in Fig. 10.1 is 1000 ns?

10.4 Determine the number of possible states in a counter composed of the following number of flip-flops:

a. 7 b. 10

c. 8

10.5 See if you can draw the waveforms for a 10-flip-flop ripple counter. What difficulties do you encounter?

10.6 What is the largest decimal number that can be stored in each counter in Prob. 10.4?

10.7 Draw the waveforms at Q_B, Q_C, and Q_D for a 7493A, assuming that a 1-MHz clock is applied at input *B*.

10.8 Draw the logic diagram, truth table, and waveforms for a two-flip-flop ripple counter operating in the count-down mode.

Section 10.2

10.9 Draw the gates necessary to decode the 16 states of a 7493A operating as in Fig. 10.3.

10.10 Assume that the clock for the ripple counter in Fig. 10.1 is a 1-MHz square wave and each flip-flop has a delay time of 0.25 μs. Carefully draw the waveforms for the clock and each flip-flop and the output decoded signals. Do you see any sources of difficulty?

10.11 Use the waveforms in Fig. 10.9 and study the remaining seven decoding gates in Fig. 10.7. Show whether glitches will appear by drawing the decoded waveform for each gate.

Section 10.3

10.12 Draw the logic diagram, truth table, and waveforms for the synchronous counter in Fig. 10.13 in the count-up mode.

10.13 Repeat Prob. 10.12, but in the count-down mode.

10.14 Write a Boolean expression for the AND gate connected to the upper leg of the OR gate that drives the clock input to flip-flop Q_D in a 74193.

10.15 Draw a complete set of waveforms for the 74191 in Fig. 10.15 operating in the count-up mode.

10.16 Repeat Prob. 10.15, but operating in the count-down mode.

Section 10.4

10.17 Determine the number of flip-flops that would be required to build the following counters:

a. Mod-6 b. Mod-11

c. Mod-15 d. Mod-19

e. Mod-31

10.18 Draw decoding gates and all waveforms for the mod-3 counter in Fig. 10.16.

10.19 Draw decoding gates and all waveforms for the mod-6 counter in Fig. 10.17.

10.20 Draw decoding gates and all waveforms for the counter in Fig. 10.18.

10.21 Draw the logic diagram, truth table, and waveforms for a mod-9 counter using two mod-3 counters connected in series.

Section 10.5

10.22 Draw decoding gates and all waveforms for the decade counter in Fig. 10.21.

10.23 Draw decoding gates and all waveforms for the counter in Fig. 10.22.

10.24 Draw waveforms for Q_B, Q_C, and Q_D, assuming that the clock is applied to input B of a 7490A.

10.25 Show how an AND gate might be used in Fig. 10.24 to count an unknown number of pulses that occur during a known time interval. This is the basic idea used in a frequency counter.

Section 10.6

10.26 Draw the logic block for a 74163 and show how to construct a mod-13 counter. Use the same technique as in Fig. 10.26. Draw the state diagram.

10.27 Repeat Prob. 10.26 for a mod-11 counter and then a mod-7 counter.

10.28 Draw the waveforms expected in Prob. 10.26.

10.29 Draw the logic block for a 74162 and show how to construct a mod-7 counter. Use the same technique as in Fig. 10.26. Draw the state diagram.

10.30 Use a 74193 presettable counter to implement a mod-8 counter. List the omitted states and normal count sequence; draw a complete logic diagram. Draw the set of waveforms you would expect, showing the clock and the four outputs. Remember that the output transitions occur on positive clock transitions.

Section 10.7

10.31 Design a modulo-3 counter using D flip-flop that counts as $01 \rightarrow 10 \rightarrow 11$. The unused state 00 goes to 01 at next clock trigger.

10.32 Design a modulo-5 counter using D flip-flop the unused states of which go to one of the valid counting state at next clock trigger.

10.33 Design a circuit using *JK* flip-flop that behaves both as a modulo-5 and modulo-3 counter depending on how it I initialized.

10.34 Design a modulo-8 counter (a) using *SR* flip-flop and (b) using *T* flip-flop.

10.35 Design a sequence generator with minimum number of flip-flops that generates sequence '110001' repetitively.

10.36 Design a sequence generator with minimum number of flip-flops that generates sequence '10110001' repetitively.

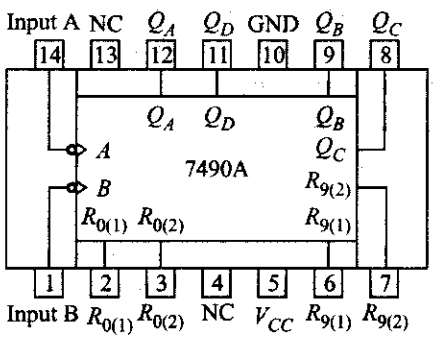
LABORATORY EXPERIMENT

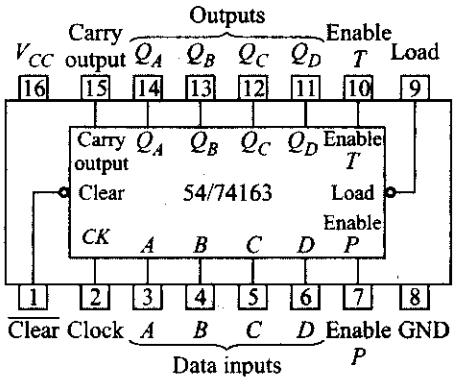
AIM: The aim of this experiment is to study counters and design a modulo-N counter.

Theory: A counter counts events happening in certain form at its input. It consists of a bank of flip-flops and also may have a combinatorial logic circuit. In ripple counter, output of one flip-flop triggers another flip-flop. In synchronous counter, all the flip-flops are triggered simultaneously by a common clock. A modulo-*N* counter generates an output at every *n* pulse occurring at its input which is usually a clock signal. With *m* number of flip-flops, a maximum of 2^m modulo number can be achieved. The CLEAR input clears a counter which can be used to have lower modulo numbers from originally designed higher modulo number counters. The LOAD input, if available, allows parallel loading of a set of data from where counting sequence can begin. This can also be used to get a lower modulo number or some specific counting states.

Apparatus: 5V DC Power supply, Multimeter, Bread Board, Clock Generator, and Oscilloscope

Work element: IC 7490 has two separate counters, a modulo-2 and a synchronous modulo-5 counter which can work independently with two different clocks being connected to input *A* and input *B*. They can be combined to work as modulo-10 (coming from 5 × 2) counter if output of one is used as clock input of other. The pair R_0 performs NAND operation and then clears (active low) the modulo-2 counter. The pair R_9 functions similarly but for modulo-5 counter. Verify 7490 truth table for individual counters and the combination. Use *R* inputs to get modulo numbers different from 2, 5 and 10. IC 74163 is modulo-16 counter with synchronous clear and data input load facility. Verify the truth table of 74163. Understand the function of carry output. Show in how many different ways 74163 can be connected to get a decade (modulo-10) counter. Use two 74163 to obtain a modulo 50 and then a modulo 100 counter.





Answers to Self-tests

1. 63
2. Ten
3. State 3 = $\overline{C}BA$
4. The primary cause of such glitches is flip-flop propagation time; one way to eliminate them is to use the clock as a "strobe."
5. In a parallel counter, all flip-flops change state in synchronism with the clock.
6. The glitches are eliminated because all gate inputs are synchronized—that is, they are all delayed from the clock by the same amount.
7. PTs
8. Four
9. 8. 7, 6, 5, 4, 3, or 2. However, mod-4 and mod-3 require only two flip-flops, and mod-2 is a single flip-flop.
10. A decade counter has 10 states—a mod-10 counter.
11. The decade counter in Fig. 10.21 has symmetrical output at D, but does not count in straight binary. The decade counter in Fig. 10.22 does not have symmetrical output at D and does count in straight binary.
12. 74161, 74163, 74191, 74193
13. The 74191 can count up or down.
14. The 74160 is a mod-10 counter.
15. The 74160 can only count up. (The 74190 can count down.)
16. Lock-out of a counter occurs when the counter remains locked into unused states and does not function properly.
17. Six.

Design of Synchronous and Asynchronous Sequential Circuits

11

OBJECTIVES

- **State machine design using Moore model and Mealy model**
- **State transition diagram and preparation of state synthesis table**
- **Derivation of design equation from state synthesis table using Karnaugh map**
- **Circuit implementation: flip-flop based approach and ROM based approach**
- **Use of Algorithm State Machine chart**
- **State reduction techniques**
- **Analysis of asynchronous sequential circuit**
- **Problems specific to asynchronous sequential circuit**
- **Design issues related to asynchronous sequential circuit**

Design problem normally starts with a word description of input output relation and ends with a circuit diagram having sequential and combinatorial logic elements. The word description is first converted to a state transition diagram or Algorithmic State Machine (ASM) chart followed by preparation of state synthesis table. For flip-flop based implementation, excitation tables are used to generate design equations through Karnaugh Map. The final circuit diagram is developed from these design equations. In Read Only Memory (ROM) based implementation, excitation tables are not required however; flip-flops are used as delay elements. In this chapter, we show how these techniques can be used in sequential circuit design.

There are two different approaches of state machine design called Moore model and Mealy model. In Moore model circuit outputs, also called primary outputs are generated solely from secondary outputs or memory values. In Mealy model circuit inputs, also known as primary inputs combine with memory elements to generate circuit output. Both the methods are discussed in detail in this chapter.

In general, sequential logic circuit design refers to *synchronous* clock-triggered circuit because of its design and implementation advantages. But there is increasing attention to *asynchronous* sequential logic

circuit, as its response is not limited by the clock frequency. But there are too many operational constraints that makes design of asynchronous circuit very complex. Except for time-critical applications synchronous circuit always remains a preferred choice for sequential logic design.

We divide this chapter in two parts. Part A presents a systematic approach towards synchronous system design while Part B is devoted to asynchronous circuit.

PART A : DESIGN OF SYNCHRONOUS SEQUENTIAL CIRCUIT

11.1 MODEL SELECTION

There are two distinct models by which a synchronous sequential logic circuit can be designed. In *Moore model* (Fig. 11.1a) the output depends only on present state and not on input. In *Mealy model* (Fig. 11.1b), the output is derived from present state as well as input. The option to include input in output generation logic gives certain advantage to Mealy model. Usually it requires less number of states and thereby less hardware to solve any problem. Also, the output is generated one clock cycle earlier. However, there is one important disadvantage associated with such circuit. The input transients, glitches etc. (if any) are directly conveyed to the output. Also if we want output transitions to be synchronized while input can change any time Mealy model is not preferred. In Moore model, the output remains stable over entire clock period and changes only when there occurs a state change at clock trigger based on input available at that time.

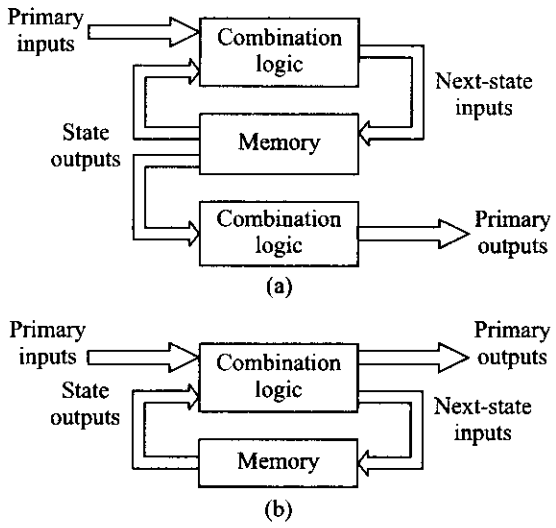


Fig. 11.1 **(a) Moore model, (b) Mealy model of sequential logic system**

In Section 11.2, we shall discuss how *conversion* from one model to other can be done through state diagram representation. Depending on application requirements we choose one of these two models or a mixed model where a part of the circuit follows Mealy model and the other Moore model.

We address all design related issues of synchronous sequential logic by solving a binary sequence detector problem in a step-by-step manner. We use both Moore model and Mealy model for this problem and note the pros and cons of each approach. Note that, any other design problem can be attempted in the same way. The solution presented in subsequent sections is particular to this problem but the approach is general in nature. The sequence detector problem is stated next.

The Problem Design a sequence detector that receives binary data stream at its input, X and signals when a combination '011' arrives at the input by making its output, Y high which otherwise remains low. Consider, data is coming from left, i.e. the first bit to be identified is 1, second 1 and third 0 from the input sequence.

11.2 STATE TRANSITION DIAGRAM

The first step in a sequential logic synthesis problem is to convert this word description to State transition diagram or Algorithm State Machine (ASM) Chart. ASM chart is discussed in Section 11.6. In this section, let us see how we arrive at state transition diagram following Moore and Mealy model. We use the problem presented in Section 11.1 for demonstration.

State Definitions: Moore Model

Since, the output is generated only from the state variables let us see how many of them are necessary. Let the detector circuit be at state *a* when initialized. State *a* can also be considered as one where none of the bit in input sequence is properly detected or the starting point of detection. Then if 1st bit is detected properly the circuit should be at a different state say, *b*. Similarly, we need two more states say, *c* and *d* to represent detection of 2nd and 3rd bit in proper order. When the detector circuit is at state *d*, output *Y* is asserted and kept high as long as circuit remains in state *d* signaling sequence detection. For other states detector output, $Y = 0$.

State Transition Diagram: Moore Model

In Moore model each state and output is defined within a circle in state transition diagram in the format *s*/*Y* where *s* represents a symbol or memory values identified with a state and *Y* represents the output of the circuit. An arrow sign marks state transition following an input value 0 or 1 that is written along the path. Note that *X* represents the binary data input from which sequence '011' is to be detected.

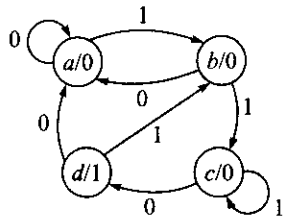


Fig. 11.2a **State transition diagram of sequence detector: Moore model**

Figure 11.2a shows the state transition diagram following Moore model. We arrive at it by following logic. The circuit is initialized with state *a*. If input data $X = 1$, the first bit of the sequence to be detected is considered detected and the circuit goes to state *b*. If $X = 0$ then it remains at state *a* to check next bit that arrives. If at state *b*, the circuit receives $X = 1$, then first two bit of the pattern is considered detected and it moves to state *c*. But at state *b*, if it receives $X = 0$ (i.e input sequence is '01') then detection has to start afresh as we need all three bits of '011' to match. Thus, the detector goes back to initial state *a*. At state *c*, if the circuit receives $X = 0$ then input bit stream is '011' and the circuit goes to state *d* and signals detection of pattern at state *d*. However, at *c* if $X = 1$, the detector is in a situation where it has received '111' in order. It stays at *c* so that if next arriving bit, $X = 0$ it should signal sequence detection. At state *d* if the circuit continues sequence detection job, receiving $X = 1$ it goes to state *b*. That ensures detection of '011' second time in input '011011'. For $X = 0$ the circuit goes to initial state *a* signifying not a single bit has been detected properly subsequent to previous detection.

State Definitions: Mealy Model

Since, the output can be derived using state as well as input we need three different states for 3-bit sequence detector circuit following Mealy model. The three states say, *a*, *b*, *c* represents none, 1st bit and 2nd bit detection. When the circuit is at state *c* if the input is as per the pattern the output is generated in state *c* itself

with proper logic combination of input. Note the difference with Moore model where output is generated one clock cycle later in state *d* and also requires one additional state.

State Transition Diagram: Mealy Model

Here, the output is written by the side of input along arrow path in the format *X*/*Y*, where *X* and *Y* represent input and output respectively. Figure 11.2b shows state transition diagram of the given problem following Mealy model. The explanation is as follows.

The circuit is initialized with state *a*. If it receives input $X = 0$, it stays at *a* else goes to state *b* that signifies first bit is detected properly. In both the cases output, $Y = 0$ signifying no detection. At state *b*, if $X = 0$, the circuit returns to initial state *a*, i.e. no bit in given order is detected and if $X = 1$, goes to state *c*, signifying two bits in order are detected. In both the cases $Y = 0$. Now when at *c*, if input received is 0 then all the three bits of the pattern are received properly and sequence detection can be signaled through $Y = 1$. Also the circuit goes to initial state *a* and prepares for a new set of detection. At state *c*, if $X = 1$ then the sequence received is '111'. An arrival of 0 in next clock can make the detection '110' possible. So, at state *c* if $X = 1$ it is considered as two bits, '11' have been detected properly and the circuit remains at state *c*. The ouput at that time is $Y = 0$ since sequence is not fully detected.

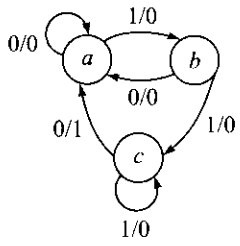


Fig. 11.2b **State transition diagram of sequence detector: Mealy model**

Conversion of Models

Conversion between Mealy and Moore models can take place as shown in Fig. 11.3 where, T_1, T_2, T_3 represent paths leading to state *a*. The path T_4 leads from state *a* when input is 1. If input is 0, state *a* leads to state *b* and there are no other paths reaching *b*. The rule of conversion is as follows. If all the transitions in a Mealy model to a particular state are associated with only one type of output then in corresponding Moore model that output becomes state output (Fig. 11.3a). If there is more than one output in Mealy model we need as many intermediate state variables, as shown in Fig. 11.3b. In Fig. 11.3c it is shown how to treat transitions that loop within a particular state. The reverse of this is applied in converting Moore model to Mealy model.

As an example, let us look at equivalence between two models of the sequence detector problem shown in Fig. 11.2a and Fig. 11.2b. In Mealy model we have paths leading to state *a*, have two different types of outputs. So state *a* of Mealy model get divided in two as *a* and *d* in Moore model. Since there is a loop in state *a* itself for one input, conversion rule shown in Fig. 11.3c is applicable. For other states there is no such conflict and a direct conversion is possible following Fig. 11.3a.

Now that we know one model can be obtained from other, i.e. logical equivalence exists between the two, we let application constraints (as discussed in Section 11.1) decide which one is to be chosen for a particular problem.

SELF-TEST

1. What is a state transition diagram?
2. How does state transition diagram of a Moore Machine differ from Mealy machine?

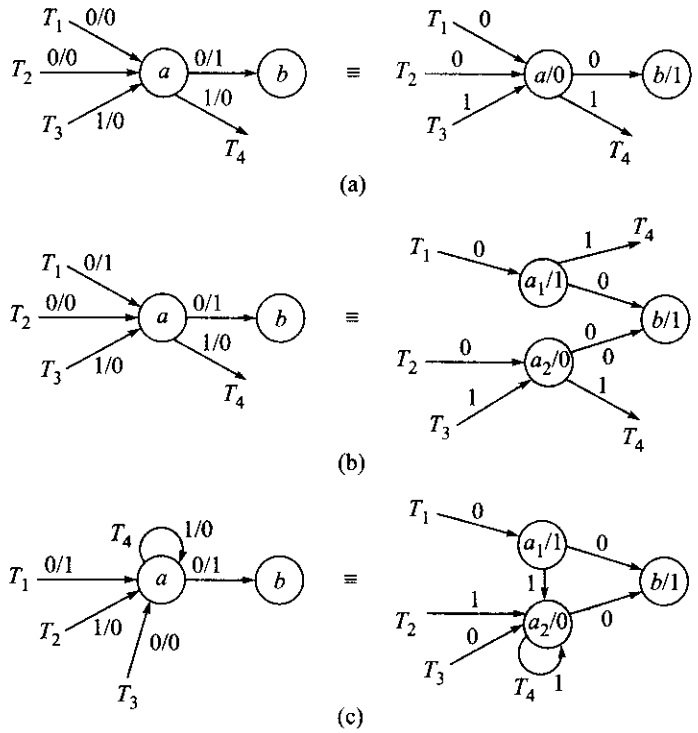


Fig. 11.3 **Conversion between Mealy and Moore model**

11.3 STATE SYNTHESIS TABLE

The next step in design process is to develop *state synthesis table*, also called *circuit excitation table* or simply *state table* from state transition diagram. Note that for m number of memory elements we can have up to 2^m number of different states in a circuit. Once we decide how many memory elements are to be used, we go for state assignment.

Often, we need to exercise state reduction technique before state assignment to remove redundancy in state description. Redundancy may come while converting word description of a complex problem to state transition diagram. We shall discuss state reduction techniques in Section 11.7.

State Assignment

Here, we allocate each state a binary combination of memory values. For the given problem, both Moore and Mealy models require minimum two flip-flops (say A and B) to define their states (4 for Moore and 3 for Mealy). Let the state assignment be as follows.

a: $B = 0$, $A = 0$ b: $B = 0$, $A = 1$ c: $B = 1$, $A = 0$ d: $B = 1$, $A = 1$

Note that Mealy model does not use state d. Assignment can be done in any order, e.g. we can make a: $B = 0$, $A = 1$ and b: $B = 0$, $A = 0$ and proceed with the design. However, one set of state assignment may give simpler final logic circuit over other. Though there is no definite state assignment rule that gives minimum

hardware for an implementation, logical adjacency between transition states often helps. In Problem 11.7 to 11.10, we shall see how a different state assignment for this sequence detector problem asks for different hardware requirement.

State Synthesis Table

The next design step is to decide what kind of memory elements are to be used for our design. Flip-flops are commonly used for this purpose. A ROM based implementation is discussed in Section 11.5. When we use flip-flops we take note of the fact that there are different types of them available. Each flip-flop has a unique characteristic equation and excitation table (Section 8.9). In synthesis problem we have to find out how flip-flop inputs are to be connected and how final output is generated from flip-flop output. For this, we use state synthesis table that gives the input requirement of all flip-flops for a given state transition diagram. Before we prepare this table we should decide which flip-flop we are going to use. We normally prefer *JK* flip-flop as it has maximum number of don't care states in its excitation table and that leads to simpler design equations. We design the given sequence detector circuit using *JK* flip-flops.

Moore Model

State synthesis table obtained from state transition diagram of Moore model (Fig. 11.2a) and excitation table of *JK* flip-flop (Fig. 8.34) is shown in Table 11.1. It has eight rows as for each of the four possible states there can be two different types of inputs. The table is prepared as follows. When the circuit is at state 00, i.e. *a* and receives $X = 0$ it remains at state 00 and output in this state $Y = 0$. Since both *B* and *A* flip-flop makes $0 \rightarrow 0$ transition both the flip-flops should have input 0× from excitation table. This way first four columns of the table (present state, input, next state, output) are filled from state transition diagram and last two columns (*B* and *A* flip-flop inputs) from flip-flop excitation table.

Table 11.1 State Synthesis Table for Moore Model

Present State		*Present Input*	*Next State*		*Output*				
B_n	A_n	X_n	B_{n+1}	A_{n+1}	Y_n	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	0	×	1	×
0	1	0	0	0	0	0	×	×	1
0	1	1	1	0	0	1	×	×	1
1	0	0	1	1	0	×	0	1	×
1	0	1	1	0	0	×	0	0	×
1	1	0	0	0	1	×	1	×	1
1	1	1	0	1	1	×	1	×	0

Mealy Model

Since, Mealy Model requires three states for this problem we have six rows in state synthesis table as in each state there can be two different types of input $X = 0$ or $X = 1$. Table 11.2 represents state synthesis table for Mealy model. The method remains the same as Moore model but we use state transition diagram (Fig. 11.2b) corresponding to Mealy model from Section 11.2.

Table 11.2 State Synthesis Table for Mealy Model

Present State		*Present Input*	*Next State*		*Present Output*				
B_n	A_n	X_n	B_{n+1}	A_{n+1}	Y_n	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	×	0	×
0	0	1	0	1	0	0	×	1	×
0	1	0	0	0	0	0	×	×	1
0	1	1	1	0	0	1	×	×	1
1	0	0	0	0	1	×	1	0	×
1	0	1	1	0	0	×	0	0	×

11.4 DESIGN EQUATIONS AND CIRCUIT DIAGRAM

In this section, we discuss how to get final circuit diagram from state synthesis table (Section 11.3) through design equation. In design equation we express flip-flop inputs as a function of present state, i.e. memory values (here, B and A) and present input (here, X). This ensures proper transfer of the circuit to next state. The design equations also give output (here, Y) equation in terms of state variables or memory elements in Moore model and state variables together with input in Mealy model. We normally use Karnaugh map technique to get a simplified form of these relations.

Moore Model

Figure 11.4a presents Karnaugh map developed from state synthesis Table 11.1 and also shows corresponding design equations. Figure 11.4b shows the sequence detector circuit diagram developed from these equations. This is done in the following manner. Equation $J_B = XA$ requires J input of flip-flop A to be fed from a two input AND gate, inputs to which are X and A. The other inputs and output are obtained in similar way. Note that, output is generated by AND operation on two flip-flop outputs and does not use X.

Mealy Model

Using state synthesis table corresponding to Mealy model (Table 11.2) we can fill six positions in each Karnaugh map (Fig. 11.5a). Locations $B_nA_nX = 110$ and $B_nA_nX = 111$ are filled with don't care(×) conditions as such a combination never occur in the detector circuit if properly initialized. The design equations are obtained from these Karnaugh maps from which circuit diagram is drawn as shown in Fig. 11.5b. Note that in this circuit, output directly uses input information.

SELF-TEST

3. What is an excitation map?
4. Is there any difference in hardware requirement between Moore and Mealy machine?
5. In the sequence detector circuit designed here, show the output in each clock cycle by completing Table 11.3.

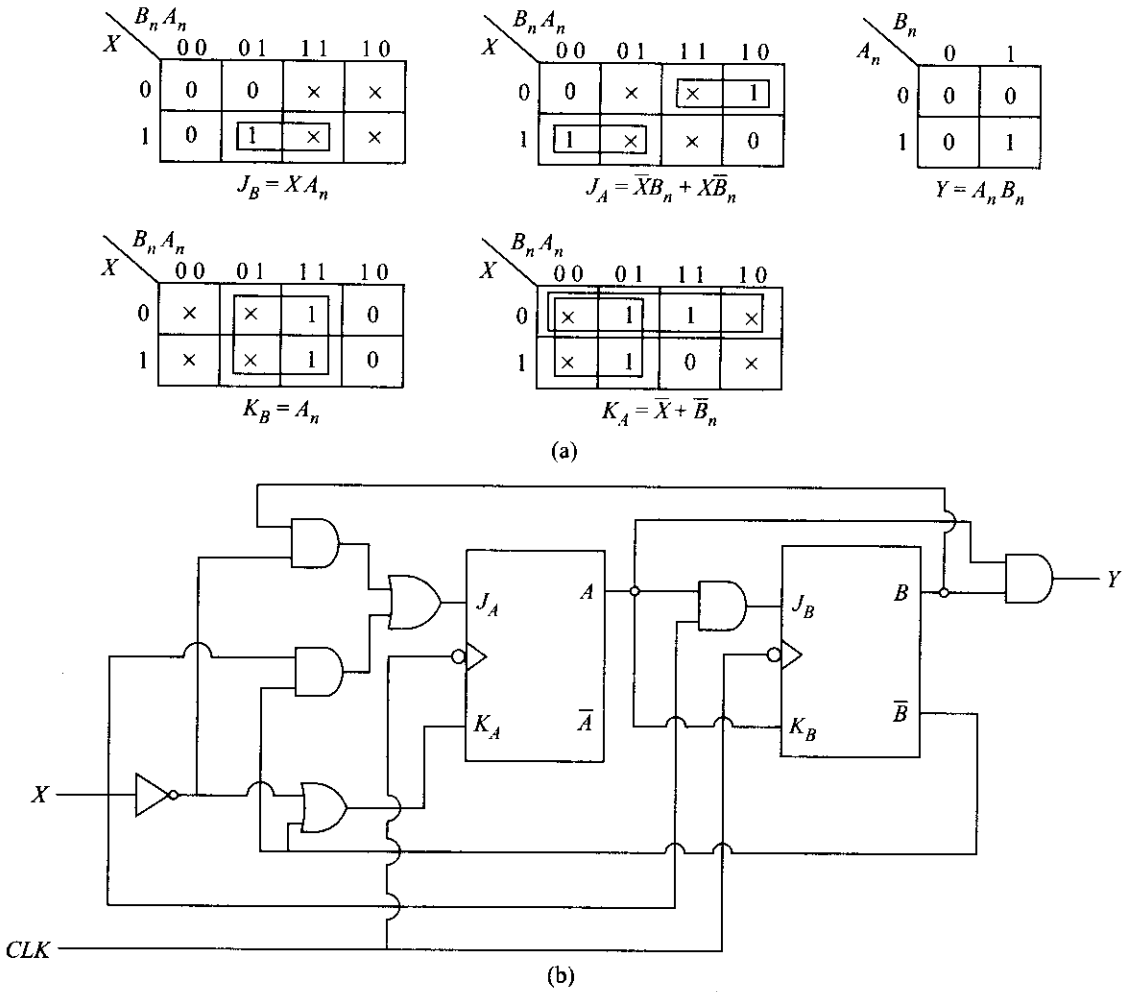


Fig. 11.4 (a) Design equations for Moore model, (b) Circuit diagram following Moore model

Table 11.3

Clock Cycle	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Input	0	1	0	1	1	0	1	0	1	1	0	1	1	0	1
Moore output															
Mealy output															

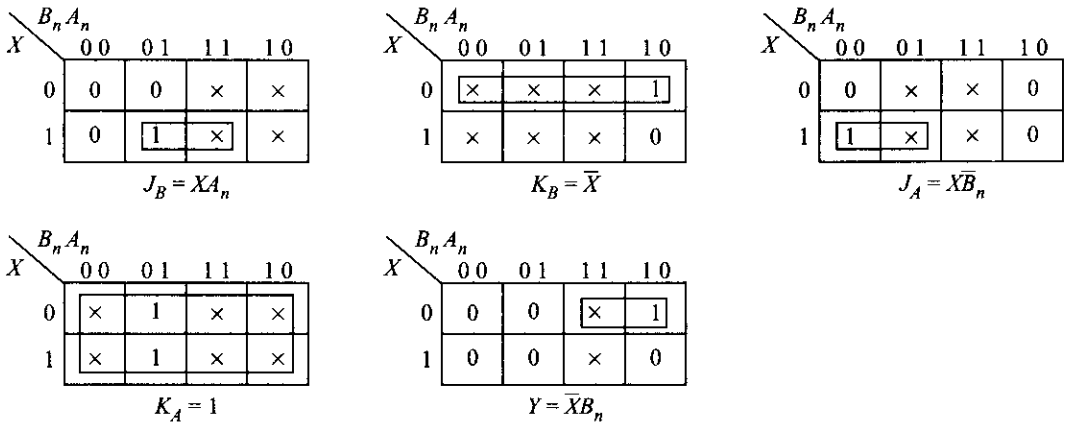


Fig. 11.5 (a) Design equations for Mealy models, (b) Circuit diagram following Mealy model

11.5 IMPLEMENTATION USING READ ONLY MEMORY

In this section we present an interesting solution to sequential logic problem using Read Only Memory (ROM) which though called memory is a combinatorial circuit. The output of ROM is immediately available when a particular location in memory is addressed. The detailed description of ROM, its architecture, type and operation is given in Sections 4.9 and 13.5.

For our design purpose we need to have a ROM that has as many memory locations as the number of rows in a state synthesis table. Note that, each row is uniquely identified by present state and present input. This present state and input combination through an address decoder points to memory spaces in ROM. In each location of ROM we store the next state value of the circuit.

We also need a bank of Delay flip-flops, the number is same as the number of state variables or memory elements. Next state information for each state variable that is stored in ROM is fed to these *D* flip-flop inputs. At clock trigger they appear at the output of the flip-flop. Now that the circuit has advanced by one clock cycle these *D* flip-flop outputs serve as present state information and fed to ROM address decoder. Together with present input they point to another location in memory that has next state information. ROM being a combinatorial circuit these next state values are immediately available to *D* flip-flop inputs and the cycle goes on. The final output is generated from state variables in Moore model and also uses direct input in Mealy model.

Moore Model

For the sequence detector problem we need 8 × 2 ROM as there are 8 rows in state synthesis Tables 11.1 and 11.2. The circuit diagram is shown in Fig. 11.6. The 3 to 8 address decoder is fed by *B*, *A* and *X*. The output of decoder is 000, 001, 010 .. in same order as they appear in state Table 11.1. For example, when $BAX = 000$ next state is 00 from state table and we store 00 in ROM corresponding to decoder output 000. Similarly, next memory values stored in ROM are 01, 00, 10, 11... in order from state table. *D* flip-flop connections are explained before and output is generated following logic equation $Y = AB$.

The circuit functions like this. The *D* flip-flops are initially cleared, i.e. $BA = 00$. If $X = 0$, the first location in ROM corresponding to $BAX = 000$ is selected and ROM output = 00 and at clock trigger next state remains

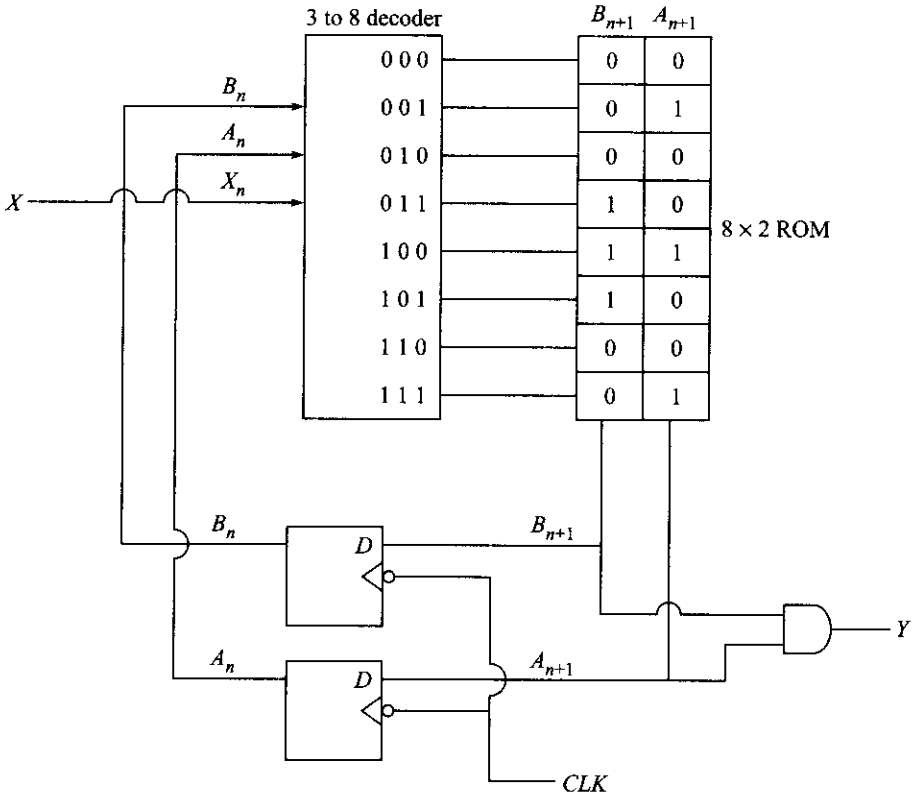


Fig. 11.6 **ROM based implementation of sequence detector: Moore model**

at $BA = 00$. Thus 00 state remains at 00 for $X = 0$. If $X = 1$, then $BAX = 001$ location in ROM is selected which stores 01, i.e. the circuit (D flip-flops) goes to $BA = 01$ state with clock trigger. For $BA = 01$, if $X = 0$ then $BAX = 010$ location in ROM is selected which stores 00 that means with NT of clock the circuit goes to state 00 or initial state. If $BA = 01$ and $X = 1$ then $BAX = 011$ location of ROM is selected which stores 10. Thus next state becomes 10 with NT. Now at $BA = 10$, if $X = 0$ then $BAX = 100$ location is selected which stores 11 and next state becomes 11. If we have recorded input values we see when 100 location in ROM is selected in ROM the pattern '011' has arrived in proper order. Stored ROM data is immediately available in the same clock cycle and we can generate circuit output from this signaling detection.

Thus, compared to previous implementation here sequence detection signal comes one cycle earlier. Also note that the design process is very straightforward. We don't need to remember flip-flop excitation table or simplify design equation, which gives different circuit for different problem. Here, for all problems circuit remains same only the content of ROM changes. The output logic may also be different but we have an option to store the output as 3rd bit in the ROM and we then don't need any output logic equation to be realized by basic gates. Refer to Problems 11.15 and 11.16.

Mealy Model

ROM based solution of Mealy model uses state synthesis table in the same way as Moore model ROM locations are selected by present state and input as appears in state table and next state value fills corresponding ROM locations (Fig. 11.7). Delay flip-flop banks are used in the same way but final output is generated from

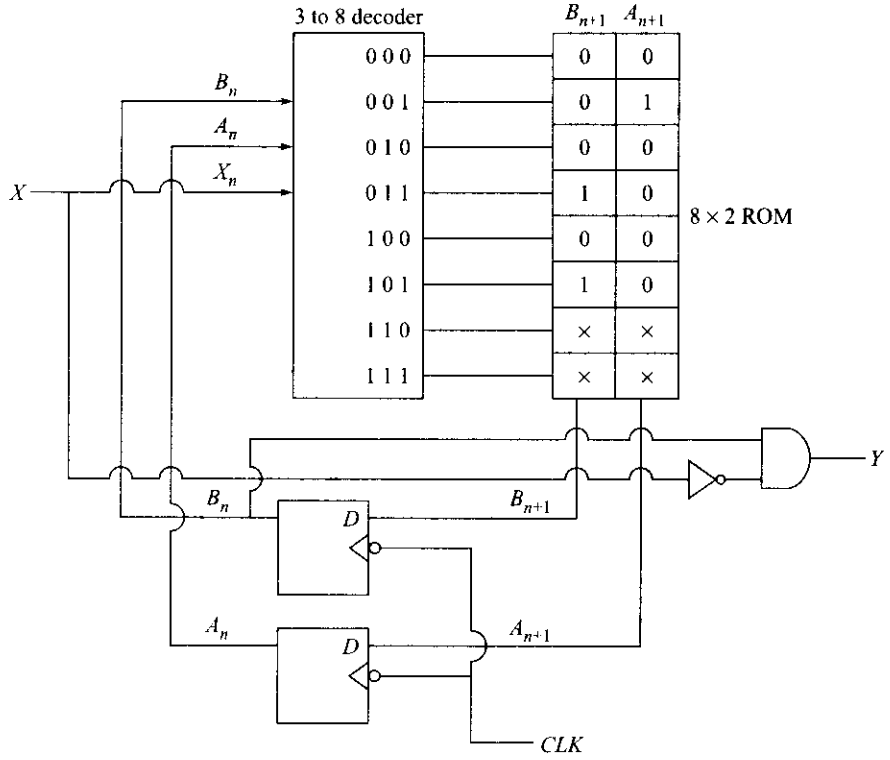


Fig. 11.7 **ROM based implementation of sequence detector: Mealy model**

D flip-flop outputs (representing present state) and data input. In Moore model we have used ROM outputs directly to generate sequence detector output. Note that ROM of size 6×2 is sufficient for Mealy model and last two locations of 8×2 ROM are not used.

Example 11.1 Give design equations for the synchronous sequential logic circuit that has two inputs X and Y. The output Z of this circuit is generated according to the timing diagram shown in Fig.11.8.

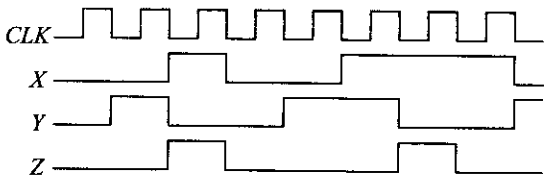
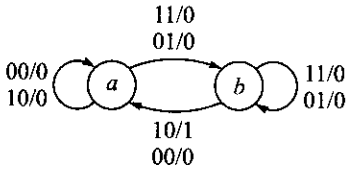


Fig. 11.8 Timing diagram for Example 11.1

Solution Instead of word description we have timing diagram explaining the problem. On careful observation we find Z remains high for one clock period when Y goes from high to low and if at that time (Y low) the other input X remains at logic high. Thus if we adopt a Mealy model, the circuit needs one memory element that remembers if previous state of Y was high for any $X = 1$, $Y = 0$ input. The state transition diagram, state synthesis table and design equations are shown in Figs. 11.9(a), (b) and (c) respectively. The design has been done with D flip-flop in which D input simply follows next state. Refer to excitation table of Fig. 8.34.



(a)

A_n	X	Y	A_{n+1}	D_n	Z
0	0	0	0	0	0
	0	1	1	1	0
	1	0	0	0	0
	1	1	1	1	0
1	0	0	0	0	0
	0	1	1	1	0
	1	0	0	0	1
	1	1	1	1	0

(b)

A_n \ XY	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$D_n = Y$

A_n \ XY	00	01	11	10
0	0	0	0	0
1	0	0	0	1

$Z = X\overline{Y}A_n$

(c)

Fig. 11.9 (a) State transition diagram, (b) State synthesis table, (c) Design equations for Example 11.1

SELF-TEST

6. How use of flip-flop is different in ROM based implementation?
7. Complete the table as shown in Q. 5 for ROM based Moore and Mealy models.

11.6 ALGORITHMIC STATE MACHINE

Algorithmic State Machine (ASM) is a flow chart like representation (ASM Chart) of the algorithm a state machine performs. State Transition diagrams though more compact in representation has certain disadvantages. For relatively more complex problem where number of inputs and states are higher the state diagram space becomes so crowded that it is difficult to read. The other advantage of ASM chart is that, it handles implementation issues with greater ease offering better timing information. In ASM chart, square boxes represents a state. If a state generates an unconditional output (Moore model) it can be specified within the square box. A diamond shaped box represents decision to be taken and normally the variable or the condition that is tested is placed inside it with a question mark. There are two exit paths of this decision box since the decision is binary in nature. For Mealy model, oval shaped boxes are used to describe the output that depends on present state as well as the present input. Circles are used to denote start, stop of the algorithm and also the connector point of an ASM chart when it becomes too large and needs to be drawn at more than one place. Entry and exit of each ASM block is shown by arrow headed connecting link.

We take a new example and discuss its design using ASM chart. ASM chart for sequence detector problem of previous section is shown in Example 11.4.

Vending Machine Problem

The task is to design a synchronous logic control unit of a vending machine. The machine can take only two types of coins of denomination 1 and 2 in any order. It delivers only one product that is priced Rs. 3. On receiving Rs. 3 the product is delivered by asserting an output $D = 1$ which otherwise remains 0. If it gets Rs. 4 then product is delivered by asserting X and also a coin return mechanism is activated by output $Y = 1$ to return a Re. 1 coin. There are two sensors to sense the denomination of the coins that give binary output as shown in the following table. The clock speed is much higher than human response time, i.e. no two coins can be deposited in same clock cycle.

I	J	Coin
0	×	No Coin dropped
1	0	One Rupee
1	1	Two Rupees

ASM Chart

The ASM chart is prepared following Mealy model and is shown in Fig. 11.10. The initial state when no coin is deposited is designated as state a. Note that, sensor output $I = 0$ indicates no coin is deposited. At every clock trigger I is tested and if found 0 the circuit retraces its path to state a and obviously none of X and Y is asserted, i.e. no product is delivered or coin returned. If $I = 1$, the controller tests J. If $J = 0$ it goes to state b that represents Re. 1 is received and if $J = 1$, goes to state c indicating Rs. 2 is received. The controller remains at state b if no further coin is deposited found by checking I. Now, if $I = 1$ and $J = 0$, the machine has received two Re. 1 coins in succession and should move to state c. But $I = 1$ and $J = 1$ means a Rs. 2 coin is received following Re. 1 totaling Rs. 3 the cost of the product. Hence, the product is delivered by asserting $X = 1$ and the circuit

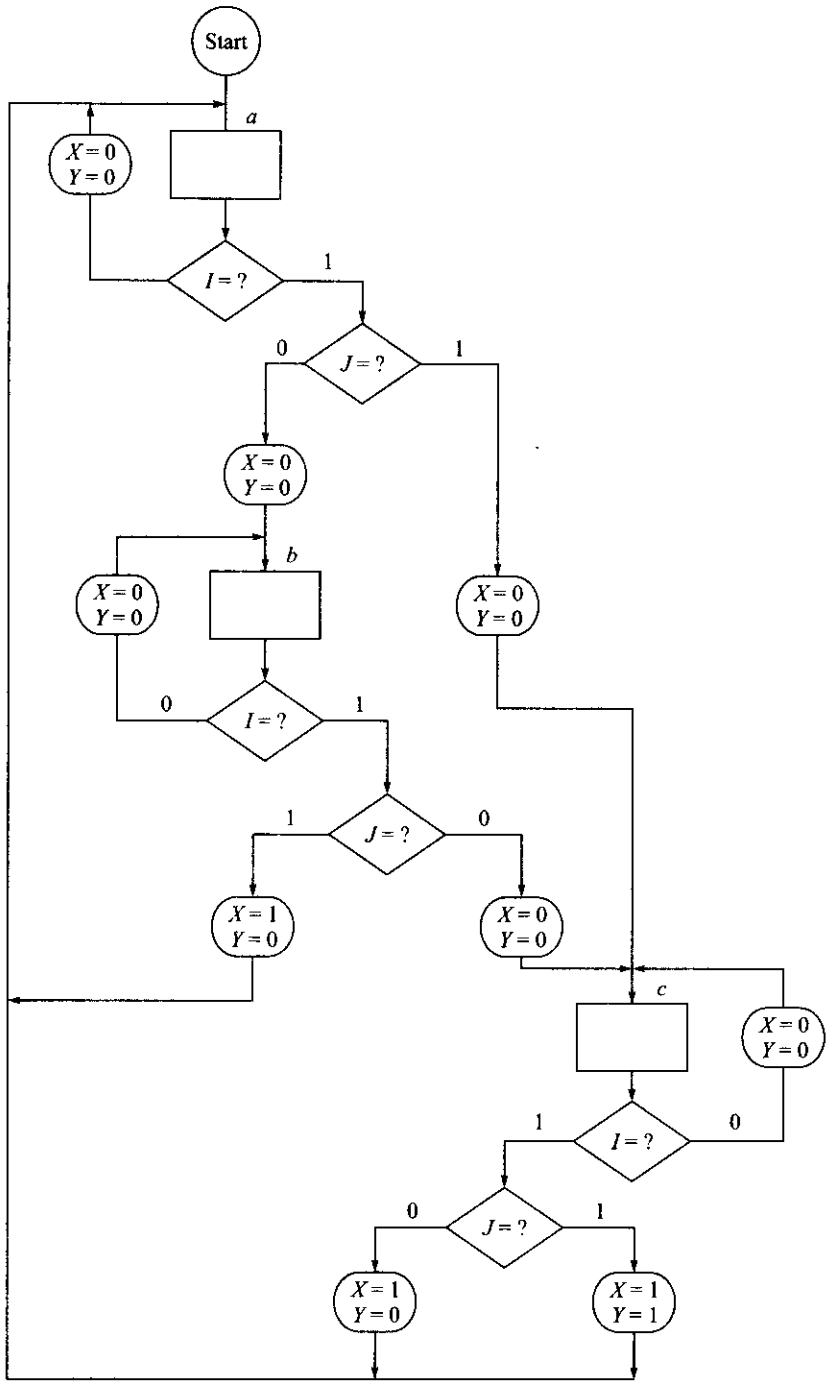


Fig. 11.10 ASM chart for vending machine problem: Mealy model

goes to initial state. At state c if on testing $I = 1$ that is a coin is deposited, the controller tests J to ascertain if it is Re. 1 or Rs. 2. If $J = 0$, Re. 1 is deposited and a total of Rs. 3 is received. The product is delivered by $X = 1$ and the circuit goes to initial state a. Now if $J = 1$ then Rs. 2 is received totaling Rs. 4. Then Re. 1 is returned by asserting $Y = 1$, also the product is delivered through $X = 1$ and the controller moves to initial state a.

State Assignment and State Synthesis Table

The subsequent design steps are same as state transition diagram based method discussed before. We prepare state table from this ASM Chart. In this example we show how to use D flip-flop as the memory element though JK flip-flop can also be used. As expected, filling up of columns that corresponds to D input in a given state is easier than JK flip-flop, also the number of Karnaugh map to be drawn for each flip-flop is half that of JK flip-flop as D flip-flop has only one data input. But all these come at a cost of increased hardware complexity. This example will highlight this aspect of design issue for synchronous sequential circuit. The state assignment is done as follows. Since there are three different states we need two flip-flops (say, B and A) to represent them. Let $BA = 00$ represent state a, $BA = 01$ state b, $BA = 10$ state c. State $BA = 11$ is not used in this problem. Table 11.4 shows the state table corresponding to ASM chart shown in Fig. 11.10 and also the D inputs corresponding to every state.

Table 11.4 State Synthesis Table for Vending Machine Problem: Mealy Model

Present	*State*	*Input*		*Next State*		*Output*		D_B	D_A
B_n	A_n	I	J	B_{n+1}	A_{n+1}	X	Y		
		0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
		1	0	0	1	0	0	0	1
		1	1	1	0	0	0	1	0
		0	0	0	1	0	0	0	1
0	1	0	1	0	1	0	0	0	1
		1	0	1	0	0	0	1	0
		1	1	0	0	1	0	0	0
		0	0	1	0	0	0	1	0
1	0	0	1	1	0	0	0	1	0
		1	0	0	0	1	0	0	0
		1	1	0	0	1	1	0	0

Design Equations from Karnaugh map and Circuit Diagram

Karnaugh maps for each flip-flop input and both the outputs are shown in Fig. 11.11a along with design equations. Note that for $BA = 11$ we have don't care states in each map that helps in minimizing design equation. The final digital controller circuit for the vending machine problem is shown in Fig. 11.11b.

Example 11.2 Draw the Delay flip-flop-Decoder-ROM based digital controller circuit for the vending machine problem.

Solution The circuit is shown in Fig. 11.12a. The values stored in ROM is derived from state stable. Here we have used higher ROM size adding two more bits in the memory location for each address but do not need any basic gate for output logic. However, we can use logic gates as shown in sequence detector problem and reduce two columns corresponding to X and Y in ROM.

IJ \ B_nA_n	00	01	11	10
00	0	0	×	1
01	0	0	×	1
11	1	0	×	0
10	0	1	×	0

$D_B = \bar{I}B_n + I\bar{J}A_n + IJ\bar{B}_n\bar{A}_n$

IJ \ B_nA_n	00	01	11	10
00	0	1	×	0
01	0	1	×	0
11	0	0	×	0
10	1	0	×	0

$D_A = \bar{I}A_n + I\bar{J}\bar{B}_n\bar{A}_n$

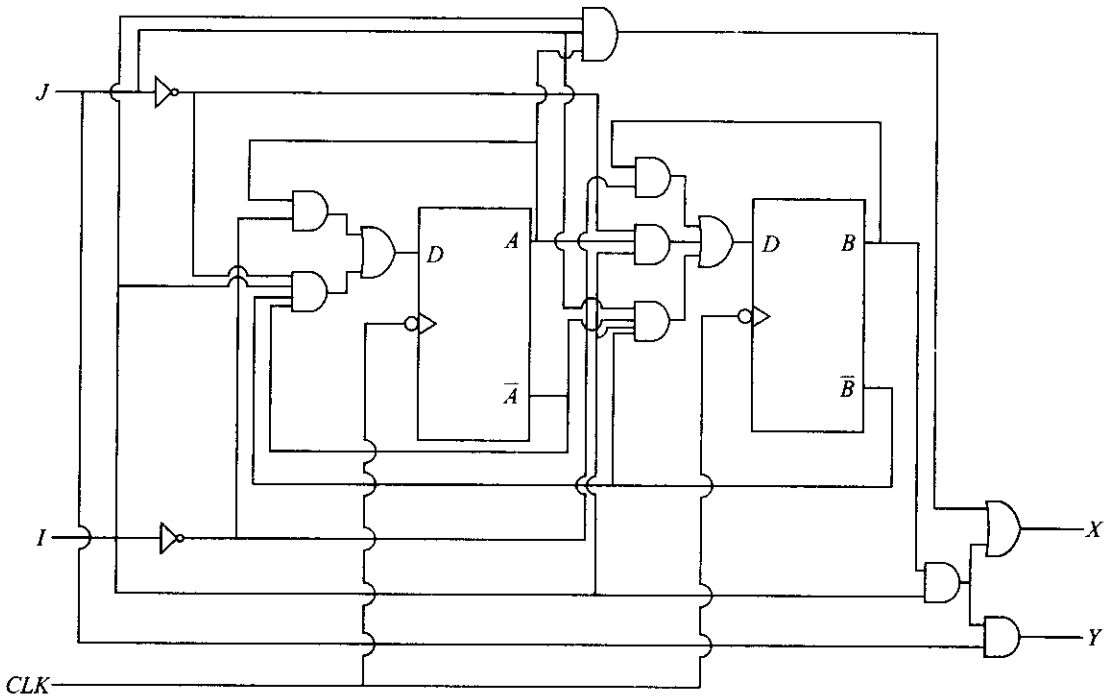
IJ \ B_nA_n	00	01	11	10
00	0	0	×	0
01	0	0	×	0
11	0	1	×	1
10	0	0	×	1

$X = IB_n + IJA_n$

IJ \ B_nA_n	00	01	11	10
00	0	0	×	0
01	0	0	×	0
11	0	0	×	1
10	0	0	×	0

$Y = IJB_n$

(a)



(b)

Fig. 11.11 **(a) Design equation, (b) Circuit diagram for vending machine problem: Mealy model**

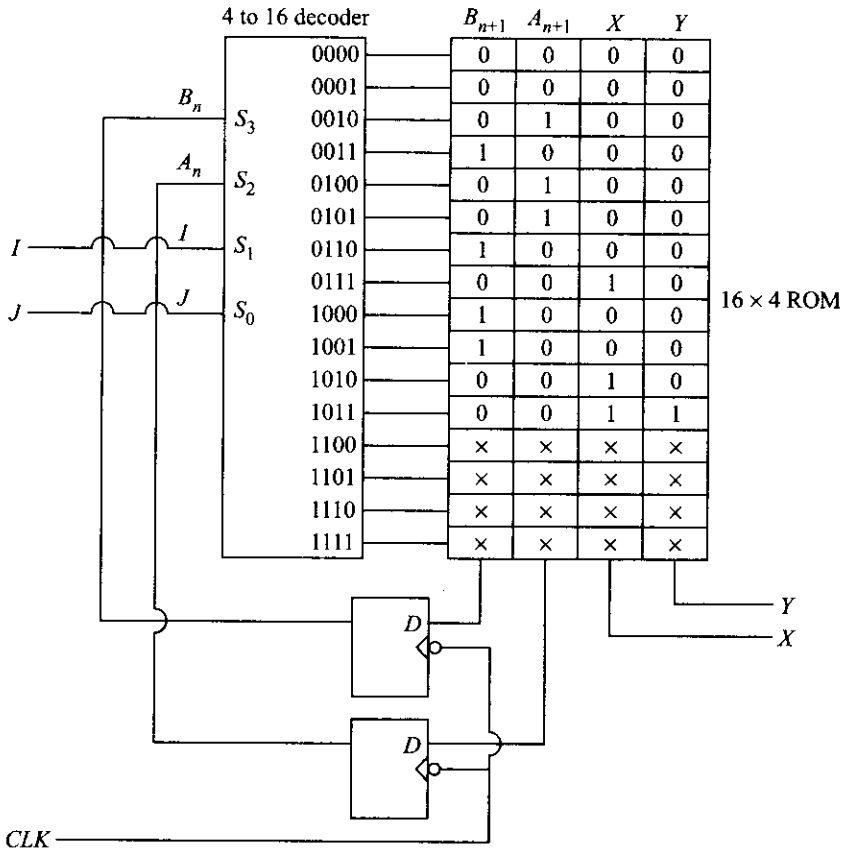


Fig. 11.12a **ROM implementation of vending machine problem**

Example 11.3 Draw state transition diagram of the Mealy model vending machine problem.

Solution The diagram can easily be drawn from ASM chart (Fig. 11.10) and is shown in Fig. 11.12b.

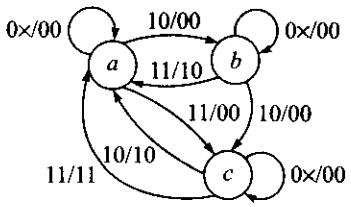


Fig. 11.12b **State transition diagram of vending machine problem**

Example 11.4 Draw ASM chart of the sequence detector problem described in Section 11.2 following Moore model.

Solution In Fig. 11.13 we show the ASM chart of the sequence detector problem described in Section 11.2 following Moore model where X denotes the input data bit and Y the detector output.

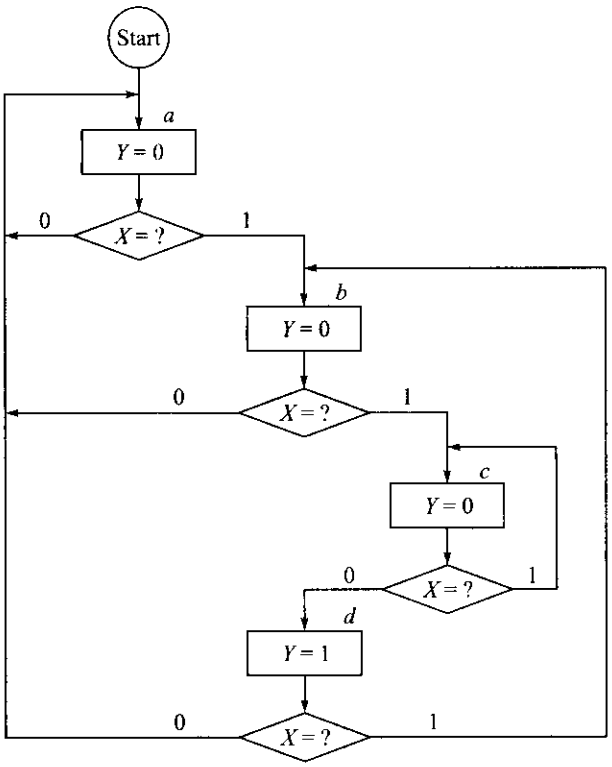


Fig. 11.13 **ASM chart of sequence detector problem: Moore model**

Note the similarity between Moore model state transition diagram of Fig. 11.2a and ASM chart shown here. Once we arrive at the ASM Chart the rest of the design procedure starting from state assignment up to final circuit diagram is same as what is discussed in Section 11.6. ASM Chart for the Mealy model sequence detector is left as exercise for the reader.

11.7 STATE REDUCTION TECHNIQUE

In design of sequential logic circuit state reduction techniques play an important role, more so for complex problems. While converting problem statement to state transition diagram or state table we may use more number of states than necessary. On removing redundant states the clarity of the problem is enhanced. This also offers simpler solution and less hardware to implement a circuit. We explain two state reduction techniques through an example.

Let the state transition diagram drawn following a Mealy model is as shown in Fig. 11.14. The goal is to identify and remove redundant states, if any and obtain the reduced state diagram.

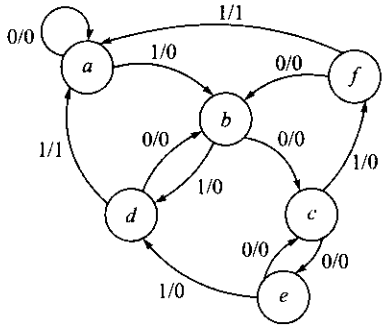


Fig. 11.14 **A state transition diagram**

Row Elimination Method

In this method, we first prepare a state table where at any given state the next state and present output(s) are written for each combination of input(s). In the present problem there are only two possible values of input $X = 0$ and $X = 1$. For 2 input circuits there will be $2^2 = 4$ such combinations in this table. Now, two states are considered equivalent if they move to same or equivalent state for every input combination and also generate same output.

Figure 11.15a shows the state table for Fig. 11.14 and we see that states b and e are equivalent as next state and output are same. Therefore, we can retain one of these two and discard the other. Let us retain b and eliminate row corresponding to present state e and in rest of the table, wherever e appears we replace it by b and get table of Fig. 11.15b. A careful look on this reduced table shows state d and f are equivalent. We retain d and eliminate row f from this table and replace f with d in rest of the rows and get Fig. 11.15c. This table places us in an interesting situation as far as equivalence between two rows are concerned. For states b and c except for next state at $X = 0$ the rest are same. Now b and c would have been equivalent if these next states are equivalent. For b, next state is c and for c, next state is b. Thus bc are equivalent if next states cb are equivalent which can always be true (from tautology). Thus, b and c are equivalent and state b is retained and row c is eliminated in the same manner and shown in Fig. 11.15d, which cannot be further reduced. The final reduced state table has three states reduced from six in the original state diagram and final reduced state diagram is shown in Fig. 11.15e.

Present state	Next state $X=0$	Next state $X=1$	Present output $X=0$	Present output $X=1$
a	a	b	0	0
$\sqrt{b}$	c	d	0	0
c	e	f	0	0
d	b	a	0	1
$\sqrt{e}$	c	d	0	0
f	b	a	0	1

(a) Original table

Present state	Next state $X=0$	Next state $X=1$	Present output $X=0$	Present output $X=1$
a	a	b	0	0
b	c	d	0	0
c	b	f	0	0
$\sqrt{d}$	b	a	0	1
$\sqrt{f}$	b	a	0	1

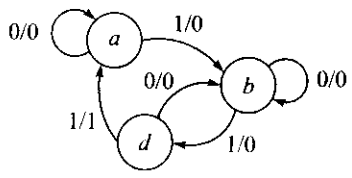
(b) After one row elimination

Present state	Next state $X=0$	Next state $X=1$	Present output $X=0$	Present output $X=1$
a	a	b	0	0
$\sqrt{b}$	c	d	0	0
$\sqrt{c}$	b	d	0	0
d	b	a	0	1

(c) After two row elimination

Present state	Next state $X=0$	Next state $X=1$	Present output $X=0$	Present output $X=1$
a	a	b	0	0
b	b	d	0	0
d	b	a	0	1

(d) Final reduced table after three row elimination

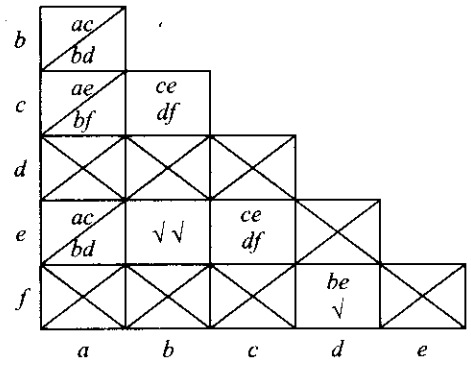


(e)

Fig. 11.15 **(a)–(d) Tables showing row elimination steps, (e) Reduced state diagram**

Implication Table Method

Implication table provides a more systematic approach towards solution of a complex state reduction problem. For n states in the initial description we have $n-1$ rows in implication table and as many number of columns. Refer to implication table of Fig. 11.16a for the given state reduction problem. The cross-point in an implication table is the location where a row and a column meet. Here, the conditions for equivalence between the states crossing each other are tested. We use state table of Fig. 11.15a derived from state transition diagram to fill up implication table. The steps to be followed are given next.



(a)

$e : e$
$d : e\,(df)$
$c : e\,(df)\,(ce) \equiv (df)\,(ce)$
$b : (df)\,(ce)\,(bc) \equiv (df)\,(bce)$
$a : a\,(df)\,(bce)$

$P = (df)\,(bce)\,(a)$

(b)

Fig. 11.16 **Implication table method of state reduction: (a) Implication table, (b) Partition table**

In **Step 1**, we identify the states, which cannot be equivalent, as their outputs do not match. This we denote by putting a double-cross in respective cross points. In this problem state d and f only have output = 1 for $X = 1$ unlike other states. Thus, intersection of d and f with others except themselves are double crossed.

In **Step 2**, for other cross points, we write necessary conditions for equivalence of intersecting states. As an example, let us look at intersection of states a and b. To get the necessary condition we refer to rows starting with a and b in state table of Fig. 11.15a. We find that at $X = 0$, a stays at a while b goes to c and at $X = 1$, a goes to b while b goes to d. Thus, a and b can only be equivalent if next states a and c are equivalent and also if b and d are equivalent. This is written at cross point of a and b in implication table. Note that output of a and b match, else, it would have got a double cross in Step 1. We similarly fill up other cross points and note that b and e are equivalent and does not require any equivalence between other states and a double tick mark is placed at that cross point.

In **Step 3**, we use relationships obtained in Steps 1 and 2, specially the ones represented by double cross and double tick mark and check if any other cross points can be crossed or ticked. Since df equivalence depends only on equivalence be which is true, they are equivalent and that cross point can be ticked. Similarly, ac cannot be equivalent, as it requires bf to be equivalent which is not true. Hence, ac intersection is crossed.

In **Step 4**, we keep repeating Step 3 and cross or tick (if possible) as many cross points in the implication table as possible. We see ab and ae cross points can be crossed as they need ac to be equivalent which is crossed in the previous step. With no further crossing and ticking possible the implication table is fully prepared and we go to Step 5.

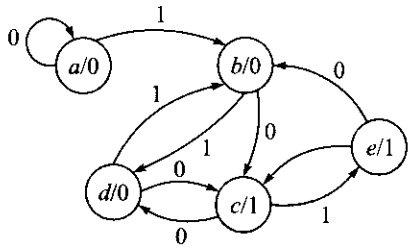
In **Step 5**, we check pairwise equivalence starting from rightmost column e of implication table. Since, the only cross point, representing ef equivalence along column e is crossed there is no equivalence possible

at column *e* and we write *e* in Fig. 11.16b in the first place. In column *d*, we find *df* are equivalent and along *d* in Fig. 11.16b the same is written. In column *c*, we find *ce* are equivalent as *df* is equivalent. In column *b*, equivalence between *be* and *bc* can be observed as *ce* and *df* are already considered equivalent and the same are written along *c* and *b*. Note that if *p* and *q* are equivalent and so are *q* and *r* then *p* and *q* are equivalent. In that case, *pqr* can form one group and any one of its members can represent the group. Since, column *a* does not give any equivalent pair the final partition table is represented as P = (*df*) (*bce*) (*a*) and has three partitions. Then three states are sufficient to solve this problem each representing one partition. If *d* represents any of *df* and *b* represents *bce* in Fig. 11.16b we get reduced state table as shown in Fig. 11.15d and corresponding reduced state diagram is shown in Fig. 11.15e.

Note that the final result is same by both the state reduction method. However, in row elimination method one has to draw many tables for a complex state reduction problem and depend a lot on observation power. The implication method being more systematic is more conclusive.

We shall discuss state reduction technique for *incompletely specified state table* in connection with asynchronous sequential circuit design in Section 11.10. In such problems some of the next states or output remains unspecified and treated as don't care condition.

Example 11.5 Reduce state transition diagram (Moore Model) of Fig. 11.17a by (i) row elimination method and (ii) implication table method



(a)

Present state	Next state $X=0$	Next state $X=1$	Present output
a	a	b	0
$\sqrt{b}$	c	d	0
c	d	e	1
$\sqrt{d}$	c	b	0
e	b	c	1

(b) Original table

Present state	Next state $X=0$	Next state $X=1$	Present output
a	a	b	0
b	c	b	0
$\sqrt{c}$	b	e	1
$\sqrt{e}$	b	c	1

(c) Table after elimination of one row

Present state	Next state $X=0$	Next state $X=1$	Present output
a	a	b	0
b	c	b	0
c	b	c	1

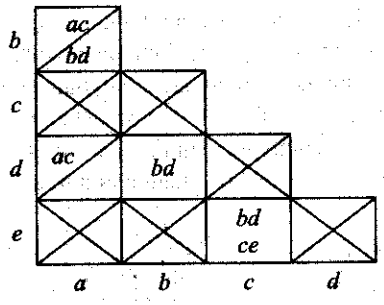
(d) Final reduced table after elimination of two rows

Fig. 11.17 Reduction by row elimination method

Solution

(i) Refer to state table of Fig. 11.17b obtained from state transition diagram. Comparing row *b* and *d* we see they are equivalent because that needs no other consideration except equivalence between themselves. Retaining *b* and replacing *d* by *b* in rest of the table we get Table of Fig. 11.17c. There we find *c* and *e* are equivalent and we retain *c* and replacing *e* by *c* get Fig. 11.17d. We see no further reduction is possible and final reduced state table that has three states.

(ii) Refer to state transition diagram and state table developed from it. Implication table is shown in Fig. 11.18. The non-compliance of output makes cross-points *de, be, ae, cd, bc, ac* non-equivalent and hence double crossed. From this we find *ab* and *ad* cannot be equivalent as that requires *ac* to be equivalent which is not true. Finally moving columnwise starting from *d* we get partition table and final partition P has three groups. Hence, the number of states is reduced to 3 from 5 by this technique.



$d : d$
$c : d\ (ce)$
$b : d\ (ce)\ (bd) \equiv (ce)\ (bd)$
$a : (ce)\ (bd)\ a$
$P = (ce)\ (bd)\ (a)$

Fig. 11.18 Reduction by implication table method

SELF-TEST

8. What is an ASM chart?
9. What is an implication table?
10. What is a partition table?
11. What is the usefulness of state reduction technique?

PART B: ASYNCHRONOUS SEQUENTIAL CIRCUIT

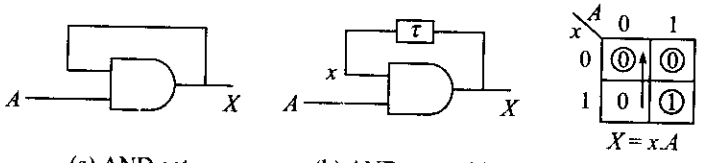
Asynchronous Sequential Circuit, also called Event Driven Circuit does not have any clock to trigger change of state. State changes are triggered by change in input signal. In clock driven circuit all the memory elements change their states together. In spite of all the advantages it offers, there are certain limitations with such circuit. The most important being the speed of operation. This is limited by the clock frequency since, state change can only take place at time $t = nT$, where T = Time period of clock signal and inverse of frequency and n is an integer. If the input changes in a manner that warrants change in the state, it cannot do that immediately and wait till the next clock trigger comes. Asynchronous sequential circuit is a solution to this however, design of such circuit is very complex and has several constraints to be taken care of, which is not required for synchronous circuit. Here, we shall discuss *fundamental mode* of operation of asynchronous sequential circuit where output change depends on change in input level. There is another type of such circuit called *pulse mode* where output change is affected by edge of the input pulse.

11.8 ANALYSIS OF ASYNCHRONOUS SEQUENTIAL CIRCUIT

As we have already noted memory is the most important element in sequential logic circuit. In synchronous system we use clock driven flip-flops which we cannot work here. This is done through feedback similar to basic latch portion of a flip-flop. Before we discuss that let us see how a two input AND gate and two input NAND gate behave with output fed back to one of the input. We shall use Karnaugh map for the analysis.

AND Gate

The two input AND gate with output fed back as one input is shown in Fig. 11.19a. The circuit can be redrawn as shown in Fig. 11.19b that includes the effect of propagation delay of the gate (say, τ), the finite time after which a gate reacts to its input. Thus, if X is current output obtained following logic relation and x is the feedback output we write, $x = X(t-\tau)$. The truth table is also called state table and each location in Karnaugh map a state of the asynchronous sequential circuit. Figure. 11.19c shows the truth table of given circuit and encircled states indicate stable condition of the circuit. For example, if $A = 0$ and by any reason previous output (that is currently fed back) $x = 0$ then, $X = x.A = 0.0 = 0$. After time $t = \tau$, x takes the value of X, i.e. 0 and because of that output X does not change. Thus, $x = 0, A = 0$ represents a stable state and is encircled. Similarly $x = 0, A = 1$ position and $x = 1, A = 1$ positions are also stable as in each of these cases $X = x$ and no change in output is necessary.



(a) AND gate (b) AND gate with propagation delay (c) Truth table

Fig. 11.19 Two input AND gate with output feedback

Let us now consider the following case. The circuit is at $x = 1, A = 1$, a stable state. Now A is made 1 and held at that value. How does the circuit react? First of all, following Karnaugh map the circuit moves one step left, i.e from $x = 1, A = 1$ to $x = 1, A = 0$ position because x, the feedback input takes finite time, τ to react. At this position $X = 0$. Therefore after time τ, x becomes 0, i.e. we move up by one position in Karnaugh map to $x = 0, A = 0$ position. Here, $X = 0$ and thus $x = X$ and as long as A does not change the circuit remains in this position, a stable one. Thus, we find $x = 1, A = 0$ position is unstable.

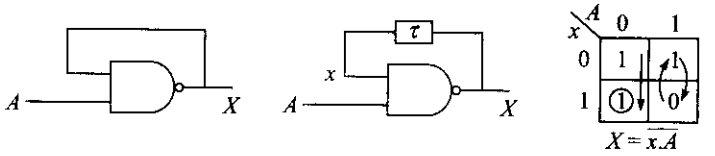
We make an important observation from this discussion which is universally true for asynchronous sequential circuit. *For any state, if $x = X$ then the circuit is stable and if $x \neq X$ it is unstable.*

NAND Gate

We extend the above observation to feedback NAND circuit shown in Fig. 11.20(a) and arrive at the Truth Table given in Fig. 11.20(c). It is interesting to note that for $A = 1$ there is no stable state and $x = X'$ for both $x = 0$ and $x = 1$. Thus there is oscillation between $x = 0, A = 1$ and $x = 1, A = 1$ state.

Two Input NAND Latch

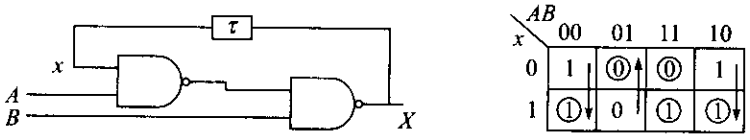
In analysis of asynchronous sequential circuit there is an important constraint to be followed. Though there can be more than one input feeding the circuit, *at a time only one input variable can change*. The other input



(a) NAND gate (b) NAND gate with propagation delay (c) Truth table

Fig. 11.20 Two input NAND gate with output feedback

can change only when the circuit is stabilized following the previous input change. The time required to stabilize the circuit is in the order of propagation delay of a gate, i.e. in nanosecond order. Similarly, if there are two or more output variables *only one output variable can change at any time instant*, as propagation delays in different paths are different. While analyzing the NAND latch given in Fig.11.21a we shall keep this in mind.



(a) NAND latch with propagation delay (b) Truth table

Fig. 11.21 Two input NAND latch

The stable and unstable states are arrived at (Fig.11.21b) following discussion in preceding section, i.e. for any given combination of x, A, B if, $X = x$, the circuit is stable otherwise not. Stables states are encircled and arrows show the movements from transient states. Now let us see how input changes affect the output. For each input combination the circuit has at least one stable state and this stable state will be the starting point of our discussion in each case.

Input AB Change from 00 to 01 The circuit moves from $xAB = 100$, a stable position to $xAB = 101$ (Note, x takes a time τ to react to a new set of input) which is unstable and then moves to $xAB = 001$, a stable state that has output 0. Therefore, a 00→01 transition in AB has output X making 1→0 transition.

Input AB Changes from 00 to 10 The circuit moves from $xAB = 100$, a stable position to $xAB = 110$, another stable state that has output 1. Therefore, a 00→10 transition in AB does not alter the value of output, $X = 1$.

Note that AB cannot change from 00 to 11 as there will be a finite delay, however small it may be between A and B change. Thus, the transition path of AB is either 00→01→11 (then output changes as 1→0→0) or 00→10→11 (output changes as 1→1→1) depending on which of A or B changes earlier. Therefore, output is 0 or 1 depending on intermediate value and in asynchronous logic design such transitions are not allowed.

Following this procedure, we look at other possible transitions of state (xAB) for input change and get transition Table 11.5. Note that, at $AB = 11$, there are two stable states $xAB = 011$ and $xAB = 111$. Transition of AB, 01→11 reaches $xAB = 011$ state while 10→11 reaches $xAB = 111$. Thus looking at output of the circuit when $AB = 11$ (also called idle input that does not force change) one can tell whether $AB = 01$ or 10 before AB becomes 11. Thus at $AB = 11$, the circuit generates output $x = X$ from memory or it has latched the

Table 11.5 Transition Table of NAND Latch

Input AB	*State(xAB) transition*		*Output X*		*Remark*
00→01	100→101→001		1→0→0		At $AB = 00$,
00→10	100→110		1→1		stable $x = 1$,
01→00	001→000→100		0→1→1		At $AB = 01$,
01→11	001→011		0→0		stable $x = 0$,
10→00	110→100,		1→1		At $AB = 10$,
10→11	110→111		1→1		stable $x = 1$,
11→01	011→001,	111→101→001	0→0,	1→0→0	At $AB = 11$,
11→10	011→010→101,	111→110	0→1→1,	1→1	stable $x = 0, 1$.

before AB becomes 11. Thus at $AB = 11$, the circuit generates output $x = X$ from memory or it has latched the information of previous input combination.

Example 11.6 (i) Analyze the Mealy model asynchronous sequential circuit of Fig. 11.22 and show its stable state and corresponding outputs. (ii) Give the state diagram of this circuit.

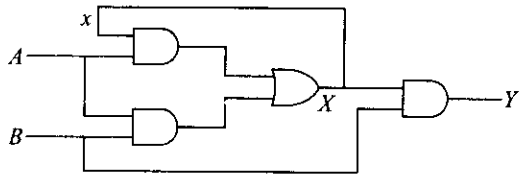
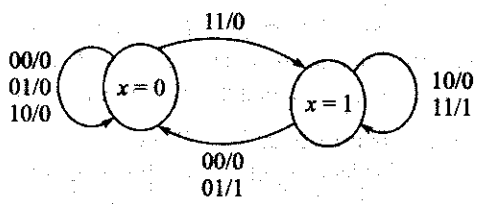


Fig. 11.22 An asynchronous sequential circuit: Mealy model

Solution To analyze the circuit we consider $x = X(t-\tau)$ where τ is the cumulative propagation delay from input side up to X. For all possible combinations of xAB we get X and Y following logic relation shown in the circuit and prepare Karnaugh map of Fig. 11.23a. States where $X = x$ are stable and encircled. Outputs corresponding to each state and input combination are shown beside. (ii) Since, there are two stable states $x = 0$ and $x = 1$ the state diagram can be drawn from Table 11.5 by considering all possible input combinations for each state as shown in Fig. 11.23b. Note that the output is dependent on inputs as well as state and is shown along the transition path beside the input.

x \ AB	00	01	11	10
0	⓪/ 0	⓪/ 0	1 / 0	⓪/ 0
1	0 / 0	0 / 1	①/ 1	①/ 0

(a)



(b)

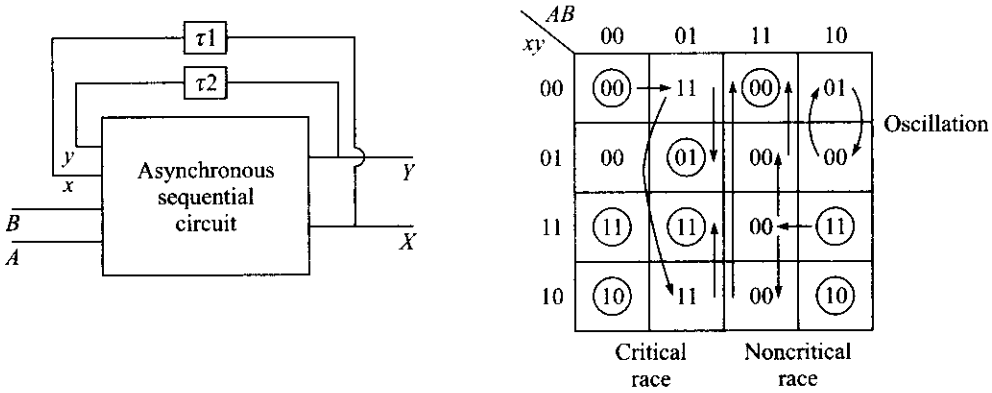
Fig. 11.23 (a) Karnaugh map, (b) State diagram for asynchronous circuit shown in Fig. 11.22

SELF-TEST

12. What is fundamental mode of operation of asynchronous sequential circuit?
13. If there are more than one input to such a circuit what constraint is imposed on them?

11.9 PROBLEMS WITH ASYNCHRONOUS SEQUENTIAL CIRCUITS

Before we go for design of asynchronous sequential circuit we would like to look into some important design related issues. These are non-issues in synchronous circuit where external clock trigger arrives after all the inputs are stabilized. Asynchronous circuit responds to all the transient values and problems like oscillation, critical race, hazards can cause major problem unless they are addressed at design stage. To explain these problems we take help of Truth Table shown in Fig.11.24 where the circuit has two external inputs A, B and two outputs X, Y. Both the outputs are fed back to the input side in the form of x and y but with different propagation delays. Thus x, y cannot change simultaneously but with time delays $\tau 1$ and $\tau 2$ respectively and we can write $x = X(t-\tau 1)$ and $y = Y(t-\tau 2)$.



(a) Block diagram (b) Truth table

Fig. 11.24 (a) Block diagram, (b) Truth table of a 2-input, 2-output circuit

Refer to the discussion in Section 11.8. The stable states are encircled in the circuit where $xy = XY$. But there are certain major problems with this truth table which we discuss in a future section.

Oscillation

Consider, the stable state $xyAB = 0000$, where $x = X$ and $y = Y$. If input AB changes from 00 to 10, the circuit goes to $xyAB = 0010$ state and then output $XY = 01$. This is a transient state because $xy \neq XY$. After time $\tau 2$, y takes the value of $Y = 1$ and the circuit goes to $xyAB = 0110$ where output $XY = 00$. This again is a transient state and after another propagation delay of $\tau 2$, the circuit goes to $xyAB = 0010$. Thus the circuit oscillates between state 0010 and 0110 and the output Y oscillates between 0 and 1 with a time gap $\tau 2$. In asynchronous sequential circuits for any given input, transitions between two unstable states like these are to be avoided to remove oscillation.

Critical Race

Next we discuss race condition that could be a major problem in asynchronous sequential circuit. This occurs when an input change tries to modify more than one output. In the truth table of Fig.11.24b, consider the stable state $xyAB = 0000$. Now, if AB changes to 01 the circuit moves to $xyAB = 0001$ where $XY = 11$. Now

depending which of $\tau 1$ and $\tau 2$ is lower, xy moves from 00 to either 01 or 10. If $\tau 1$ is lower, x changes earlier and the circuit goes to $xyAB = 1001$ which is a unstable state with output $XY = 11$ and $xy \neq XY$. The circuit next moves to state $xyAB = 1101$ which is a stable state and final output $XY = 11$. If $\tau 2$ is lower, y changes earlier and the circuit goes to $xyAB = 0101$, a stable state and the final output is 01. Thus, depending on propagation delays in feedback path, the circuit settles at two different states generating two different set of outputs. Such a situation is called critical race condition and is to be avoided in asynchronous sequential circuit.

Race can be non-critical too, in which case its presence does not pose any problem for the circuit behavior. In the truth table, consider stable state $xyAB = 1110$. If input AB changes to 11, the circuit goes to $xyAB = 1111$ where output $XY = 00$. Note that both the output variables are supposed to change which cannot happen. Again depending on propagation delays xy becomes either 01 or 10. If $xy = 01$ then the circuit moves to $xyAB = 0111$ and then to 0011 and settles there. If $xy = 10$ then the transition path is $1111 \rightarrow 1011 \rightarrow 0011$. In both the cases final state is 0011 and output is 00. Since, the race condition does not lead to two different state it is termed as non-critical race.

Hazards

Static and dynamic hazards causes malfunctioning of asynchronous sequential circuit. Situations like $Y = A + A'$ or $Y = AA'$ are to be avoided for any input output combination with the help of hazard covers in truth table. A detailed discussion on how to avoid hazard appears in Section 3.9. In circuit with feedback even when these hazards are adequately covered there can be another problem called *essential hazard*. This occurs when change in input does not reach one part of the circuit while from other part one output fed back to the input side becomes available. Essential hazard is avoided by adding delay, may be in the form of additional gates that does not change the logic level, in the feedback path. This ensures effect of input change propagates to the all parts of the circuit and then only feed back output, generated from that input-change makes its presence felt.

Example 11.7 In an asynchronous sequential circuit, the state variable outputs of X and Y are related with primary inputs A and B and its own feedback x and y as shown in Karnaugh map of Fig. 11.25. Can the circuit face any problem in its operation?

Solution Yes, the circuit may face problem in its operation. When the circuit is at stable state $xyAB = 1111$ and input AB changes from $11 \rightarrow 10$ the circuit oscillates between $xyAB = 1110$ and $xyAB = 1010$. Also there can be a critical race problem if at stable state $xyAB = 0001$, input AB change from 01 to 00. The circuit may settle at $xyAB = 0100$ or $xyAB = 1000$ depending on which of x and y changes first at the feedback path. Non-critical race situation occurs if at stable state $xyAB = 0010$ the input AB change from 10 to 00.

xy \ AB	00	01	11	10
00	11	(00)	11	(00)
01	(01)	11	11	(01)
11	10	11	(11)	10
10	(10)	(10)	11	11

Fig. 11.25 Karnaugh map for Example 11.7

SELF-TEST

14. What is racing? What is the difference between critical and non-critical race?
15. What is essential hazard?

11.10 DESIGN OF ASYNCHRONOUS SEQUENTIAL CIRCUIT

The discussions in previous sections show several design constraints for asynchronous sequential circuit. This makes the design of such circuit complex and cumbersome and if benefits like speed are not of critical importance, synchronous design is preferred to asynchronous design. In this section we explain the design steps of asynchronous sequential circuit through an example. The problem we attempt to solve is described next.

The Problem

A digital logic circuit is to be designed that has two inputs A, B and one output X. X goes high if at $A = 1$, B makes a transition $1 \rightarrow 0$. X remains high as long as this $A = 1$, $B = 0$ are maintained. If any of A or B changes at this time output X goes low. It becomes high again when at $A = 1$, B goes from 1 to 0. The timing diagram corresponding to this problem is shown in Fig. 11.26.

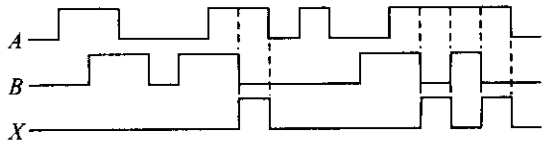


Fig. 11.26 Timing diagram of the problem

State Transition Diagram

From the problem statement we first develop a state transition diagram, say using Moore model. The state symbol and output at that state is shown together within a circle in this diagram (Fig. 11.27). Let the initial state be considered as a when $AB = 00$ with output 0. As long as AB remains 00, the circuit remains at a. Note that in synchronous sequential circuit between two clock trigger input might change but the state of the circuit remains same. Here, as soon as one of A or B changes the circuit may immediately move to different states. Note that A and B cannot change together, a constraint we have to adhere to in asynchronous design. At state a, if input $AB = 01$, the circuit goes to state b and if input $AB = 10$ it goes to c ($00 \rightarrow 11$ prohibited). Both b and c generate output 0 as they have not yet fulfilled the condition stated in the problem for assertion of output. The circuit remains at b for $AB = 01$. If AB changes to 11, the circuit moves to state d with output $X = 0$ and if AB becomes 00 the circuit goes back to a. Similarly the circuit stays at c if input stays at 10 but goes to d receiving 11 and to a receiving 00. Note that at state d, the input $AB = 11$ and now if $B \rightarrow 0$ then condition for output $X = 1$ is fulfilled and next state e for $AB = 10$ shows output as 1. The circuit remains at state e as long as $AB = 10$. It goes back to state d if AB becomes 11 because if B again goes to 0 output should be high. However at e, if AB changes to 00 the circuit goes to initial state a as AB becoming 10 following 00 will not assert output.

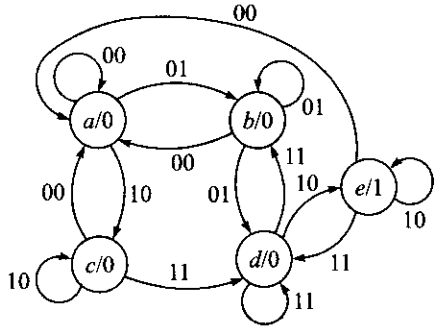


Fig. 11.27 State transition diagram of the problem

Primitive Table

The next step is to form state table from state transition diagram. In this table if all the rows representing a state has only one stable state for all possible input combinations and it is termed as *primitive table*, or *primitive flow table* or simply *flow table*. Often we can skip step one and directly go to primitive table from problem statement. Primitive table prepared from state transition diagram is shown in table Fig. 11.28.

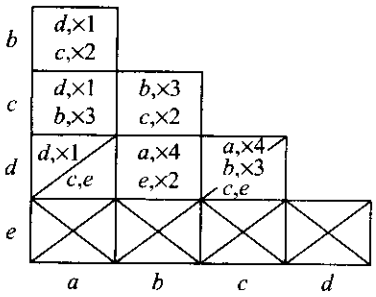
Note that each row in this table has one don't care state. The don't care state in each row comes which asks for both the input variables to change to move from stable state, a condition not allowed in asynchronous sequential logic. The don't care states have been given suffix like 1, 2 which is not a must. However, this helps in next step where we check state redundancy.

	AB: 00	01	11	10	X
a	(a)	b	×1	c	0
b	a	(b)	d	×2	0
c	a	×3	d	(c)	0
d	×4	b	(d)	e	0
e	a	×5	d	(e)	1

Fig. 11.28 Primitive table for the problem

State Reduction

It is always useful to check state redundancy before going for actual circuit design. Removing redundant states helps in generating the circuit in a simpler way and with less hardware. We use implication table for this example to remove redundant state, if there is any. The implication table, drawn from primitive table and state reduction is shown in Fig. 11.29. For preparation of implication table refer to discussion in Section 11.7. Note that in asynchronous design, when there are don't care states in state table, this is called incompletely specified table. For this, state reduction can be done as follows.



$d : d$

$c : cd$

$b : cd\ (bc) \equiv d\ (bc)$ $\quad \times 2 \equiv c, \times 3 \equiv b$

$a : d\ (bc)\ (ab)\ (ac) \equiv (abc)$ $\quad \times 1 \equiv d$

$P = (abc)\ (d)\ (e)$

Fig. 11.29 State reduction by implication table

Since state e cannot be equivalent with any of a, b, c and d (output being different) we put double cross right in the beginning for row e of implication table. Next we find a and d cannot be equivalent as that requires c and e to be equivalent which is not. Similarly, c and d cannot be equivalent and we cross these two places in row d of implication table with single line. Now let us try to find equivalence by moving along columns. In column d and c there is no equivalence possible. In column b, equivalence between either of b and c or b and d is possible. But, both (bc and bd) equivalences are not possible as it requires don't care state ×2 to be made equivalent to c and e while ce themselves are not equivalent. In column a, we see (c, ×2) equivalence

may make a and b equivalent. Therefore, from column b, we get (bc) equivalence by making $c, \times 2$ and $b, \times 3$ equivalent. In column a, by assigning $\times 1$ to d we can make (ab) and (ac) equivalent and there is no conflict with assignment of don't care states. Since, $(bc)\ (ab)\ (ac) \equiv (abc)$ partition table has three different groups (abc), (d) and (e). Thus the states are reduced to 3 from original 5. Let state a represent the group (abc). Now the reduced state table is as shown in Fig. 11.30a and reduced state transition diagram in Fig. 11.30b.

State Assignment

This step in asynchronous sequential circuit design has to be done very carefully so that a valid state transition does not require two or more output variables to change simultaneously which may lead to racing problem. In this problem there are three states in the reduced state diagram which needs two variables to represent them. Figure 11.30 shows that we cannot avoid two variables changing together in one or more occasions for the reduced state transition diagram. If $\{a,d,e\}$ is represented by $\{00,01,10\}$ it occurs twice for $d \rightarrow e$ and $e \rightarrow d$ transitions in state transition diagram. A representation of $\{00,01,11\}$ requires two variables to change only once when $e \rightarrow a$ transition occurs. The solution to this may be found if the unused fourth combination of two variable representation is used as a dummy state, say ϕ. We include ϕ between e and a. Note that, if one dummy state is not enough we may need to use a third variable to represent the states that will make $2^3-3 = 5$ dummy variable available for this purpose. Let us represent the states in this problem by two variables PQ in the following way

$$a: 00 \quad d: 01 \quad e: 11 \quad \phi: 10$$

The modified state diagram and state table with dummy variable $\phi = 10$ included are shown in Fig. 11.31. Note that ϕ is an unstable state and before the input can change it goes to next stable state a. We represent state variables by P and Q, the corresponding feedback variables are represented by p and q respectively.

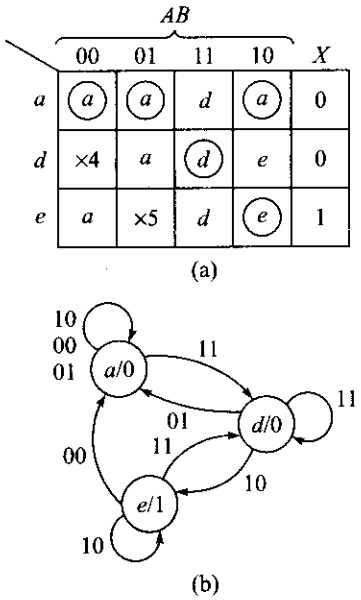


Fig. 11.30 **Reduced (a) State table, (b) State transition diagram**

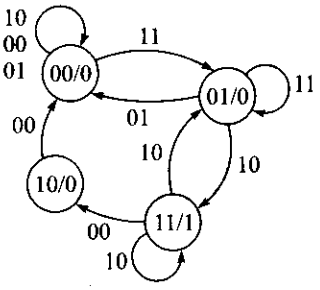


Fig. 11.31 **Modified state transition diagram**

Design Equations and Circuit Diagram

We use Karnaugh map to get expression of state variables P and Q as a function of input A,B and feedback variables p and q. The equations derived from Karnaugh map are shown in Fig. 11.32. The equation of output X is generated from P and Q as we use Moore model. The final circuit is developed from these equations and is shown in Fig. 11.33.

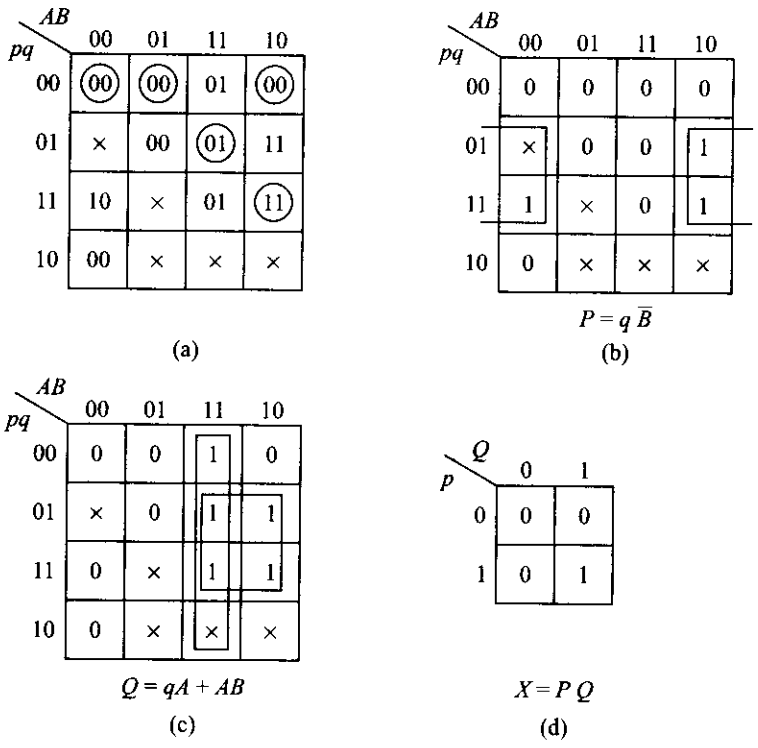


Fig. 11.32 **(a) Reduced state diagram from Fig. 11.27, (b)–(d) Karnaugh map and design equations**

It is left to the reader to analyze this circuit and verify the timing diagram shown along with the problem statement. Now, that we have seen all the steps in asynchronous sequential logic design we are in a position to appreciate how complex the process is compared to synchronous sequential logic design. Thus the later is preferred if issues like speed, clock skew, etc. are not of critical importance.

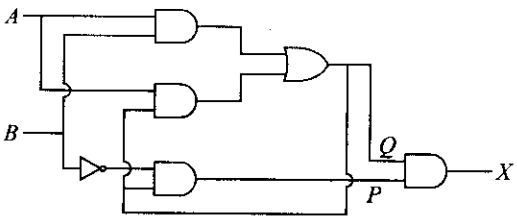


Fig. 11.33 **Circuit diagram of asynchronous sequential logic for the problem in Fig. 11.23**

Example 11.8 Design an asynchronous sequential logic circuit for state transition diagram shown in Fig. 11.34.

Solution Let the two input variables be termed A and B in order. Figure 11.35a shows state table through Karnaugh map. Since the state transition diagram has two states we need one ($\log_2 2$) output feedback serving as memory. Let the output variable be termed X and its feedback x. If we represent current state a as $x = 0$ and b as $x = 1$ then output

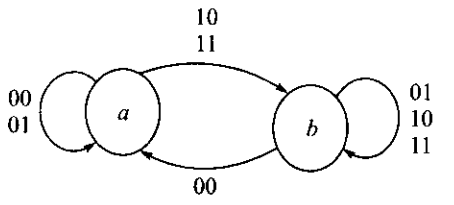


Fig. 11.34 **State transition diagram for Example 11.8**

X can be expressed as shown in Fig. 11.35b. The asynchronous sequential logic circuit drawn from design equation is shown in Fig. 11.35c.

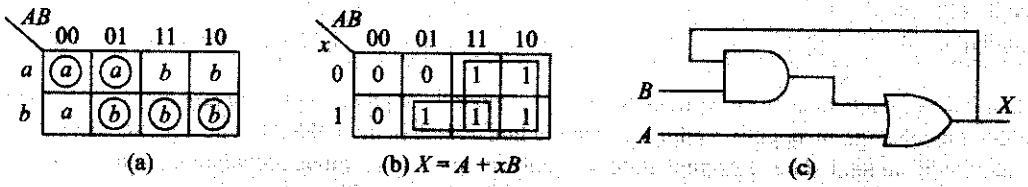


Fig. 11.35 Solution for Example 11.8

SELF-TEST

16. What is a primitive flow table?
17. What is an incompletely specified table?
18. What is a dummy variable?
19. What is the advantage and disadvantage of asynchronous over synchronous sequential circuits?

11.11 FSM IMPLEMENTATION IN HDL

We shall conclude our discussion on HDL showing how one can represent a Finite State Machine (FSM) in HDL. We take up the Mealy Model shown in Fig. 11.2b as illustration and the code is given next. Note that, we have introduced two variables Clock and Reset, which is not explicit in the figure. The clock is used for synchronous state transition of the circuit (at negative edge) and active low asynchronous Reset is used to initialize the circuit to state a.

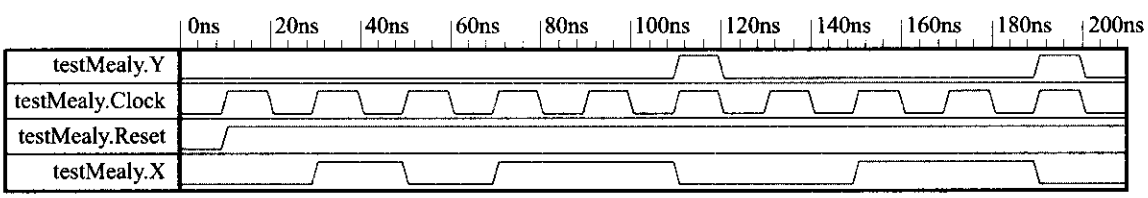
```
module MealyFSM(X, Clock,Reset,Y);
input X,Clock,Reset;
output  Y;
reg Y;
reg [1:0] PS,NS; //PS represents Present State, NS Next State
parameter a=2'b00, b=2'b01, c=2'b10;  //States shown in fig. given binary value
always @ (negedge Clock or negedge Reset) //Reset or state change
  if (~Reset) PS=a;
  else PS=NS;
always @ (PS or  X) //Determines next state
  if (PS==a && X==0) NS=a;
  else if (PS==a && X==1)  NS=b;
  else if (PS==b && X==0)  NS=a;
  else if (PS==b && X==1)  NS=c;
  else if (PS==c && X==0)  NS=a;
  else if (PS==c && X==1)  NS=c;
```

```
always @ (PS or  X)  //Determines output which is dependent both on present
  if (PS==c && X==0) Y=1; //state and present input as Mealy Model
  else Y=0;
Endmodule
```

We have defined each state by an equivalent binary number through keyword **parameter**. The first **always** block does state change at negative edge of clock when Reset is held HIGH. The second **always** block decides what will be next state if current state and current input is in some combination. The third always block decides output based on current state and current input. Note that all these assignments follow the conditions stated in FSM Mealy model of Fig. 11.2b.

Now let us try to test this circuit by feeding an input X = 01011001101(first value of X fed is 0 as if data is coming from right) by creating a test bench and appending it to above code. Let us verify whether the circuit can detect pattern '110' (as data is considered to come from right) and generate appropriate output. The following test bench can generate such pattern and the output is plotted against clock and input X in the subsequent timing diagram, obtained from Verilog simulation.

```
module testMealy();
reg Clock, Reset, X;
wire Y;
initial
begin  Reset = 0; //Initial value of reset=0, this resets the FSM and PS=a.
X=0;  // input is 0 at start
#10 Reset = 1;
#20 X=1;  #20 X=0;  #20 X=1;  #20 X=1;  //Change in input data at
#20 X=0;  #20 X=0;  #20 X=1;  #20 X=1;  //odd multiple of 10 ns
#20 X=0;  #20 X=1;                      //starting from 30 ns.
end
// Clock generator follows
initial
   begin
  Clock = 1'b0;
  repeat (21)
  #10 Clock = ~Clock; //Clock inverts at every 10ns so that
   end      //negative edge of clock comes at even multiple of 10 ns
MealyFSM MFSM(X,Clock,Reset,Y);
endmodule
```



Find from the timing diagram $X = 0$ up to 30 ns from start and remains 1 till 50 ns and so on. Clock negative edge comes at 20 ns, 40 ns etc. When X remains 1 at two negative edges of clock and a 0 follows like in between 110–120 ns $Y = 1$ and similarly between 190–200 ns. $Y = 0$ elsewhere and this is what the FSM is supposed to perform, i.e. detect '110' (data from right, if from left '011') that is input bit 1, followed by another 1 followed by 0.

Example 11.9 Give Verilog HDL description of the Moore Model shown in Fig. 11.2a

Solution

```
module MooreFSM(X, Clock,Reset,Y);
input X,Clock,Reset;
output  Y; reg Y;
reg [1:0] PS,NS; //PS represents Present State, NS Next State
parameter a=2'b00, b=2'b01, c=2'b10, d=2'b11; //Four states given binary value
always @ (negedge Clock or negedge Reset) //Reset or state change
   if (~Reset) PS=a;
   else PS=NS;
always @ (PS or  X) //Determines next state
   if (PS==a && X==0) NS=a;
   else if (PS==a && X==1)  NS=b;
   else if (PS==b && X==0)  NS=a;
   else if (PS==b && X==1)  NS=c;
   else if (PS==c && X==0)  NS=d;
   else if (PS==c && X==1)  NS=c;
   else if (PS==d && X==0)  NS=a;
   else if (PS==d && X==1)  NS=b;
always @ (PS)  //Determines output which is dependent only  on
   if (PS==d) Y=1; //present state due to Moore Mealy Model
   else Y=0;
endmodule
```

We conclude our discussion on HDL here. The objective had been to make one get started with basics of HDL design. Dedicated books and courses deal with this subject in greater details. One should note that free or student version of Verilog compliler has limited ability and trial full versions are free only for the trial period. Also the hardware device on which the design is exported is not cheap. It is thus not useful for simple design problem except for finding functional error through simulation. But it definitely is cheaper and convenient if one considers a large complex design problem. The hardware devices commonly used to load HDL codes are discussed in Section 13.6 of Chapter Memory.

PROBLEM SOLVING WITH MULTIPLE METHODS

Problem A part of the simplistic digital control unit of a hypothetical Automatic Teller Machine (ATM) works like this. The ATM senses ATM card insertion by assertion of an input I. A correct typing of Personal Identification Number (PIN) is sensed by P. Transaction is done by asserting

TR and card return by *CR* is if $P = 1$. If the process is cancelled by pressing push button 'Return' that asserts *R*, transaction does not take place but card is returned. If not cancelled, the user gets two more opportunities to enter PIN. But third incorrect entry locks the card by asserting *CL*.

Solution The input and output variables assertions are as follows.

Input:

Card inserted $I = 1$, Card not inserted $I = 0$.
'Return' button pressed $R = 1$, 'Return' button not pressed $R = 0$.
PIN correctly entered $P = 1$, PIN incorrectly entered $P = 0$.

Output:

Card to be returned $CR = 1$, Card not to be returned $CR = 0$.
Transaction allowed $TR = 1$, Transaction not allowed $TR = 0$.
Card to be locked $CL = 1$, Card not to be locked $CL = 0$.

In Method-1, we make use of Moore Model and ASM chart which is shown in Fig. 11.36.

We find there are six rectangular boxes or six states (*a* to *f*). Thus, it requires three flip-flops which can handle up to eight states. The three inputs and three present states of flip-flops would require total 3 + 3 = 6 variable Karnaugh Map for design. The ROM-Delay flip-flop approach may thus be preferred.

Let the state assignments of three flip-flops, say *C*, *B*, and *A* be as follows

$a : CBA = 000$, $b : CBA = 001$, $c : CBA = 010$,
$d : CBA = 011$, $e : CBA = 100$, $f : CBA = 101$.

Since, it is a Moore Model, the output corresponding to each state are

$a : CR = 0, TR = 0, CL = 0$ $b : CR = 0, TR = 0, CL = 0$
$c : CR = 0, TR = 0, CL = 0$ $d : CR = 0, TR = 0, CL = 1$
$e : CR = 1, TR = 0, CL = 0$ $f : CR = 1, TR = 1, CL = 0$

Then the state transition table can be as given in Fig. 11.37.

Three present states and three inputs call for a 6 to 64 decoder. Three next states require a storage of three bits in each ROM address. Thus this implementation will require 3 flip-flops, one 6 to 64 decoder and one 64×3 ROM.

The outputs are derived from present state, i.e. flip-flops in a Moore Model. The combinatorial circuit required for this is arrived from Karnaugh Maps presented in Fig. 11.38.

The final realization is shown in Fig. 11.39.

Note that the requirement for decoder, ROM, etc. can be reduced noting that not all 64 combinations are required for the solution, e.g., if $I = 0$, no matter what the other values are, the circuit remains in state *a*.

In Method-2, this is ASM chart based approach targeting a Mealy Model. The output is asserted as soon as the condition is fulfilled. Figure 11.40 shows the ASM chart.

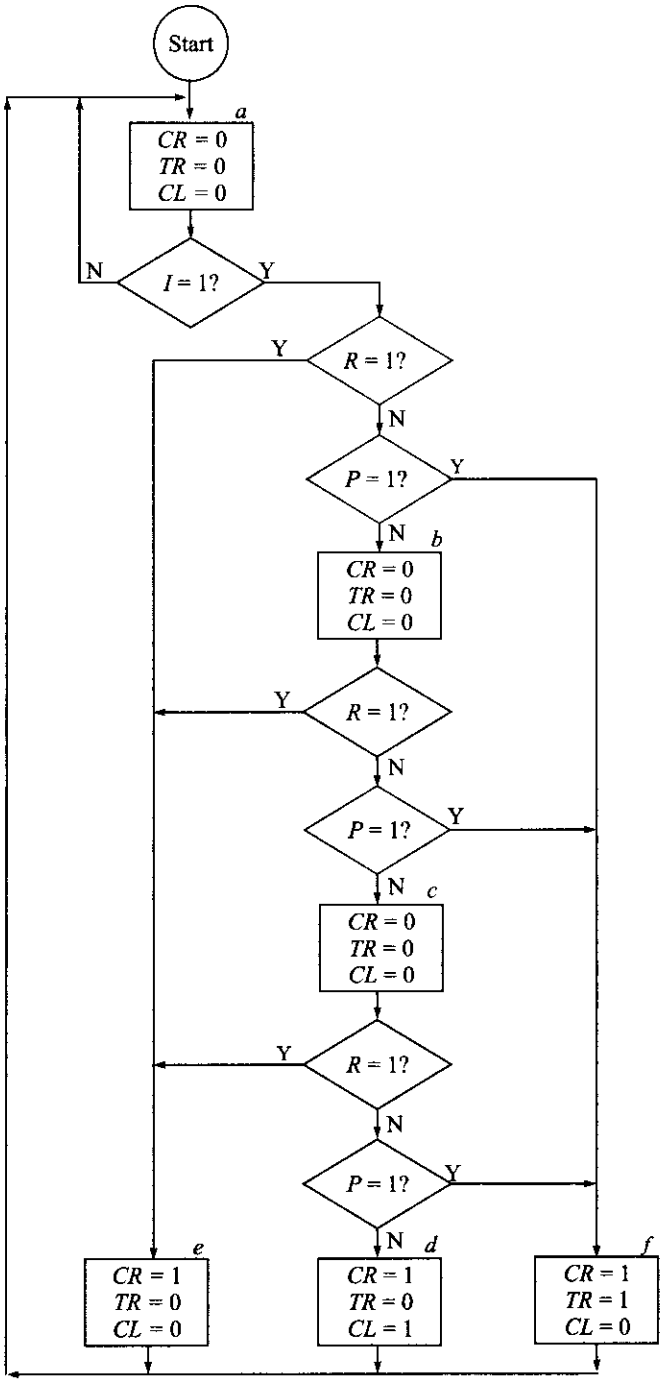


Fig. 11.36 ASM chart for Moore Model: Solution using Method-1

	Present State			Input			Next State		
	C_n	B_n	A_n	I	R	P	C_{n+1}	B_{n+1}	A_{n+1}
a:	0	0	0	0	X	X	0	0	0
a:	0	0	0	1	0	0	0	0	1
b:	0	0	1	1	1	X	1	0	0
b:	0	0	1	1	0	1	1	0	1
b:	0	0	1	1	0	0	0	1	0
c:	0	1	0	1	1	X	1	0	0
c:	0	1	0	1	0	1	1	0	1
c:	0	1	0	1	0	0	0	1	1
d:	0	1	1	X	X	X	0	0	0
e:	1	0	0	X	X	X	0	0	0
f:	1	0	1	X	X	X	0	0	0

Fig. 11.37 State Transtion Table for Moore Model: Solution using Method-1

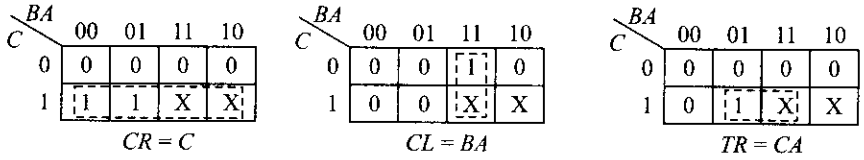


Fig. 11.38 Combinatorial Logic for Moore Model: Solution using Method-1

We find there are three rectangular boxes or three states (a to c). Thus it requires two flip-flops which can handle up to four states. We continue with the ROM-Delay flip-flop approach.

Let the state assignments of two flip-flops, say B and A be as follows

$$a: BA = 00, \qquad b: BA = 01, \qquad c: BA = 10$$

Then the state transition table can be as given in Fig. 11.41.

Two present states and three inputs call for a 5 to 32 decoder. Two next states require storage of two bits and in each ROM address. The three outputs can also be directly generated from ROM which require additional three bits in each location. Thus this implementation will require 2 flip-flops, one 5 to 32 decoder and one 32×5 ROM. The final realization is shown in Fig. 11.42.

Other Methods: As already mentioned, both Moore and Mealy Model can be realized by flip-flops and combinatorial circuits, as done for vending machine problem in Fig. 11.11. But this becomes cumbersome when the problem is relatively more complex with Karnaugh Map requiring solution for more than four variables. But then, QM algorithm can be used for which computer code also exists. The state transition diagram is avoided here as for larger number of input variables, every node will have too many branches spreading out, making the diagram very complex.

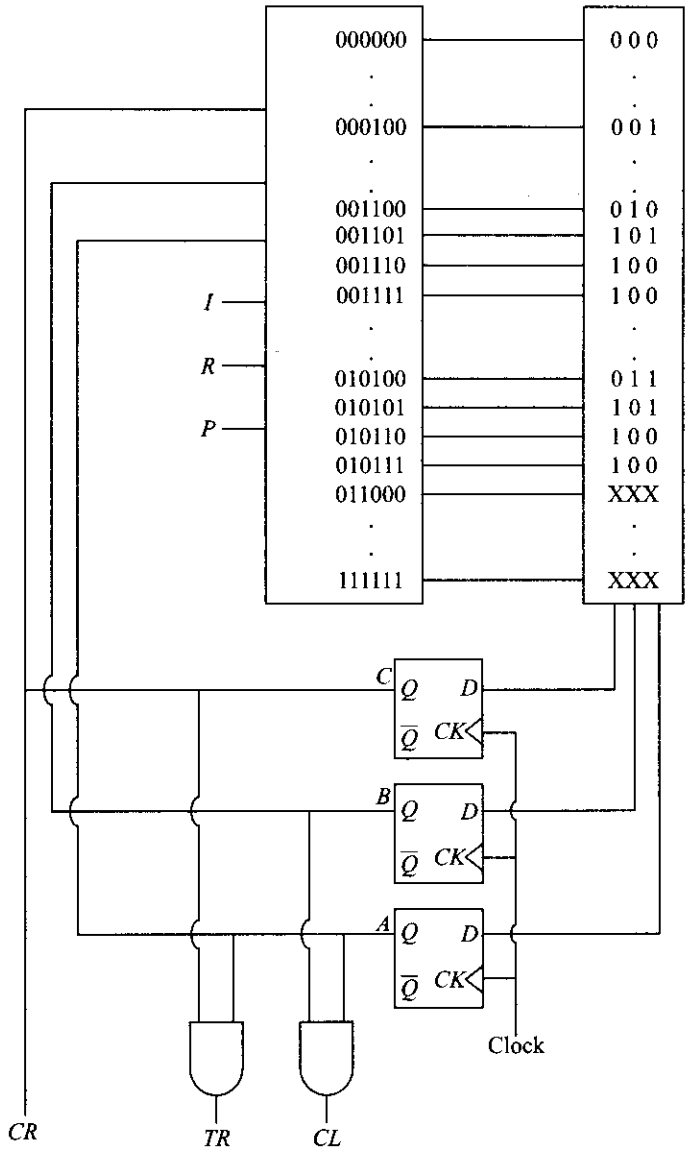


Fig. 11.39 **Realization using ROM: Solution using Method-1**

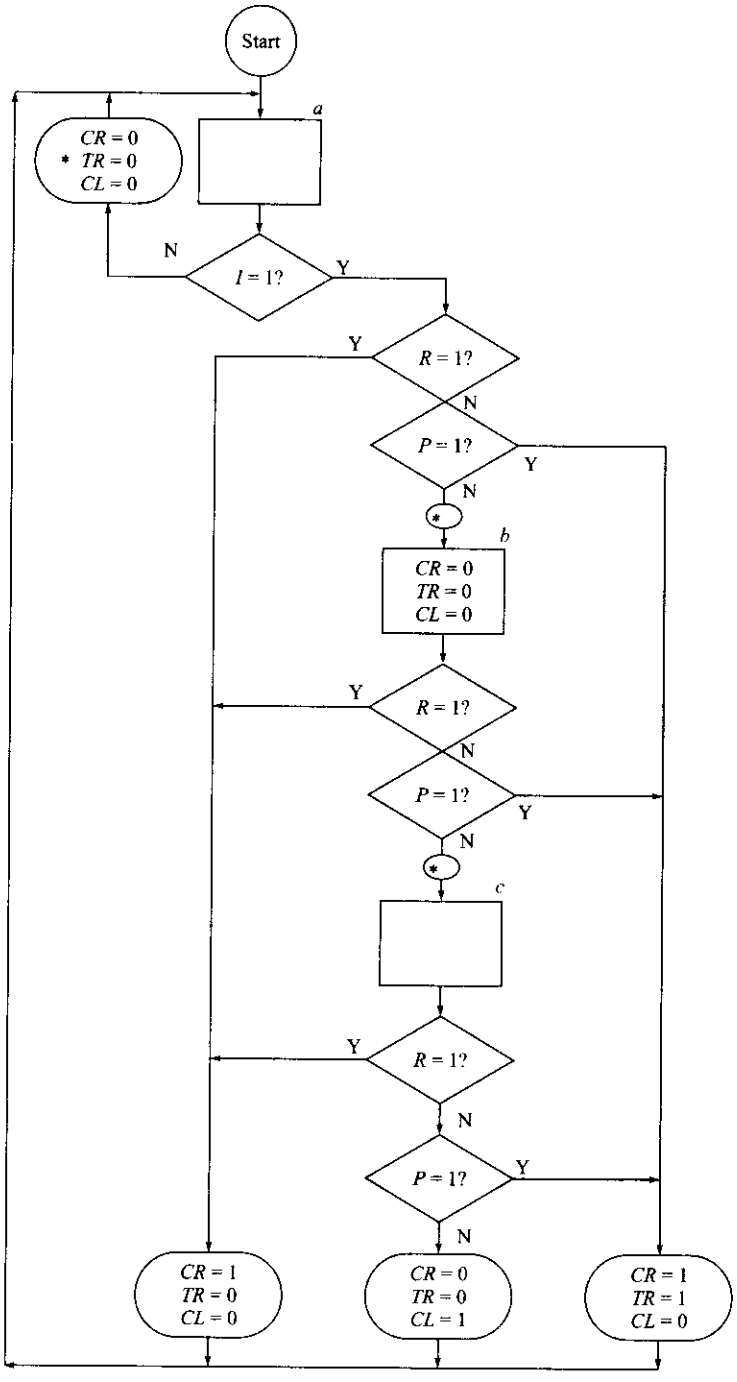


Fig. 11.40 ASM chart for Mealy Model: Solution using Method-2, *represents *CR* = 0, *TR* = 0, *CL* = 0

	Present State		Input			Next State		Output		
	B_n	A_n	I	R	P	B_{n+1}	A_{n+1}	CR	TR	CL
a:	0	0	0	X	X	0	0	0	0	0
a:	0	0	1	0	0	0	1	0	0	0
a:	0	0	1	0	1	0	0	1	1	0
a:	0	0	1	1	X	0	0	1	0	0
b:	0	1	1	0	0	1	0	0	0	0
b:	0	1	1	0	1	0	0	1	1	0
b:	0	1	1	1	X	0	0	1	0	0
c:	1	0	1	0	0	0	0	0	0	1
c:	1	0	1	0	1	0	0	1	1	0
c:	1	0	1	1	X	0	0	1	0	0

Fig. 11.41 State Table for Mealy Model: Solution using Method-2

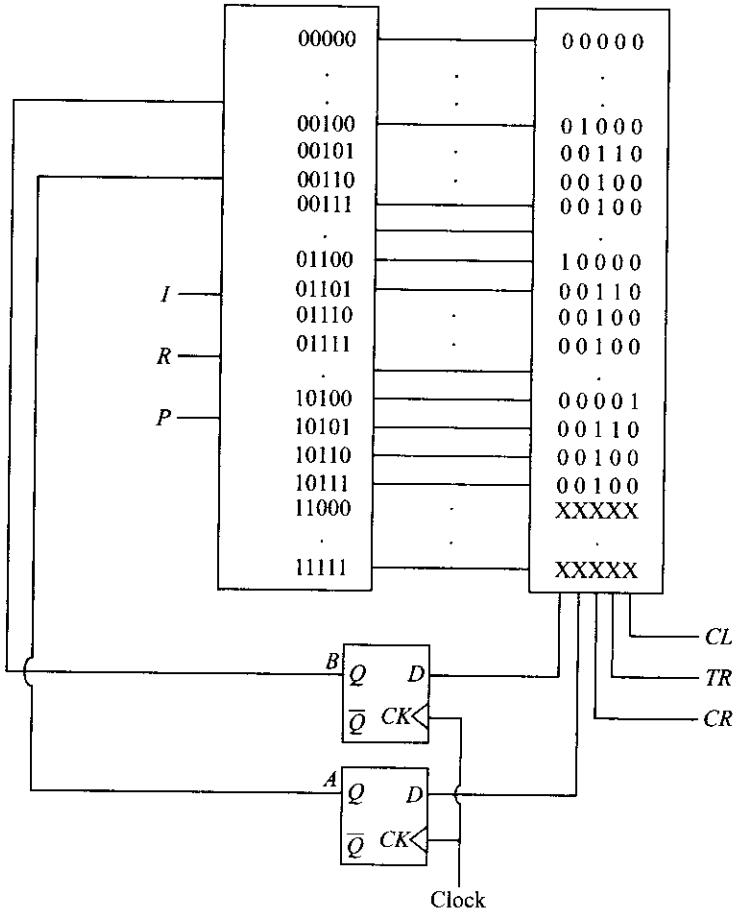


Fig. 11.42 Realization using ROM: Solution using Method-1

SUMMARY

The most popular sequential logic circuit design uses an external clock for triggering and the state changes occur synchronously with clock. In Moore model, such design output is derived directly from state outputs, also called secondary outputs. In Mealy model, output is generated from primary or actual input to circuits and secondary inputs. Word or timing description of a problem is first converted to state transition diagram or ASM chart followed by state synthesis table. In one implementation circuit can be realized using any type of flip-flop using flip-flop excitation table and excitation map. In other ROM and delay flip-flops are used where ROM stores the next state information. State reduction techniques are used to remove redundant states thereby can reduce the circuit complexity. Asynchronous system is not dependent on any external clock thus operates at a higher speed. But the circuit reacts to any and every change in the inputs and are prone to problems like racing, oscillations and different types of hazards. Design of such circuits are much more difficult compared to synchronous circuit and are attempted only for time critical applications where a very fast response to any input change is required.

GLOSSARY

- ***ASM chart*** a flow chart describing state transition with timing information
- ***asynchronous sequential circuit*** not synchronous with any external clock
- ***critical race*** leads to two different outputs of circuit depending on which feedback variable changes earlier
- ***dummy variable*** an additional variable preventing simultaneous change of two state variables in asynchronous sequential circuit
- ***essential hazard*** a condition when following an input change, one feedback variable tries to change the output before the other part of the circuit could respond to change in input.
- ***excitation map*** relationship that gives design equation for flip-flop inputs
- ***incompletely specified table*** state table where some of the next state or output or both remain unspecified.
- ***Mealy model*** where output depends both on state variable and input
- ***Moore model*** where output depends only on state variables
- ***non critical race*** leads to same output irrespective of propagation delay in a race condition
- ***oscillation*** circuit moving between two unstable states
- ***primitive flow table*** that directly maps state transition diagram in a state table where each row has only one stable state.
- ***racing*** a condition when more than one feedback variables try to change its value
- ***ROM*** Read Only Memory
- ***state*** memory values of a sequential circuit
- ***state transition diagram*** depicts state transition of a circuit pictorially
- ***synchronous sequential circuit*** works synchronously with external clock trigger
- ***state synthesis table*** state transition and flip-flop input description that leads to synthesis of sequential logic circuit

PROBLEMS

Section 11.1 and Section 11.2

11.1 Draw state transition diagram of synchronous sequential logic circuit using Mealy model that detects three consecutive zeros from an input data stream, X and signals detection by making output, $Y = 1$.

11.2 Convert Mealy model of Problem 11.1 to Moore model using conversion rules.

11.3 Using Moore model draw state transition diagram of a serial parity checker circuit. If the number of '1's received at input X is even, parity checker output, $Y = 0$. If odd number of '1's are received at X then $Y = 1$.

11.4 Convert Moorem model of Problem 11.3 to a Mealy model.

11.5 Using Moore model draw state transition diagram of the circuit that generates a single pulse of width equal to clock period when enabled by $E = 1$. The circuit is reset by $E = 0$ at any stage.

11.6 Draw state transition diagram of sequence detector circuit that detects '1101' from input data stream using both Mealy and Moore model.

Section 11.3 and Section 11.4

11.7 For Mealy model state transition diagram of sequence detector problem shown in Fig. 11.2b use following state assignment and get corresponding state synthesis table for JK flip-flop based solution.

a: $B = 0, A = 0$ b: $B = 0, A = 1$

c: $B = 1, A = 1$

11.8 For Mealy model state transition diagram of sequence detector Problem shown in Fig. 11.2b use following state assignment and get corresponding state synthesis table for JK flip-flop based solution.

a: $B = 0, A = 0$ b: $B = 0, A = 1$

c: $B = 1, A = 1$ d: $B = 1, A = 0$

11.9 Give design equations for Problem 11.7. Compare this with solution given in Section 11.4.

11.10 Give design equations for Problem 11.8. Compare this with solution given in Section 11.4.

11.11 Give design equations for Problem 11.1 for implementation with D flip-flops.

11.12 Implement circuit diagram for Problem 11.1 using JK flip-flops.

11.13 How many memory elements are necessary for Mealy and Moore models in sequence detector Problem 11.6.

11.14 Implement (a) parity checker circuit of Problem 11.3 and (b) single pulse generator circuit of Problem 11.5.

Section 11.5 to 11.7

11.15 Show how using an additional column in ROM the combinatorial circuit of Fig. 11.7 for sequence detector problem can be dispensed with.

11.16 Implement ROM based solution for Problem 11.6 where output is directly derived from ROM.

11.17 In vending machine problem of Section 11.6 we want to add an additional function. We give the customer an option to get back the coins he has deposited if he finds himself short of money or changes his mind midway. However, this function does not work if the cost of the product is reached. A push button switch, P is used for this which when pressed generates $P = 1$ and returns the coin deposited thus far by activating $C = 1$. Show what changes in ASM chart of Fig. 11.10 are necessary for this.

11.18 Draw ASM chart for Problem 11.5 and implement the circuit using ROM.

11.19 Find the minimum number of states necessary to represent following state table both by row elimination and implication table method.

Table 11.6

Present State	Next State		Present Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	f	d	0	1
b	c	f	1	1
c	f	b	1	1
d	e	g	1	1
e	a	d	1	1
f	g	b	0	1
g	a	d	0	1

11.20 Reduce following state table using implication table method.

Table 11.7

Present State	Next State		Present Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	h	c	1	0
b	c	d	0	1
c	h	b	0	0
d	f	h	0	0
e	c	f	0	1
f	f	g	0	0
g	g	c	1	0
h	a	c	1	0

Section 11.8

11.21 State the condition of stability in asynchronous sequential logic.

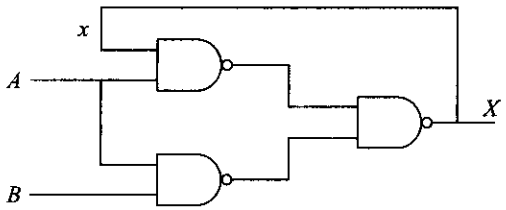
11.22 One of the two inputs of a two input NOR gate is fed back from the output. Write its state stable and encircle stable states, if any.

11.23 For state table in Problem 11.30 show the stable states, if any.

11.24 For state table in Problem 11.30 show how the circuit behaves when $xy = 11$ and A changes as $1 \rightarrow 0$.

11.25 There are three inputs A, B and C to an asynchronous sequential logic system. If $ABC = 111$ at any given time write the allowed combination of inputs that can follow.

11.26 Draw state table of adjacent asynchronous sequential logic circuit.



Section 11.9

11.27 When does oscillation occur in an asynchronous sequential logic circuit?

11.28 How can essential hazard be prevented in asynchronous sequential logic circuit?

11.29 There are two inputs A, B and three feedback outputs x, y and z of an asynchronous sequential logic system. If $xyzAB = 10011$ gives a stable state and input AB changes as $11 \rightarrow 10$, which of the following next state does not give racing problem – 10110, 00110, 11010, 00010 and 11110?

11.30 Find out potential problems in following state table where A is input and x and y are output feedbacks.

A \ xy	00	01	11	10
0	01	00	10	10
1	00	01	11	01

Section 11.10

11.31 The T flip-flop has a single input T, and single output Q. For $T = 0$, output does not change. For $T = 1$, output complements and remains at that value as long as $T = 1$. Draw its (a) state diagram and (b) primitive flow table.

11.32 For Problem 11.31, use state reduction technique to check if a reduced flow table is possible.

11.33 Find design equations for Problem 11.31 after appropriate state assignment.

11.34 Design a parity generator using asynchronous sequential logic that gives output = 1 when it receives odd number of pulses and output = 0

if the number of pulses received is even. (**Hint:** State transition diagram is same as Problem 11.31.)

11.35 Draw state transition diagram of a modulo-3 counter for asynchronous sequential logic. The counter counts number of pulses appearing at its input and generates output = 1 when three pulses arrive else output = 0.

11.36 Design modulo-3 counter stated in Problem 11.35 using asynchronous sequential logic.

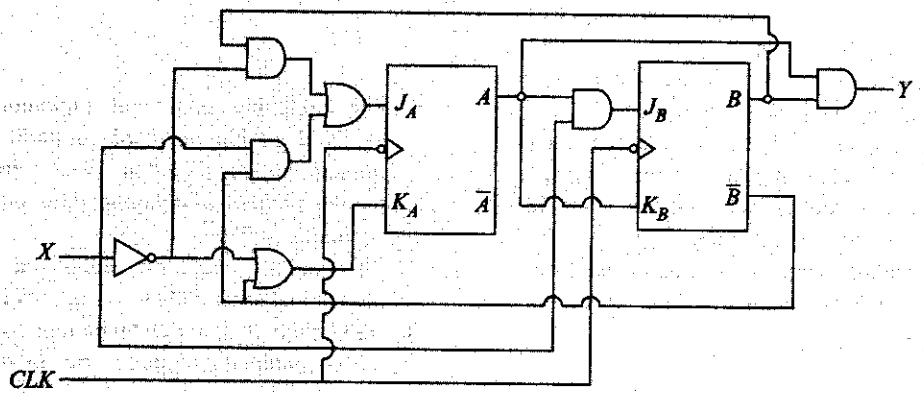
11.37 We require a circuit which will suppress narrow positive spikes on a signal line. The output of the circuit will be an inverted and slightly delayed version of the input minus the spikes. Construct the primitive flow table and show one state assignment scheme.

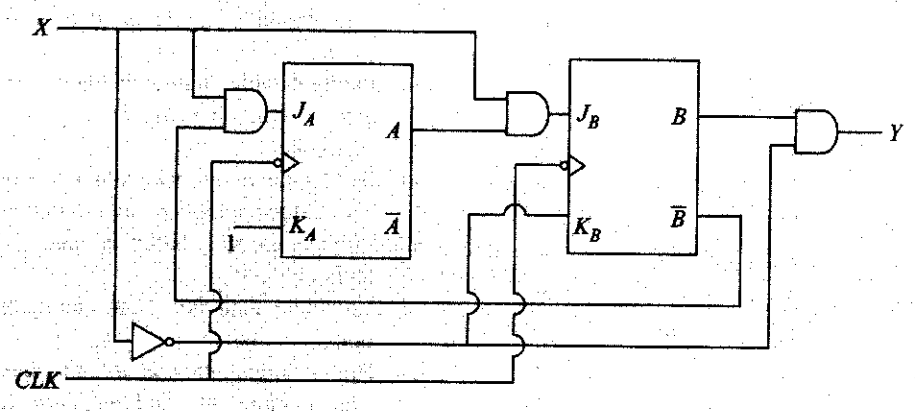
11.38 Get design equations for Problem 11.37 and implement the circuit. Verify how it does noise suppression.

LABORATORY EXPERIMENT

AIM: **The aim of this experiment is to implement a Moore Model and a Mealy Model for a sequence detector that detects a sequence '110' from the incoming data stream.**

Theory: The Moore Model generates final output solely from flip-flop states while Mealy Model can use input data too. Moore Model, usually takes more hardware but in





Mealy Model, unwanted fluctuations in input get directly reflected at the output. Section 11.4 of the book explains the design of these models and reproduces the circuits to be implemented.

Apparatus: +5V DC Power supply, Multimeter, Bread Board, Clock Generator, and Oscilloscope.

Work element: Connect the circuit as shown. You can manually give input, or a counter can be used to generate a repetitive sequence that contains '110'. Check the output. For manual input, check the performance for both the Models when debounce switch is used and when it is not used. Learn to debug the circuit by verifying outputs of individual building blocks say flip-flops and logic gates for different inputs.

Answers to Self-tests

1. State transition diagram is a visual description of how state of sequential circuit change in each clock cycle.
2. In Moore machine, output is associated with a state and written inside a circle. In Mealy machine it is associated with input and written along the arrow-headed transition path.
3. Excitation map is Karnaugh map representation of flip-flop inputs in terms of present state and present circuit input that gives design equations for flip-flops.
4. Moore model normally require more hardware as it needs more number of states to describe a problem.
5.

Clock Cycle	Input	Moore output	Mealy output
1	0	0	0
2	1	0	0
3	0	0	0
4	1	0	0
5	1	0	0
6	0	0	1
7	1	1	0
8	0	0	0
9	1	0	0
10	1	0	0
11	0	0	1
12	1	1	0
13	1	0	0
14	0	0	1
15	1	1	0

6. The excitation table and excitation map of flip-flops are not used in ROM based implementation. Flip-flops used there only for the purpose of delaying information by one clock period.
7. The output is same for Moore and Mealy models and same as Mealy output of Q. 5.
8. ASM chart is flow chart type representation of sequential logic circuit with better time indexation.
9. Implication table is a mapping of state variables that identifies state redundancy easily.
10. Partition tables partitions state variables in groups which consists of equivalent state variables.
11. By this, the redundant states can be removed. This in turn reduces hardware requirement.
12. Output change is based on change in input level.
13. Not more than one input can change at a time.
14. Racing occurs when an input change tries to change more than one feedback variables.

Depending on which changes earlier the circuit may stabilize in two different states of the circuit resulting critical race. If it stabilizes in same state it is called Non-critical race.

15. This is the condition when following an input change, one feedback variable tries to change the output before the other part of the circuit could respond to change in input.
16. This directly maps state transition diagram in a state table where each row has only one stable state.
17. State table where some of the next state or output or both remain unspecified.
18. An additional variable, which is not required as such but prevents simultaneous change of two state variables in asynchronous sequential circuit.
19. Asynchronous circuit does not depend on clock trigger, hence faster. But there are several practical constraints that makes design of such circuit very complex.

D/A Conversion and A/D Conversion

12

OBJECTIVES

- **Be able to do calculations related to variable resistor and binary ladder networks**
- **Recall some of the sections of a typical D/A converter and calculate D/A resolution**
- **Understand A/D conversion using the simultaneous, counter, continuous, and dual-slope methods**
- **Discuss the accuracy and resolution of A/D converters**

Digital-to-analog (D/A) and analog-to-digital (A/D) conversion form two very important aspects of digital data processing. Digital-to-analog conversion involves translation of digital information into equivalent analog information. As an example, the output of a digital system might be changed to analog form for the purpose of driving a pen recorder. Similarly, an analog signal might be required for the servomotors which drive the cursor arms of a plotter. In this respect, a D/A converter is sometimes considered a decoding device.

The process of changing an analog signal to an equivalent digital signal is accomplished by the use of an A/D converter. For example, an A/D converter is used to change the analog output signals from transducers (measuring temperature, pressure, vibration, etc.) into equivalent digital signals. These signals would then be in a form suitable for entry into a digital system. An A/D converter is often referred to as an *encoding device* since it is used to encode signals for entry into a digital system.

Digital-to-analog conversion is a straightforward process and is considerably easier than A/D conversion. In fact. a D/A converter is usually an integral part of any A/D converter. For this reason, we consider the D/A conversion process first.

12.1 VARIABLE, RESISTOR NETWORKS

The basic problem in converting a digital signal into an equivalent analog signal is to change the n digital voltage levels into one equivalent analog voltage. This can be most easily accomplished by designing a resistive network that will change each digital level into an equivalent binary weighted voltage (or current).

Binary Equivalent Weight

As an example of what is meant by *binary equivalent weight*, consider the truth table for the 3-bit binary signal shown in Fig. 12.1. Suppose that we want to change the eight possible digital signals in this figure into equivalent analog voltages. The smallest number represented is 000, let us make this equal to 0 V. The largest number is 111: let us make this equal to +7 V. This then establishes the range of the analog signal to be developed. (There is nothing special about the voltage levels chosen; they were simply selected for convenience.)

2^2	2^1	2^0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Fig. 12.1

Now, notice that between 000 and 111 there are seven discrete levels to be defined. Therefore, it will be convenient to divide the analog signal into seven levels. The smallest incremental change in the digital signal is represented by the least-significant bit (LSB), 2^0. Thus we would like to have this bit cause a change in the analog output that is equal to one-seventh of the full-scale analog output voltage. The resistive divider will then be designed such that a 1 in the 2^0 position will cause $+7 \times \frac{1}{7} = +1$ V at the output.

Since $2^1 = 2$ and $2^0 = 1$, it can be clearly seen that the 2^1 bit represents a number that is twice the size of the 2^0 bit. Therefore, a 1 in the 2^1 bit position must cause a change in the analog output voltage that is twice the size of the LSB. The resistive divider must then be constructed such that a 1 in the 2^1 bit position will cause a change of $+7 \times \frac{2}{7} = +2$ V in the analog output voltage.

Similarly, $2^2 = 4 = 2 \times 2^1 = 4 \times 2^0$, and thus the 2^2 bit must cause a change in the output voltage equal to four times that of the LSB. The 2^2 bit must then cause an output voltage change of $+7 \times \frac{4}{7} = +4$ V.

The process can be continued, and it will be seen that each successive bit must have a value twice that of the preceding bit. Thus the LSB is given a binary equivalent weight of $\frac{1}{7}$ or 1 part in 7. The next LSB is given a weight of $\frac{2}{7}$, which is twice the LSB, or 2 parts in 7. The MSB (in the case of this 3-bit system) is given a weight of $\frac{4}{7}$, which is 4 times the LSB or 4 parts in 7. Notice that the sum of the weights must equal 1. Thus $\frac{1}{7} + \frac{2}{7} + \frac{4}{7} = \frac{7}{7} = 1$. In general, the binary equivalent weight assigned to the LSB is $1/(2^n - 1)$, where n is the number of bits. The remaining weights are found by multiplying by 2, 4, 8, and so on. Remember,

$$\text{LSB weight} = \frac{1}{(2^n - 1)}$$

Example 12.1 Find the binary equivalent weight of each bit in a 4-bit system.

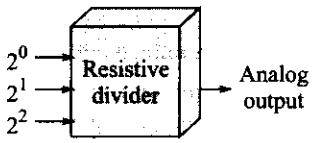
Solution The LSB has a weight of $1/(2^4 - 1) = 1/(16 - 1) = \frac{1}{15}$, or 1 part in 15. The second LSB has a weight of $2 \times \frac{1}{15} = \frac{2}{15}$. The third LSB has a weight of $4 \times \frac{1}{15} = \frac{4}{15}$, and the MSB has a weight of $8 \times \frac{1}{15} = \frac{8}{15}$. As a check, the sum of the weights must equal 1. Thus $\frac{1}{15} + \frac{2}{15} + \frac{4}{15} + \frac{8}{15} = \frac{15}{15} = 1$. The binary equivalent weights for 3-bit and 4-bit systems are summarized in Fig. 12.2.

Bit	Weight
2^0	1/7
2^1	2/7
2^2	4/7
Sum	7/7

(a)

Bit	Weight
2^0	1/15
2^1	2/15
2^2	4/15
2^3	8/15
Sum	15/15

(b)



(a)

Digital input	Analog output
0 0 0	+0 V
0 0 1	+1 V
0 1 0	+2 V
0 1 1	+3 V
1 0 0	+4 V
1 0 1	+5 V
1 1 0	+6 V
1 1 1	+7 V

(b)

Fig. 12.2 **Binary equivalent weights**

Fig. 12.3

Resistive Divider

What is now desired is a resistive divider that has three digital inputs and one analog output as shown in Fig. 12.3a. Assume that the digital input levels are 0 = 0 V and 1 = +7 V. Now, for an input of 001, the output will be +1 V. Similarly, an input of 010 will provide an output of +2 V and an input of 100 will provide an output of +4 V. The digital input 011 is seen to be a combination of the signals 001 and 010. If the +1 V from the 2^0 bit is added to the +2 V from the 2^1 bit, the desired +3 V output for the 011 input is achieved. The other desired voltage levels are shown in Fig. 12.3b; they, too, are additive combinations of voltages.

Thus the resistive divider must do two things in order to change the digital input into an equivalent analog output voltage:

1. The 2^0 bit must be changed to +1 V, and 2^1 bit must be changed to +2 V, and 2^2 bit must be changed to +4 V.
2. These three voltages representing the digital bits must be summed together to form the analog output voltage.

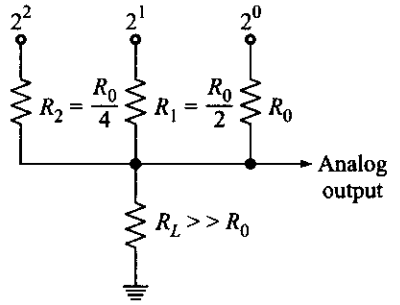


Fig. 12.4

A resistive divider that performs these functions is shown in Fig. 12.4. Resistors R_0, R_1, and R_2 form the divider network. Resistance R_L represents the load to which the divider is connected and is considered to be large enough that it does not load the divider network.

Assume that the digital input signal 001 is applied to this network. Recalling that 0 = 0 V and 1 = +7 V, you can draw the equivalent circuit shown in Fig. 12.5. Resistance R_L is considered large and is neglected. The analog output voltage V_A can be most easily found by use of Millman's theorem, which states that the voltage appearing at any node in a resistive network is equal to the summation of the currents entering the node (found by assuming that the node voltage is zero) divided by the summation of the conductances connected to the node. In equation form, Millman's theorem is

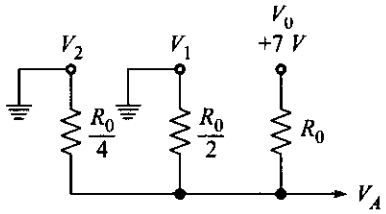


Fig. 12.5

$$V = \frac{V_1/R_1 + V_2/R_2 + V_3/R_3 + \cdots}{1/R_1 + 1/R_2 + 1/R_3 + \cdots}$$

Applying Millman's theorem to Fig. 12.5, we obtain

$$V_A = \frac{V_0/R_0 + V_1/(R_0/2) + V_2/(R_0/4)}{1/R_0 + 1/(R_0/2) + 1/(R_0/4)}$$

$$= \frac{7/R_0}{1/R_0 + 2/R_0 + 4/R_0} = \frac{7}{7} = +1\,V$$

Drawing the equivalent circuits for the other 7-input combinations and applying Millman's theorem will lead to the table of voltages shown in Fig. 12.3 (see Prob. 12.3).

To summarize, a resistive divider can be built to change a digital voltage into an equivalent analog voltage. The following criteria can be applied to this divider:

1. There must be one input resistor for each digital bit.
2. Beginning with the LSB, each following resistor value is one-half the size of the previous resistor.
3. The full-scale output voltage is equal to the positive voltage of the digital input signal. (The divider would work equally well with input voltages of 0 and $-V$.)
4. The LSB has a weight of $1/(2^n - 1)$, where n is the number of input bits.
5. The change in output voltage due to a change in the LSB is equal to $V/(2^n - 1)$, where V is the digital input voltage level.
6. The output voltage V_A can be found for any digital input signal by using the following modified form of Millman's theorem:

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + \cdots + V_{n-1} 2^{n-1}}{2^n - 1} \tag{12.1}$$

where $V_0, V_1, V_2, V_3, \ldots, V_{n-1}$ are the digital input voltage levels (0 or V) and n is the number of input bits.

Example 12.2 For a 5-bit resistive divider, determine the following: (a) the weight assigned to the LSB; (b) the weight assigned to the second and third LSB; (c) the change in output voltage due to a change in the LSB, the second LSB, and the third LSB; (d) the output voltage for a digital input of 10101. Assume 0 = 0 V and 1 = +10 V.

Solution

(a) The LSB weight is $1/(2^5 - 1) = 1/31$.
(b) The second LSB weight is 2/31, and the third LSB weight is 4/31.
(c) The LSB causes a change in the output voltage of 10/31 V. The second LSB causes an output voltage change of 20/31 V, and the third LSB causes an output voltage change of 40/31 V.
(d) The output voltage for a digital input of 10101 is

$$V_A = \frac{10\times 2^0 + 0\times 2^1 + 10\times 2^2 + 0\times 2^3 + 10\times 2^4}{2^5 - 1}$$

$$= \frac{10(1 + 4 + 16)}{32 - 1} = \frac{210}{31} = +6.77\ \text{V}$$

This resistive divider has two serious drawbacks. The first is the fact that each resistor in the network has a different value. Since these dividers are usually constructed by using precision resistors, the added expense becomes unattractive. Moreover, the resistor used for the MSB is required to handle a much greater current than that used for the LSB resistor. For example, in a 10-bit system, the current through the MSB resistor is

approximately 500 times as large as the current through the LSB resistor (see Prob. 12.5). For these reasons, a second type of resistive network, called a *ladder*, has been developed.

SELF-TEST

1. What is the LSB weight of a 6-bit resistive ladder?
2. What is the value of V_A in Example 12.2 if the MSB is 0?

12.2 BINARY LADDERS

The *binary ladder* is a resistive network whose output voltage is a properly weighted sum of the digital inputs. Such a ladder, designed for 4 bits, is shown in Fig. 12.6. It is constructed of resistors that have only two values and thus overcomes one of the objections to the resistive divider previously discussed. The left end of the ladder is terminated in a resistance of $2R$, and we shall assume for the moment that the right end of the ladder (the output) is open-circuited.

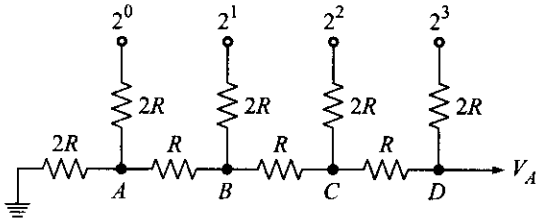


Fig. 12.6 Binary ladder

Let us now examine the resistive properties of the network, assuming that all the digital inputs are at ground. Beginning at node A, the total resistance looking into the terminating resistor is $2R$. The total resistance looking out toward the 2^0 input is also $2R$. These two resistors can be combined to form an equivalent resistor of value R as shown in Fig. 12.7.

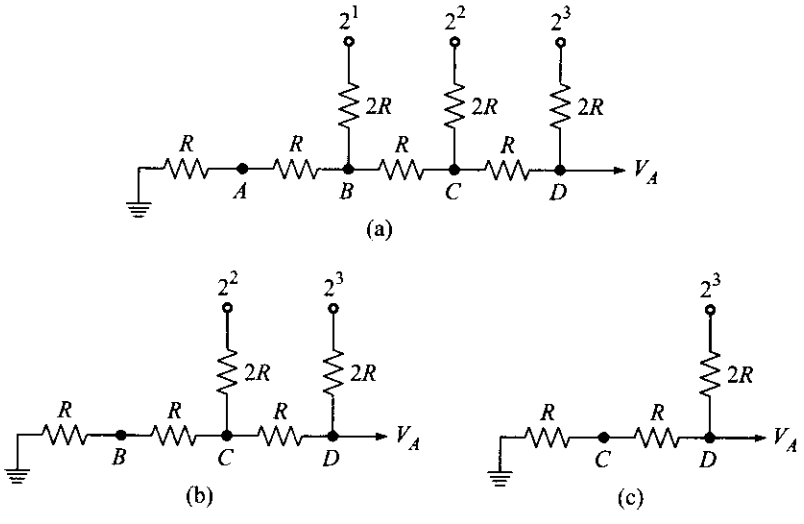


Fig. 12.7

Now, moving to node B, we see that the total resistance looking into the branch toward node A is $2R$, as is the total resistance looking out toward the 2^1 input. These resistors can be combined to simplify the network as shown in Fig. 12.7.

From Fig. 12.7, it can be seen that the total resistance looking from node C down the branch toward node B or out the branch toward the 2^2 input is still $2R$. The circuit in Fig. 12.7 can then be reduced to the equivalent as shown in Fig. 12.7.

From this equivalent circuit, it is clear that the resistance looking back toward node C is $2R$, as is the resistance looking out toward the 2^3 input.

From the preceding discussion, we can conclude that the total resistance looking from any node back toward the terminating resistor or out toward the digital input is $2R$. Notice that this is true regardless of whether the digital inputs are at ground or $+V$. The justification for this statement is the fact that the internal impedance of an ideal voltage source is 0 Ω, and we are assuming that the digital inputs are ideal voltage sources.

We can use the resistance characteristics of the ladder to determine the output voltages for the various digital inputs. First, assume that the digital input signal is 1000. With this input signal, the binary ladder can be drawn as shown in Fig. 12.8a. Since there are no voltage sources to the left of node D, the entire network to the left of this node can be replaced by a resistance of $2R$ to form the equivalent circuit shown in Fig. 12.8b. From this equivalent circuit, it can be easily seen that the output voltage is

$$V_A = V \times \frac{2R}{2R + 2R} = \frac{+V}{2}$$

Thus a 1 in the MSB position will provide an output voltage of $+V/2$.

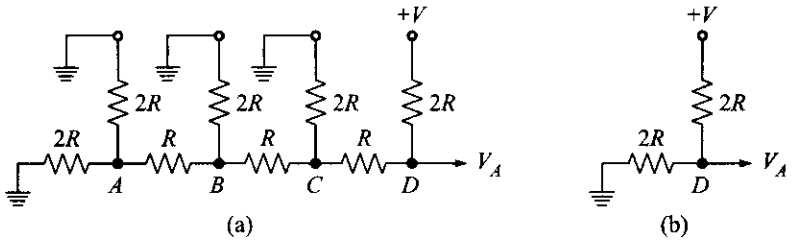


Fig. 12.8 **(a) Binary ladder with a digital input of 1000, (b) Equivalent circuit for a digital input of 1000**

To determine the output voltage due to the second MSB, assume a digital input signal of 0100. This can be represented by the circuit shown in Fig. 12.9a. Since there are no voltage sources to the left of node C, the entire network to the left of this node can be replaced by a resistance of $2R$, as shown in Fig. 12.9b. Let us now replace the network to the left of node C with its Thévenin equivalent by cutting the circuit on the jagged line shown in Fig. 12.9b. The Thévenin equivalent is clearly a resistance R in series with a voltage source $+V/2$. The final equivalent circuit with the Thévenin equivalent included is shown in Fig. 12.9c. From this circuit, the output voltage is clearly

$$V_A = \frac{+V}{2} \times \frac{2R}{R + R + 2R} = \frac{+V}{4}$$

Thus the second MSB provides an output voltage of $+V/4$.

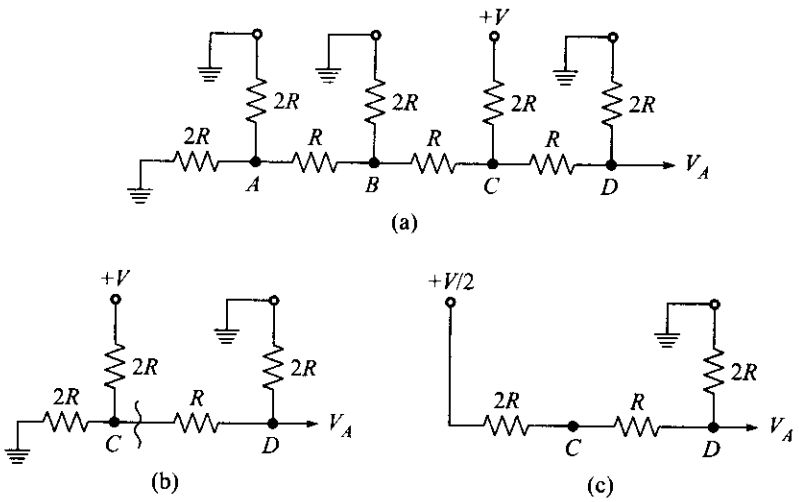


Fig. 12.9 **(a) Binary ladder with a digital input of 0100, (b) Partially reduced equivalent circuit, (c) Final equivalent circuit using Thévenin's theorem**

This process can be continued, and it can be shown that the third MSB provides an output voltage of $+V/8$, the fourth MSB provides an output voltage of $+V/16$, and so on. The output voltages for the binary ladder are summarized in Fig. 12.10; notice that each digital input is transformed into a properly weighted binary output voltage.

Example 12.3 What are the output voltages caused by each bit in a 5-bit ladder if the input levels are 0 = 0 V and 1 = +10 V?

Solution The output voltages can be easily calculated by using Fig. 12.10. They are

$$\text{First MSB } V_A = \frac{V}{2} = \frac{+10}{2} = +5 \text{ V}$$

$$\text{Second MSB } V_A = \frac{V}{4} = \frac{+10}{4} = +2.5 \text{ V}$$

$$\text{Third MSB } V_A = \frac{V}{8} = \frac{+10}{8} = +1.25 \text{ V}$$

$$\text{Fourth MSB } V_A = \frac{V}{16} = \frac{+10}{16} = +0.625 \text{ V}$$

$$\text{LSB = fifth MSB } V_A = \frac{V}{32} = \frac{+10}{32} = +0.3125 \text{ V}$$

Bit position	Binary weight	Output voltage
MSB	1/2	$V/2$
2d MSB	1/4	$V/4$
3d MSB	1/8	$V/8$
4th MSB	1/16	$V/16$
5th MSB	1/32	$V/32$
6th MSB	1/64	$V/64$
7th MSB	1/128	$V/128$
.	.	.
.	.	.
.	.	.
Nth MSB	$1/2^N$	$V/2^N$

Fig. 12.10 **Binary ladder output voltages**

Since this ladder is composed of linear resistors, it is a linear network and the principle of superposition can be used. This means that the total output voltage due to a combination of input digital levels can be found by simply taking the sum of the output levels caused by each digital input individually.

In equation form, the output voltage is given by

$$V_A = \frac{V}{2} + \frac{V}{4} + \frac{V}{8} + \frac{V}{16} + \cdots + \frac{V}{2^n} \tag{12.2}$$

where n is the total number of bits at the input.

This equation can be simplified somewhat by factoring and collecting terms. The output voltage can then be given in the form

$$V_A = \frac{V_0 2^0 + V_1 2^1 + V_2 2^2 + V_3 2^3 + \cdots + V_{n-1} 2^{n-1}}{2^n} \tag{12.3}$$

where $V_0, V_1, V_3, \ldots, V_{n-1}$ are the digital input voltage levels. Equation (12.3) can be used to find the output voltage from the ladder for any digital input signal.

Example 12.4 Find the output voltage from a 5-bit ladder that has a digital input of 11010. Assume that 0 = 0 V and 1 = +10 V.

Solution By Eq. (12.3):

$$V_A = \frac{0\times 2^0 + 10\times 2^1 + 0\times 2^2 + 10\times 2^3 + 10\times 2^4}{2^5}$$

$$= \frac{10(2+8+16)}{32} = \frac{10\times 26}{32} = +8.125 \text{ V}$$

This solution can be checked by adding the individual bit contributions calculated in Example 12.3.

Notice that Eq. (12.3) is very similar to Eq. (12.1), which was developed for the resistive divider. They are, in fact, identical with the exception of the denominators. This is a subtle but very important difference. Recall that the full-scale voltage for the resistive divider is equal to the voltage level of the digital input 1. On the other hand, examination of Eq. (12.2) reveals that the full-scale voltage for the ladder is given by

$$V_A = V\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots + \frac{1}{2^n}\right)$$

The terms inside the brackets form a geometric series whose sum approaches 1, given a sufficient number of terms. *However, it never quite reaches 1.* Therefore, the full-scale output voltage of the ladder approaches V in the limit, but never quite reaches it.

Example 12.5 What is the full-scale output voltage of the 5-bit ladder in Example 12.4?

Solution The full-scale voltage is simply the sum of the individual bit voltages. Thus

$$V = 5 + 2.5 + 1.25 + 0.625 + 0.3125 = +9.6875 \text{ V}$$

To keep the ladder in perfect balance and to maintain symmetry, the output of the ladder should be terminated in a resistance of $2R$. This will result in a lowering of the output voltage, but if the $2R$ load is maintained constant, the output voltages will still be a properly weighted sum of the binary input bits. If the load is varied, the output voltage will not be a properly weighted sum, and care must be exercised to ensure that the load resistance is constant.

Terminating the output of the ladder with a load of $2R$ also ensures that the input resistance to the ladder seen by each of the digital voltage sources is constant. With the ladder balanced in this manner, the resistance

looking into any branch from any node has a value of $2R$. Thus the input resistance seen by any input digital source is $3R$. This is a definite advantage over the resistive divider, since the digital voltage sources can now all be designed for the same load.

Example 12.6 Suppose that the value of R for the 5-bit ladder described in Example 12.3 is 1000 Ω. Determine the current that each input digital voltage source must be capable of supplying. Also determine the full-scale output voltage, assuming that the ladder is terminated with a load resistance of 2000 Ω.

Solution The input resistance into the ladder seen by each of the digital sources is $3R = 3000\ \Omega$. Thus, for a voltage level of +10 V, each source must be capable of supplying $I = 10/(3 \times 10^3) = 3\frac{1}{3}$ mA (without the $2R$ load resistor, the resistance looking into the MSB terminal is actually $4R$). The no-load output voltage of the ladder has already been determined in Example 12.5. This open-circuit output voltage along with the open-circuit output resistance can be used to form a Thevénin equivalent circuit for the output of the ladder. The resistance looking back into the ladder is clearly $R = 1000\ \Omega$. Thus the Thévenin equivalent is as shown in Fig. 12.11. From this figure, the output voltage is

$$V_A = +9.6875 \times \frac{2R}{2R+R} = +6.4583 \text{ V}$$

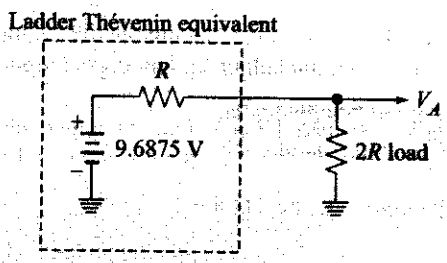
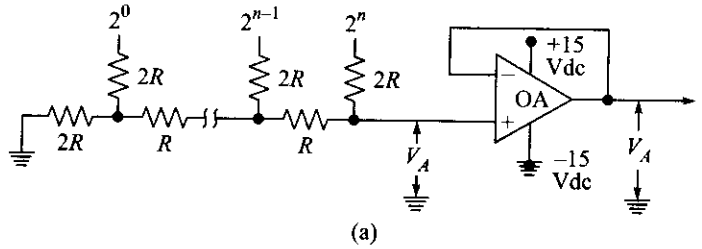
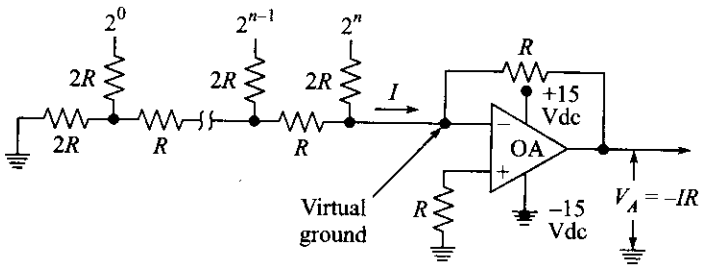


Fig. 12.11 Example 12.6

The operational amplifier (OA) shown in Fig. 12.12a is connected as a unity-gain noninverting amplifier. It has a very high input impedance, and the output voltage is equal to the input voltage. It is thus a good buffer amplifier for connection to the output of a resistive ladder. It will not load down the ladder and thus will not disturb the ladder output voltage V_A; V_A will then appear at the output of the OA.



(a)



(b)

Fig. 12.12

Connecting an OA with a feedback resistor R as shown in Fig. 12.12b results in an amplifier that acts as an inverting current-to-voltage amplifier. That is, the output voltage V_A is equal to the negative of the input current I multiplied by R. The input impedance to this amplifier is essentially 0 Ω: thus, when it is connected to an R-$2R$ ladder, the connecting point is virtually at ground potential. In this configuration, the R-$2R$ ladder will produce a current output I that is a binary weighted sum of the input digital levels. For instance, the MSB produces a current of $V/2R$. The second MSB produces a current of $V/4V$, and so on. But the OA multiplies these currents by $-R$, and thus V_A is

$$V_A = (-R)\left(\frac{V}{2R} + \frac{V}{4R} + \cdots\right) = -\frac{V}{2} - \frac{V}{4} - \cdots$$

This is exactly the same expression given in Eqs. (12.2) and (12.3) except for the sign. Thus the D/A converter in Fig. 12.12a and b will provide the same output voltage V_A except for sign. In Fig. 12.12a, the R-$2R$ ladder and OA are said to operate in a voltage mode, while the connection in Fig. 12.12b is said to operate in a current mode.

SELF-TEST

3. **If the ladder in Example 12.4 is increased to 6 bits, what is the output voltage due to the sixth bit alone?**
4. **If the ladder in Example 12.4 is increased to 6 bits, what is its full-scale output voltage?**

12.3 D/A CONVERTERS

Either the resistive divider or the ladder can be used as the basis for a digital-to-analog (D/A) converter. It is in the resistive network that the actual translation from a digital signal to an analog voltage takes place. There is, however, the need for additional circuitry to complete the design of the D/A converter.

As an integral part of the D/A converter there must be a register that can be used to store the digital information. This register could be any one of the many types discussed in previous chapters. The simplest register is formed by use of *RS* flip-flops, with one flip-flop per bit. There must also be level amplifiers between the register and the resistive network to ensure that the digital signals presented to the network are all of the same level and are constant. Finally, there must be some form of gating on the input of the register such that the flip-flops can be set with the proper infomation from the digital system. A complete D/A converter in block-diagram form is shown in Fig. 12.13a.

Let us expand on the block diagram shown in this Fig. 12.13a by drawing the complete schematic for a 4-bit D/A converter as shown in Fig. 12.13b. You will recognize that the resistor network used is of the ladder type.

The level amplifiers each have two inputs: one input is the +10 V from the precision voltage source, and the other is from a flip-flop. The amplifiers work in such a way that when the input from a flip-flop is high, the output of the amplifier is at +10 V. When the input from the flip-flop is low, the output is 0 V.

The four flip-flops form the register necessary for storing the digital information. The flip-flop on the right represents the MSB, and the flip-flop on the left represents the LSB. Each flip-flop is a simple *RS* latch and requires a positive level at the R or S input to reset or set it. The gating scheme for entering information into the register is straightforward and should be easy to understand. With this particular gating scheme, the flip-flops need not be reset (or set) each time new information is entered. When the READ IN line goes high, only

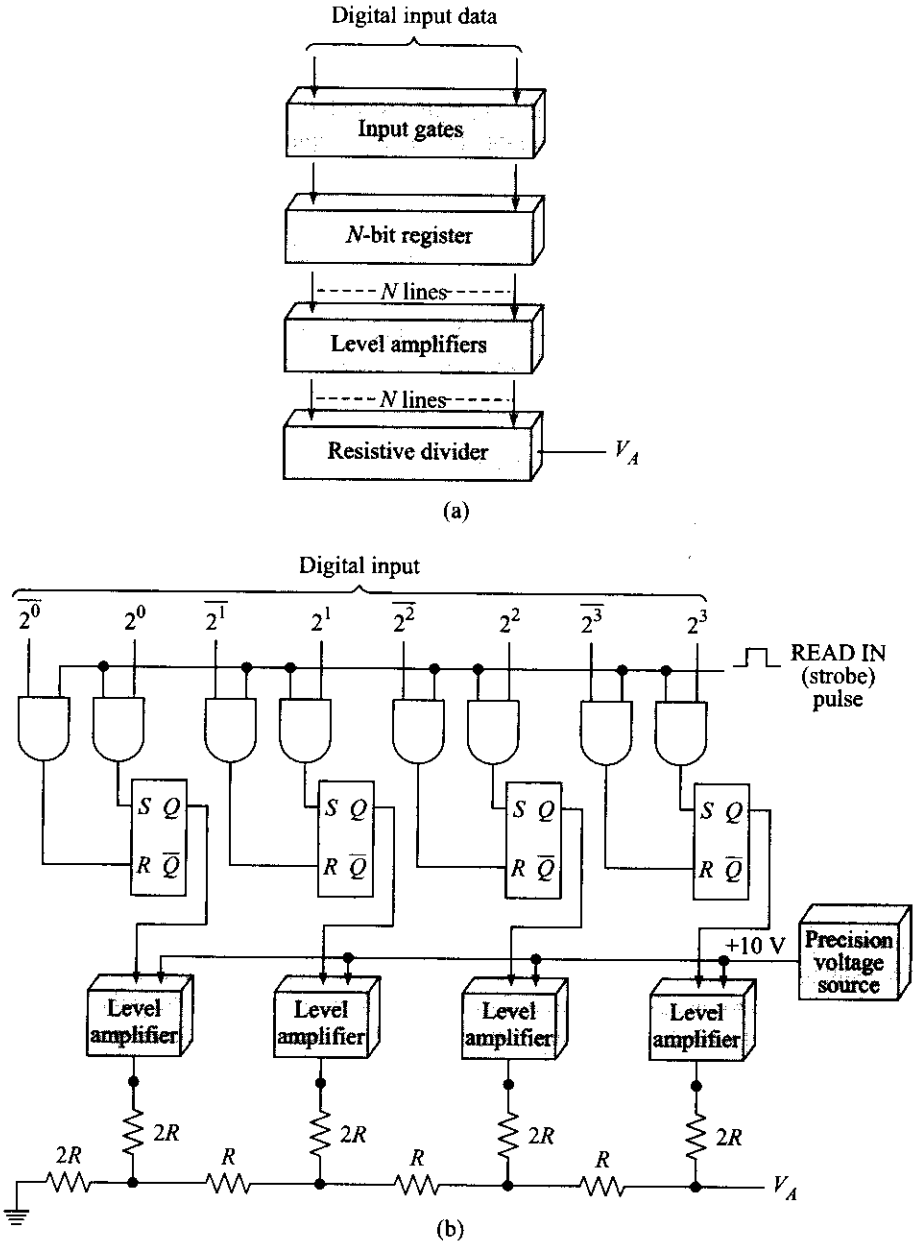


Fig. 12.13 4-bit D/A converter

one of the two gate outputs connected to each flip-flop is high, and the flip-flop is set or reset accordingly. Thus data are entered into the register each time the READ IN (strobe) pulse occurs. *D* flip-flops could be used in place of the *RS* flip-flops.

Multiple Signals

Quite often it is necessary to decode more than one signal—for example, the X and Y coordinates for a plotting board. In this event, there are two ways in which to decode the signals.

The first and most obvious method is simply to use one D/A converter for each signal. This method, shown in Fig. 12.14a, has the advantage that each signal to be decoded is held in its register and the analog output voltage is then held fixed. The digital input lines are connected in parallel to each converter. The proper converter is then selected for decoding by the select lines.

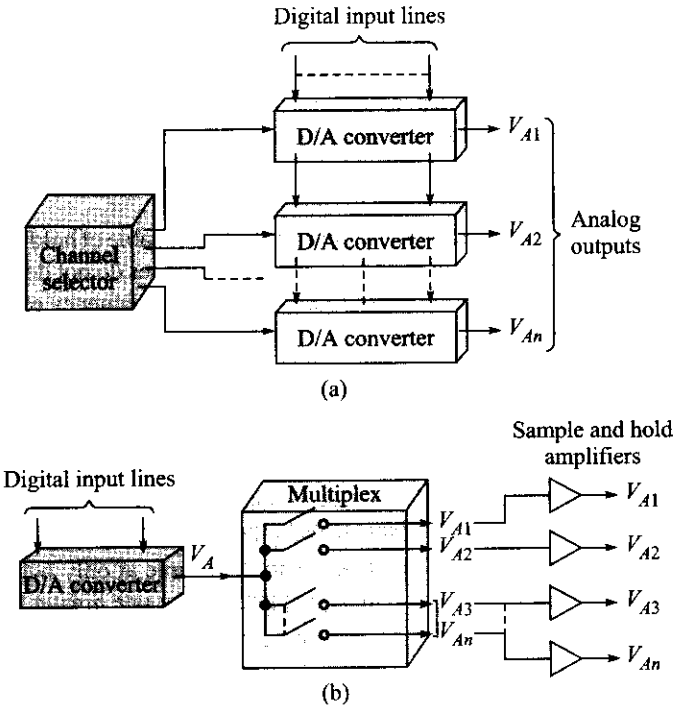


Fig. 12.14 **Decoding a number of signals: (a) Channel selection method, (b) Multiplex method**

The second method involves the use of only one D/A converter and switching its output. This is called *multiplexing*, and such a system is shown in Fig. 12.14b. The disadvantage here is that the analog output signal must be held between sampling periods, and the outputs must therefore be equipped with sample-and-hold amplifiers.

Sample and Hold Circuit

An OA connected as in Fig. 12.15a is a unity-gain noninverting voltage amplifier—that is, $V_o = V_i$. Two such OAs are used with a capacitor in Fig. 12.15b to form a sample-and-hold amplifier. When the switch is closed, the capacitor charges to the D/A converter output voltage. When the switch is opened, the capacitor holds the voltage level until the next sampling time. The operational amplifier provides a large input impedance so as not to discharge the capacitor appreciably and at the same time offers gain to drive external circuits.

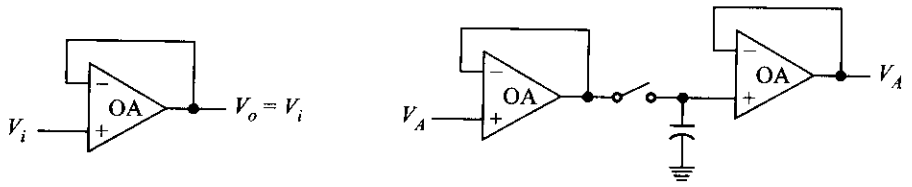


Fig. 12.15 **(a) Unity gain amplifier, (b) Sample-and-hold circuit**

When the D/A converter is used in conjunction with a multiplexer, the maximum rate at which the converter can operate must be considered. Each time data is shifted into the register, transients appear at the output of the converter. This is due mainly to the fact that each flip-flop has different rise and fall times. Thus a settling time must be allowed between the time data is shifted into the register and the time the analog voltage is read out. This settling time is the main factor in determining the maximum rate of multiplexing the output. The worst case is when all bits change (e.g., from 1000 to 0111).

Naturally, the capacitors on the sample-and-hold amplifiers are not capable of holding a voltage indefinitely; therefore, the sampling rate must be sufficient to ensure that these voltages do not decay appreciably between samples. The sampling rate is a function of the capacitors as well as the frequency of the analog signal which is expected at the output of the converter.

At this point, you might be curious to know just how fast a signal must be sampled in order to preserve its integrity. Common sense leads to the conclusion that the more often the signal is sampled, the less the sample degrades between samples. On the other hand, if too few samples are taken, the signal degrades too much (the sample-and-hold capacitors discharge too much), and the signal information is lost. We would like to reduce the sampling rate to the minimum necessary to extract all the necessary information from the signal. The solution to this problem involves more than we have time for here, but the results are easy enough to apply.

First, if the signal in question is sinusoidal, it is necessary to sample at only *twice* the signal frequency. For instance if the signal is a 5-kHz sine wave, it must be sampled at a rate greater than or equal to 10 kHz. In other words, a sample must be taken every $\frac{1}{1000}$ s = 100 μs. What if the waveform is not sinusoidal? Any waveform that is periodic can be represented by a summation of sine and cosine terms, with each succeeding term having a higher frequency. In this case, it will be necessary to sample at a rate equal to twice the highest frequency of interest.

D/A Converter Testing

Two simple but important tests that can be performed to check the proper operation of the D/A converter are the *steady-state accuracy test* and the *monotonicity test*.

The steady-state accuracy test involves setting a known digital number in the input register, measuring the analog output with an accurate meter, and comparing with the theoretical value.

Checking for monotonicity means checking that the output voltage increases regularly as the input digital signal increases. This can be accomplished by using a counter as the digital input signal and observing the analog output on an oscilloscope. For proper monotonicity, the output waveform should be a perfect staircase waveform, as shown in Fig. 12.16. The steps on the staircase waveform must be equally spaced and of the exact same amplitude. Missing steps, steps of different amplitude, or steps in a downward fashion indicate malfunctions.

The monotonicity test does not check the system for accuracy, but if the system passes the test, it is relatively certain that the converter error is less than 1 LSB. Converter accuracy and resolution are the subjects of the next section.

A D/A converter can be regarded as a logic block having numerous digital inputs and a single analog output as seen in Fig. 12.16b. It is interesting to compare this logic block with the potentiometer shown in Fig. 12.16c. The analog output voltage of the D/A converter is controlled by the digital input signals while the analog output voltage of the potentiometer is controlled by mechanical rotation of the potentiometer shaft. Considered in this fashion, it is easy to see how a D/A converter could be used to generate a voltage waveform (sawtooth, triangular, sinusoidal, etc.). It is, in effect, a digitally controlled voltage generator!

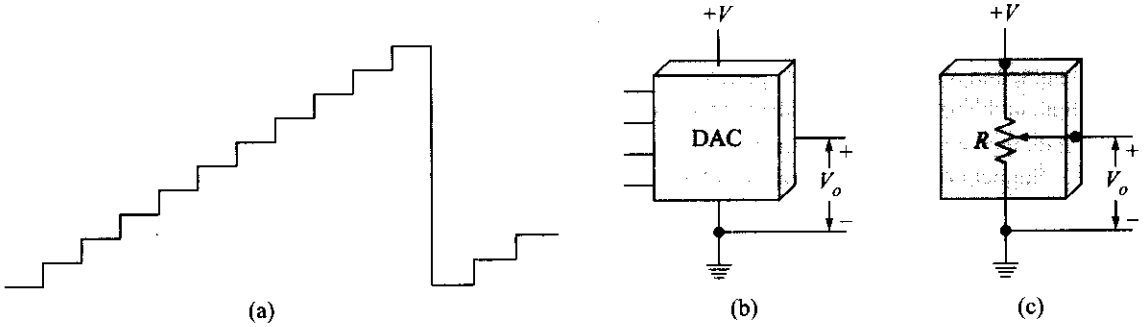


Fig. 12.16 **Correct output voltage waveform for monotonicity test**

Example 12.7 Suppose that in the course of a monotonicity check on the 4-bit converter in Fig. 12.13 the waveform shown in Fig. 12.17 is observed. What is the probable malfunction in the converter?

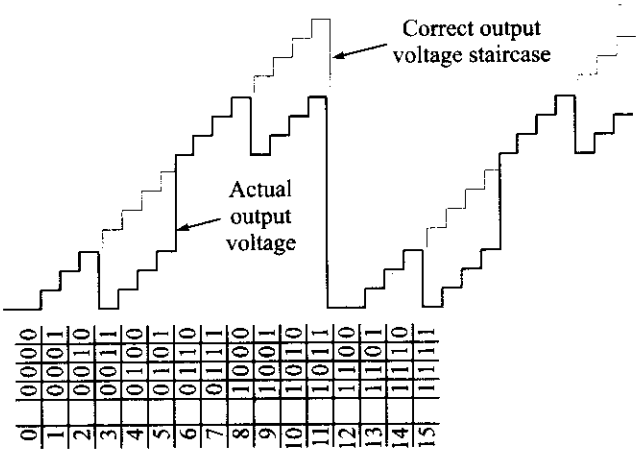


Fig. 12.17 **Irregular output voltage for Example 12.7**

Solution There is obviously some malfunction since the actual output waveform is not continuously increasing as it should be. The actual digital inputs are shown directly below the wave-form. Notice that the converter functions

correctly up to count 3. At count 4, however, the output should be 4 units in amplitude. Instead, it drops to 0. It remains 4 units below the correct level until it reaches count 8. Then, from count 8 to 11, the output level is correct. But again at count 12 the output falls 4 units below the correct level and remains there for the next four levels. If you examine the waveform carefully, you will note that the output is 4 units below normal during the time when the 2^2 bit is supposed to be high. This then suggests that the 2^2 bit is being dropped (i.e., the 2^2 input to the ladder is not being held high). This means that the 2^2-level amplifier is malfunctioning or the 2^2 AND gate is not operating properly. In any case, the monotonicity check has clearly shown that the second MSB is not being used and that the converter is not operating properly.

Available D/A Converters

D/A converters, as well as sample-and-hold amplifiers, are readily obtainable commercial products. Each unit is constructed in a single package; general-purpose economy units are available with 6-, 8-, 10-, and 12-bit resolution, and high-resolution units with up to 16-bit resolution are available.

An inexpensive and very popular D/A converter is the DAC0808, an 8-bit D/A converter available from National Semiconductor. Motorola manufactures an 8-bit D/A converter, the MC1508/1408. In Fig. 12.18, a DAC0808 is connected to provide a full-scale output voltage of $V_o = +10$ Vdc when all 8 digital inputs are 1s (high). If the 8 digital inputs are all 0s (low), the output voltage will be $V_o = 0$ Vdc. Let's look at this circuit in detail.

First of all, two dc power-supply voltages are required for the DAC0808: $V_{CC} = +5$ Vdc and $V_{EE} = -15$ Vdc. The 0.1-μF capacitor is to prevent unwanted circuit oscillations, and to isolate any variations in V_{EE}. Pin 2 is ground (GND), and pin 15 is also referenced to ground through a resistor.

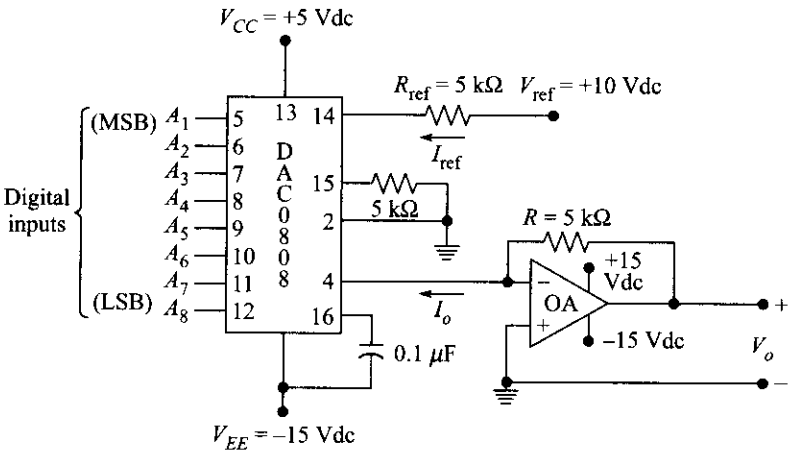


Fig. 12.18

The output of the D/A converter on pin 4 has a very limited voltage range (+0.5 to −0.6 V). Rather, it is designed to provide an output current I_o. The minimum current (all digital inputs low) is 0.0 mA, and the maximum current (all digital inputs high), is I_{ref}. This reference current is established with the resistor at pin 14 and the reference voltage as

$$I_{ref} = V_{ref}/R_{ref} \tag{12.4}$$

The D/A converter output current I_o is given as

$$I_o = I_{\text{ref}}\left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \cdots + \frac{A8}{256}\right) \tag{12.5}$$

where $A_1, A_2, A_3, \ldots, A_8$ are the digital input levels (1 or 0).

The OA is connected as a current-to-voltage converter, and the output voltage is given as

$$V_o = I_o \times R \tag{12.6}$$

Substituting Eqs. (12.4) and (12.5) into Eq. (12.6),

$$V_o = V_{\text{ref}} / R_{\text{ref}} \times \left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \cdots + \frac{A8}{256}\right) \times R \tag{12.7}$$

If we set the OA feedback resistor R equal to R_{ref}, then

$$V_o = V_{\text{ref}}\left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \cdots + \frac{A8}{256}\right) \tag{12.8}$$

Let's try out Eq. (12.8). Suppose all digital inputs are 0s (all low). Then

$$V_o = V_{\text{ref}} \times \left(\frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \cdots + \frac{0}{256}\right)$$
$$= V_{\text{ref}} \times 0 = 0.0 \text{ Vdc}$$

Now, suppose all digital inputs are 1s (all high). Then

$$V_o = V_{\text{ref}} \times \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{256}\right)$$
$$= (V_{\text{ref}}) \times \left(\frac{255}{256}\right) = 0.996 \times V_{\text{ref}}$$

Since V_{ref} in Fig. 12.18 is +10 Vdc, the output voltage is seen to have a range between 0.0 and +9.96 Vdc. It doesn't quite reach +10 Vdc, but this is characteristic of this type of circuit. This circuit is essentially the current-mode operation discussed in the previous section and illustrated in Fig. 12.12b.

Example 12.8 In Fig. 12.16, $A1$ is high, $A2$ is high, $A5$ is high and $A7$ is high. The other digital inputs are all low. What is the output voltage V_o?

Solution

$$V_o = 10 \times \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \frac{1}{128}\right) = 10 \times 0.789 = 7.89 \text{ V}$$

SELF-TEST

5. What is a monotonicity test?
6. What would be the full-scale output voltage in Fig. 12.18 if V_{ref} were changed to +5 Vdc?

12.4 D/A ACCURACY AND RESOLUTION

Two very important aspects of the D/A converter are the resolution and the accuracy of the conversion. There is a definite distinction between the two, and you should clearly understand the differences.

The accuracy of the D/A converter is primarily a function of the accuracy of the precision resistors used in the ladder and the precision of the reference voltage supply used. Accuracy is a measure of how close the actual output voltage is to the theoretical output value.

For example, suppose that the theoretical output voltage for a particular input should be +10 V. An accuracy of 10 percent means that the actual output voltage must be somewhere between +9 and +11 V. Similarly, if the actual output voltage were somewhere between +9.9 and +10.1 V, this would imply an accuracy of 1 percent.

Resolution, on the other hand, defines the smallest increment in voltage that can be discerned. Resolution is primarily a function of the number of bits in the digital input signal; that is, the smallest increment in output voltage is determined by the LSB.

In a 4-bit system using a ladder, for example, the LSB has a weight of $\frac{1}{16}$. This means that the smallest increment in output voltage is $\frac{1}{16}$ of the input voltage. To make the arithmetic easy, let us assume that this 4-bit system has input voltage levels of +16 V. Since the LSB has a weight of $\frac{1}{16}$, a change in the LSB results in a change of 1 V in the output. Thus the output voltage changes in steps (or increments) of 1 V. The output voltage of this converter is then the staircase shown in Fig. 12.16 and ranges from 0 to +15V in 1-V increments. This converter can be used to represent analog voltages from 0 to +15 V, but it cannot resolve voltages into increments smaller than 1 V. If we desired to produce +4.2 V using this converter, therefore, the actual output voltage would be +4.0 V. Similarly, if we desired a voltage of +7.8 V, the actual output voltage would be +8.0 V. It is clear that this converter is not capable of distinguishing voltages finer than 1 V, which is the resolution of the converter.

If we wanted to represent voltages to a finer resolution, we would have to use a converter with more input bits. As an example, the LSB of a 10-bit converter has a weight of $\frac{1}{1024}$. Thus the smallest incremental change in the output of this converter is approximately $\frac{1}{1000}$ of the full-scale voltage. If this converter has a +10-V full-scale output, the resolution is approximately $+10 \times \frac{1}{1000} = 10$ mV. This converter is then capable of representing voltages to within 10 mV.

Example 12.9 What is the resolution of a 9-bit D/A converter which uses a ladder network? What is this resolution expressed as a percent? If the full-scale output voltage of this converter is +5 V, what is the resolution in volts?

Solution The LSB in a 9-bit system has a weight of $\frac{1}{512}$. Thus this converter has a resolution of 1 part in 512. The resolution expressed as a percentage is $\frac{1}{512} \times 100$ percent $\cong$ 0.2 percent. The voltage resolution is obtained by multiplying the weight of the LSB by the full-scale output voltage. Thus the resolution in volts is $\frac{1}{512} \times 5 \cong 10$ mV.

Example 12.10 How many bits are required at the input of a convener if it is necessary to resolve voltages to 5 mV and the ladder has +10 V full scale?

Solution The LSB of an 11-bit system has a resolution of $\frac{1}{2048}$. This would provide a resolution at the output of $\frac{1}{2048} \times +10 \cong +5$ mV.

It is important to realize that resolution and accuracy in a system should be compatible. For example, in the 4-bit system previously discussed, the resolution was found to be 1 V. Clearly it would be unjustifiable to construct such a system to an accuracy of 0.1 percent. This would mean that the system would be accurate to 16 mV but would be capable of distinguishing only to the nearest 1 V.

Similarly, it would be wasteful to construct the 11-bit system described in Example 12.19 to an accuracy of only 1 percent. This would mean that the output voltage would be accurate only to 100 mV, whereas it is capable of distinguishing to the nearest 5 mV.

SELF-TEST

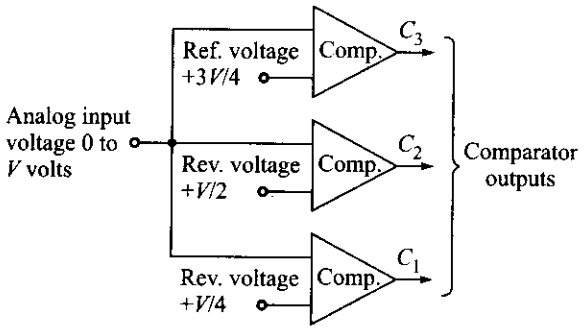
7. What is the resolution of the DAC0808 in Fig. 12.18?

12.5 A/D CONVERTER—SIMULTANEOUS CONVERSION

The process of converting an analog voltage into an equivalent digital signal is known as *analog-to-digital (A/D) conversion*. This operation is somewhat more complicated than the converse operation of D/A conversion. A number of different methods have been developed, the simplest of which is probably the simultaneous method. This is also known as *A/D converter, flash type*, the reason for which will be clear shortly.

The simultaneous method of A/D conversion is based on the use of a number of comparator circuits. One such system using three comparator circuits is shown in Fig. 12.19 below. The analog signal to be digitized serves as one of the inputs to each comparator. The second input is a standard reference voltage. The reference voltages used are $+V/4$, $+V/2$, and $+3V/4$. The system is then capable of accepting an analog input voltage between 0 and $+V$.

If the analog input signal exceeds the reference voltage to any comparator, that comparator turns on. (Let's assume that this means that the output of the comparator goes high.) Now, if all the comparators are off, the analog input signal must be between 0 and $+V/4$. If C_1 is high (comparator C_1 is on) and C_2 and C_3 are low, the input must be between $+V/4$ and $+V/2$ V. If C_1 and C_2 are high while C_3 is low, the input must be between $+V/2$ and $+3V/4$. Finally, if all comparator outputs are high, the input signal must be between $+3V/4$ and $+V$. The comparator output levels for the various ranges of input voltages are summarized in Fig. 12.19.



(a)

Input voltage	Comparator output		
	C_1	C_2	C_3
0 to $+V/4$	Low	Low	Low
$+V/4$ to $+V/2$	High	Low	Low
$+V/2$ to $+3V/4$	High	High	Low
$+3V/4$ to $+V$	High	High	High

(b)

Fig. 12.19 Simultaneous A/D conversion: (a) Logic diagram, (b) Comparator outputs for input voltage ranges

Examination of Fig. 12.19 reveals that there are four voltage ranges that can be detected by this converter. Four ranges can be effectively discerned by two binary digits (bits). The three comparator outputs can then be fed into a coding network to provide 2 bits which are equivalent to the input analog voltage. The bits of the coding network can then be entered into a flip-flop register for storage. The complete block diagram for such an A/D converter is shown in Fig. 12.20.

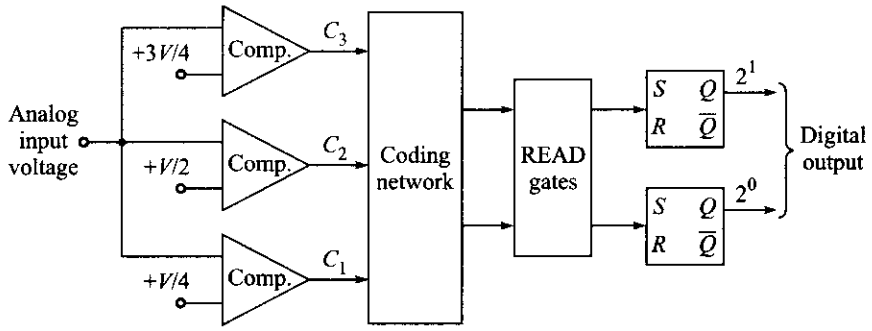


Fig. 12.20 **2-bit simultaneous A/D converter**

In order to gain a clear understanding of the operation of the simultaneous A/D converter, let us investigate the 3-bit converter shown in Fig. 12.21a. Notice that in order to convert the input signal to a digital signal having 3 bits, it is necessary to have seven comparators (this allows a division of the input into eight ranges). For the 2-bit converter, remember that three comparators were necessary for defining four ranges. In general, it can be said that $2^n - 1$ comparators are required to convert to a digital signal that has n bits. Some of the comparators have inverters at their outputs since both C and $\overline{C}$ are needed for the encoding matrix.

The encoding matrix must accept seven input levels and encode them into a 3-bit binary number (having eight possible states). Operation of the encoding matrix can be most easily understood by examination of the table of outputs in Fig. 12.22.

The 2^2 bit is easiest to determine since it must be high (the 2^2 flip-flop must be set) whenever C_4 is high.

The 2^1 line must be high whenever C_2 is high and $\overline{C}_4$ is high, or whenever C_6 is high. In equation form, we can write $2^1 = C_2\overline{C}_4 + C_6$.

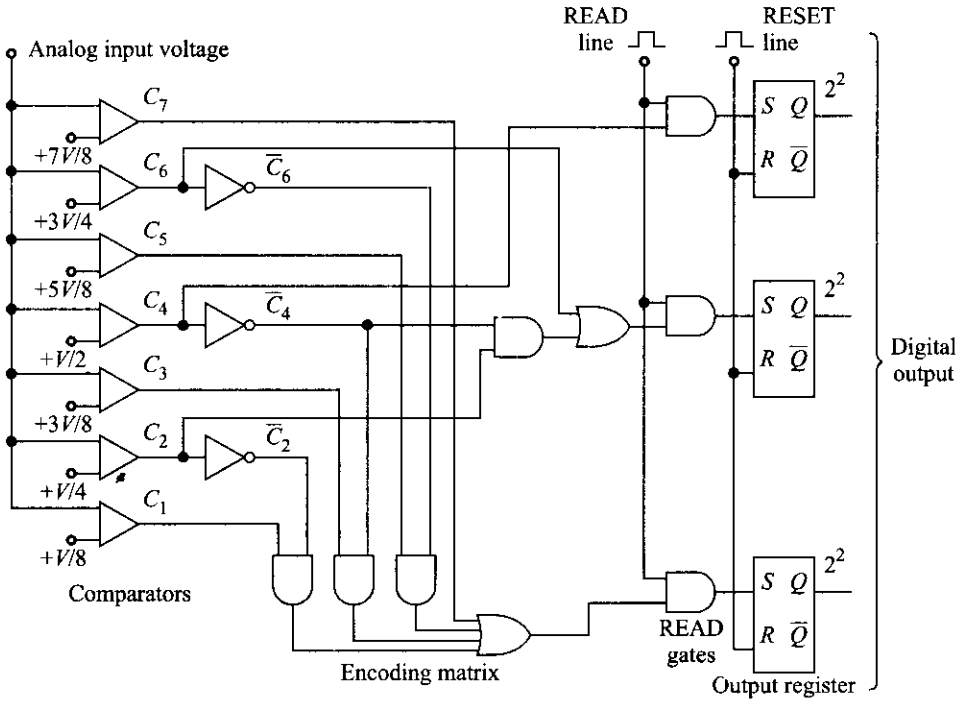
The logic equation for the 2^0 bit can be found in a similar manner; it is

$$2^0 = C_1\overline{C}_2 + C_3\overline{C}_4 + C_5\overline{C}_6 + C_7$$

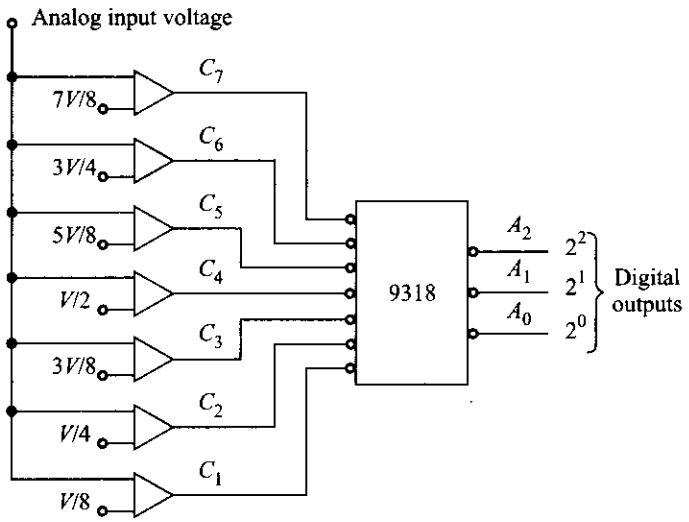
The transfer of data from the encoding matrix into the register must be carried out in two steps. First, a positive reset pulse must appear on the RESET line to reset all the flip-flops low. Then, a positive READ pulse allows the proper READ gates to go high and thus transfer the digital information into the flip-flops.

Interestingly, a convenient application for a 9318 priority encoder is to use it to replace all the digital logic as shown in Fig. 12.21b. Of course, the inputs $C_1, C_2, \ldots, C_7$ must be TTL-compatible. In essence, the output of the 9318 is a digital number that reflects the highest-order zero input; this corresponds to the lowest reference voltage that still exceeds the input analog voltage.

The construction of a simultaneous A/D converter is quite straightforward and relatively easy to understand. However, as the number of bits in the desired digital number increases, the number of comparators increases very rapidly ($2^n - 1$), and the problem soon becomes unmanageable. Even though this method is simple and is capable of extremely fast conversion rates, here are preferable methods for digitizing numbers having more than 3 or 4 bits. Because it is so fast, this type of converter is frequently called a *flash converter*.



(a)



(b)

Fig. 12.21 **3-bit simultaneous A/D converter: (a) Logic diagram, (b) Using a 9318 priority encoder**

Input voltage	Comparator for level							Binary output		
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	2^2	2^1	2^0
0 to $V/8$	Low	Low	Low	Low	Low	Low	Low	0	0	0
$V/8$ to $V/4$	High	Low	Low	Low	Low	Low	Low	0	0	1
$V/4$ to $3V/8$	High	High	Low	Low	Low	Low	Low	0	1	0
$3V/8$ to $V/2$	High	High	High	Low	Low	Low	Low	0	1	1
$V/2$ to $5V/8$	High	High	High	High	Low	Low	Low	1	0	0
$5V/8$ to $3V/4$	High	High	High	High	High	Low	Low	1	0	1
$3V/4$ to $7V/8$	High	High	High	High	High	High	Low	1	1	0
$7V/8$ to V	High	High	High	High	High	High	High	1	1	1

Fig. 12.22 **Logic table tor the converter in Fig. 12.19(a)**

The Motorola MC10319 is an example of an 8-bit flash A/D converter. The input has 256 parallel comparators connected to a precision voltage divider network. The comparator outputs are fed to latches and then to an encoder network that captures the digital signal in Gray code. Gray code is used to ensure that small input errors do not result in large digital signal errors. The Gray code is then decoded into straight binary and presented to the outputs, which are tri-state TTL = compatible. The flash A/D converter is capable of operation with a 25-MHz clock! It comes in a 24-pin DIP and requires two dc supply voltages—typically +5 Vdc and –5 Vdc. Possible applications include radar signal processing, video displays, high-speed instrumentation, and television broadcasting.

SELF-TEST

8. **Why is a simultaneous A/D converter called a flash converter?**
9. **What is one application for a flash converter?**

12.6 A/D CONVERTER–COUNTER METHOD

A higher-resolution A/D converter using only one comparator could be constructed if a variable reference voltage were available. This reference voltage could then be applied to the comparator, and when it became equal to the input analog voltage, the conversion would be complete.

To construct such a converter, let us begin with a simple binary counter. The digital output signals will be taken from this counter, and thus we want it to be an n-bit counter, where n is the desired number of bits. Now let us connect the output of this counter to a standard binary ladder to form a simple D/A converter. If a clock is now applied to the input of the counter, the output of the binary ladder is the familiar staircase waveform shown in Fig. 12.16. This waveform is exactly the reference voltage signal we would like to have for the comparator! With a minimum of gating and control circuitry, this simple D/A converter can be changed into the desired A/D converter.

Figure 12.23 shows the block diagram for a counter-type A/D converter. The operation of the counter is as follows. First, the counter is reset to all 0s. Then, when a convert signal appears on the START line, the gate opens and clock pulses are allowed to pass through to the input of the counter. The counter advances through its normal binary count sequence, and the staircase waveform is generated at the output of

the ladder. This waveform is applied to one side of the comparator, and the analog input voltage is applied to the other side. When the reference voltage equals (or exceeds) the input analog voltage, the gate is closed, the counter stops, and the conversion is complete. The number stored in the counter is now the digital equivalent of the analog input voltage.

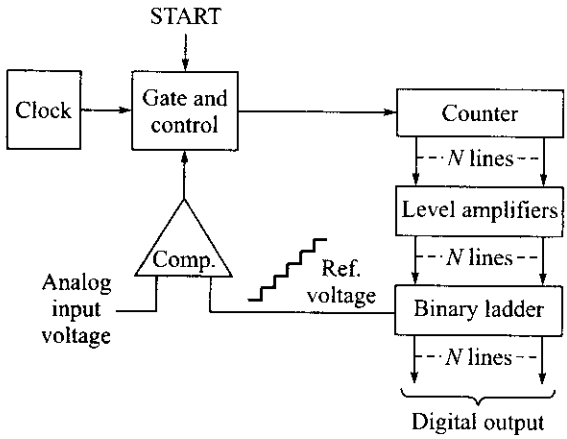


Fig. 12.23 **Counter type A/D converter**

Notice that this converter is composed of a D/A converter (the counter, level amplifiers, and the binary ladder), one comparator, a clock, and the gate and control circuitry. This can really be considered as a closed-loop control system. An error signal is generated at the output of the comparator by taking the difference between the analog input signal and the feedback signal (staircase reference voltage). The error is detected by the control circuit, and the clock is allowed to advance the counter. The counter advances in such a way as to reduce the error signal by increasing the feedback voltage. When the error is reduced to zero, the feedback voltage is equal to the analog input signal, the control circuitry stops the clock from advancing the counter, and the system comes to rest.

The counter-type A/D converter provides a very good method for digitizing to a high resolution. This method is much simpler than the simultaneous method for high resolution, but the conversion time required is longer. Since the counter always begins at zero and counts through its normal binary sequence, as many as 2^n counts may be necessary before conversion is complete. The average conversion time is, of course, $2^n/2$ or 2^{n-1} counts.

The counter advances one count for each cycle of the clock, and the clock therefore determines the conversion rate. Suppose, for example, that we have a 10-bit converter. It requires 1024 clock cycles for a full-scale count. If we are using a 1-MHz clock, the counter advances 1 count every microsecond. Thus, to count full scale requires $1024 \times 10^{-6} = 1.024$ ms. The converter reaches one-half full scale in half this time, or in 0.512 ms. The time required to reach one-half full scale can be considered the *average* conversion time for a large number of conversions.

Example 12.11 Suppose that the converter shown in Fig. 12.23 is an 8-bit converter driven by a 500-kHz clock. Find (a) the maximum conversion time; (b) the average conversion time; (c) the maximum conversion rate.

Solution

(a) An 8-bit converter has a maximum of $2^8 = 256$ counts. With a 500-kHz clock, the counter advances at the rate of 1 count each 2 μs. To advance 256 counts requires $256 \times 2 \times 10^{-6} = 512 \times 10^{-6} = 512$ μs.

(b) The average conversion time is one-half the maximum conversion time. Thus it is $1/2 \times 0.512 \times 10^{-3} = 0.256$ ms.

(c) The maximum conversion rate is determined by the longest conversion time. Since the converter has a maximum conversion time of 0.512 ms, it is capable of making at least $1/(0.512 \times 10^{-3}) \cong 1953$ conversions per second.

Figure 12.24 shows one method of implementing the control circuitry for the converter shown in Fig. 12.23. The waveforms for one conversion are also shown. A conversion is initiated by the receipt of a START signal.

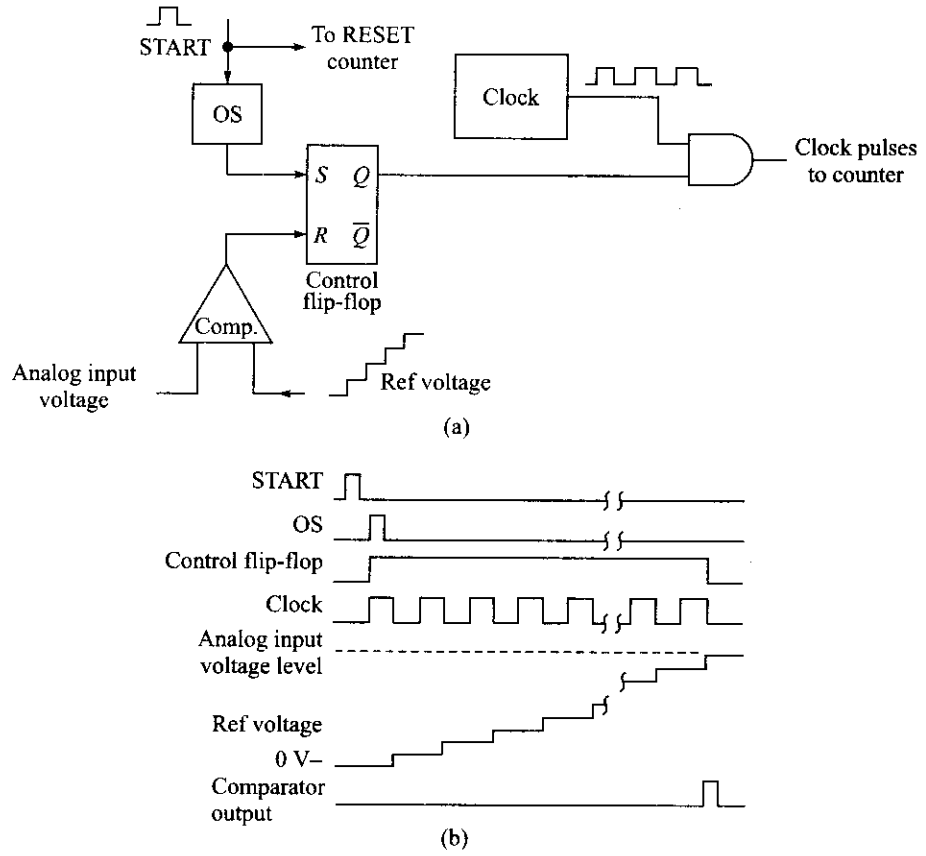


Fig. 12.24 Control of the A/D converter in Fig. 12.21

The positive edge of the START pulse is used to reset all the flip-flops in the counter and to trigger the one-shot. The output of the one-shot sets the control flip-flop, which makes the AND gate true and allows clock pulses to advance the counter.

The delay between the RESET pulse to the flip-flops and the beginning of the clock pulses (ensured by the one-shot) is to ensure that all flip-flops are reset before counting begins. This is a definite attempt to avoid any racing problems.

With the control flip-flop set, the counter advances through its normal count sequence until the staircase voltage from the ladder is equal to the analog input voltage. At this time, the comparator output changes state, generating a positive pulse which resets the control flip-flop. Thus the AND gate is closed and counting ceases. The counter now holds a digital number which is equivalent to the analog input voltage. The converter remains in this state until another conversion signal is received.

If a new start signal is generated immediately after each conversion is completed, the converter will operate at its maximum rate. The converter could then be used to digitize a signal as shown in Fig. 12.25a Notice that the conversion times in digitizing this signal are not constant but depend on the amplitude of the input signal. The analog input signal can be reconstructed from the digital information by drawing straight

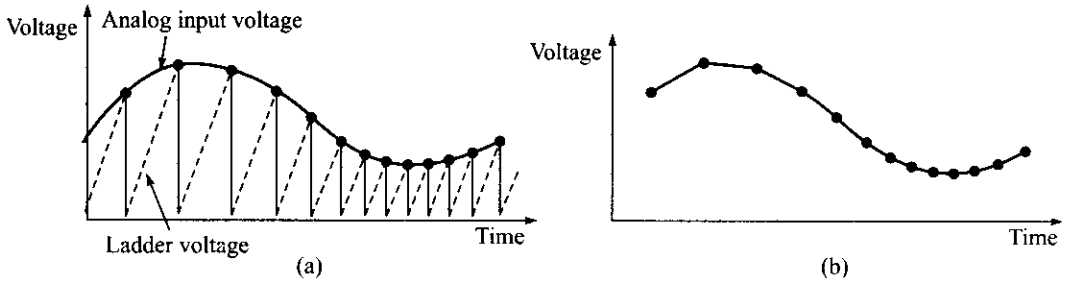


Fig. 12.25 (a) Digitizing an analog voltage. (b) Reconstructed signal from the digital data.

lines from each digitized point to the next. Such a reconstruction is shown in Fig. 12.25b; it is, indeed, a reasonable representation of the original input signal. In this case, it is important to note that the conversion times are smaller than the transient time of the input waveform.

On the other hand, if the transient time of the input waveform approaches the conversion time, the reconstructed output signal is not quite so accurate. Such a situation is shown in Fig. 12.26a and b. In this case, the input waveform changes at a rate faster than the converter is capable of recognizing. Thus the need for reducing conversion time is apparent.

SELF-TEST

10. The A/D converter in Fig. 12.23 has 8 bits and is driven by a 2-MHz clock. What is the maximum conversion time?
11. What is the average conversion time for the converter in question 10?

12.7 CONTINUOUS A/D CONVERSION

An obvious method for speeding up the conversion of the signal as shown in Fig. 12.26 is to eliminate the need for resetting the counter each time a conversion is made. If this were done. the counter would not begin at zero each time, but instead would begin at the value of the last converted point. This means that the counter would have to be capable of counting either up or down. This is no problem; we are already familiar with the operation of up-down counters.

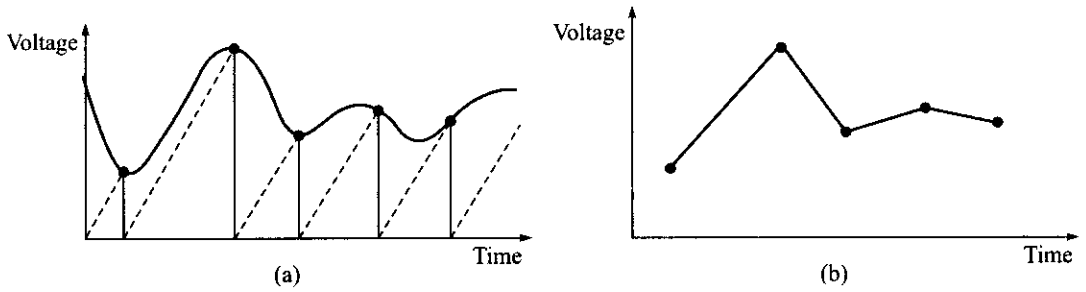
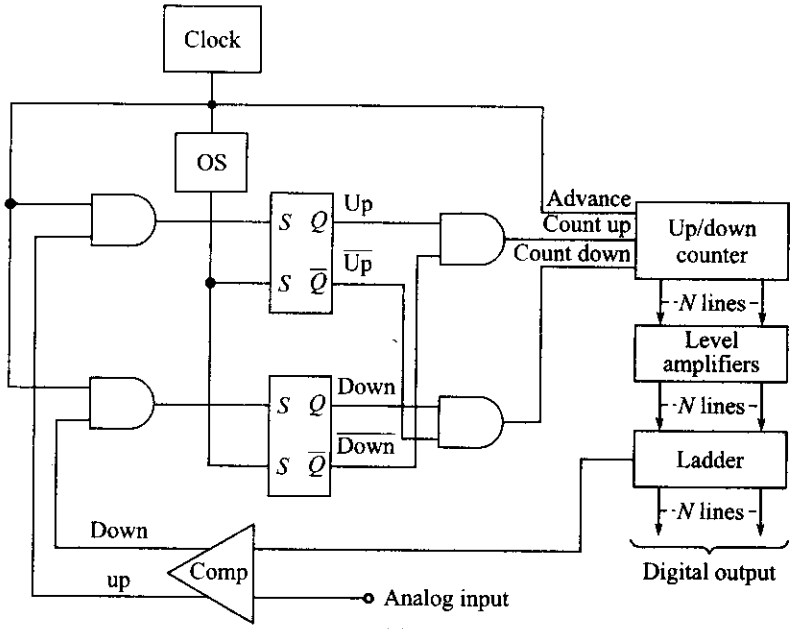
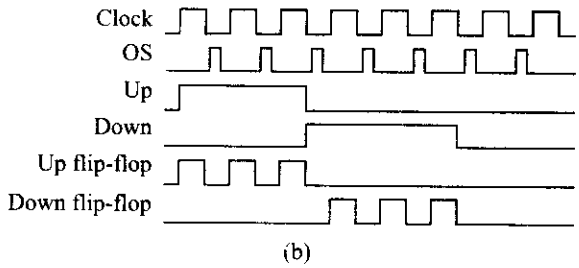


Fig. 12.26 (a) Digitizing an analog voltage, (b) Reconstructed signal from the digital data

There is, however, the need for additional logic circuitry, since we must decide whether to count up or down by examining the output of the comparator. An A/D converter which uses an up-down counter is shown in Fig. 12.27 below. This method is known as *continuous conversion*, and thus the converter is called a *continuous-type A/D converter*. Since the converter's digital output always tries to track the analog input to the converter, this is also known as *A/D converter-tracking type*.



(a)



(b)

Fig. 12.27 Continuous A/D converter

The D/A portion of this converter is the same as those previously discussed, with the exception of the counter. It is an up-down counter and has the up and down count control lines in addition to the advance line at its input.

The output of the ladder is fed into a comparator which has two outputs instead of one as before. When the analog voltage is more positive than the ladder output, the *up* output of the comparator is high. When the analog voltage is more negative than the ladder output, the *down* output is high.

If the *up* output of the comparator is high, the AND gate at the input of the *up* flip-flop is open, and the first time the clock goes positive, the *up* flip-flop is set. If we assume for the moment that the *down* flip-flop is reset, the AND gate which controls the *count-up* line of the counter will be true and the counter will advance one count. The counter can advance only one count since the output of the one-shot resets both the *up* and the *down* flip-flops just after the clock goes low. This can then be considered as one count-up conversion cycle.

Notice that the AND gate which controls the *count-up* line has inputs of *up* and $\overline{down}$. Similarly, the count-down line AND gate has inputs of *down* and $\overline{up}$. This could be considered an exclusive-OR arrangement and ensures that the count-down and count-up lines cannot both be high at the same time.

As long as the *up* line out of the comparator is high, the converter continues to operate one conversion cycle at a time. At the point where the ladder voltage becomes more positive than the analog input voltage, the *up* line of the comparator goes low and the *down* line goes high. The converter then goes through a count-down conversion cycle. At this point, the ladder voltage is within 1 LSB of the analog voltage, and the converter oscillates about this point. This is not desirable since we want the converter to cease operation and not jump around the final value. The trick here is to adjust the comparator such that its outputs do not change at the same time.

We can accomplish this by adjusting the comparator such that the *up* output will not go high unless the ladder voltage is more than 1/2 LSB below the analog voltage. Similarly, the *down* output will not go high unless the ladder voltage is more than 1/2 LSB above the analog voltage. This is called *centering on the LSB* and provides a digital output which is within 1/2 LSB.

A waveform typical of this type of converter is shown in Fig. 12.28. You can see that this converter is capable of following input voltages that change at a much faster rate.

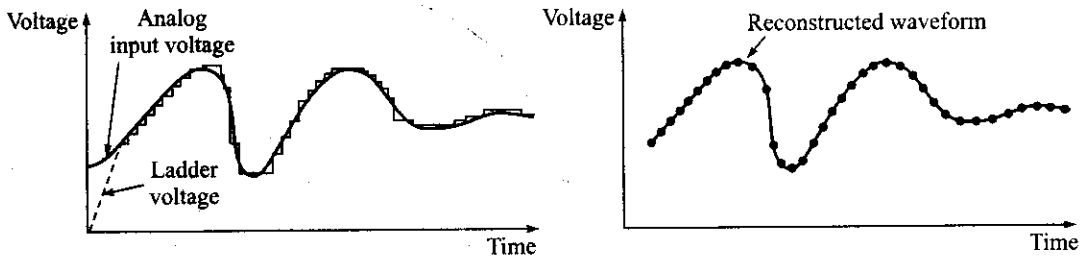


Fig. 12.28 Continuous A/D conversion

Example 12.12 Quite often, additional circuitry is added to a continuous converter to ensure that it cannot count off scale in either direction. For example, if the counter contained all 1s, it would be undesirable to allow it to progress through a count-up cycle, since the next count would advance it to all 0s. We would like to design the logic necessary to prevent this.

Solution The two limit points which must be detected are all 1s and all 0s in the counter. Suppose that we construct an AND gate having the 1 sides of all the counter flip-flops as its inputs. The output of this gate will be true whenever the counter contains all 1s. If the gate is then connected to the reset side of the *up* flip-flop, the counter will be unable to count beyond all 1s.

Similarly, we might construct an AND gate in which the inputs are the 0 sides of all the counter flip-flops. The output of this gate can be connected to the reset side of the *down* flip-flop, and the counter will then be unable to count beyond all 0s. The gates are shown in Fig. 12.29.

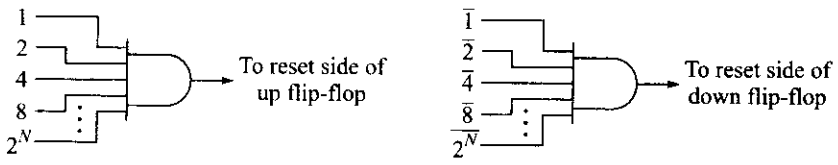


Fig. 12.29 **Count-limiting gates for the converter in Fig. 12.25**

SELF-TEST

12. How does the continuous-type A/D converter differ from the simple counter-type A/D converter?
13. What advantage does the continuous-type A/D converter offer over the counter-type A/D converter?

12.8 A/D TECHNIQUES

There are a variety of other methods for digitizing analog signals—too many to discuss in detail. Nevertheless, we shall take the time to examine two more techniques and the reasons for their importance.

Probably the most important single reason for investigating other methods of conversion is to determine ways to reduce the conversion time. Recall that the simultaneous converter has a very fast conversion time. The counter converter is simple logically but has a relatively long conversion time. The continuous converter has a very fast conversion time once it is locked on the signal but loses this advantage when multiplexing inputs.

Successive Approximation

If multiplexing is required, the *successive-approximation converter* is most useful. The block diagram for this type of converter is shown in Fig. 12.30a. The converter operates by successively dividing the voltage ranges in half. The counter is first reset to all 0s, and the MSB is then set. The MSB is then left in or taken out (by resetting the MSB flip-flop) depending on the output of the comparator. Then the second MSB is set in, and a comparison is made to determine whether to reset the second MSB flip-flop. The process is repeated down to the LSB, and at this time the desired number is in the counter. Since the conversion involves operating on one flip-flop at a time, beginning with the MSB, a ring counter may be used for flip-flop selection.

The successive-approximation method thus is the process of approximating the analog voltage by trying 1 bit at a time beginning with the MSB. The operation is shown in diagram form in Fig. 12.30b. It can be seen from this diagram that each conversion takes the same time and requires one conversion cycle for each bit. Thus the total conversion time is equal to the number of bits, n, times the time required for one conversion cycle. One conversion cycle normally requires one cycle of the clock. As an example, a 10-bit converter operating with a 1-MHz clock has a conversion time of $10 \times 10^{-6} = 10^{-5} = 10\ \mu s$.

When dealing with conversion times this short, it is usually necessary to take into account the other delays in the system (e.g. switching time of the multiplexer, settling time of the ladder network, comparator delay, and settling time).

All the logic blocks inside the dashed line in Fig. 12.30a, or some equivalent arrangement, are frequently constructed on a single MSI chip; this chip is called a *successive-approximation register* (SAR). For example, the Motorola MC6108 shown in Fig. 12.28c is an 8-bit microprocessor-compatible A/D converter that includes an SAR, D/A conversion capabilities, control logic, and buffered digital outputs, in a 28-pin DIP.

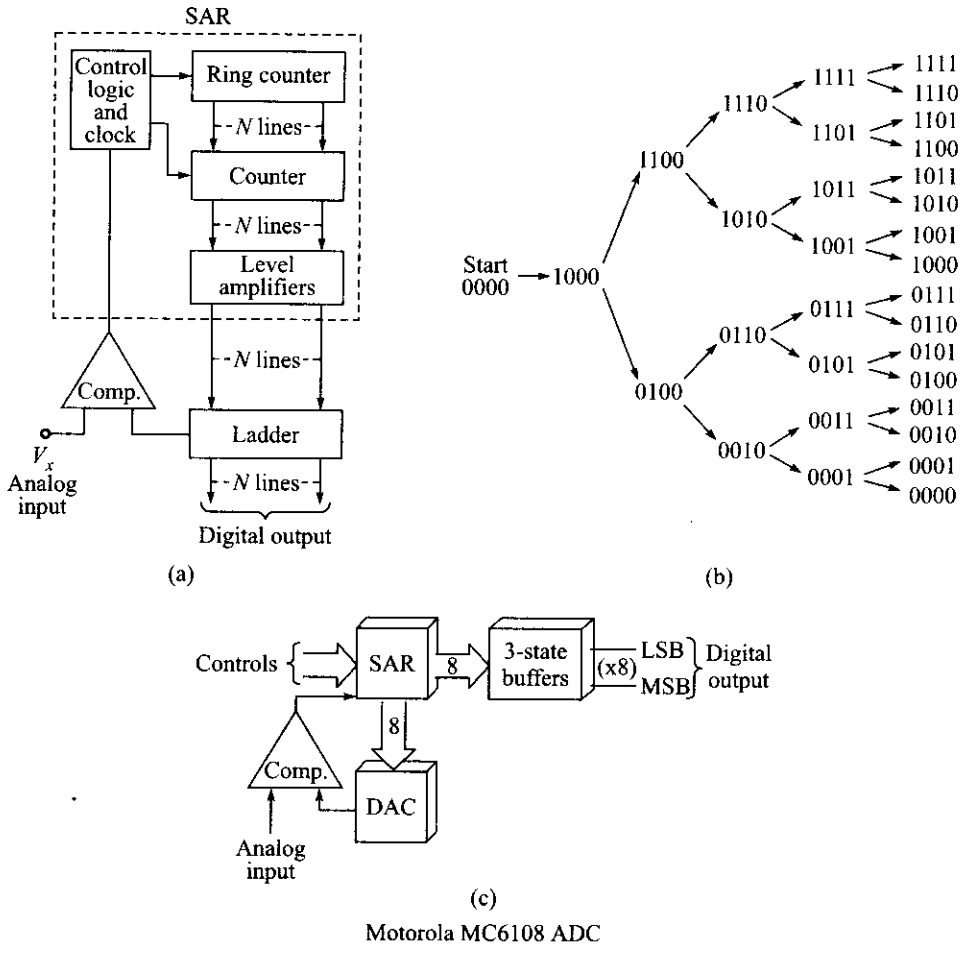


Fig. 12.30 **Successive approximation converter**

The ADC0804

The ADC0804 is an inexpensive and very popular A/D converter which is available from a number of different manufacturers, including National Semiconductor. The ADC0804 is an 8-bit CMOS microprocessor compatible successive-approximation A/D converter that is supplied in a 20-pin DIP. It is capable of digitizing an analog input voltage within the range 0 to +5 Vdc, and it only requires a single dc supply voltage—usually +5 Vdc. The digital outputs are both TTL- and CMOS-compatible.

The block diagram of an ADC0804 is shown in Fig. 12.31. In this case, the controls are wired such that the converter operates continuously. This is the so-called *free-running mode*. The 10-kΩ resistor, along with the

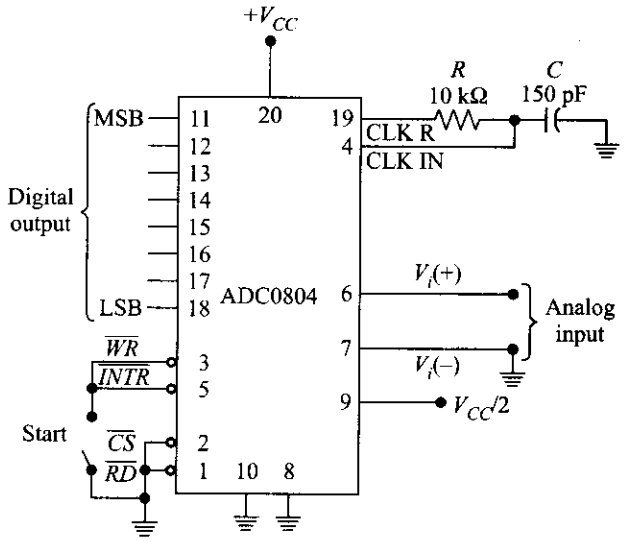


Fig. 12.31

150-pF capacitor, establishes the frequency of operation according to $f \approx 1/1.1(RC)$. In this case,

$$f \approx \frac{1}{1.1 \times (10\ \text{k}\Omega \times 150\ \text{pF})}$$

$$= \frac{1}{1.1 \times (10^4 \times 1.5 \times 10^{-12})} = 607\ \text{kHz}$$

A momentary activation of the START switch is necessary to begin operation. A detailed discussion of the ADC0804 is given in Section 15.4.

Section Counters

Another method for reducing the total conversion time of a simple counter converter is to divide the counter into sections. Such a configuration is called a *section counter*. To determine how the total conversion time might be reduced by this method, assume that we have a standard 8-bit counter. If this counter is divided into two equal counters of 4 bits each, we have a section converter. The converter operates by setting the section containing the four LSBs to all 1s and then advancing the other sections until the ladder voltage exceeds the input voltage. At this point the four LSBs are all reset, and this section of the counter is then advanced until the ladder voltage equals the input voltage.

Notice that a maximum of $2^4 = 16$ counts is required for each section to count full scale. Thus this method requires only $2 \times 2^4 = 2^5 = 32$ counts to reach full scale. This is a considerable reduction over the $2^8 = 256$ counts required for the straight 8-bit counter. There is, of course, some extra time required to set the counters initially and to switch from counter to counter during the conversion. This logical operation time is very small, however, compared with the total time saved by this method.

This type of converter is quite often used for digital voltmeters, since it is very convenient to divide the counters by counts of 10. Each counter is then used to represent one of the digits of the decimal number appearing at the output of the voltmeter. We discuss this subject in detail in the next chapter.

SELF-TEST

14. What does SAR stand for in Fig. 12.30c?
15. What is an ADC0804?

12.9 DUAL-SLOPE A/D CONVERSION

Up to this point, our interest in different methods of A/D conversion has centered on reducing the actual conversion time. If a very short conversion time is not a requirement, there are other methods of A/D conversion that are simpler to implement and much more economical. Basically, these techniques involve comparison of the unknown input voltage with a reference voltage that begins at zero and increases linearly with time. The time required for the reference voltage to increase to the value of the unknown voltage is directly proportional to the magnitude of the unknown voltage, and this time period is measured with a digital counter. This is referred to as a *single-ramp method*, since the reference voltage is sloped like a ramp. A variation on this method involves using an operational amplifier integrating circuit in a dual-ramp configuration. The dual-ramp method is very popular, and widely used in digital voltmeters and digital panel meters. It offers good accuracy, good linearity, and very good noise-rejection characteristics.

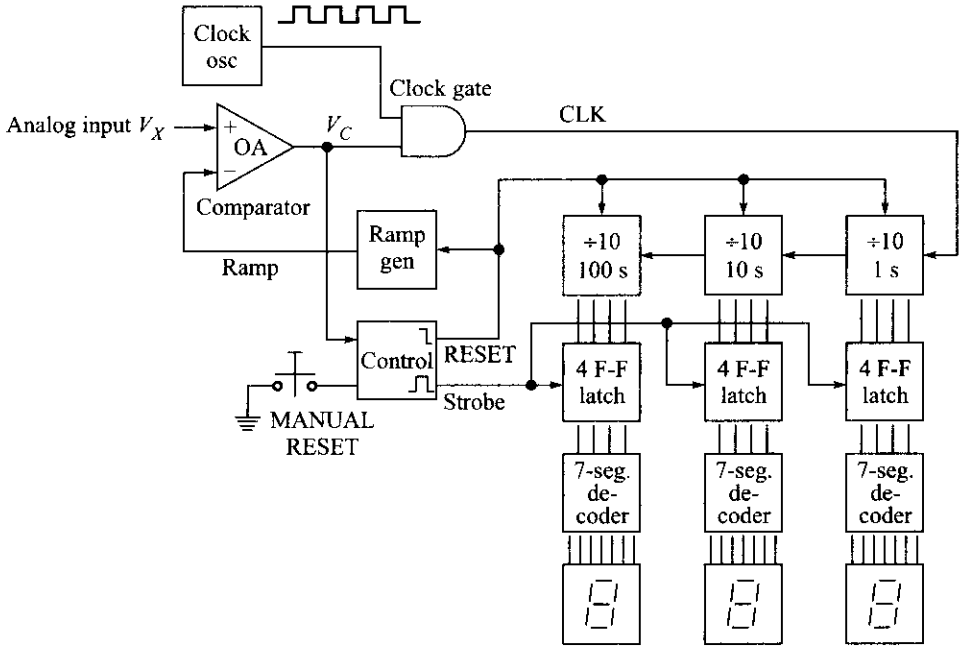
Single-Ramp A/D Converter

Let's take a look at the single-ramp A/D converter in Fig. 12.32. The heart of this converter is the *ramp generator*. This is a circuit that produces an output voltage ramp as shown in Fig. 12.33a. The output voltage begins at zero and increases linearly up to a maximum voltage V_m. It is important that this voltage be a straight line—that is, it must have a constant slope. For instance, if V_m = 1.0 Vdc, and it takes 1.0 ms for the ramp to move from 0.0 up to 1.0 V, the slope is 1 V/ms, or 1000 V/s.

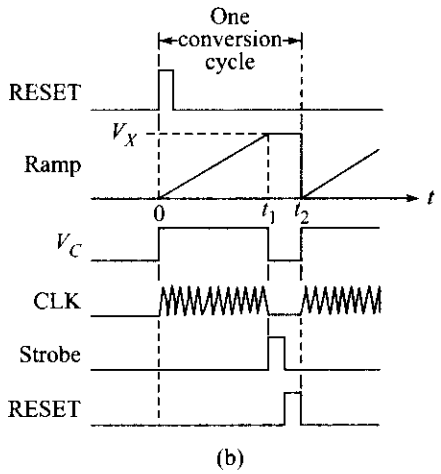
This ramp generator can be constructed in a number of different ways. One way might be to use a D/A converter driven by a simple binary counter. This would generate the staircase waveform previously discussed and shown in Fig. 12.16a. A second method is to use an operational amplifier (OA) connected as an integrator as shown in Fig. 12.33b. For this circuit, if V_i is a constant, the output voltage is given by the relationship $V_o = (V_i/RC)t$. Since V_i, R, and C are all constants, this is the equation of a straight line that has a slope (V_i/RC) as shown in Fig. 12.33a. Now that we have a way to generate a voltage ramp and we understand its characteristics, let's return to the converter in Fig. 12.32.

We assume that the clock is running continuously and that any input voltage V_X that we wish to digitize is positive. If it is not, there are circuits that we can use to adjust for negative input signals. The three decade counters are connected in cascade, and their outputs can be strobed into three 4-flip-flop latch circuits. The latches are then decoded by seven-segment decoders to drive the LED displays as units, tens, and hundreds of counts. We can begin a conversion cycle by depressing the MANUAL RESET switch.

Refer carefully to the logic diagram and the waveforms in Fig. 12.32. MANUAL RESET generates a RESET pulse that clears all the decade counters to 0s and resets the ramp voltage to zero. Since V_X is positive and RAMP begins at zero, the output of the comparator OA, V_c, must be high. This voltage enables the



(a)



(b)

Fig. 12.32 Single-slope A/D converter

CLOCK gate allowing the clock, CLK, to be applied to the decade counter. The counter begins counting upward, and the RAMP continues upward until the ramp voltage is equal to the unknown input V_X.

At this point, time t_1, the output of the comparator V_c goes low, thus disabling the CLOCK gate and the counters cease to advance. Simultaneously, this negative transition on V_c generates a STROBE signal in the CONTROL box that shifts the contents of the three decade counters into the three 4-flip-flop latch circuits.