Digital electronics

1-Sequential circuit counters

Such a group of flip- flops is a counter. The number of flip-flops used and the way in which they are connected determine the number of states and also the specific sequence of states that the counter goes through during each complete cycle.

Counters are classified according to the way they are clocked: asynchronous and synchronous. Within each of these two categories, counters are classified primarily by the type of sequence, the number of states, or the number of flip-flops in the counter.

1-Asynchronous Counter Operation

Asynchronous counters called ripple counters, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. The term asynchronous refers to events that do not have a fixed time relationship with each other . An asynchronous counter is one in which the flip-flops within the counter do not change states at exactly the same time because they do not have a common clock pulse.

A 2-Bit Asynchronous Binary Counter

Fig1-1 shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) is applied to the clock input (C) of only the first flop-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the $\bar{Q}_0$ out-put of FF0. FF0 changes state at the positive-going edge of each clock pulse. But FF1 changes only when triggered by a positive-going transition of the $\bar{Q}_0$ output of FF0. Because of the inherent propagation delay tie through a flip-flop, a transition of the input clock pulse (CLK) and a transition of the $\bar{Q}_0$ output of FF0 can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter operation is asynchronous.

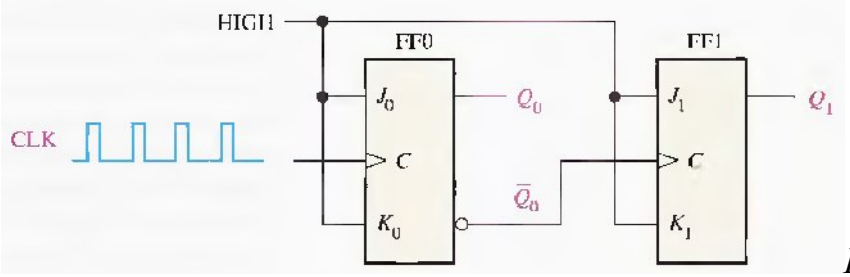


Fig1-1 2-bit asynchronous counter

The Timing Diagram

Applying 4clock pulses to FF0, Both flip-flops are connected for toggle operation (J=1, K=1) and initially RESET (Q LOW). The positive-going edge of CLK1 (clock pulse1) causes the Q_0 output of FF0 to go HIGH. At the same time the $\bar{Q}_0$ output goes LOW, but it has no effect on FF1 because a positive-going transition must occur to trigger the flip-flop. After the leading edge of CLK1, Q_0=1 & Q_1=0. The positive-going edge of CLK2 causes Q_0 to go LOW.

$\bar{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go HIGH. After the leading edge of CLK2, Q_0=0 & Q_1=1. The positive-going edge of CLK3 causes Q_0 to go HIGH again. Output $\bar{Q}_0$ goes LOW and has no effect on FF1. Thus, after the leading edge of CLK3, $\bar{Q}_0$=1 & Q_1=1. The positive-going edge of CLK4 causes Q_0 to go LOW, while $\bar{Q}_0$ goes HIGH and triggers

FF1, causing Q_1 to go LOW. After the leading edge of CLK4, Q_0 =0 & Q_1=0. The 2-bit counter exhibits four different states, as you would expect with two flip-flops (2^2 = 4).

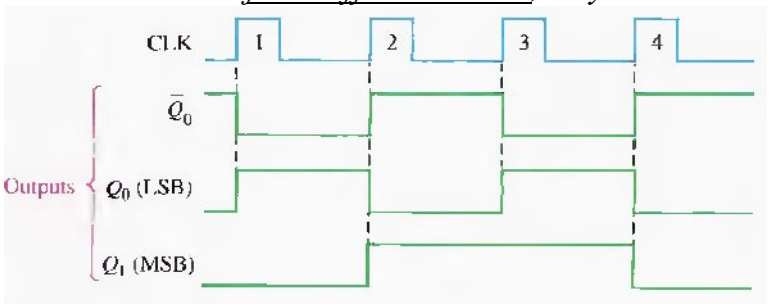


Fig 1-2 Timing diagram for the counter of Fig1-1

CLOCK PULSE	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Table 1-1Binary state sequence

The fourth pulse it recycles to its original state (Q_0=0, Q_1=0). The term recycles; it refers to the transition of the counter from its final state back to its original state.

A 3-Bit Asynchronous Binary Counter

CLOCK PULSE	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

Table1-2 State sequence for a 3-bit binary counter

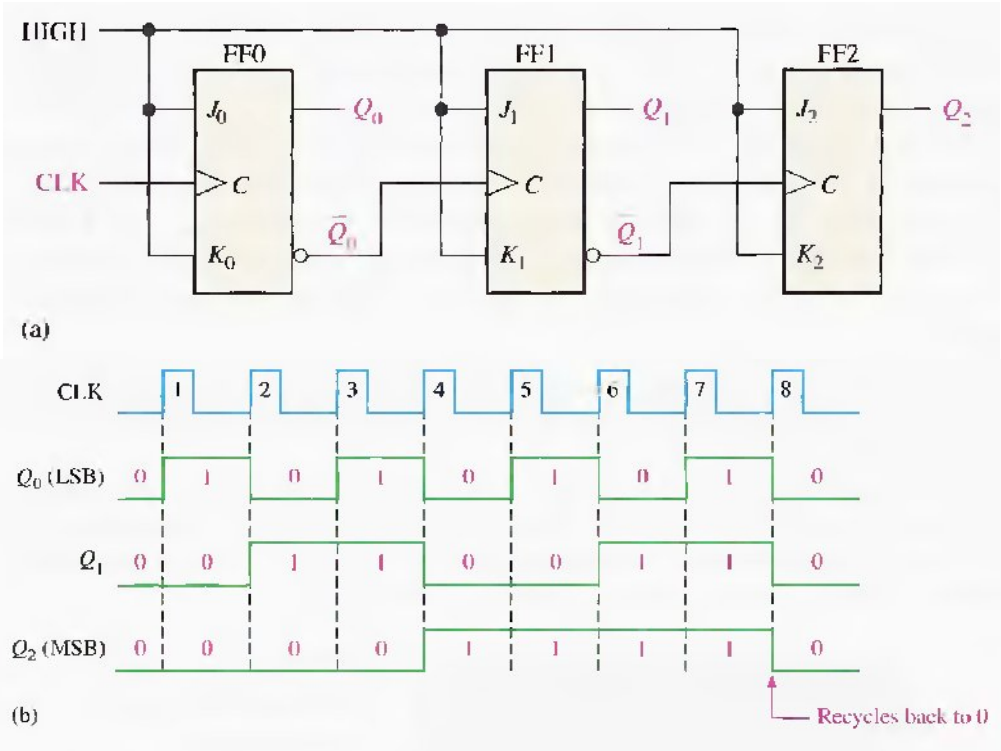


Fig 1-3 Three-bit asynchronous binary counter and its timing diagram for one cycle.

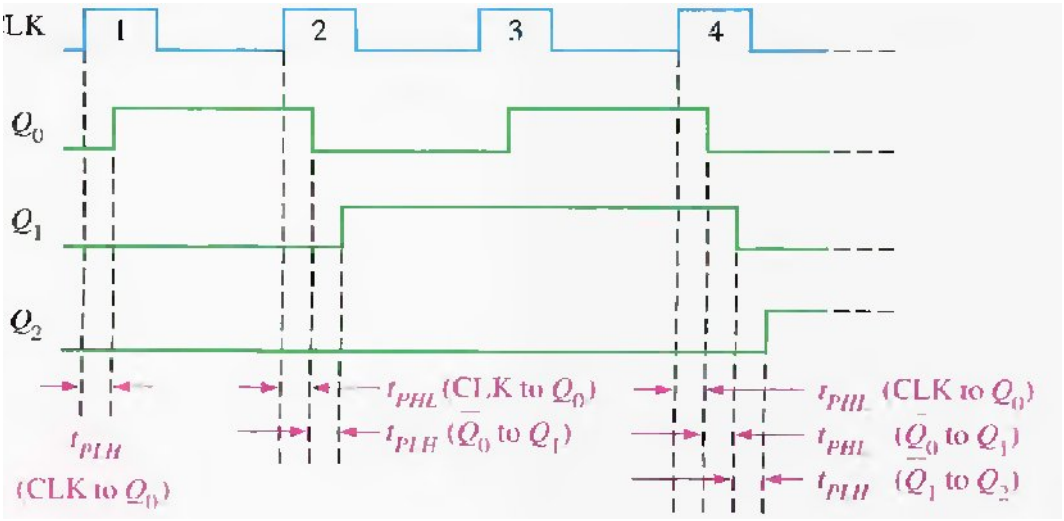


Fig 1-4Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.

***Example 1-1**: A 4-bit asynchronous binary counter is shown in Fig1-5(a). Each flip-flop is negative edge-triggered and has a propagation delay for 10 ns. Develop a timing diagram showing the Q output of each flip-flop, and determine the total propagation delay time from the triggering edge of a clock pulse until a corresponding change can occur in the state of Q_3, Also determine the maximum clock frequency at which the counter can be operated.*

***Solution**: The timing diagram with delays omitted is as shown in Fig1-5(b). For the total delay time, the effect of CLK8 or CLK 16 must propagate through four flip-flops before Q_3 changes, so* $t_{P(tot)} = 4 \times 10\ ns = 40\ us$

The maximum clock frequency is $f_{max} = 1/t_{P(tot)} = 1/40\ ns = 25\ MHz$

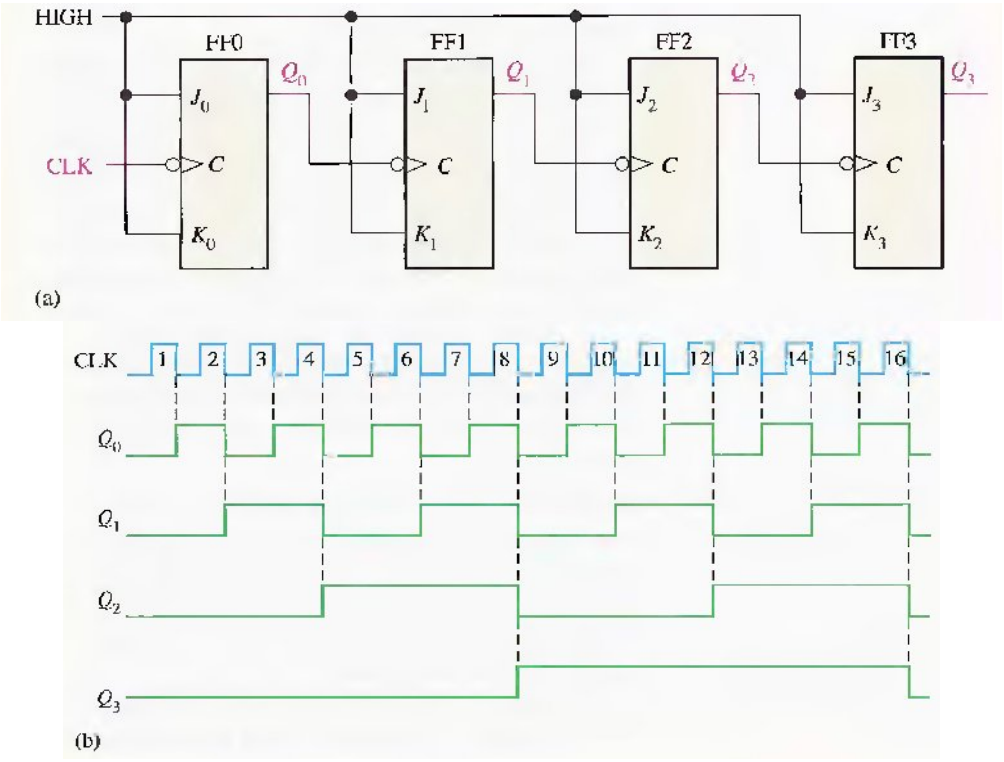


Fig 1-5 4-bit asynchronous binary counter and its timing diagram.

Related Problem: *Show the timing diagram if all of the flip-flops in Fig1-5(a) are positive edge- triggered.*

Asynchronous Decade Counters

The modulus is the number of unique states through which the counter will sequence. The maximum possible number of states of a counter is 2^n where n is the number of flip-flops. Counters can be designed to have a number of states in their sequence that is less than the maximum of 2^n. This type of sequence is called a truncated sequence. One common modulus for counters with truncated sequences is 10 (Modules10). A decade counter with a count sequence of zero (0000) through 9 (1001) is a BCD decade counter because its 10-state sequence produces the BCD code. To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. A decade counter requires 4 flip-flops. One way to make the counter recycle after the count of 9 (1001) is to decode count 10 (1010) with a NAND gate and connect the output of the NAND gate to the clear (CLR) inputs of the flip-flops, as shown in Fig1-6(a).

Partial Decoding: *in Fig1-6(a) only Q_1 & Q_3 are connected to the NAND gate inputs. This arrangement is an example of partial decoding, in which the two unique states (Q_1=1 & Q_3=1) are sufficient to decode the count of 10 because none of the other states (0 through 9) have both Q_1 & Q_3 HIGH at the same time.*

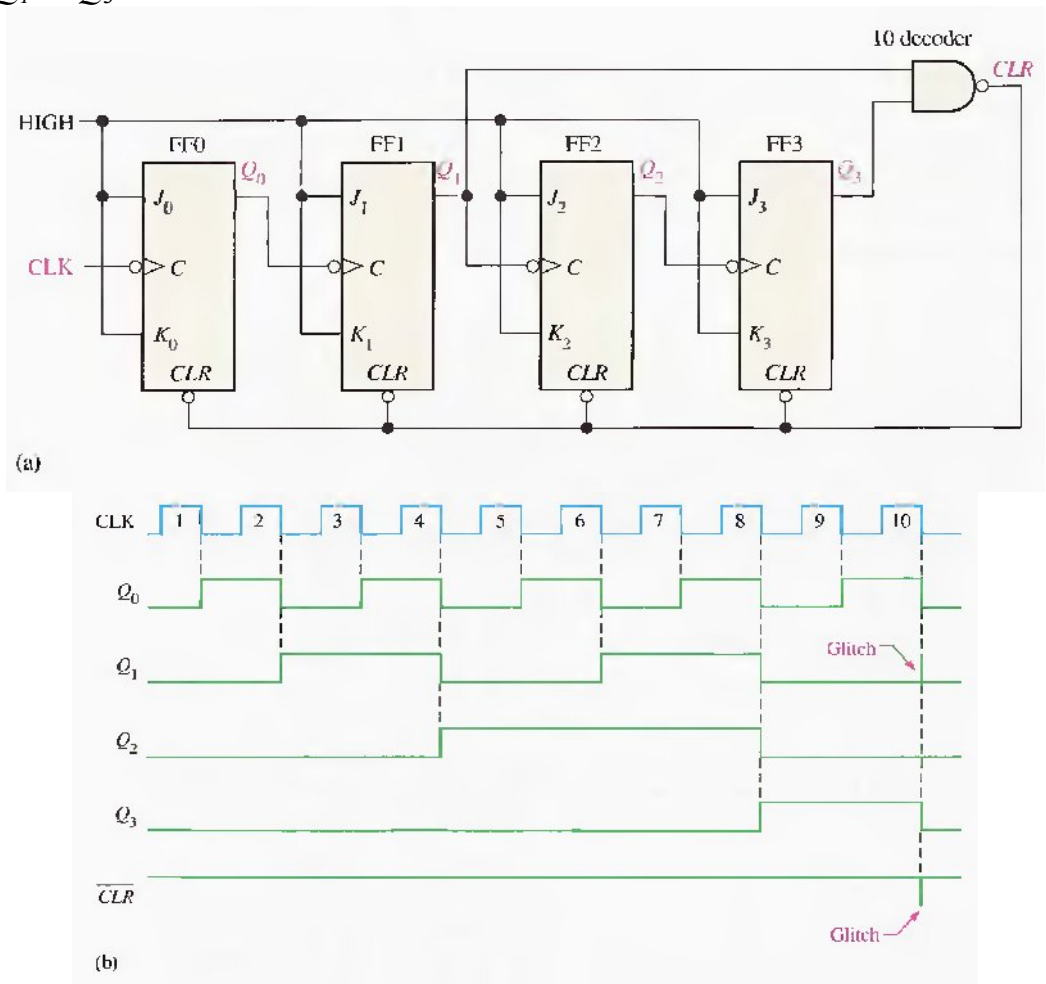
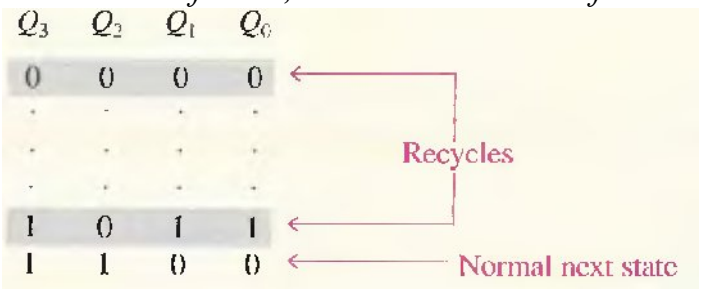


Fig 1-6 an asynchronously clocked decade counter with asynchronous recycling.

Example1-2: *Show how an asynchronous counter can be implemented having a modulus of 12 with a straight binary sequence from 0000 through 1011*

Solution*: 4 flip-flops are required to produce any modulus greater than 8 but less than or equal to 16. When the counter gets to its last state.1011, it must recycle back to 0000 rather than going to its normal next state of 1100, as illustrated in the following sequence chart:*



Observe that Q_0 & Q_1 both go to 0 anyway, but Q_2 & Q_3 must be forced to 0 on the 12 clock pulse. Fig1-7a shows the modulus-12 counter. The NAND gate partially decodes count 12 (1100) and resets flip-flop2 & flip-flop 3. Thus, on the 12 clock pulse, the counter is forced to recycle from count 11 to count 0, as shown in the timing diagram of fig1-7b.

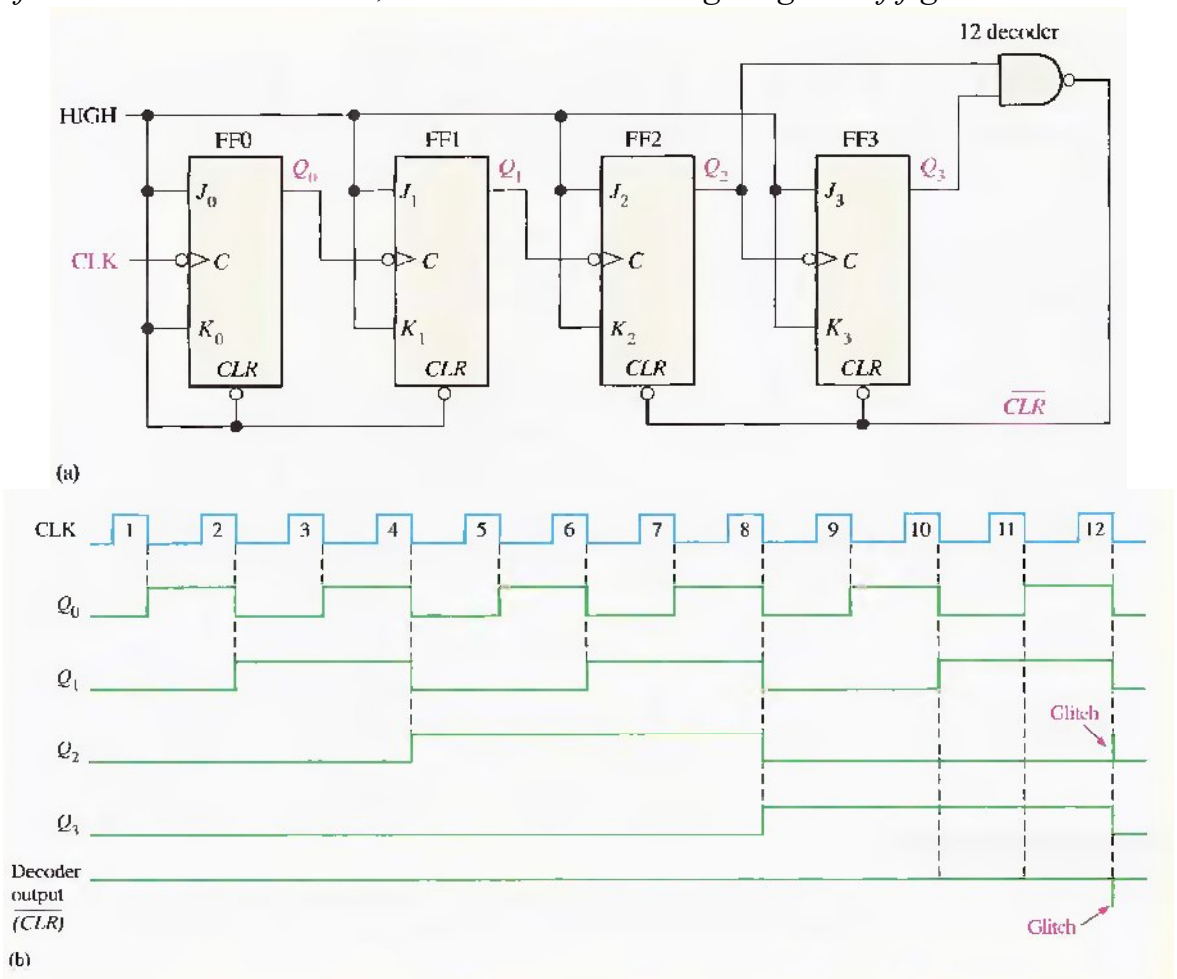


Fig1-7 asynchronously clocked modulus-12 counter with asynchronous recycling

Related Problem *How can the counter in fig1-7a be modified to make it a modulus-13 counter?*

2- Synchronous counter operation

In synchronous counters, the clock input is connected to all of the flip-flops so that they are clocked simultaneously. The term synchronous refers to events that have a fixed time relationship with each other.

A 2-Bit Synchronous Binary Counter

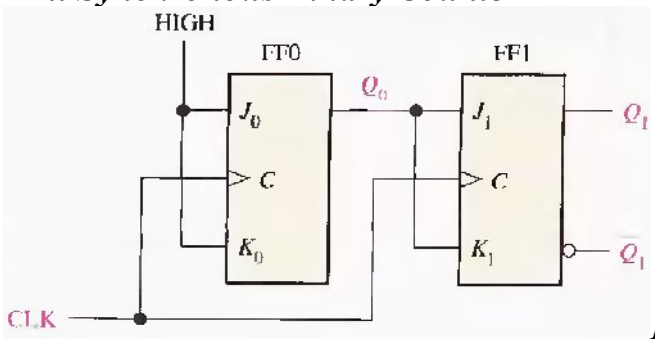


Fig 1-8 A 2-bit synchronous binary counter

First, when the positive edge of the first clock pulse is applied, FF0 will toggle and Q_0 will therefore go HIGH. When FF1at the positive-going edge of CLK1 inputs J_1 & K_1 are both LOW because Q_0 has not yet gone HIGH. So, J_1=0 & K_1=0 . This is a no-change condition, and therefore FF1 does not change state (fig1-9a).

When the leading edge of CLK2 occurs, FF0 will toggle and Q_0 will go LOW and Q_1 goes HIGH. Thus, after CLK2, Q_0=0 & Q_1=1 (Fig1-9b).

When the leading edge of CLK3 occurs, FF0 again toggles to the SET state (Q_0= 1), and FF1 remains SET (Q_1=1 . After this triggering edge, Q_0 =1 & Q_1 =1 (Fig1-9c).

at the leading edge of CLK4, Q_0 & Q_1 go LOW (Fig1-9d).

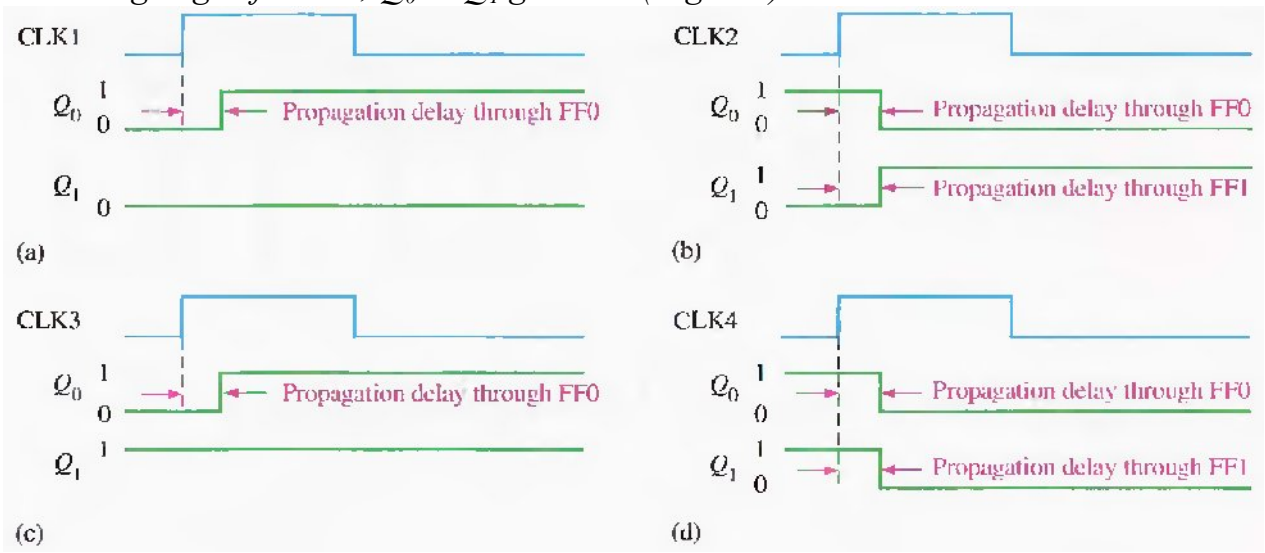


Fig 1-9 timing details for the1-bit synchronous counter operation

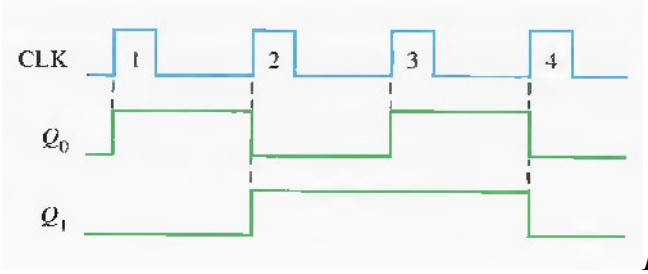


Fig 1-10 complete timing diagram for the counter

A 3-Bit Synchronous Binary Counter

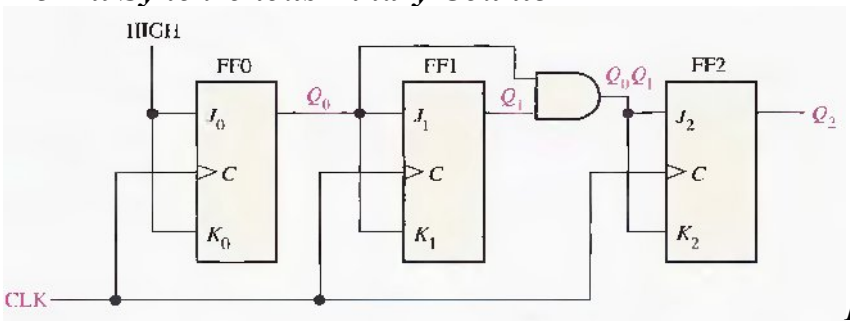


Fig1-11 3-bit synchronous counter

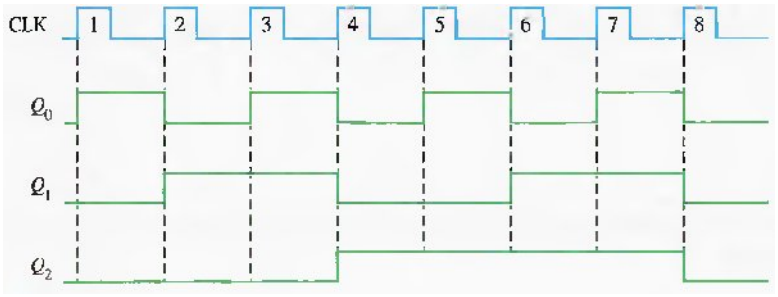
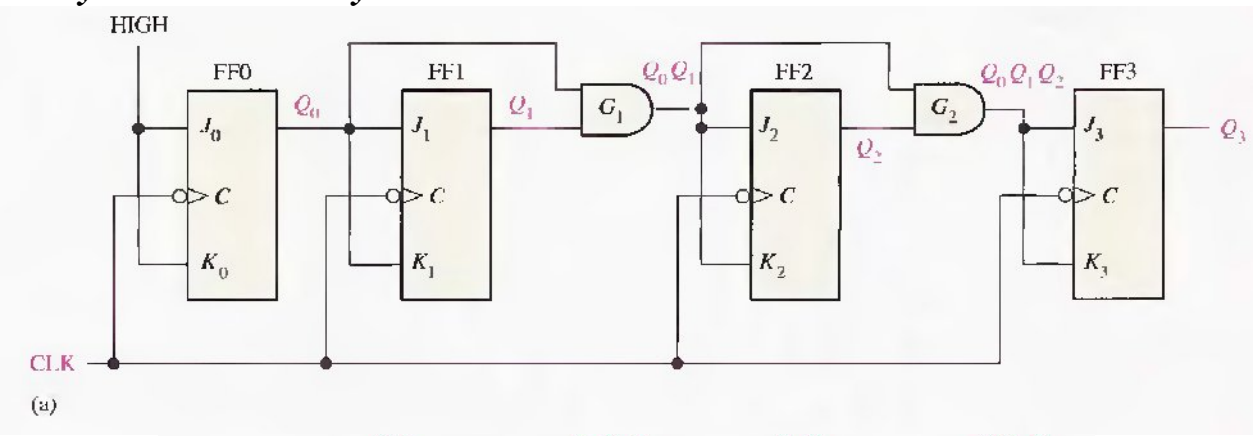


Fig1-12 the timing diagram

CLOCK PULSE	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

Table1-3 Binary state sequence

A 4-Bit Synchronous Binary Counter



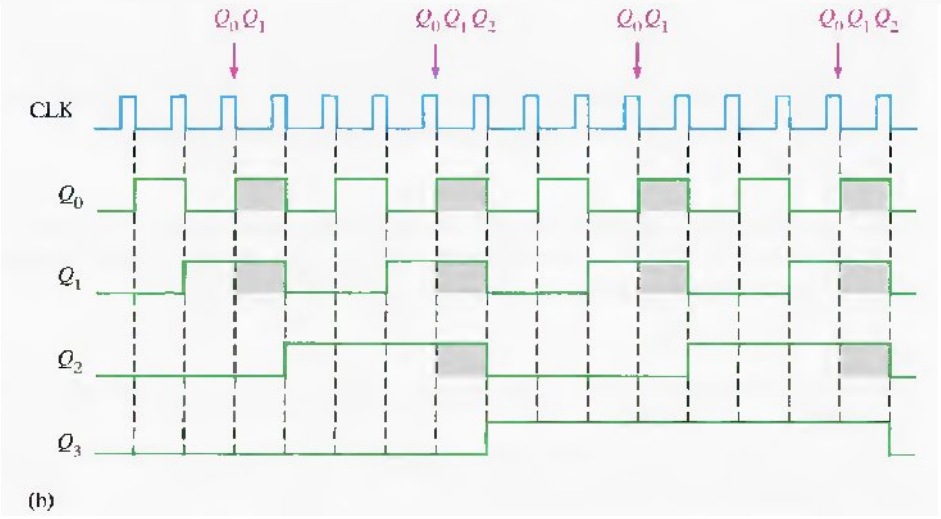


Fig1-13 a 4-bit synchronous binary counter and timing diagram. Points where the AND gate outputs are HIGH are indicated by the shaded areas.

A 4-Bit Synchronous Decade Counter

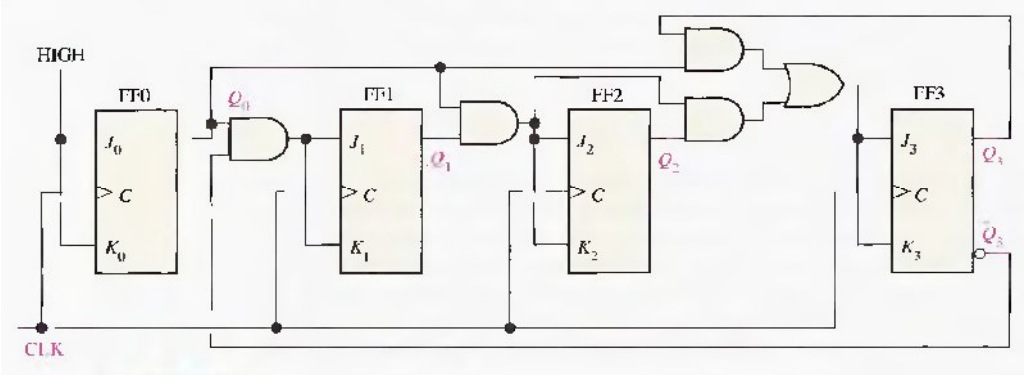


Fig1-14 a synchronous BCD decade counters

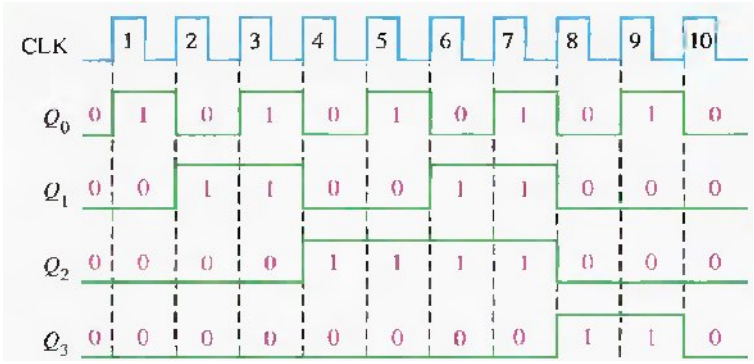


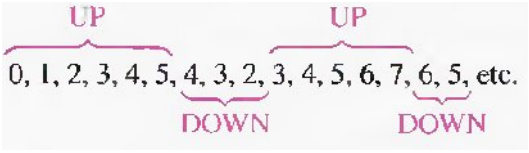
Fig1-15 Timing diagram for the BCD Counter

CLOCK PULSE	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycles)	0	0	0	0

table 1-4 States of a BCD decade

3-Up-Down counter

An up/down (bidirectional) counter is one that is capable of progressing in either direction through a certain sequence.



An examination of Q_0 for both the up and down sequences shows that FF0 toggles on each clock pulse. Thus, the J_0 & K_0 inputs of FF0 are J_0=K_0=1. For the up sequence, Q_1 changes state on the next clock pulse when Q_0 =1. For the down sequence, Q_1 changes on the next clock pulse when Q_0= 0. Thus, the J_1 & K_1 inputs of FF1 must equal 1 under the conditions expressed by the following equation:

$$J_1 = K_1 = (Q_0 .UP) + (\bar{Q}_0 .DOWN)$$

For the up sequence, Q_2 changes state on the next clock pulse when Q_0=Q_1=1. For the down sequence, Q_2 changes on the next clock pulse when Q_0=Q_1= 0. Thus, the J_2 & K_2 inputs of FF2 must equal 1 under the conditions expressed by the following equation:

$$J_2 = K_2 = (Q_0 .Q_1.UP) + (\bar{Q}_0 .\bar{Q}_1 .DOWN)$$

Notice that the UP/ $\bar{}$DOWN control input is HIGH for UP and LOW for DOWN.

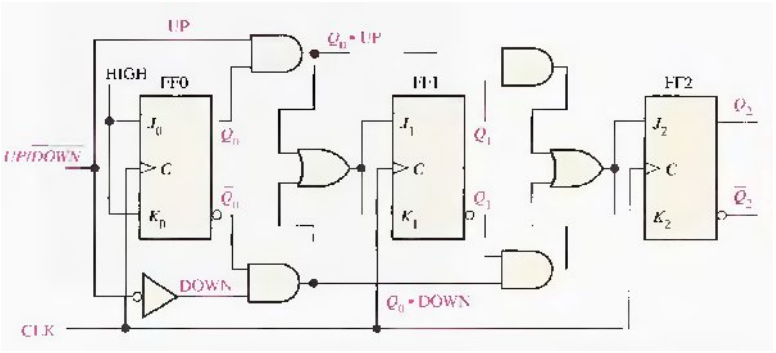


Fig1-16 a basic 3-bit up/down synchronous counter

CLOCK PULSE	UP	Q_2	Q_1	Q_0	DOWN
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

table1-5 Up/Down sequence

***Example1-4**: Show the timing diagram and determine the sequence of a 4-bit synchronous binary up/down counter if the clock and UP/¯DOWN control inputs have waveforms as shown in fig1-7a. The counter starts in the all 0s state and is positive edge-triggered.*
***Solution**: From the waveforms of fig2-17b, the counter sequence is as shown in Table 1-6.*

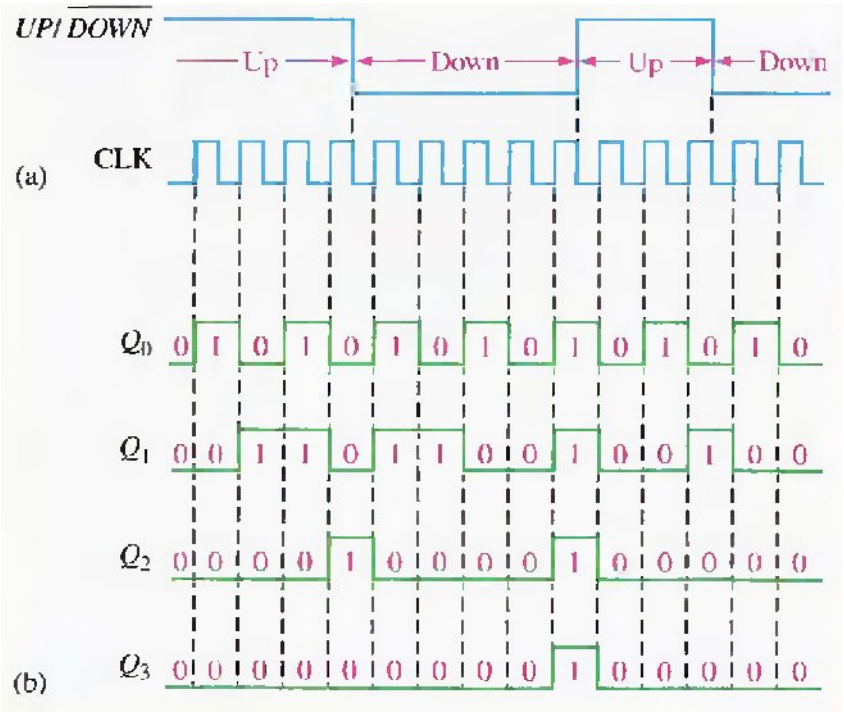


Fig 1-17

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	
0	0	0	1	
0	0	1	0	UP
0	0	1	1	
0	1	0	0	
0	0	1	1	
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	
1	1	1	1	
0	0	0	0	
0	0	0	1	UP
0	0	1	0	
0	0	0	1	
0	0	0	0	DOWN

Table 1-6

***Related Problem** Show the timing diagram if the UP/ ¯DAWN control waveform in fig1-17a is inverted.*

Design of synchronous counters
General Model of a Sequential Circuit
a general sequential circuit consists of a combinational logic section and a memory section (flip-flops), as shown in Fig 1-18.

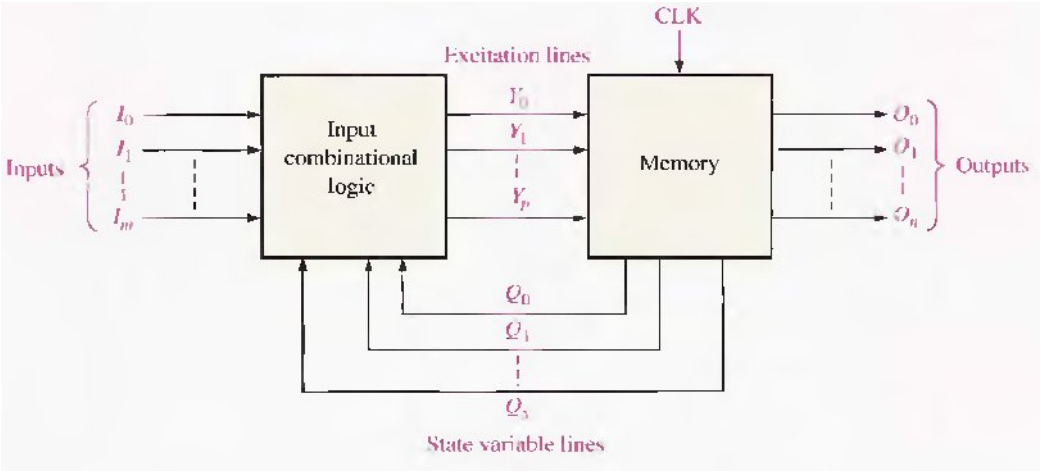


Figure 1-18 General clocked sequential circuit

The information stored in the memory section, as well as the inputs to the combinational logic (I_0, I_1, . . ., I_m), is required for proper operation of the circuit. At any given time the memory is in a state called the present state and will advance to a next state on a clock pulse a determined by conditions on the excitation lines (Y_0, Y_1 , . . . , Y_p) the present state of the memory is represented by the state variables (Q_0 , Q_1, . . . , Q_t). These state variables, along with the inputs (I_0, I_1,. . . . , I_m) determine the system outputs (O_0 , O_1, . . . , O_n).

a general design procedure for sequential circuits is applied to synchronous counters in a series of steps:

Step 1: State Diagram

The first step in the design of a counter is to create a state diagram. A state diagram shows the progression of states through which the counter advances when it is clocked. As an example, Fig1-19 is a state diagram for a basic 3-bit Gray code counter. This particular circuit has no inputs other than the clock and no outputs other than the outputs taken off each flip-flop in the counter.

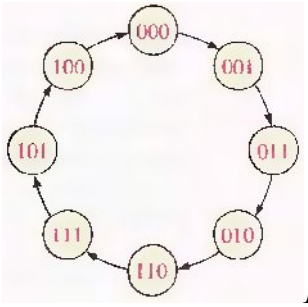


Fig 1-19 State diagram for a 3-bit Gray code counter

Step 2: Next-State Table

Once the sequential circuit is defined by a state diagram, the second step is to derive a next-state table, which lists each state of the counter (present state) along with the corresponding next state. The next state is the state that the counters guess to from its present state upon application of a clock pulse the next-state table is derived from the state diagram and is shown in Table 1-7 for the 3-bit Gray code counter. Q_0 is the least significant bit.

PRESENT STATE			NEXT STATE		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Table1-7 Next-state table for 3-bit Gray code counter.

Step 3: Flip-Flop Transition Table

Table 1-8 is a transition table for the J-K flip-flop. All possible output transitions are listed by showing the Q output of the flip-flop going from present states to next states. Q_N is the present state of the flip-flop (before a clock pulse) and Q_{N+1} is the next state (after a clock pulse). For each output transition the J & K inputs that will cause the transition to occur are listed. An X indicates a "don't care" (the input can be either a 1 or 0).

OUTPUT TRANSITIONS			FLIP-FLOP INPUTS	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Q_N: present state

Q_{N+1}: next state

X: "don't care"

Table1-8 Transition table for a J-K flip-flop

To design the counter, the transition table is applied to each of the flip-flops in the counter, based on the next-state table (Table 1-7). For example, for the present state 000, Q_0 goes from a present state of 0 to a next state of 1. To make this happen J_0 must be a 1 and you don't care what K_0 is (J_0= 1, K_0 = X), as you can see in the transition table (Table 1-8). Next, Q_1 is 0 in the present state and remains 0 in the next state. For this transition, J_1 = 0 & K_1 = X. Q_2 is 0 in the present state and remains 0 in the next state. Therefore, J_2= 0 & K_2 = X.

Step 4: Karnaugh Maps

Karnaugh maps can be used to determine the logic required for the J & K inputs of each flip-flop in the counter. There is a Karnaugh map for the J input & a Kamaugh map for the K input of each flip-flop. Each cell in a Karnaugh Maps represents one of the present states in the counter sequence listed in Table 1-7. From the J & K states in the transition table (Table 1-8) a 1,0, or X is entered into each present state cell on the maps depending on the transition of the Q output for a particular flip- flop. To illustrate this procedure, two sample entries are shown for the J_0 & K_0 inputs to the least significant flip-flop (Q_0) in Fig 1-20.

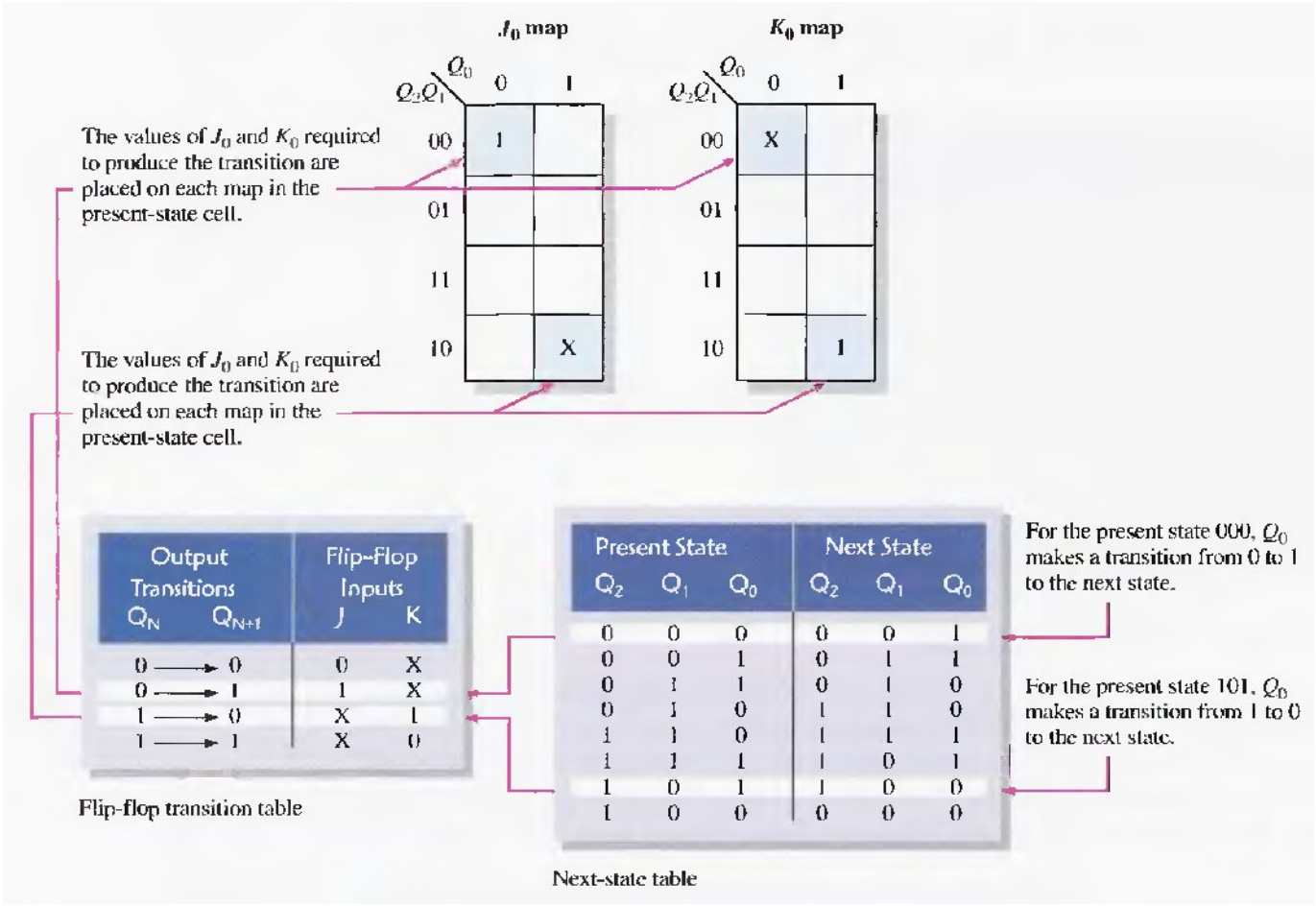


Fig1-20Examples of the mapping procedure for the counter sequence

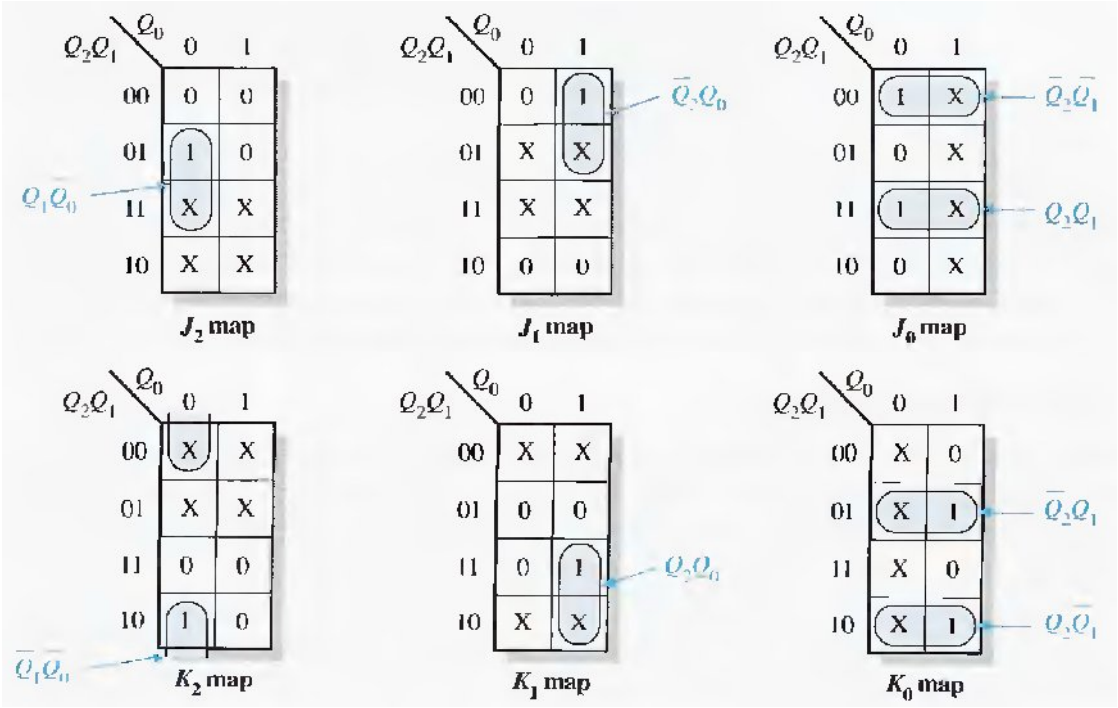


Fig1-21 Q( ) Karnaugh maps for present-state J & K inputs.

*The completed **Karnaugh Maps** for all three flip-flops in the counter are shown in fig1-21. The cells are grouped as indicated and the corresponding Boolean expressions for each group are derived.*

Step 5: logic Expressions for Flip-Flop Inputs

From the Karnaugh maps of fig1-21 you obtain the following expressions for the J & K inputs of each flip-flop:

$$J_0 = Q_2Q_1 + \overline{Q}_2\overline{Q}_1 = \overline{Q_2 \oplus Q_1}$$
$$K_0 = Q_2Q_1 + Q_2Q_1 = Q_2 \oplus Q_1$$
$$J_1 = \overline{Q}_2Q_0$$
$$K_1 = Q_2Q_0$$
$$J_2 = Q_1\overline{Q}_0$$
$$K_2 = \overline{Q}_1\overline{Q}_0$$

Step 6: Counter Implementation

The final step is to implement the combinational logic from the expressions for the J & K inputs and connect the flip-flops to form the complete 3-bit Gray code counter as in fig 1-22.

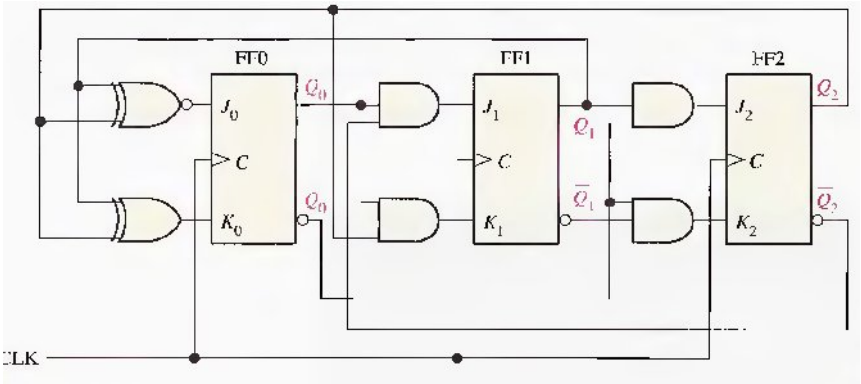


Fig1-22 3-bit Gray code counters

A summary of steps used in the design of this counter follows. In general, these steps can be applied to any sequential circuit:

1. Specify the counter sequence and draw a state diagram.

2. Derive a next-state table from the state diagram.

3. Develop a transition table showing the flip-flop inputs required for each transition, The transition table is always the same for a given type of flip-flop.

4. Transfer the J & K states from the transition table to Karnaugh maps. There is a Karnaugh map for each input of each flip-flop.

5. Group the Karnaugh map cells to generate and derive the logic expression for each flip-flop input.

6. implement the expressions with combinational logic, and combine with the flip-flops to create the counter.

Example1-5:*Design a counter with the binary count sequence shown in the state diagram of Fig 1-23. Use J-K flip-flops.*

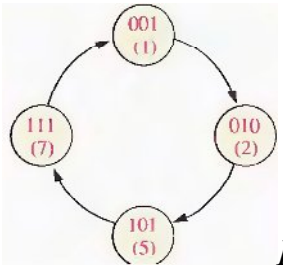


Fig1-23 state diagram for Ex5

Solution:

Step 1*: Although there are only 4 states, a 3-bit counter is required to implement this sequence because the maximum binary count is 7. Since the required sequence dues not include all the possible binary, states, the invalid states (0, 3, 4, & 6) can be treated as "don't cares" in the design. However, if the counter should erroneollsly get into an invalid state, you must make sure that it goes back to a valid state.*

Step 2*: The next-state table is developed from the state diagram and is given in Table 1-9.*

PRESENT STATE			NEXT STATE		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	1	0	1	0
0	1	0	1	0	1
1	0	1	1	1	1
1	1	1	0	0	1

Table 1-9Next state table

Step 3:

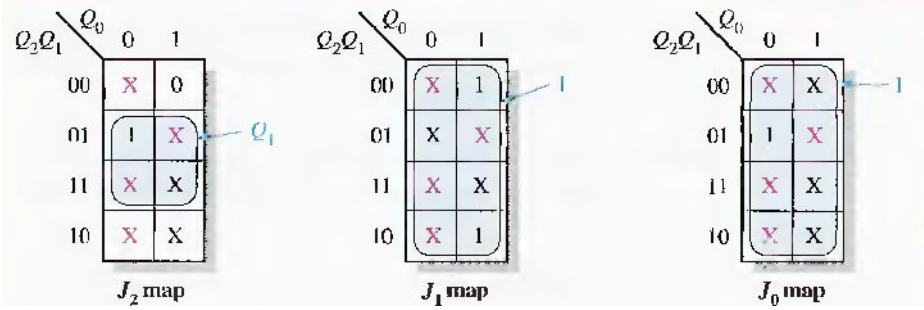
The transition table for the J-K flip-flop is repeated in Table 1-10.

OUTPUT TRANSITIONS			FLIP-FLOP INPUTS	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Table 1-10 transition table for J-K flip-flop

Step 4*:*

The J & K inputs are plotted on the present-state Karnaugh maps in fig1-24. Also "don't cares" can be placed in the cells corresponding to the invalid states of 000, 011, 100, and 110, as indicated by the Xs.



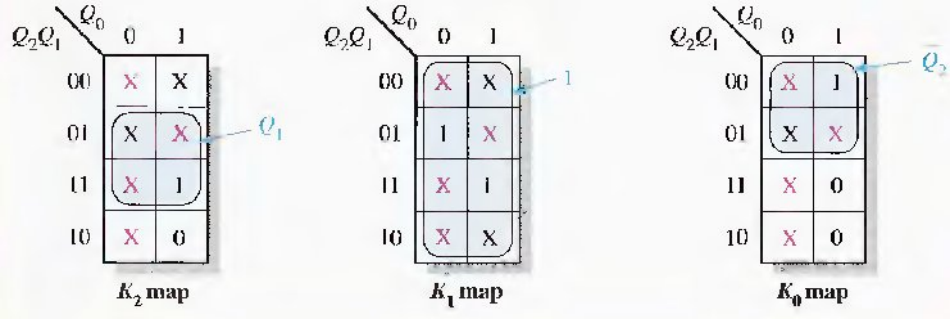


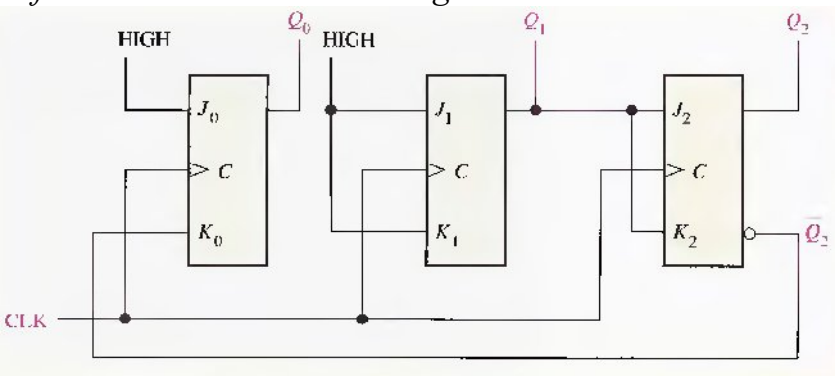
Fig1-24

***Step 5**:*

Group the 1s, taking advantage of as many of the "don't care" states as possible for maximum simplification(fig1-24). Notice that when all cells in a map are grouped, the expression is simply equal to1. The expression for each J & K input taken from the maps is as follows:

***Step 6**:*

The implementation of the counter is shown in Fig1-25.



$J_0 = 1, K_0 = \overline{Q}_2$

$J_1 = K_1 = 1$

$J_2 = K_2 = Q_1$

Fig1-25

Analysis shows that if the counter, by accident, gets into one of the invalid states (0, 3, 4, 6), it will always return to a valid state according to the following sequences: 0⟶3⟶ 4 ⟶7. & 6⟶ 1

Cascaded counter

Counters can be connected in cascade to achieve higher-modulus operation. In essence, cascading means that the last age output of one counter drives the input of the next counter.

An example of two counters connected in cascade is shown in fig1-26 for a 2-bit & a 3-bit ripple counter.

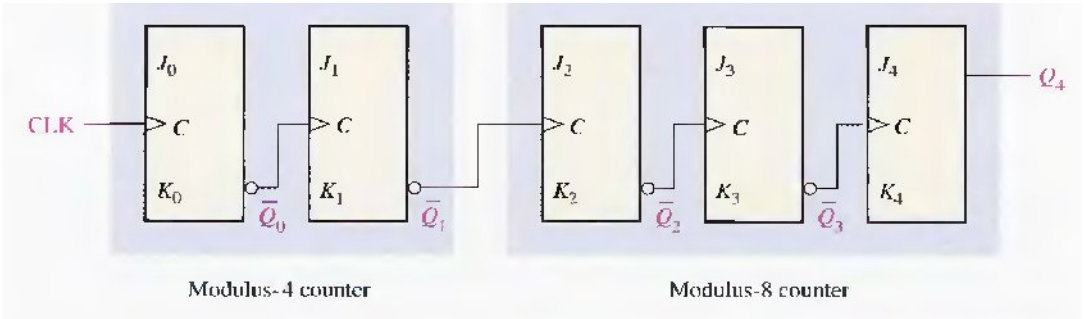


Fig1-26 Two cascaded counters (all J & K inputs are HIGH).

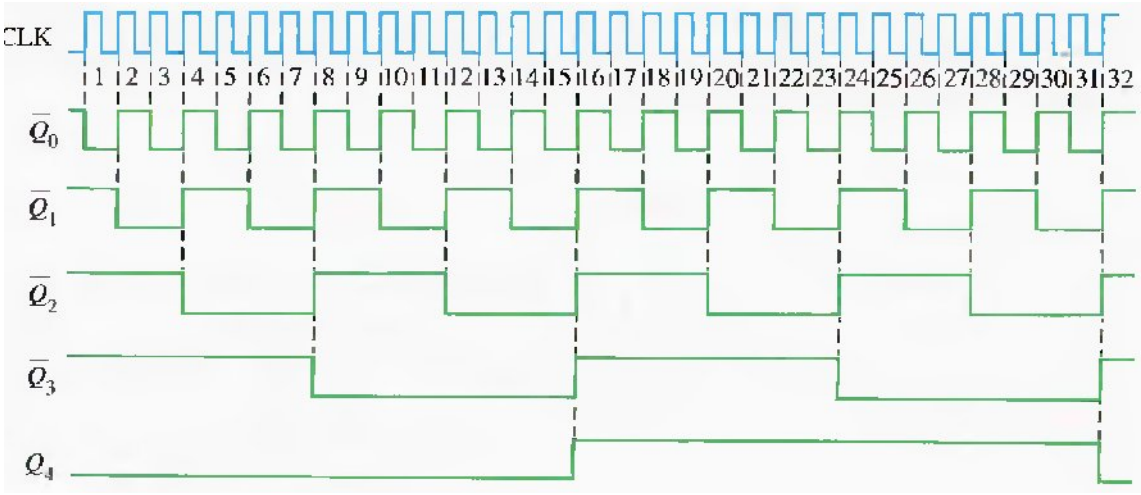


Fig1-27 Timing diagram for the cascaded counter configuration of Fig1-26

The timing diagram is shown in fig1-27. Notice that the final output of the modulus-8 counter, Q_4 , occurs once for every 32 input clock pulses. The overall modulus of the cascaded counters is 32; that is, they act as a divide-by-32 counter.
*When operating synchronous counters in a cascaded configuration, it is necessary to **e** the count enable and the terminal count functions to achieve higher-modulus operation. On some devices the count enable is CTEN (or G), and terminal count (TC) is analogous to ripple clock output (RCO) on some IC counters.*
Fig1-28 shows two decade counters connected in cascade. The terminal count (TC) output of counter 1 is connected to the count enable (CTEN) input of counter 2.
Counter 2 is inhibited by the LOW on its CTEN input until counter 1 reaches its last, or terminal, state and its terminal count output goes HIGH. This HIGH now enables counter 2, so that when the first clock pulse after counter 1 reaches its terminal count (CLK10), counter 2 goes from its initial state to its second state. Upon completion of the entire second cycle of counter 1 (when counter 1 reaches terminal count the second time), counter 2 is again enabled and advances to its next state. This sequence continues. Since these are decade counters. Counter1 must go through ten complete cycles be fore counter 2 completes its first cycle. In other words, for every ten cycles of counter 1, counter 2 goes through one cycle. Thus, counter 2 will complete one cycle after one hundred clock pulses. The overall modulus of these two cascaded counters is 10 x 10 = 100.

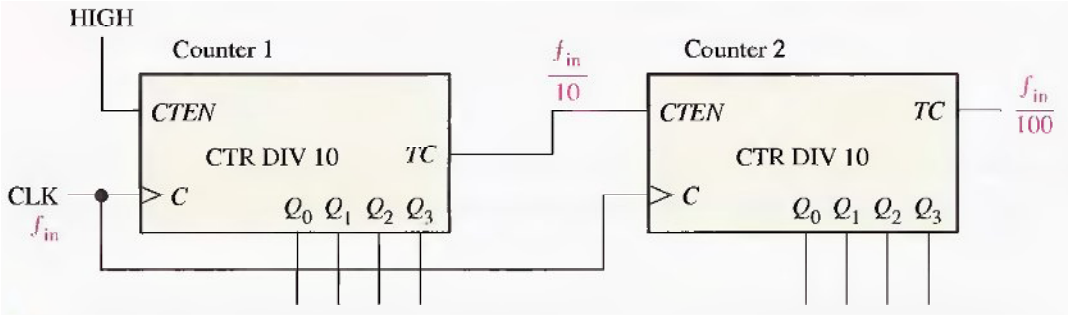


Fig1-28 A modulus-100 counter using two cascaded decade counters.

When viewed as a frequency divider, the circuit of fig1-28 divides the input clock frequency by 100. Cascaded counters are often used to divide a high-frequency clock signal to obtain highly accurate pulse frequencies. Cascaded counter configurations used for such purposes are sometimes called countdown chains. For example, suppose that you have a basic clock frequency of 1 MHz and you wish to obtain 100 kHz, 10 kHz, and 1 kHz: a series of cascaded decade counters can be used. If the 1 MHz signal is divided by 10, the output is 100 kHz. Then if the 100 kHz signal is divided by 10, the output is 10 kHz. Another division by 10 produces the 1 kHz frequency. The general implementation of this countdown in fig1-29

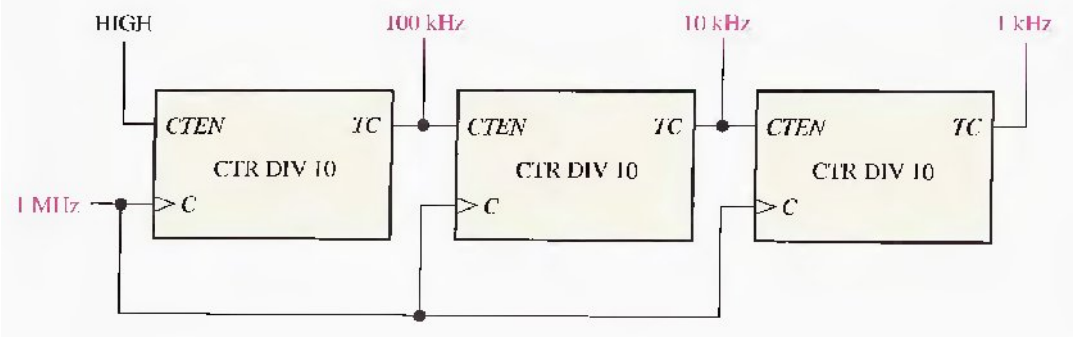


Fig1-29 Three cascaded decade counters forming a divide-by-1000 frequency divider with intermediate divide-by-10 and divide-by-100 outputs.

Example1-7 :*Determine the overall modulus of the two cascaded counter configurations in fig1-30.*

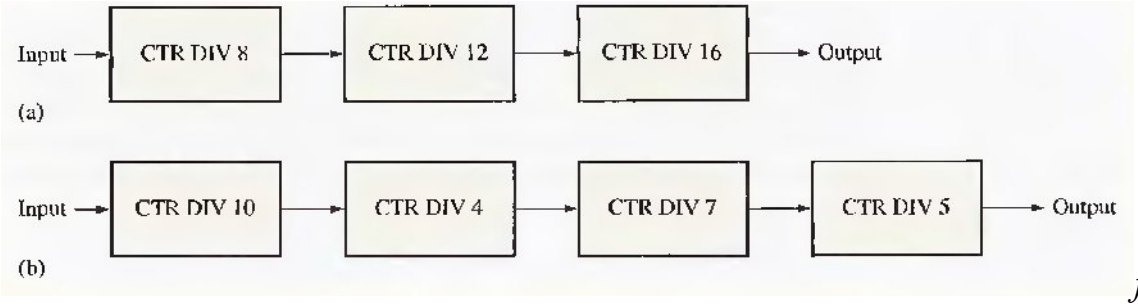


fig 1-30

Solution: *In fig1-30(a), the overall modulus for the 3-counter configuration is:*

$$8 \times 12 \times 16 = 1536$$

In fig1-3(b), the overall modulus for the 4-counter configuration is:

$$10 \times 4 \times 7 \times 5 = 1400$$

Related Problem: *How many cascaded decade counters are required to divide a clock frequency by 100,000?*

Counter decoding

In many applications, it is necessary that some or all of the counter states be decoded. The decoding of a counter involves using decoders or logic gates to determine when the counter is in a certain binary state in its sequence.

Suppose that you wish to decode binary state 6(110) of a 3-bit binary counter. When $Q_2 = 1$, $Q_1 = 1$. & $Q_0 = 0$. a HIGH appears on the output of the decoding gate, indicating that the

counter is at state 6(fig1-31). This is called active-HIGH decoding. Replacing the AND gate with a NAND gate provides active-LOW decoding.

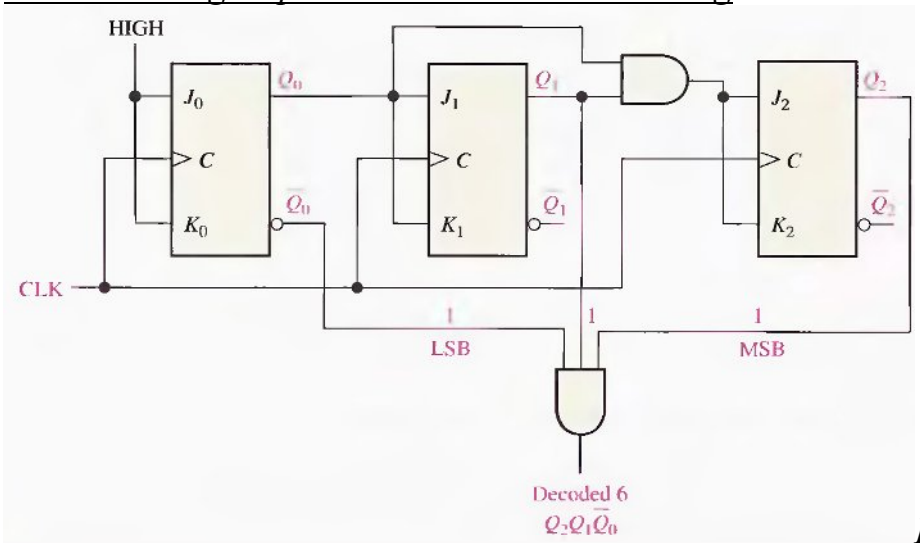
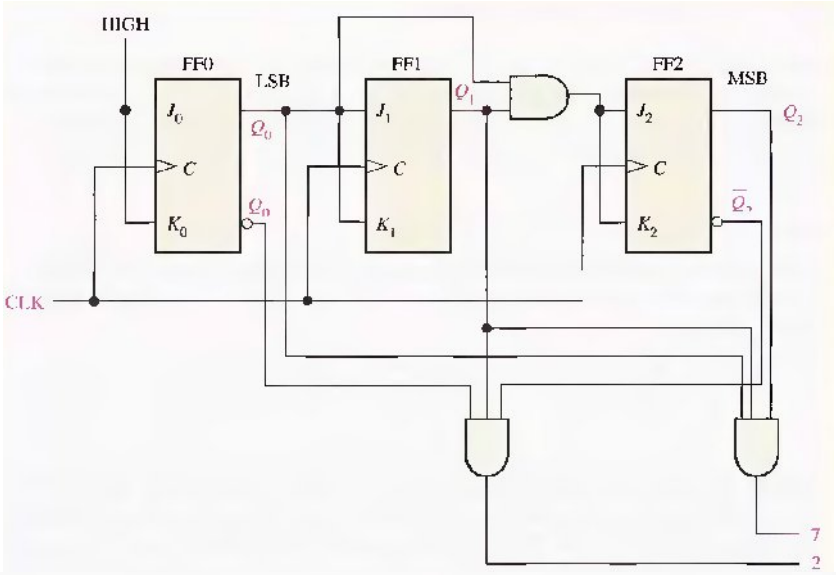


Fig 1-31 Decoding of state 6

Example:1-9:*Implement the decoding of binary state 2 and binary state 7 of a 3-bit synchronous counter. Show the entire counter timing diagram and the output waveforms of the decoding gates. Binary 2 =* $\bar{Q}_2\bar{Q}_1\bar{Q}_0$ *and binary 7 =* $Q_2Q_1Q_0$.



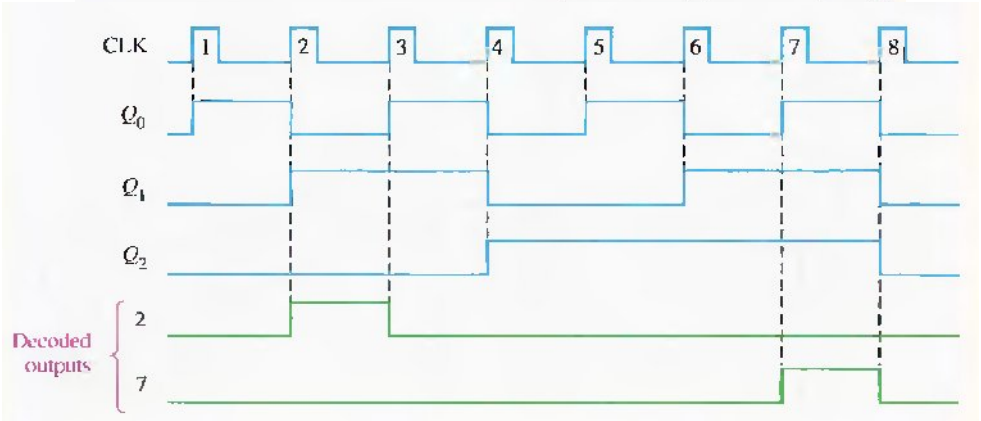


Fig1-32 A 3-bit counter with active HIGH decoding of count 2 and count 7.

Related Problem *Show the logic for decoding state 5 in the 3-bit counter.*

1-3 Counter application

A Digital Clock :*fig1-33 is a simplified logic diagram of a digital clock that displays seconds, minutes, and hours. First, a 60 Hz sinusoidal ac voltage is converted to a 60 Hz pulse waveform and divided own to a 1Hz pulse waveform by a divide-by-60 counter formed by a divide-by-10 counter allowed by a divide-by-6 counter. Both the seconds and minutes counts are also produced by divide-by-60 counters. These counters count from 0 to 59 and then recycle to 0 . Notice that the divide-by-6 is formed with a decade counter with a truncated sequence achieved by using the decoder count 6 to asynchronously clear the counter. The terminal count, 59, is also decoded to enable the next counter in the chain.*

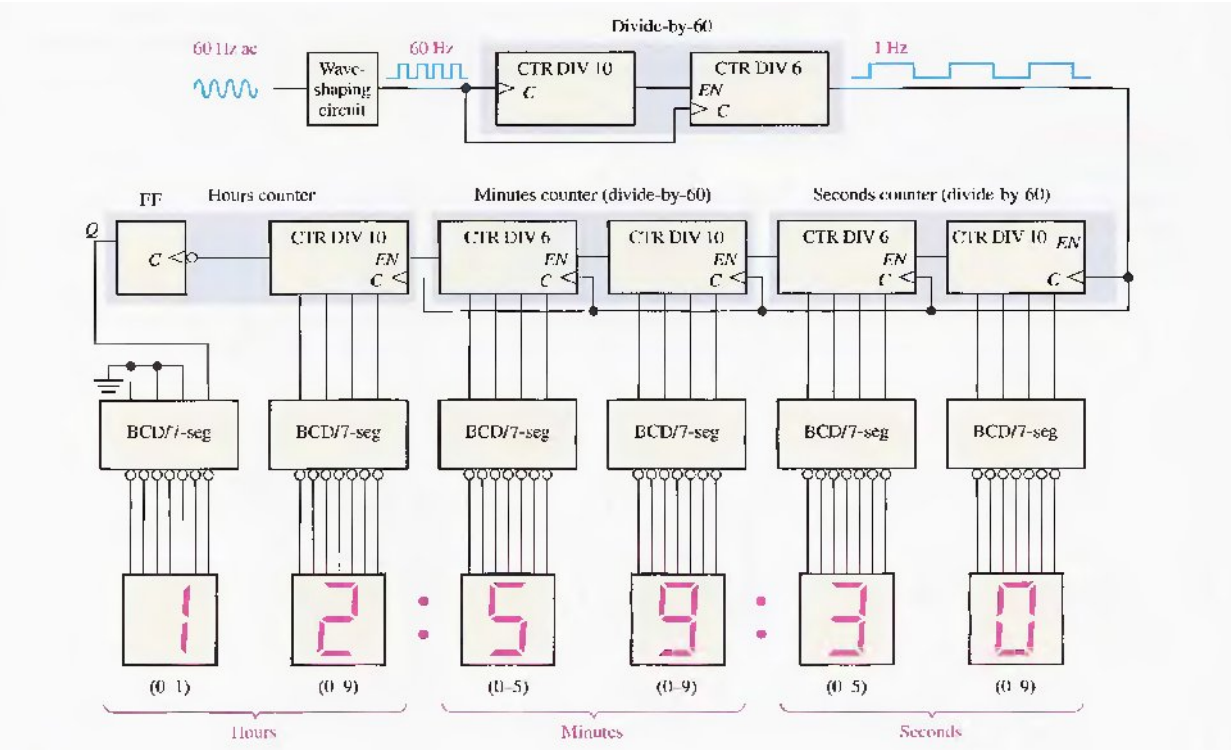


Fig 1-33 Simplified logic diagram for a 12-hour digital clock

The hours counter is implemented with a decade counter and a flip-flop as shown in fig1-34. Consider that initially both the decade counter and the flip-flop are RESET, and the decode-12 gate and decode-9 gate outputs are HIGH. The decade counter advances through all of its states from zero to nine, and on the clock pulse that recycles it from nine back to zero, the flip-flop goes to the SET state (J=1, K=0). This illuminates a 1 on the tens-of-hours display. The total count is now ten (the decade counter is in the zero state and the flip-flop is SET).
Next, the total count advances to eleven and then to twelve. In state 12 the Q2 output of the decade counter is HIGH, the flip-flop is still SET, and thus the decode-12 gate output is LOW. This activates the $\overline{PE}$ input of the decade counter. On the next clock pulse, the decade Counter is preset to state I by the data inputs, and the flip-flop is RESET (J=0, K=1).
As you can see, this logic always causes the counter to recycle from twelve back to one rather than back to zero.

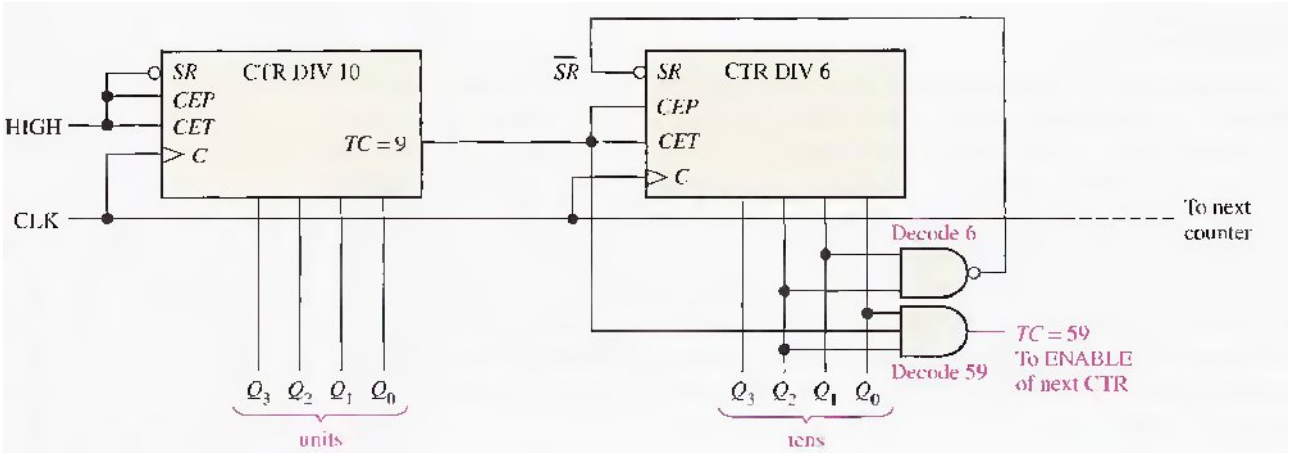


Fig 1-34 Logic diagram of typical divide-by-60 counter using synchronous decade counters. Note that the outputs are in binary order (the right-most bit is the LSB).

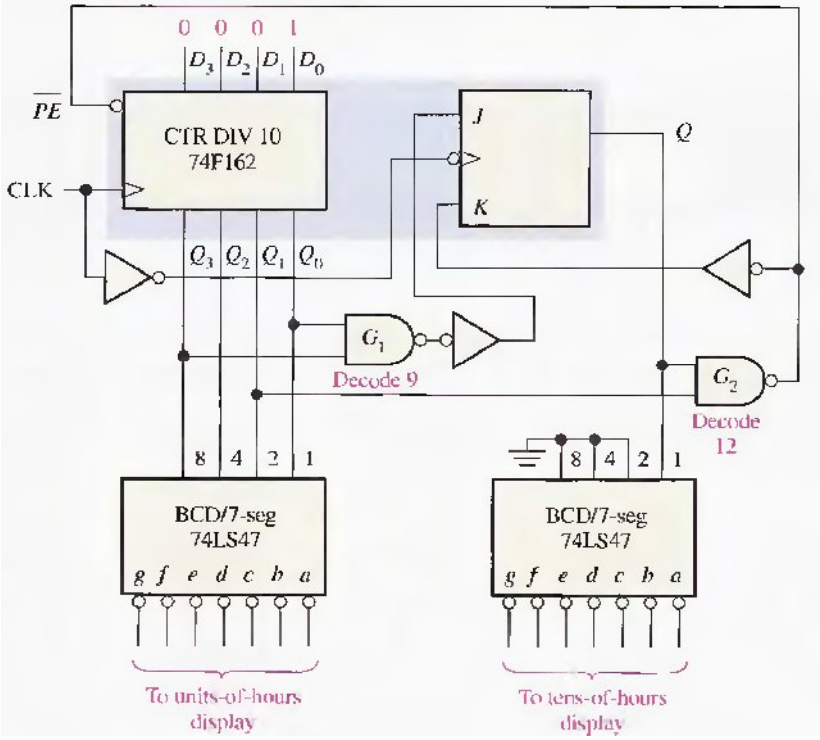
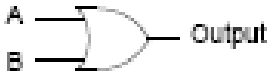
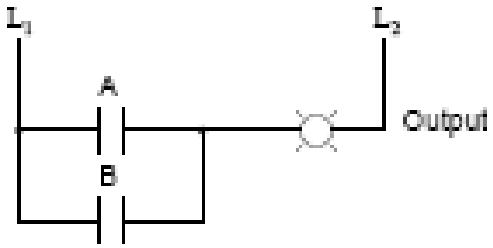


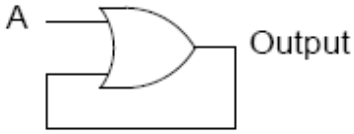
Fig1-35 Logic diagram for hours counter and decoders. Note that on the counter inputs and outputs, the right-most bit is the LSB.

Digital logic with feedback

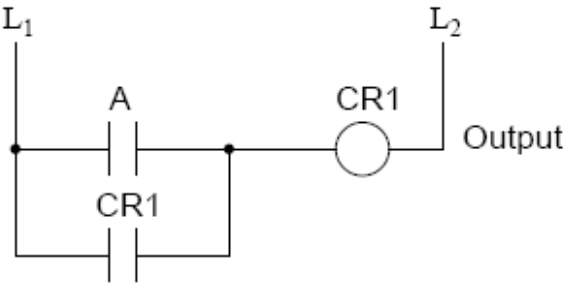


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1





A	Output
0	?
1	1

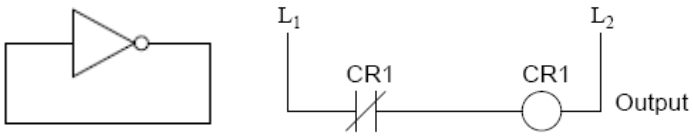


the nature of an OR gate: any high (1) input forces the output high (1). If A is low (0), however, we cannot guarantee the logic level or state of the output in our truth table. Since the output feeds back to one of the OR gate's inputs, and we know that any 1 input to an OR gates makes the output 1, this circuit will latch in the 1 output state after any time that A is 1. When A is 0, the output could be either 0 or 1, depending on the circuit's prior state! The proper way to complete the above truth table would be to insert the word latch in place of the question mark, showing that the output maintains its last state when A is 0.

Any digital circuit employing feedback is called a multivibrator. The example we just explored with the OR gate was a very simple example of what is called a bistable multivibrator. It is called bistable because it can hold stable in one of two possible output states, either 0 or 1.

There are also monostable multivibrators, which have only one stable output state (that other state being momentary), and astable multivibrators, which have no stable state (oscillating back and forth between an output of 0 and 1).

A very simple astable multivibrator is an inverter with the output fed directly back to the input:

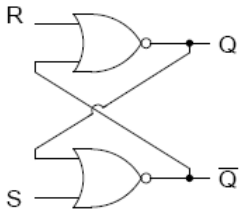


When the input is 0, the output switches to 1. That 1 output gets fed back to the input as a 1. When the input is 1, the output switches to 0. That 0 output gets fed back to the input as a 0, and the cycle repeats itself. The result is a high frequency (several megahertz) oscillator

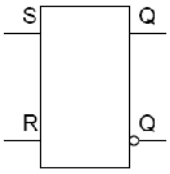
The S-R latch

Any circuit with two or more vibrators linked together is a multivibrator , a bistable multivibrator has two stable states, as indicated by the prefix bi in its name. Typically, one state is referred to as set and the other as reset. The simplest bistable device, therefore,is known as a set-reset, or S-R, latch.

To create an S-R latch, we can wire two NOR gates in such a way that the output of one feeds back to the input of another, and vice versa, like this:



S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0



The Q & ¯Q outputs are supposed to be in opposite states. I say "supposed to" because making both the S and R inputs equal to 1 results in both Q & ¯Q being 0. For this reason, having both S and R equal to 1 is called an invalid or illegal state for the S-R multivibrator.

Otherwise, making S=1 and R=0 (sets) the multivibrator so that Q=1 & ¯Q=0. Conversely, making R=1 and S=0 (resets) the multivibrator in the opposite state. When S and R are both equal to 0, the multivibrator's outputs (latch) in their prior states.

By definition, a condition of Q=1 &¯Q=0 is set. A condition of Q=0& ¯Q =1 is reset. These terms are universal in describing the output states of any multivibrator circuit. Note that the initial power-up condition of either the gate of S-R latch is such that both gates (coils) start in the de-energized mode. As such, one would expect that the circuit will start up in an invalid condition, with both Q & ¯Q outputs being in the same state. Actually, this is true! However, the invalid condition is unstable with both S and R inputs inactive, and the circuit will quickly stabilize in either the set or reset condition because one gate is bound to react a little faster than the other.

Race conditions should be avoided in circuit design primarily for the unpredictability that will be created. One way to avoid such a condition is to insert a time-delay relay into the circuit to disable one of the competing relays for a short time, giving the other one a clear advantage. In other words, by purposely slowing down the de-energization of one relay, we ensure that the other relay will always win and the race results will always be predictable.

Review:

1- A bistable multivibrator is one with two stable output states.

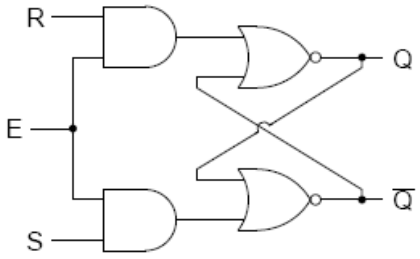
2-In a bistable multivibrator, the condition of Q=1 & ¯Q =0 is defined as set. A condition of Q=0 & ¯Q =1 is conversely defined as reset. If Q & ¯Q happen to be forced to the same state (both 0 or both 1), that state is referred to as invalid.

3-In an S-R latch, activation of the S input sets the circuit, while activation of the R input resets the circuit. If both S and R inputs are activated simultaneously, the circuit will be in an invalid condition.

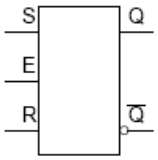
4-A race condition is a state in a sequential system where two mutually-exclusive events are simultaneously initiated by a single cause.

The gated S-R latch

It is sometimes useful in logic circuits to have a multivibrator which changes state only when certain conditions are met, regardless of its S and R input states. The conditional input is called the enable, and is symbolized by the letter E. Study the following example to see how this works:



E	S	R	Q	$\overline{Q}$
0	0	0	latch	latch
0	0	1	latch	latch
0	1	0	latch	latch
0	1	1	latch	latch
1	0	0	latch	latch
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

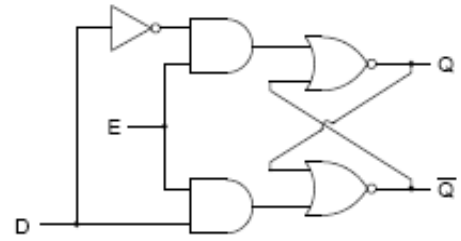


When the E=0, the outputs of the two AND gates are forced to 0, regardless of the states of either S or R. Consequently, the circuit behaves as though S and R were both 0, latching the Q & ¯Q outputs in their last states. Only when the enable input is activated (1) will the latch respond to the S and R inputs. Note the identical function in ladder logic:
A practical application of this might be the same motor control circuit (with two normallyopen pushbutton switches for start and stop), except with the addition of a master lockout input (E) that disables both pushbuttons from having control over the motor when it's low (0).

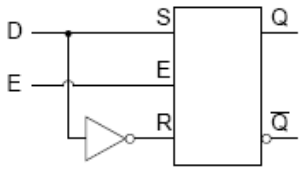
REVIEW:
1-The enable input on a multivibrator must be activated for either S or R inputs to have any effect on the output state.
2-This enable input is sometimes labeled 'E', and other times as 'EN'.

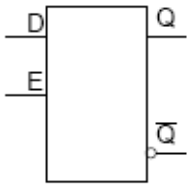
The D latch

Since the enable input on a gated S-R latch provides a way to latch the Q & ¯Q outputs without regard to the status of S or R, we can eliminate one of those inputs to create a multivibrator latch circuit with no "illegal" input states. Such a circuit is called a D latch, and its internal logic looks like this:



E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0





Note that the R input has been replaced with the complement (inversion) of the old S input, and the S input has been renamed to D. As with the gated S-R latch, the D latch will not respond to a signal input if the enable input is 0 – it simply stays latched in its last state. When the enable input is 1, however, the Q output follows the D input. Since the R input of the S-R circuitry has been done away with, this latch has no 'invalid' or "illegal" state. Q & ¯Q are always opposite of one another.

Like both the S-R and gated S-R latches, the D latch circuit may be found as its own prepackaged circuit, complete with a standard symbol:

The D latch is nothing more than a gated S-R latch with an inverter added to make R the complement (inverse) of S. Let's explore the ladder logic equivalent of a D latch, modified from the basic ladder diagram of an S-R latch:

An application for the D latch is a 1-bit memory circuit. You can write (store) a 0 or 1 bit in this latch circuit by making the enable input high (1) and setting D to whatever you want the stored bit to be. When the enable input is made low (0), the latch ignores the status of the D input and merrily holds the stored bit value, outputting at the stored value at Q, and its inverse on output $\bar{Q}$.

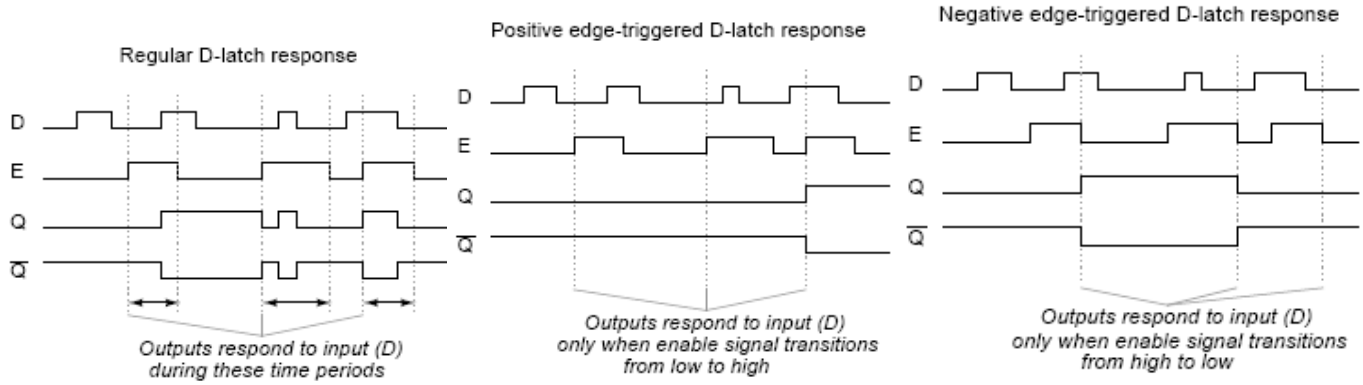
REVIEW:

1-A D latch is like an S-R latch with only one input: the D input. Activating the D input sets the circuit, and de-activating the D input resets the circuit. Of course, this is only if the enable input (E) is activated as well. Otherwise, the output(s) will be latched, unresponsive to the state of the D input.

2-D latches can be used as 1-bit memory circuits, storing either a high or a low state when disabled, and reading new data from the D input when enabled.

Edge-triggered latches: Flip-Flops

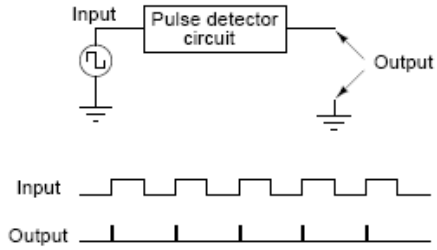
The latch responds to the data inputs (S-R or D) only when the enable input is activated. In many digital applications, it is desirable to limit the responsiveness of a latch circuit to a very short period of time instead of the entire duration that the enabling input is activated. One method of enabling a multivibrator circuit is called edge triggering, where the circuit's data inputs have control only during the time that the enable input is transitioning from one state to another.Let's compare timing diagrams for a normal D latch versus one that is edge-triggered:



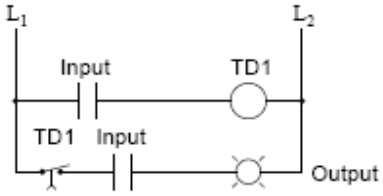
In the first timing diagram, the outputs respond to input D whenever the enable (E) input is high, for however long it remains high. When the enable signal falls back to a low state, the circuit remains latched. In the second timing diagram, we note a distinctly different response in the circuit output(s): it only responds to the D input during that brief moment of time when the enable signal changes, or transitions, from low to high. This is known as positive edge triggering. There is such a thing as negative edge triggering as well, and it produces the following response to the same input signals:

Whenever we enable a multivibrator circuit on the transitional edge of a square-wave enable signal, we call it a flip-flop instead of a latch. Consequently, and edge-triggered S-R circuit is more properly known as an S-R flip-flop, and an edge-triggered D circuit as a D flip-flop. The enable signal is renamed to be the clock signal. Also, we refer to the data inputs (S, R, and D, respectively) of these flip-flops as synchronous inputs, because they have effect only at the time of the clock pulse edge (transition), thereby synchronizing any output changes with that clock pulse, rather than at the data inputs.

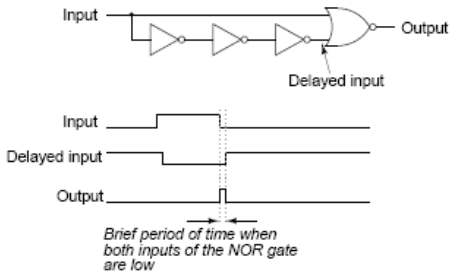
But, how do we actually accomplish this edge-triggering? What we need is a digital circuit that outputs a brief pulse whenever the input is activated for an arbitrary period of time, and we can use the output of this circuit to briefly enable the latch. We're getting a little ahead of ourselves here, but this is actually a kind of monostable multivibrator, which for now we'll call a pulse detector.

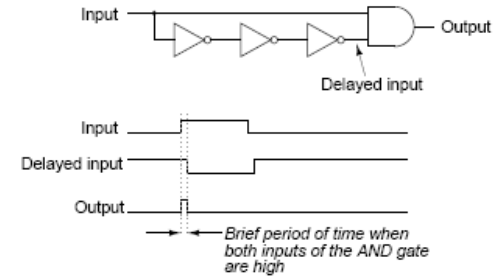


The duration of each output pulse is set by components in the pulse circuit itself. In ladder logic, this can be accomplished quite easily through the use of a time-delay relay with a very short delay time:

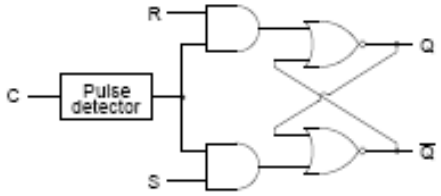


Implementing this timing function with semiconductor components is actually quite easy, as it exploits the inherent time delay within every logic gate (known as propagation delay). What we do is take an input signal and split it up two ways, then place a gate or a series of gates in one of those signal paths just to delay it a bit, then have both the original signal and its delayed counterpart enter into a two-input gate that outputs a high signal for the brief moment of time that the delayed signal has not yet caught up to the low-to-high change in the non-delayed signal. An example circuit for producing a clock pulse on a low-to-high input signal transition is shown here:



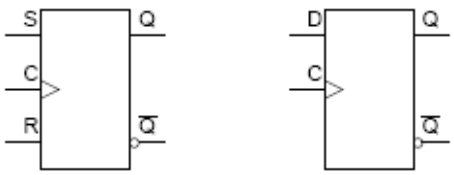


This circuit may be converted into a negative-edge pulse detector circuit with only a change of the final gate from AND to NOR:
Now that we know how a pulse detector can be made, we can show it attached to the enable input of a latch to turn it into a flip-flop. In this case, the circuit is a S-R flip-flop:

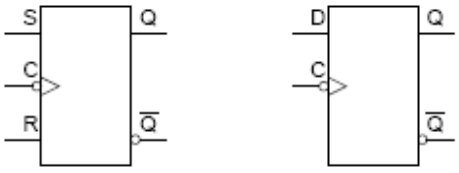


C	S	R	Q	$\overline{Q}$
⌐	0	0	latch	latch
⌐	0	1	0	1
⌐	1	0	1	0
⌐	1	1	0	0
x	0	0	latch	latch
x	0	1	latch	latch
x	1	0	latch	latch
x	1	1	latch	latch

Only when the clock signal (C) is transitioning from low to high is the circuit responsive to the S and R inputs. For any other condition of the clock signal ("x") the circuit will be latched. A ladder logic version of the S-R flip-flop is shown here:
It is important to note that the invalid state for the S-R flip-flop is maintained only for the short period of time that the pulse detector circuit allows the latch to be enabled. After that brief time period has elapsed, the outputs will latch into either the set or the reset state. Once again, the problem of a race condition manifests itself. With no enable signal, an invalid output state cannot be maintained. However, the valid "latched" states of the multivibrator – set and reset – are mutually exclusive to one another. Therefore, the two gates of the multivibrator circuit will "race" each other for supremacy, and whichever one attains a high output state first will "win."The block symbols for flip-flops are slightly different from that of their respective latch counterparts:



The positive edge-triggered-



Negative edge-triggered with a bubble on the clock input
The triangle symbol next to the clock inputs tells us that these are edge-triggered devices, and consequently that these are flip-flops rather than latches.

Both of the above flip-flops will "clock" on the falling edge (high-to-low transition) of the clock signal.

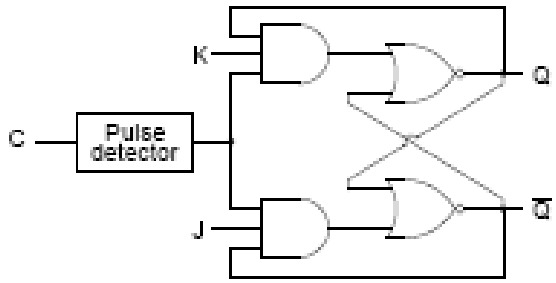
Review:

1-A flip-flop is a latch circuit with a "pulse detector" circuit connected to the enable (E) input, so that it is enabled only for a brief moment on either the rising or falling edge of a clock pulse.

2-Pulse detector circuits may be made from time-delay relays for ladder logic applications, or from semiconductor gates (exploiting the phenomenon of propagation delay).

The J-K flip-flop

Another variation on a theme of bistable multivibrators is the J-K flip-flop. Essentially, this is a modified version of an S-R flip-flop with no "invalid" or "illegal" output state. Look closely at the following diagram to see how this is accomplished



C	J	K	Q	$\overline{Q}$
⌐	0	0	latch	latch
⌐	0	1	0	1
⌐	1	0	1	0
⌐	1	1	toggle	toggle
x	0	0	latch	latch
x	0	1	latch	latch
x	1	0	latch	latch
x	1	1	latch	latch

What used to be the S and R inputs are now called the J and K inputs, respectively. The old two-input AND gates have been replaced with 3-input AND gates, and the third input of each gate receives feedback from the Q & ¯Q outputs. What this does for us is permit the J input to have effect only when the circuit is reset, and permit the K input to have effect only when the circuit is set. In other words, the two inputs are interlocked, to use a relay logic term, so that they cannot both be activated simultaneously. If the circuit is "set," the J input is inhibited by the 0 status of ¯Q through the lower AND gate; if the circuit is "reset," the K input is inhibited by the 0 status of Q through the upper AND gate.

When both J and K inputs are 1, however, something unique happens. Because of the selective inhibiting action of those 3-input AND gates, a"set" state inhibits input J so that the flip-flop acts as if J=0 while K=1 when in fact both are 1. On the next clock pulse, the outputs will switch ("toggle") from set (Q=1 & ¯Q=0) to reset (Q=0 & ¯Q=1). Conversely, a "reset" state inhibits input K so that the flip-flop acts as if J=1 and K=0 when in fact both are 1. The next clock pulse toggles the circuit again from reset to set.

The end result is that the S-R flip-flop's "invalid" state is eliminated (along with the race condition it engendered) and we get a useful feature as a bonus: the ability to toggle between the two (bistable) output states with every transition of the clock input signal.

There is no such thing as a J-K latch, only J-K flip-flops. Without the edge-triggering of The end result is that the S-R flip-flop's "invalid" state is eliminated (along with the race condition it engendered) and we get a useful feature as a bonus: the ability to toggle between the two (bistable) output states with every transition of the clock input signal.

There is no such thing as a J-K latch, only J-K flip-flops. Without the edge-triggering of The end result is that the S-R flip-flop's "invalid" state is eliminated (along with the race condition

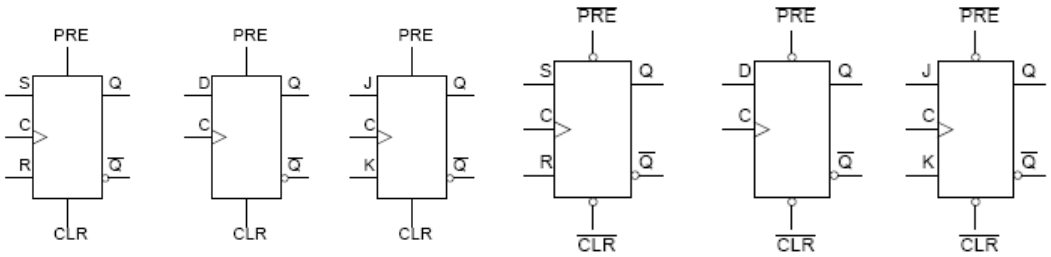
it engendered) and we get a useful feature as a bonus: the ability to toggle between the two (bistable) output states with every transition of the clock input signal.
the clock input, the circuit would continuously toggle between its two output states when both J and K were held high (1), making it an astable device instead of a bistable device in that circumstance. If we want to preserve bistable operation for all combinations of input states, we must use edge-triggering so that it toggles only when we tell it to, one step (clock pulse) at a time.The block symbol for a J-K flip-flop is a whole lot less frightening than its internal circuitry, and just like the S-R and D flip-flops, J-K flip-flops come in two clock varieties (negative and positive edge-triggered):

REVIEW:

1- A J-K flip-flop is nothing more than an S-R flip-flop with an added layer of feedback. This feedback selectively enables one of the two set/reset inputs so that they cannot both carry an active signal to the multivibrator circuit, thus eliminating the invalid condition.
2-When both J and K inputs are activated, and the clock input is pulsed, the outputs (Q & ¯Q) will swap states. That is, the circuit will toggle from a set state to a reset state, or vice versa.

Asynchronous flip-flop inputs

The normal data inputs to a flip flop (D, S and R, or J and K) are referred to as synchronous nputs because they have effect on the outputs (Q & ¯Q) only in step, or in sync, with the clock signal transitions. These extra inputs that I now bring to your attention are called asynchronous because they can set or reset the flip-flop regardless of the status of the clock signal. Typically, they're called preset and clear:



When the preset input is activated, the flip-flop will be set (Q=1, ¯Q=0) regardless of any of the synchronous inputs or the clock. When the clear input is activated, the flip-flop will be reset (Q=0, & ¯Q=1), regardless of any of the synchronous inputs or the clock. So, what happens if both preset and clear inputs are activated? Surprise, surprise: we get an invalid state on the output, where Q & ¯Q go to the same state, the same as our old friend, the S-R latch! Preset and clear inputs find use when multiple flip-flops are ganged together to perform a function on a multi-bit binary word, and a single line is needed to set or reset them all at once.
Asynchronous inputs, just like synchronous inputs, can be engineered to be active-high or active-low. If they're active-low, there will be an inverting bubble at that input lead on the block symbol, just like the negative edge-trigger clock inputs
Sometimes the designations "PRE" and "CLR" will be shown with inversion bars above them, to further denote the negative logic of these inputs:

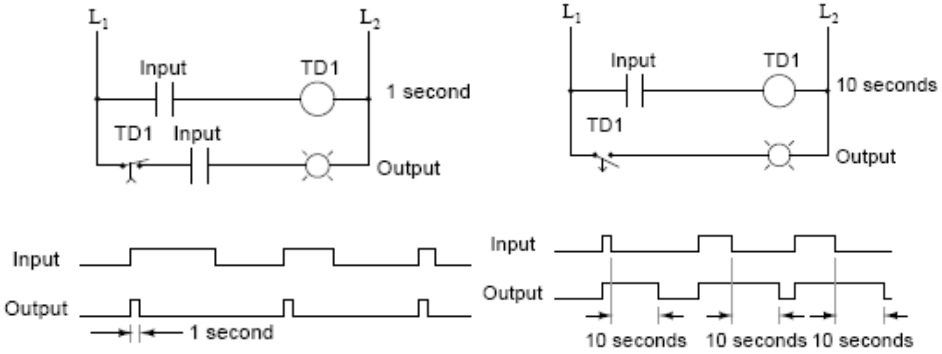
Review:

1-Asynchronous inputs on a flip-flop have control over the outputs (Q & ¯Q) regardless of clock input status.
2-These inputs are called the preset (PRE) and clear (CLR). The preset input drives the flip-flop to a set state while the clear input drives it to a reset state.
3-It is possible to drive the outputs of a J-K flip-flop to an invalid condition using the asynchronous inputs, because all feedback within the multivibrator circuit is overridden.

Monostable multivibrators

We've already seen one example of a monostable multivibrator in use: the pulse detector used within the circuitry of flip-flops, to enable the latch portion for a brief time when the clock input signal transitions from either low to high or high to low. The pulse detector is classified as a monostable multivibrator because it has only one stable state. By stable, I mean a state of output where the device is able to latch or hold to forever, without external prodding. A latch or flip-flop, being a bistable device, can hold in either the "set" or "reset" state for an indefinite period of time. Once its set or reset, it will continue to latch in that state unless prompted to change by an external input. A monostable device, on the other hand, is only able to hold in one particular state indefinitely. Its other state can only be held momentarily when triggered by an external input.

A mechanical analogy of a monostable device would be a momentary contact pushbutton switch, which spring-returns to its normal (stable) position when pressure is removed from its button actuator. Likewise, a standard wall (toggle) switch, such as the type used to turn lights on and off in a house, is a bistable device. It can latch in one of two modes: on or off.

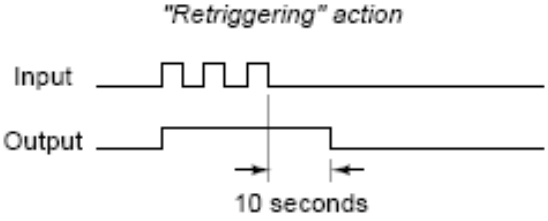


All monostable multivibrators are timed devices. That is, their unstable output state will hold only for a certain minimum amount of time before returning to its stable state. With semiconductor monostable circuits, this timing function is typically accomplished through the use of resistors and capacitors, making use of the exponential charging rates of RC circuits. A comparator is often used to compare the voltage across the charging (or discharging) capacitor with a steady reference voltage, and the on/off output of the comparator used for a logic signal. With ladder logic, time delays are accomplished with time-delay relays, which can be constructed with semiconductor/RC circuits like that just mentioned, or mechanical delay devices which impede the immediate motion of the relay's armature. Note the design and operation of the pulse detector circuit in ladder logic:

No matter how long the input signal stays high (1), the output remains high for just 1 second of time, then returns to its normal (stable) low state. For some applications, it is necessary to have

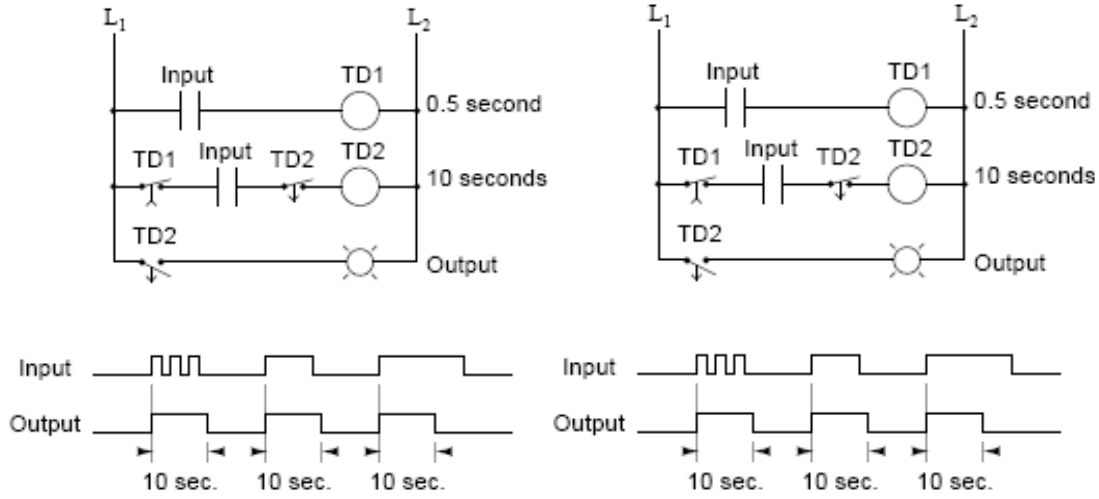
a monostable device that outputs a longer pulse than the input pulse which triggers it. Consider the following ladder logic circuit:

When the input contact closes, TD1 contact immediately closes, and stays closed for 10 seconds after the input contact opens. No matter how short the input pulse is, the output stays high (1) for exactly 10 seconds after the input drops low again. This kind of monostable multivibrator is called a one-shot. More specifically, it is a retriggerable one-shot, because the timing begins after the input drops to a low state, meaning that multiple input pulses within 10 seconds of each other will maintain a continuous high output:



One application for a retriggerable one-shot is that of a single mechanical contact debouncer. As you can see from the above timing diagram, the output will remain high despite "bouncing" of the input signal from a mechanical switch. Of course, in a real-life switch debouncer circuit, you'd probably want to use a time delay of much shorter duration than 10 seconds, as you only need to "debounce" pulses that are in the millisecond range.

What if we only wanted a 10 second timed pulse output from a relay logic circuit, regardless of how many input pulses we received or how long-lived they may be? In that case, we'd have to couple a pulse-detector circuit to the retriggerable one-shot time delay circuit, like this:



Time delay relay TD1 provides an "on" pulse to time delay relay coil TD2 for an arbitrarily short moment (in this circuit, for at least 0.5 second each time the input contact is actuated).

As soon as TD2 is energized, the normally-closed, timed-closed TD2 contact in series with it prevents coil TD2 frombeing re-energized as long as its timing out (10 seconds). This effectively makes it unresponsive to any more actuations of the input switch during that 10 second period.

Only after TD2 times out does the normally-closed, timed-closed TD2 contact in series with it allow coil TD2 to be energized again. This type of one-shot is called a nonretriggerable oneshot.

One-shot multivibrators of both the retriggerable and nonretriggerable variety find wide application in industry for siren actuation and machine sequencing, where an intermittent input signal produces an output signal of a set time.

Review:

1-A monostable multivibrator has only one stable output state. The other output state can only be maintained temporarily.

2-Monostable multivibrators, sometimes called one-shots, come in two basic varieties: retriggerable and nonretriggerable.

3-One-shot circuits with very short time settings may be used to debounce the ”dirty” signals created by mechanical switch contacts.

Digital Signal Processing Basics

***D**igital **s**ignal **p**rocessing converts signals that naturally occur in analog form, such as sound, video. and information from sensors, to digital form and uses digital techniques to enhance and modify analog signal data for various applications. A digital signal processing system first translates a continuously varying analog signal into a series of discrete levels. This series of levels follows the variations of the analog signal and resembles a staircase, fig 4-1.*

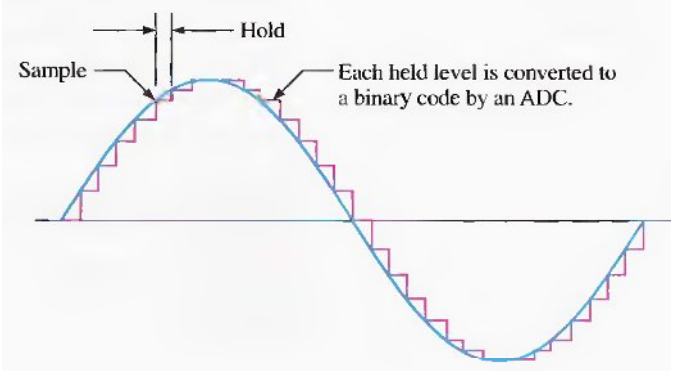


fig4-1 An original analog signal(sin wave) and its 'stairstep' approximation

The "stairstep" approximation is quantized into binary codes that represent each discrete step on the "stairsteps" by a process called analog-to-digital conversion. The circuit that performs conversion is an analog-to-digital converter (ADC). Once the analog signal has been converted to a binary coded form, it is applied to a DSP (digital signal processor). The DSP can perform various operations on the incoming data, such as removing unwanted interference, increasing the amplitude of some signal frequencies and reducing others, encoding the data for secure transmissions, and detecting and correcting errors in transmitted codes. DSPs make possible, among many other things, the cleanup of sound recordings, the removal of echos from communications lines, the enhancement of images from CT scans for better medical diagnosis. After a DSP processes a signal, the signal can be converted back to a much improved version of the original analog signal. This is accomplished by a digital-to-analog converter (DAC).

fig4-2 Basic block diagram of a typical digital signal processing (DSP)system

DSPs are actually a specialized type of microprocessor. Typically, microprocessors are designed for general-purpose functions and operate with large software packages. DSPs are used for special-purpose applications; they are very fast number crunchers that must work in real time by processing information as it happens using specialized algorithms (programs). ADC converter in a system must take samples of the incoming analog data often enough to

catch all the relevant fluctuations in the signal amplitude, and the DSP must keep pace with the sampling rate of the ADC by doing its calculations as fast as the sampled data are received.

Converting analog signal to digital

Sampling and Filtering: *The first two blocks in the system diagram of fig4-2 are the anti-aliasing filter and the sample-and-hold circuit. The sample-and-hold function does two operations, the first of which is sampling. Sampling is the process of taking a sufficient number of discrete values at points on a waveform that will define the shape of waveform. The* ***more samples*** *you take, the* ***more accurately*** *you can define a waveform. Sampling converts an analog signal into a series of* ***impulses****, each representing the* ***amplitude*** *of the signal at a given instant in time.*

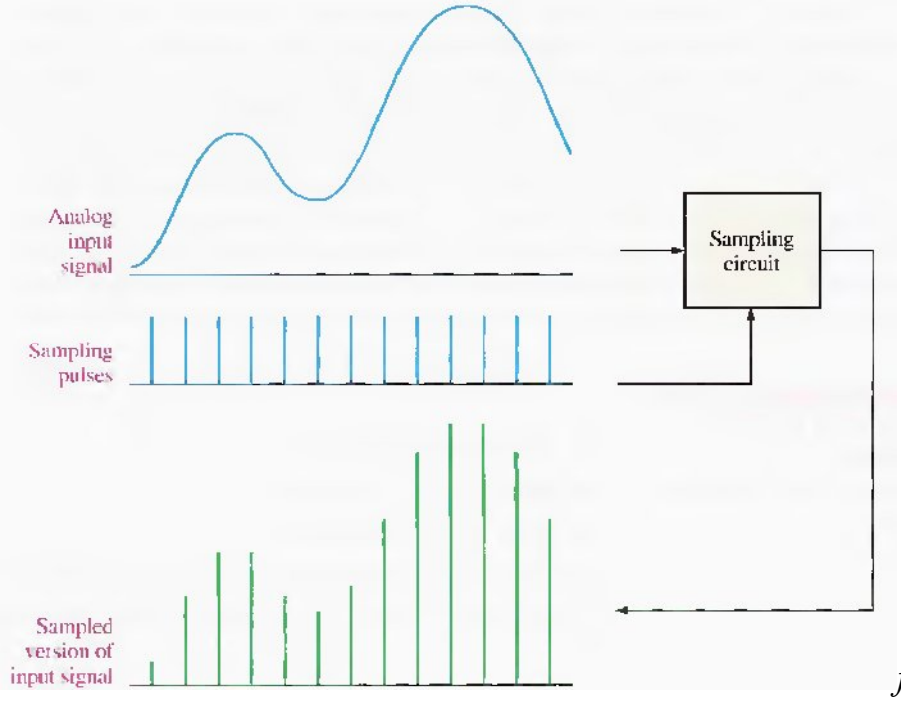


fig 4-3

When an analog signal is to be sampled, there are certain criteria that must be met in order to accurately represent the original signal. ***All*** *analog signals (except a pure sine wave) contain a spectrum of component frequencies called harmonics. The harmonics of an analog signal are sine waves of different frequencies and amplitudes. When the harmonics of a given periodic waveform are added, the result is the original signal before a signal can be sampled; it must be passed through a low-pass filter (anti-aliasing filter) to eliminate harmonic frequencies above a certain value as determined by the Nyquist frequency.*

The Sampling Theorem Notice in fig4-3 that there are two input waveforms. One is the analog signal and the other is the sampling pulse waveform. In order to represent an analog signal, the sampling frequencies and amplitudes are added, the result is the original signal. Before a signal can be sampled, it must be passed through a low-pass filter (anti-aliasing filter) to ***eliminate*** *harmonic frequencies above a certain value as determined by the Nyquist frequency. The highest analog frequency can be no greater than one-half the sampling frequency. The*

frequency $f_{a(max)}$ is known as the Nyquist frequency and is expressed in Eq 4-1. The sampling frequency should be more than twice the highest analog frequency.

$$f_{sample} \geq 2f_{a(max)} \text{ --------4-1}$$

Low-pass filtering is necessary to remove all frequency components (harmonics) of the analog signal that exceed the Nyquist frequency. An unwanted condition known as aliasing will occur. An alias is a signal produced when the sampling frequency is not at least twice the signal frequency. An alias signal has a frequency that is less than the highest frequency in the analog signal being sampled and therefore falls within the spectrum or frequency band of the input analog signal causing distortion. fig4-4 If the analog signal contains frequencies above the Nyquist frequency, these frequencies overlap into the spectrum of the sample waveform. The lower frequency components of the sampling waveform become mixed in with the frequency spectra of the analog waveform, resulting in an aliasing error.

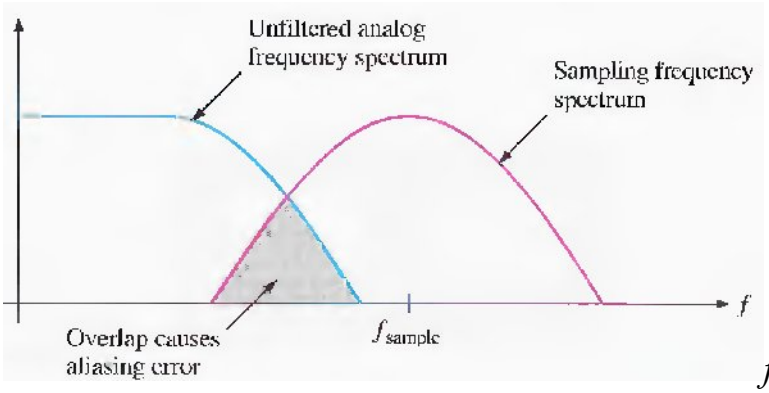


fig 4-4 the condition $f_{sample} < 2\, f_{a(max)}$

To avoid an aliasing error, the filter must at least eliminate all analog frequencies above the minimum frequency in the sampling spectrum, fig4-5. Aliasing can also be avoided by sufficiently increasing the sampling frequency.

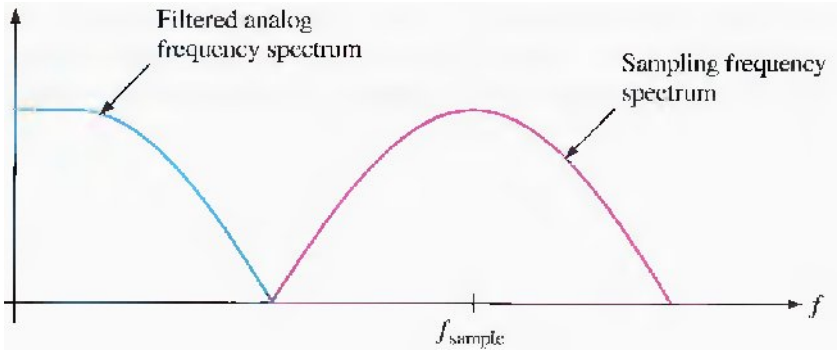


Fig4-5 after low-pass filtering the frequency spectra of the analog and the sampling signals do not overlap thus eliminating aliasing error

Many applications do not require a wide frequency range to obtain reproduced sound that is acceptable. For example, human speech contains some frequencies near 10 kHz and, therefore, requires a sampling rate of at least 20 kHz. However, if only frequencies up to 4 kHz (ideally

requiring an 8 kHz minimum sampling rate) are reproduced, voice is very understandable. On the other hand, if a sound signal is not sampled at a high enough rate, the effect of aliasing will become noticeable with background noise and distortion.

Holding the Sampled Value: *After filtering and sampling the sampled level must be held constant until the next sample occurs. This is necessary for the ADC to have time to process the sampled value.*

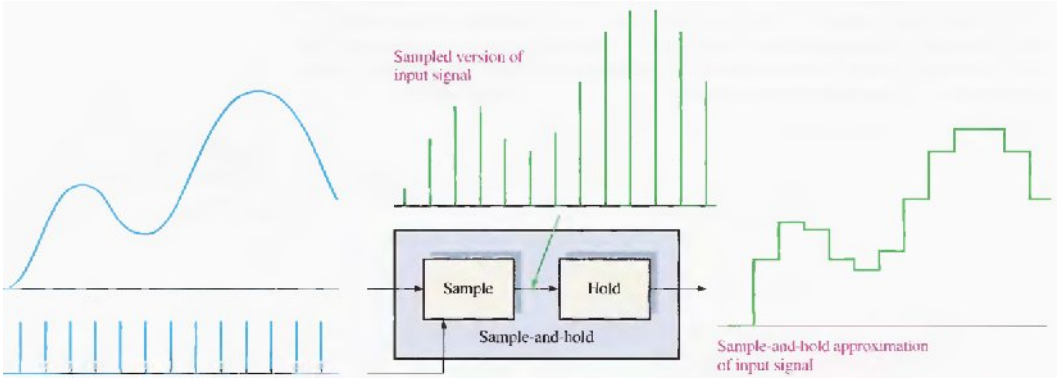


fig4-6 illustration of a sample and hold operation

Analog-to-Digital Conversion: *Analog-to-digital conversion is the process of converting the output of the sample-and hold circuit to a series of binary codes that represent the amplitude of the analog input at each of the sample times. The sample-and-hold process keeps the amplitude of the analog input signal constant between sample pulses; therefore, the analog-to-digital conversion can be done using a constant value rather than having the analog signal change during a conversion interval, which is the time between sample pulses fig4-7.*

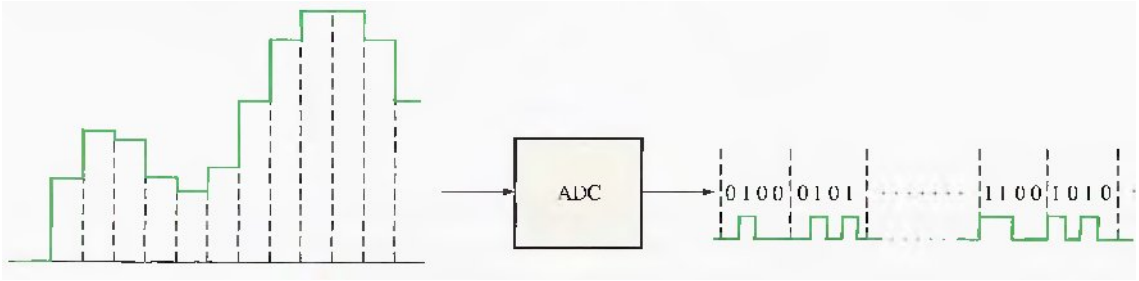


Fig4-7 Basic function of an analog to digital (ADC) converter, the ADC output waveform that represents the binary codes

Quantization: *The process of converting an analog value to a code is called quantization. During the quantization process, the ADC converts each sampled value of the analog signal to a binary code. The more bits that are used to represent a sampled value, the more accurate is the representation. Let's quantize a reproduction of the analog waveform into four levels (0-3). 2 bits are required, and each sample interval is numbered along the horizontal axis. The quantization process is summarized in table 4-1.*

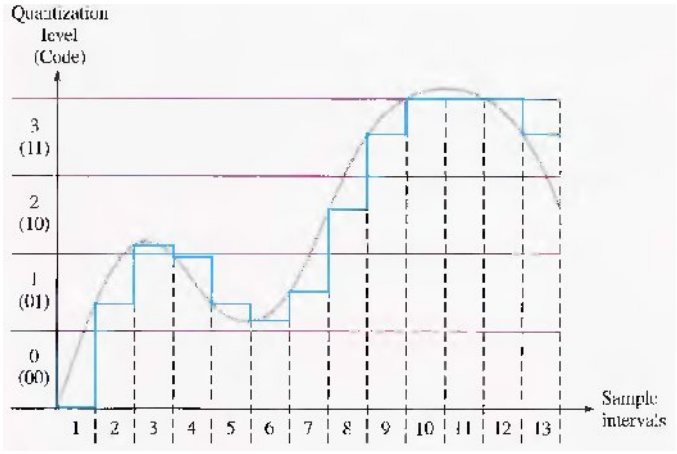


fig4-8 Sample and hold output waveform with four quantization levels.

SAMPLE INTERVAL	QUANTIZATION LEVEL	CODE
1	0	00
2	1	01
3	2	10
4	1	01
5	1	01
6	1	01
7	1	01
8	2	10
9	3	11
10	3	11
11	3	11
12	3	11
13	3	11

Table4-1 2bit quantization for the waveform

If the resulting 2-bit digital codes are used to reconstruct the original waveform, which is done by DAC, you would get the waveform shown in fig4-9. As you can see, quite a bit of accuracy is lost using only two bits to represent the sampled values.

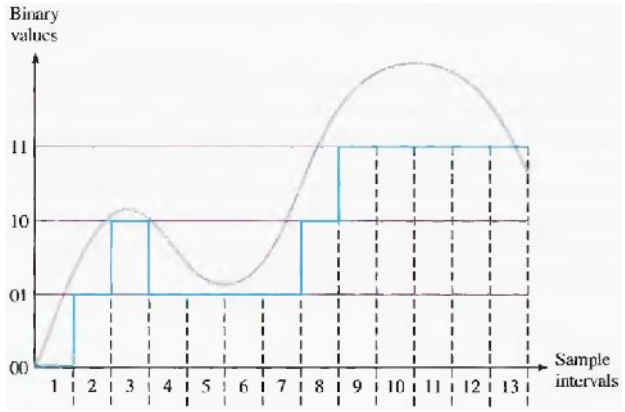


fig4-9 The reconstructed waveform in fig4-8 using four quantization levels (2 bits).

Now, let's see how more bits will improve the accuracy. Fig4-10 shows the same waveform with sixteen quantization levels (4 bits). The 4-bit quantization process is summarized in table4-2.

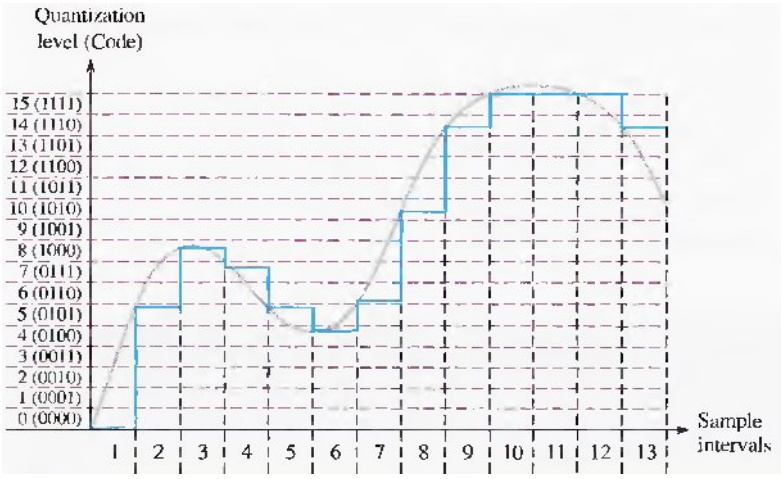


fig4-10 Sample-and-hold output waveform with 16 quantization levels.

SAMPLE INTERVAL	QUANTIZATION LEVEL	CODE
1	0	0000
2	5	0101
3	8	1000
4	7	0111
5	5	0101
6	4	0100
7	6	0110
8	10	1010
9	14	1110
10	15	1111
11	15	1111
12	15	1111
13	14	1110

Table4-2 4bit quantization for the waveform

If the resulting 4-bit digital codes are used to reconstruct the original waveform, you would get the waveform shown in fig4-11. The result is much more like the original waveform than for the case of 4 quantization levels in fig4-9. This shows that greater accuracy is achieved with more quantization bits.

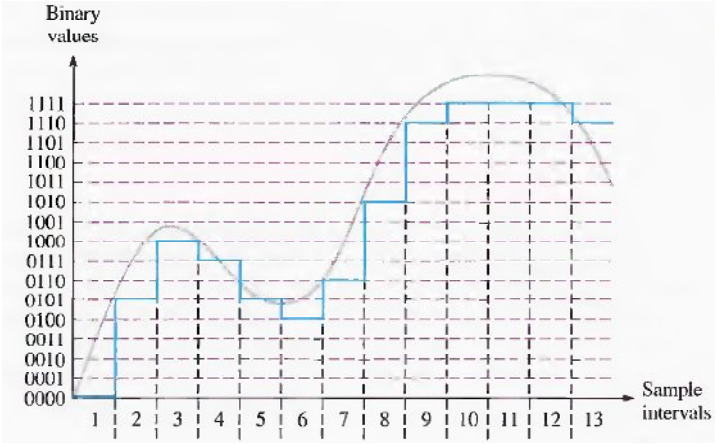


fig4-11 The reconstructed waveform in fig4-10 using 16 quantization levels (4 bits).

Analog-to-digital conversion methods: *It is necessary when measured quantities must be in digital form for processing or for display or storage. Two important ADC parameters are* ***resolution****, which is the number of bits, and* ***throughput,*** *which is the sampling rate an ADC can handle in units of samples per second (sps).*

1-Flash (Simultaneous) Analog-to-Digital Converter : *The flash method utilizes comparators that compare reference voltages with the analog input voltage. When the input voltage exceeds the reference voltage for a given comparator, a HIGH is generated. Fig4-12 shows a 3-bit converter that uses 7 comparator circuits; a comparator is not needed for the all-0s condition. A 4-bit converter of this type requires 15 comparators. In general, 2^n-1 comparators are required for conversion to an n-bit binary code. The number of bits used in an ADC is its resolution. The large number of comparators necessary for a reasonable-sized binary number is one of the* ***disadvantages*** *of the flash ADC. Its chief* ***advantage*** *is that it provides a fast conversion time because of a high throughput, measured in samples per second (sps). The reference voltage for each comparator is set by the resistive voltage-divider circuit. The output of each comparator is connected to an input of the priority encoder. The encoder is enabled by a pulse on the* ***EN*** *input. And a 3-bit code representing the value of the input appears on the encoder's outputs. The binary code is determined by the highest-order input having a HIGH level.*

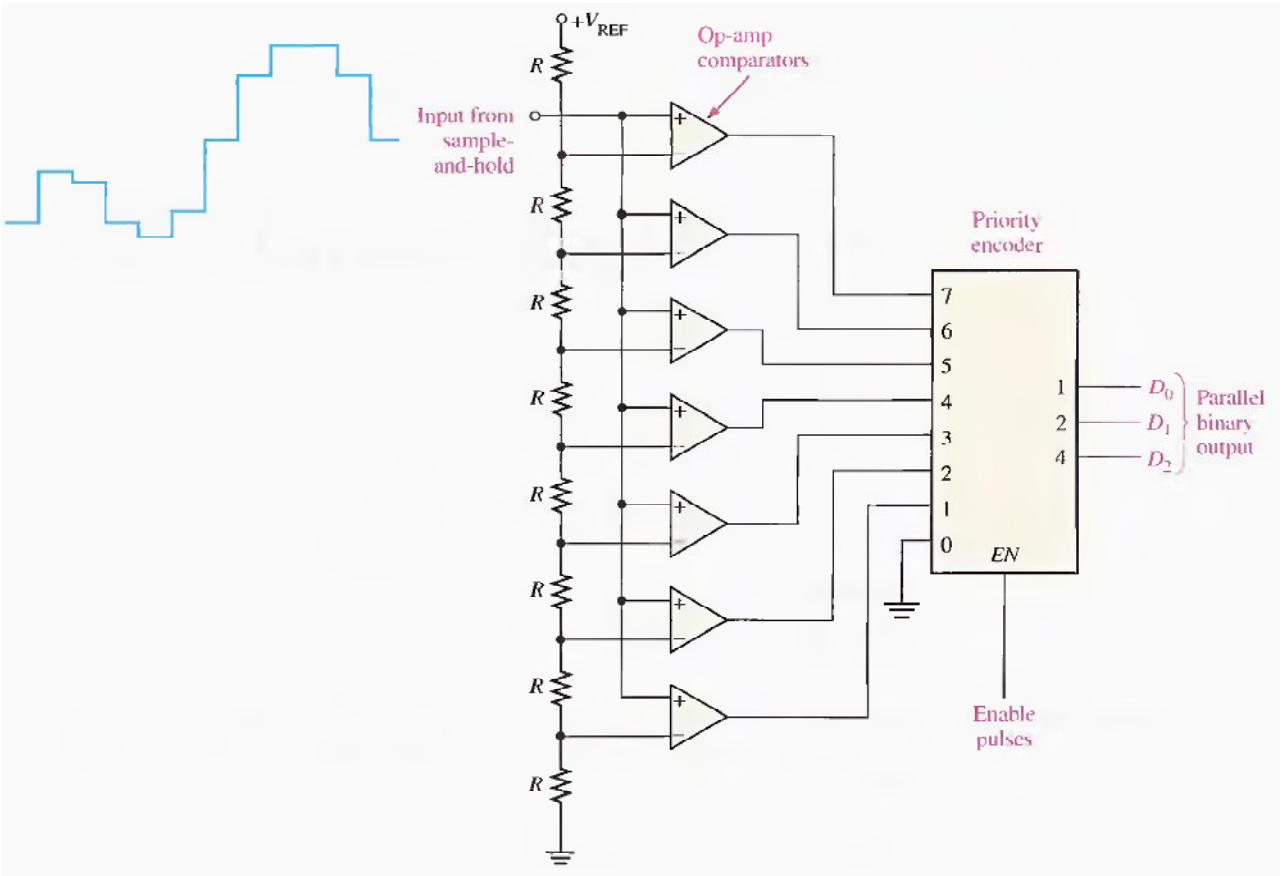


fig4-12 A 3- bit flash ADC

The frequency of the enable pulses and the number of bits in the binary code determine the accuracy with which the sequence of binary codes represents the input of the ADC.

Example4-1: Determine the binary code output of the 3-bit flash ADC in fig4-13 for the input signal in fig4-13 and the encoder enable pulses shown. For this example, V REF = +8 V.

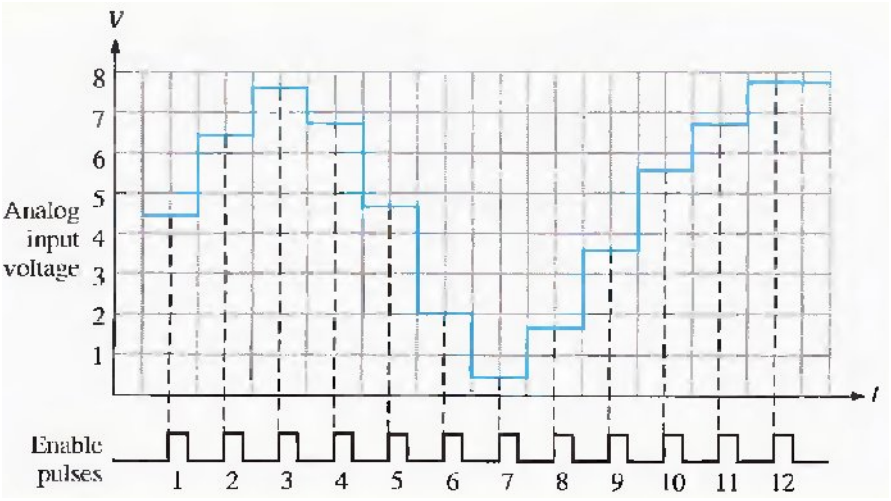


fig4-13 Sampling of values on a waveform for conversion to binary code

***Solution: The** resulting digital output sequence is listed as follows and shown in the waveform diagram of fig4-14 in relation to the enable pulses:*

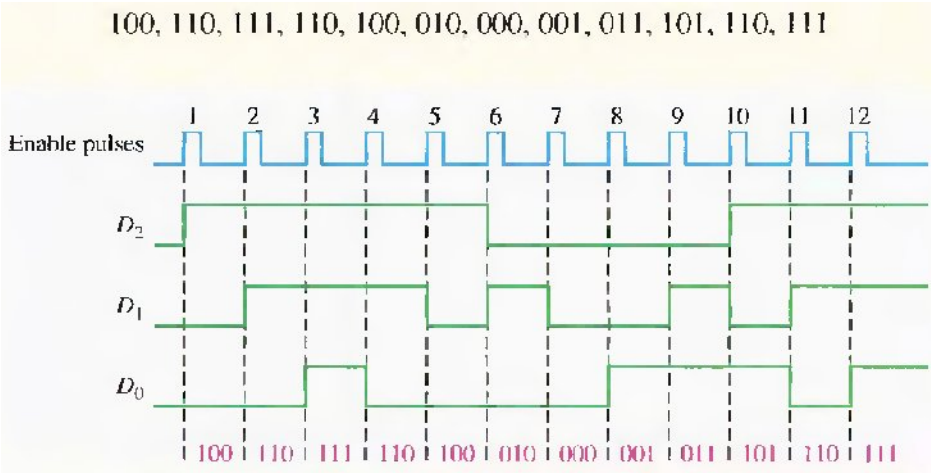


fig4-14 Resulting digital outputs for sample-and-hold values. Output Do is the LSB of the 3-bit binary code.

***2-Successive-Approximation Analog-to-Digital Converter:** One of the most widely used methods of analog-to-digital conversion is successive- approximation. It has a much faster conversion time than the dual-slope conversion, but it is slower than the flash method. It also has a fixed conversion time that is the same for any value of the analog input. Fig4-15 shows a basic block diagram of a 4-bit successive approximation ADC. It consists of a DAC, a successive-approximation register (SAR), and a comparator. The basic operation is as follows:*

The input bits of the DAC are enabled (made equal to a 1) one at a time, starting with the most significant bit (MSB). As each bit is enabled, the comparator produces an output that indicates whether the input signal voltage is greater or less than the output of the DAC. If the DAC output is greater than the input signal, the comparator's output is LOW, causing the bit in the register to reset. If the output is less than the input signal, the 1 bit is retained in the register. The system does this with the MSB first, then the next most significant bit, then the next, and so on. After all the bits of the DAC have been tried, the conversion cycle is complete, example of a 4-bit conversion.

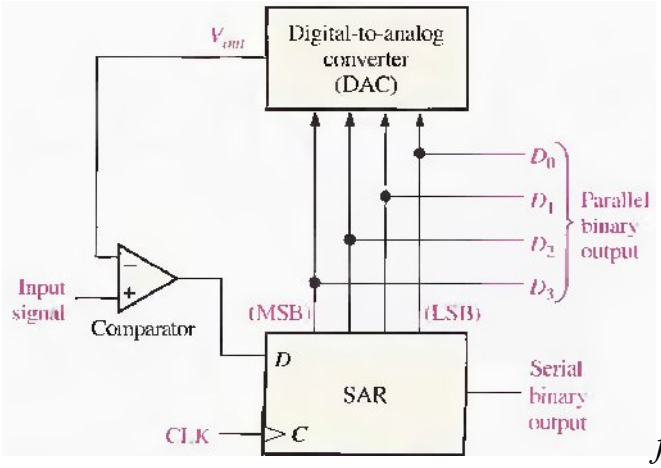


fig4-15 Successive-approximation ADC

Fig4-16 illustrates the step-by-step con- version of a constant input voltage (5.1V in this case). Let's assume that the DAC has the following output characteristic: V_{out} = 8V for the 2^3 bit (MSB), V_{out} = 4V for the 2^2 bit, V_{out} = 2V for the 2^1 bit. and V_{out} = 1V for the 2° bit (LSB).

Fig4-16(a) shows the first step in the conversion cycle with the MSB = 1. The output of the DAC is 8 V. Since this is greater than the input of 5.1V, the output of the comparator is LOW, causing the MSB in the SAR to be reset to a 0.

Fig4-16(b) shows the second step in the conversion cycle with the 2^2 bit equal to a 1. The output of the DAC is 4V. Since this is less than the input of 5.1V, the output of the comparator switches to a HIGH, causing this bit to be retained in the SAR.

Fig4-16(c) shows the third step in the conversion cycle with the 2^1 bit equal to a 1. The output of the DAC is 6V because there is a 1 on the 2^2 bit input and on the 2^1 bit input;4V + 2V = 6V. Since this is greater than the input of 5.1V, the output of the comparator switches to a LOW, causing this bit to be reset to a 0.

Fig4-16(d) shows the fourth and final step in the conversion cycle with the 2° bit equal to a 1. The output of the DAC is 5V because there is a 1 on the 2^2 bit input and on the 2° bit input; 4V + 1V = 5V. The four bits have all been tried, thus completing the conversion cycle. At this point the binary code in the register is 0101, which is approximately the binary value of the input of 5.1V. Additional bits will produce an even more accurate result. Another conversion cycle now begins, and the basic process is repeated. The SAR is cleared at the beginning of each cycle.

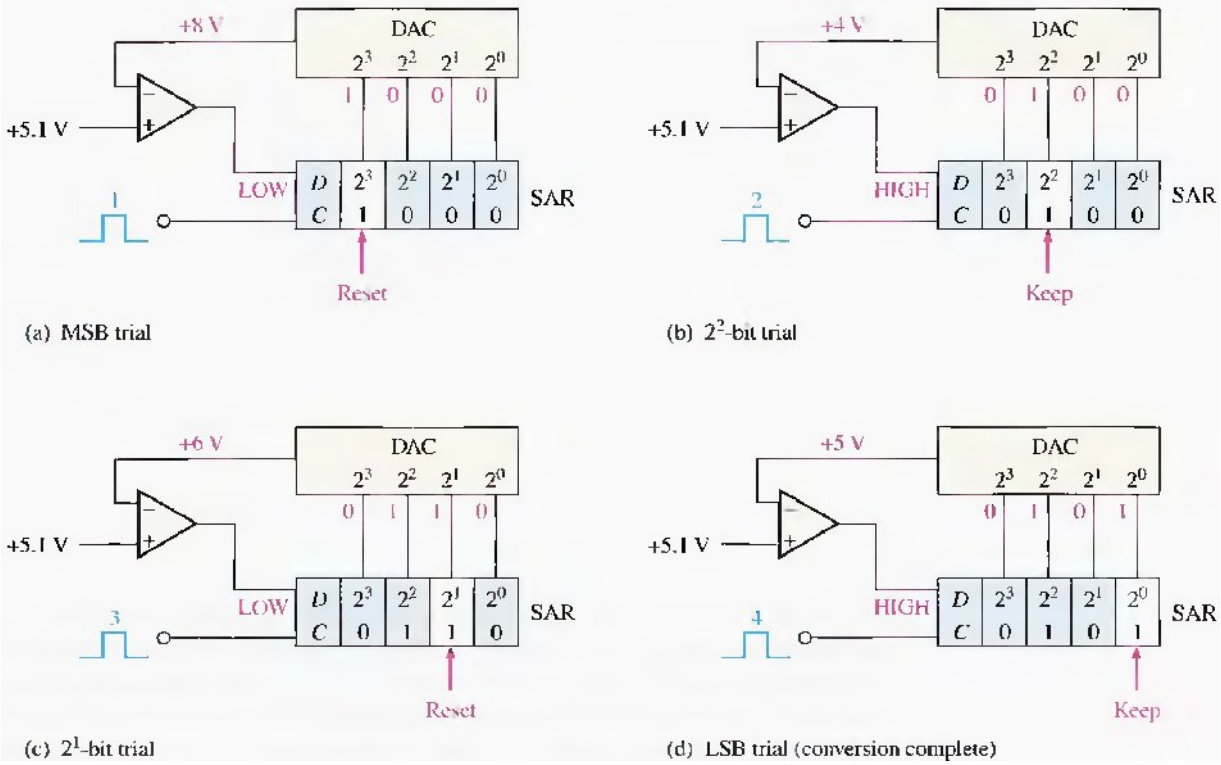


fig4-16 illustration of the successive approximation conversion process

The Digital signal processor (DSP)

A digital signal processor (DSP) is a special type of microprocessor that processes data in real time is typically programmed in either assembly language or in C language. Because programs written in assembly language can usually execute faster and because speed is critical in most DSP applications.

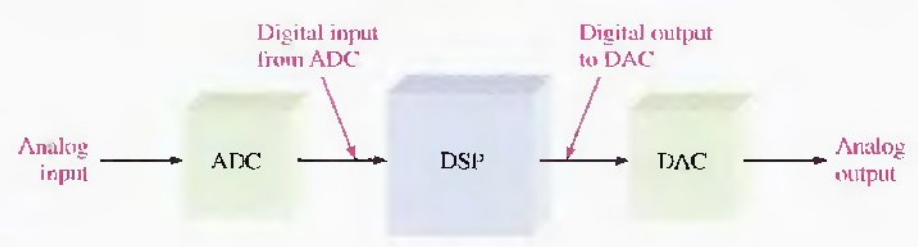


Fig4-17 The DSP has a digital input and produces a digital output.

DSP Applications:

***1-Telecommunications**: The field of telecommunications involves transferring all types of information from one location to another, including telephone conversations. Television signals, and digital data, the DSP facilitates multiplexing many signals onto one transmission channel because information in digital form is relatively easy to multiplex and demultiplex at*

the receiving end of a telecommunications system, the DSP decompresses the data to restore the signal to its original form.

2-Echoes: *a problem in many long distance telephone connections, Occur when a portion of a voice signal is returned with a delay as the distance between the transmitter and the receiver increase. DSPs are used to effectively cancel the annoying echo, which results in a clear, undisturbed voice signal.*

3-Music Processing: *the DSP is used in the music industry to provide filtering, signal addition and subtraction, and signal editing in music preparation and recording*

4-Radar: *In radio detection and ranging radar applications, DSPs provide more accurate determination of distance using* ***dam*** *compression techniques, decrease noise using filtering techniques, thereby increasing the range, and optimize the ability of the radar system to identity specific types of targets.*

5-Image Processing: *The DSP is used in image-processing applications such as the computed tomography (CT) and magnetic resonance imaging (MRI), which are widely used in the medical field for looking inside the human body. (In CT, X-rays are passed through a section of the body from many directions; MRI uses magnetic fields in conjunction with radio waves to probe inside the human body) MRI and CT DSP uses image compression to reduce the number of bits needed.*

6-Filtering: *DSPs are commonly used to implement digital filters for the purposes of separating signals that have been combined with other signals or with interference and noise and for restoring signals that are distorted.*

Basic DSP Architecture

DSP is a specialized microprocessor optimized for speed in order to process data in real time. Many DSPs are based on what is known as the Harvard architecture, which consists of a central processing unit (CPU) and two memories, one for data and the other for the program, as shown by the block diagram in fig4-18.

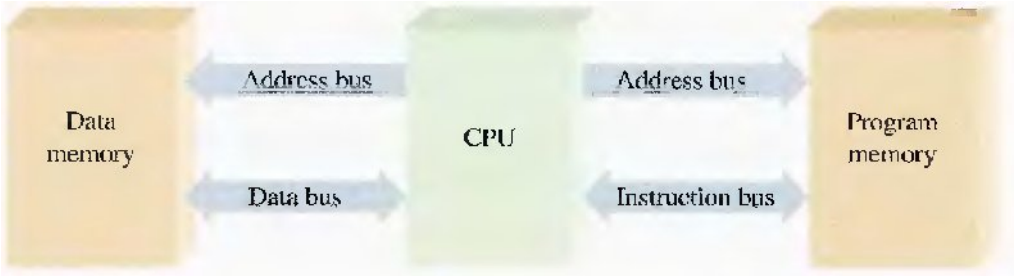


Fig4-18 Many DSPs use the Harvard architecture (two memories)

Specific DSPs-The TMS320C6000 Series:

The DSPs have a central processing unit (CPU) that contains 64 general-purpose 32-bit registers or 32 general-purpose 32-bit registers. Each DSP has eight functional units that contain two 16-bit multipliers and six arithmetic logic units (ALUs). The performance of the DSPs in terms MIPS (Million Instructions per Second), MFLOPS (Million Floating-point Operations per Second), and MMACS (Million Multiply/Accumulates per Second)

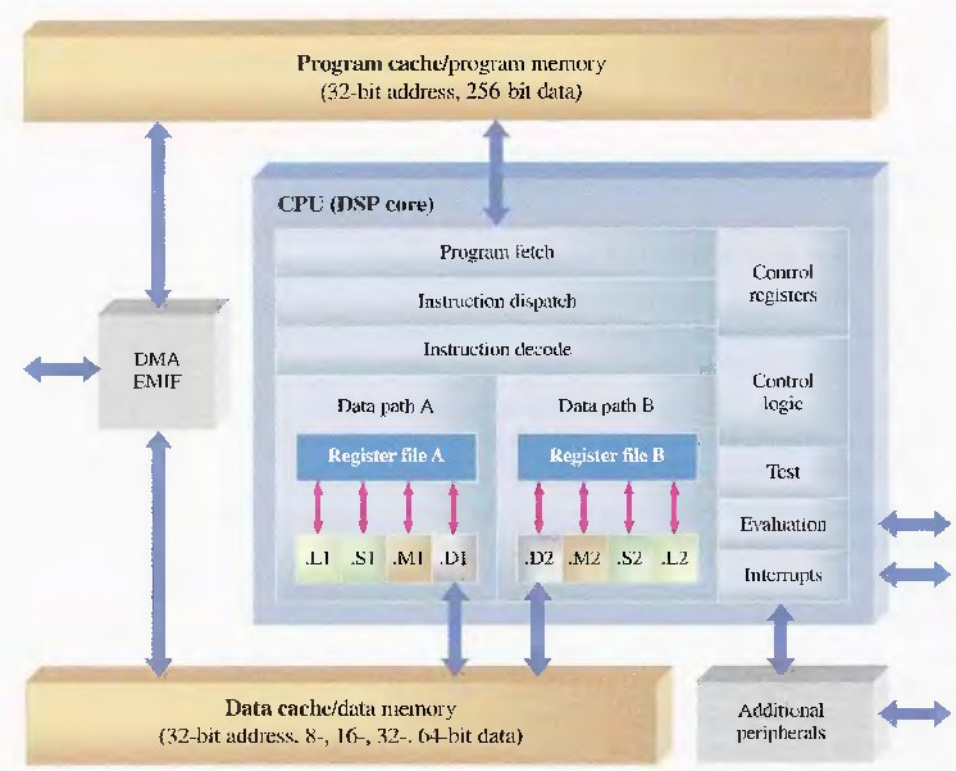


fig4-19 General block diagram of the TMS320C6000 series DSP.

Data Paths in the CPU:*A pipeline allows multiple instructions to be processed simultaneously. A pipeline operation consists of three stages through which all instructions flow: fetch, decode, execute. Eight instructions at a time are first fetched from the program memory then decoded. Finally, they are executed.*

Internal DSP Memory and Interfaces: *There are two internal memories, one for data and one for program. The program memory is organized in 256-bit packets (eight 32-bit instructions) and there are 64 KBit of capacity.*

The data memory also has a capacity of 64 kB and can be accessed in 8-, 16-, 32-, or 64- bit word lengths.

Both internal memories are accessed with a 32-bit address. The DMA (Direct Memory Access) is used to transfer data without going through the CPU. The EMIF (External Memory Interface) is used to support external memories when required in an application.

Timers*: There are two general-purpose timers in the DSP that can be used for timed events, counting, pulse generation.*

Digital-to-Analog Conversion methods

1-Binary-Weighted-Input Digital-to-Analog Converter

This method uses a resistor network with resistance values that represent the binary weights of the input bits of the digital code. Fig4-20 shows a 4-bit DAC. Each of the input resistors will either have current or have no current, depending on the input voltage level. If the input voltage is zero (binary 0), the current is also zero. If the input voltage is HIGH (binary 1), the amount of current depends on the input resistor value and is different for each input resistor, as indicated in the figure.

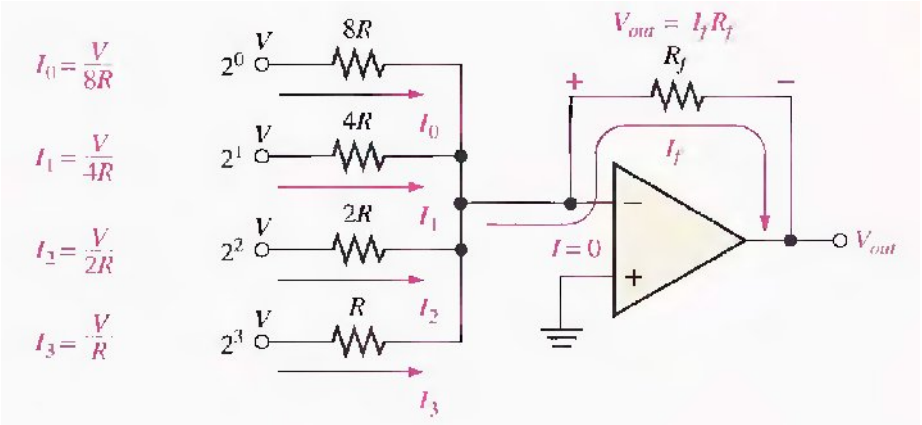


Fig 4-20 A 4-bit DAC with binary-weighted inputs

Since there is practical1y no current into the op-amp inverting (-) input, all of the input currents sum together and go through R_f. Since the inverting input is at 0V (virtual ground), the drop across R_f is equal to the output voltage, so $V_{out} = I_f R_f$.

The values of the input resistors are chosen to be inversely proportional to the binary weights of the corresponding input bits. The lowest-value resistor (R) corresponds to the highest binary-weighted input (2^3). The other resistors are multiples of R (that is, 2R, 4R, and 8R) and correspond to the binary weights 2^2, 2^1, and 2^0, respectively.

The input currents are also proportional to the binary weights. Thus, the output voltage is proportional to the sum of the binary weights because the sum of the input currents is through R_f. Disadvantages of this type of DAC are the number of different resistor values and the fact that the voltage levels must be exactly the same for all inputs. For example, an 8-bit converter requires eight resistors, ranging from some value of R to 128R in binary-weighted steps.

Example: *Determine the output of the DAC in Fig4-21(a) if the waveforms representing a sequence of 4-bit numbers in Fig4-21(b) are applied to the inputs; Input D_o is the least significant bit (LSB).*

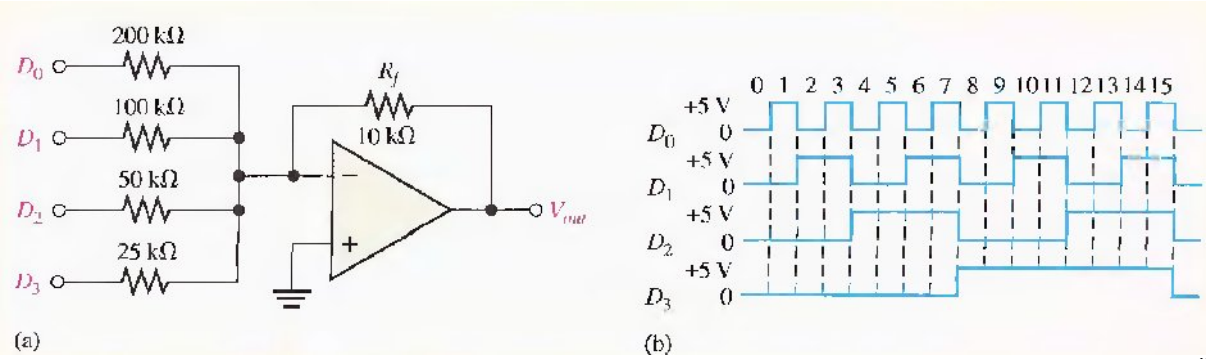


fig 4-21

Solution: *First, determine the current for each of the weighted inputs. Since the inverting (-) input of the op-amp is at 0V (virtual ground) and a binary 1 corresponds to +5V. The current through any of the input resistors is 5V divided by the resistance value.*

$$I_0 = \frac{5\text{ V}}{200\text{ k}\Omega} = 0.025\text{ mA}$$

$$I_1 = \frac{5\text{ V}}{100\text{ k}\Omega} = 0.05\text{ mA}$$

$$I_2 = \frac{5\text{ V}}{50\text{ k}\Omega} = 0.1\text{ mA}$$

$$I_3 = \frac{5\text{ V}}{25\text{ k}\Omega} = 0.2\text{ mA}$$

Almost no current goes into the inverting op-amp input because of its extremely high impedance. Therefore, assume that all of the current goes through the feedback resistor R_f since one end of R_f is at 0V (virtual ground), the drop across R_f equals the output voltage, which is negative with respect to virtual ground.

Fig4-21(b), the first binary input code is 0000, which produces an output voltage of zero V.

The next input code is 0001, which produces an output voltage of (-0.25V).

The next code is 0010, which produces an output voltage of (-0.5 V).

The next code is 0011, which produces an output voltage of (-0.25V) + (-0.5V) = -0.75V.

Each successive binary code increases the output voltage by (-0.25V), so for this particular straight binary sequence on the inputs, the output is a stair step waveform going from 0V to (-3.75V) in (-0.25V) steps.

$$V_{out(D0)} = (10\text{ k}\Omega)(-0.025\text{ mA}) = -0.25\text{ V}$$
$$V_{out(D1)} = (10\text{ k}\Omega)(-0.05\text{ mA}) = -0.5\text{ V}$$
$$V_{out(D2)} = (10\text{ k}\Omega)(-0.1\text{ mA}) = -1\text{ V}$$
$$V_{out(D3)} = (10\text{ k}\Omega)(-0.2\text{ mA}) = -2\text{ V}$$

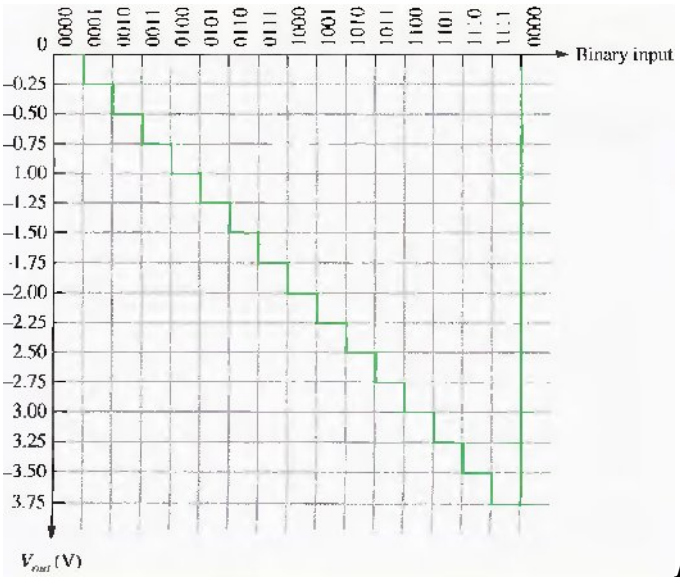


Fig4-22 Output of the DAC in fig4-21.

2-The R/2R ladder Digital-to-Analog Converter

The R/2R ladder is overcomes one of the problems in the binary-weighted-input DAC in that it requires only two resistor values, as shown in fig4-23 for 4 bits.

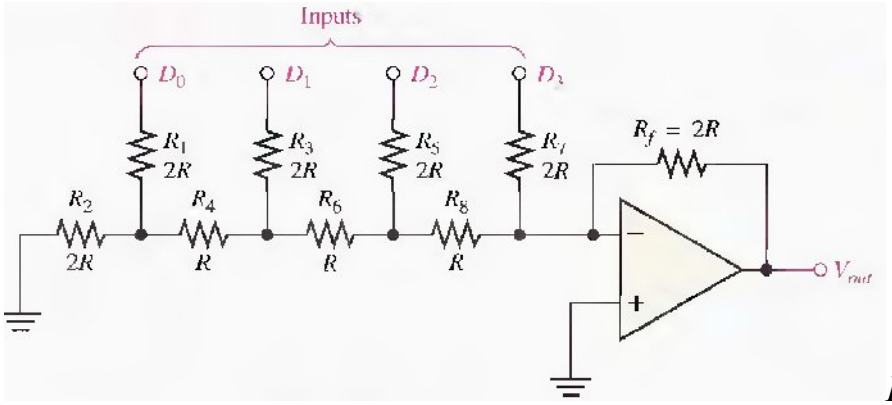


Fig 4-23 An R/2R ladder DAC

Start by assuming that the D_3 input is HIGH (+5V) and the others are LOW (ground, 0V). This condition represents the binary number 1000. A circuit analysis will show that this reduces to the equivalent form shown in Fig4-24(a). Essentially no current goes through the 2R equivalent resistance because the inverting input is at virtual ground. Thus, all of the current (I = 5V/2R) through R_7 also goes through R_f the output voltage is (-5V). The operational amplifier keeps the inverting (-) input near zero volts (0 V) because of negative feedback. Therefore, all current goes through R rather than into the inverting input.

Fig4-24(b) shows the equivalent circuit when D_2 input is at +5V and the others are at ground. This condition represents 0100. If we theveninzing looking from R_s , we get 2.5V in series with R, this results in a current through R_f of I=2.5V/2R, which gives an output voltage of(-2.5V).

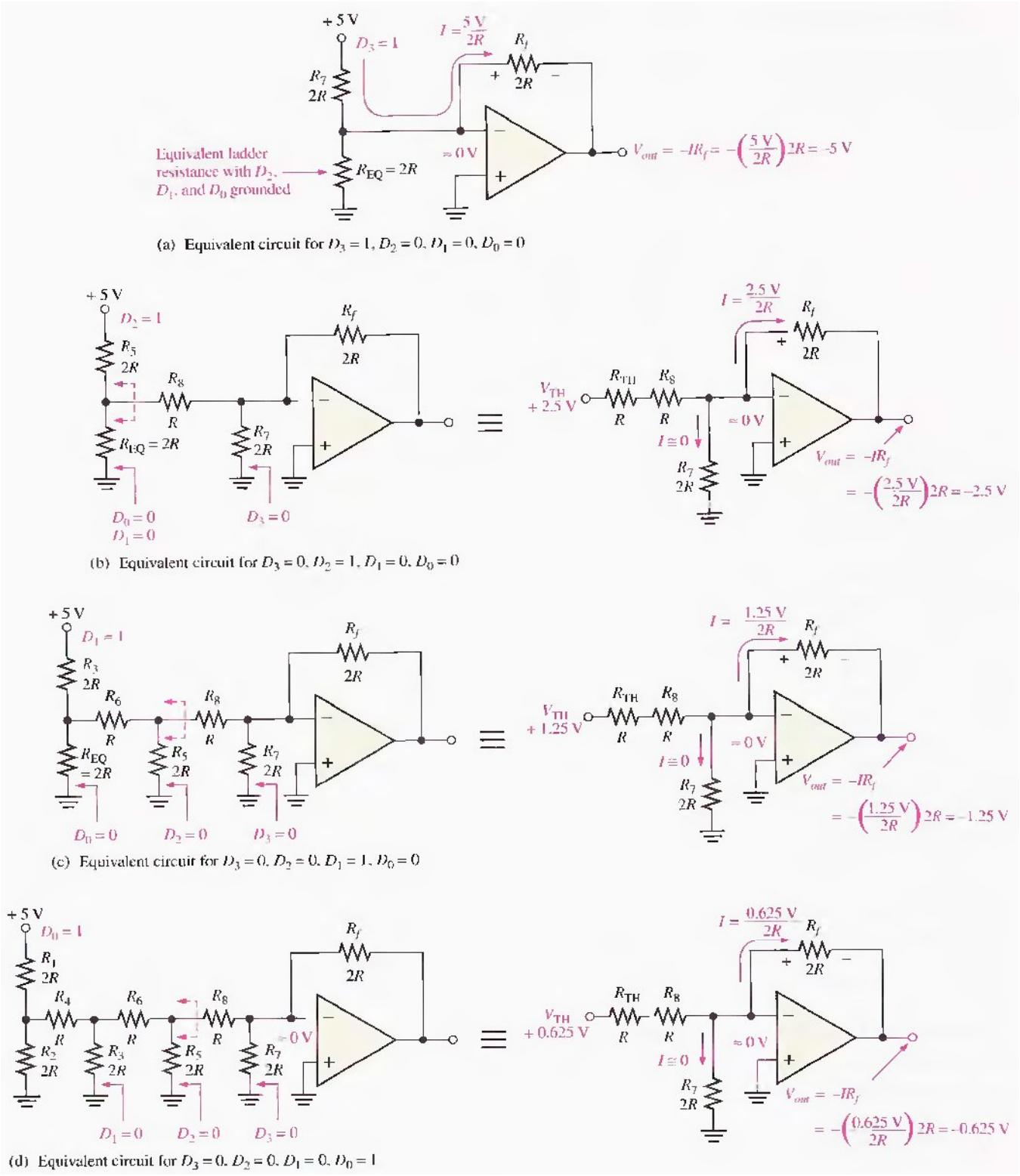


fig 4-24 Analysis of the R/2R ladder DAC.

There is no current into the op-amp inverting input and that there is no current through the equivalent resistance to ground because it has 0V across it, due to the virtual ground.

Fig4-24(c) shows the equivalent circuit when D_1 input is at +5V and the others are at ground. This condition represents 0010. theveninzing looking from R_s , you get 1.25V in series with R, this results in a current through R_f of I=1.25V/2R, which gives an output voltage of (- 1.25V).

Fig4-24(d) the equivalent circuit representing the case where D_o is at +5V and the other inputs are at ground. This condition represents 0001. Theveninzing from R_s gives an equivalent of 0.625V in series with R. The resulting current through R_f is I =0.625V/2R, which gives an output voltage of (-0.625V). Notice that each successively lower-weighted input produces an output voltage that is halfed so that the output voltage is proportional to the binary weight of the input bits.

Performance Characteristics of Digital-to-Analog Converters:-

***1-Resolution**: The resolution of a DAC is the reciprocal of the number of discrete steps in the output. This is dependent on the number of input bits. For example, a 4-bit DAC has a resolution of one part in 2^4 - 1 (one part in fifteen). Resolution can also be expressed as the number of bits that are converted.*

***2-Accuracy**: Accuracy is derived from a comparison of the actual output of a DAC with the expected output. It is expressed as a percentage of a full-scale, or maximum, output voltage. For example if a converter has a full-scale output of 10V and the accuracy is ±1 %, then the maximum error for any output voltage is (10V)(0.001) = 10 mV. Ideally the accuracy should be no worse than ± 1/2 of a least significant bit.*

***3-Linearity:** A linear error is a deviation from the ideal straight-line output of a DAC. A special case is an offset error, which is the amount of output voltage when the input bits are all zeros*

***4-Monotonicity**: A DAC is monotonic if it does not take any reverse steps when it is sequenced over its entire range of input bits.*

***5-Settling time**: Settling time is normally defined as the time it takes a DAC to settle within ± 1/2 LSB of its final value when a change occurs in the input code.*

***Example**: Determine the resolution, expressed as a percentage, of the following:*

(a) an 8-bit DAC (b) a 12-bit DAC

***Solution**: (a) For the 8-bit converter,*

$$\frac{1}{2^8 - 1} \times 100 = \frac{1}{255} \times 100 = 0.392\%$$

(b) For the 12-bit converter,

$$\frac{1}{2^{12} - 1} \times 100 = \frac{1}{4095} \times 100 = 0.0244\%$$

Related Problem Calculate the resolution for a 16-bit

The Reconstruction filter

The output of the DAC is a "stairstep" approximation of the original analog signal after it has been processed by the DSP. The purpose of the low-pass reconstruction filter (some-times called a post filter) is to smooth out the DAC output by eliminating the higher frequency content that results from the fast transitions of the "stairsteps," as roughly illustrated in Fig4-25.

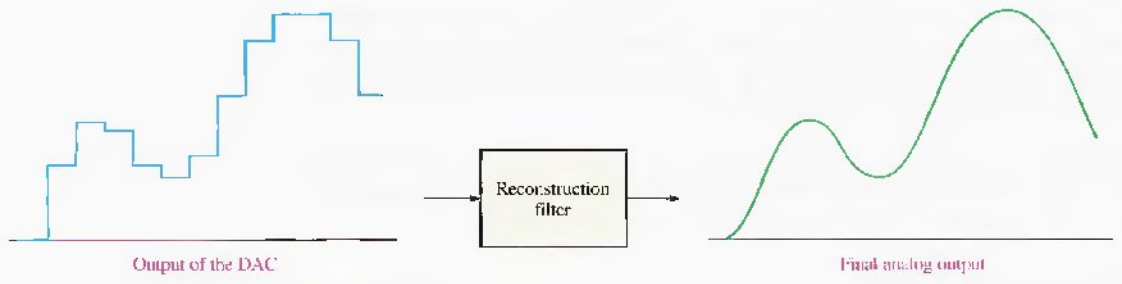


Fig4-25 the reconstruction filter smoothes the output of the DAC

Programmable logic: SPLDs &CPLDs

Two major types of simple programmable logic devices (SPLDs) are the PAL and the GAL. ***PAL*** *(programmable array logic).****GAL*** *(generic array logic). PAL is one-time programmable (OTP), and a GAL is a type of PAL that is reprogrammable. The basic structure of both PALs and GALs is a programmable AND array and a fixed OR array, which is a basic sum-of-products architecture. The complex programmable logic device (CPLD) is basically a single device with multiple SPLDs that provides more capacity for larger logic designs.*

SPLD: the PAL

A PAL (programmable array logic) consists of a programmable array of AND gates that connects to a fixed array of OR gates. PALs are implemented with ***fuse*** *process technology and are, therefore one-time programmable (OTP). The PAL structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented. any combinational logic function can be expressed in SOP form.*

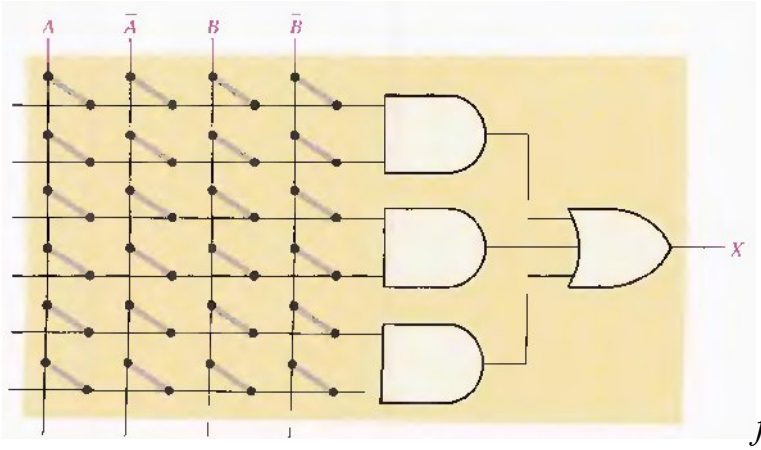


fig3-1 Basic AND/OR structure of a PAL

a programmable array is essentially a grid or matrix of conductors that form rows and columns with a programmable link at each cross point. Each programmable link, which is a fuse in the case of a PAL. is called a ***cell****. Each row is connected to the input of an AND gate, and each column is connected to an input variable or it complement. By programming the presence or absence of a fuse connection, any combination of input variables or complements can be applied to an AND gate to form any desired product term. The AND gates are connected to an OR gate, creating a sum-of- products (SOP) output.*

Implementing a Sum-of-Products Expression

The produced term AB is produced by the top AND gate. A¯B is produced by the middle AND gate, and ¯A¯B is produced by the bottom AND gate. the fuses are left intact to connect the desired variables or their complements to the appropriate AND gate inputs. The fuses are opened where a variable or its complement is not used in a given product term. The final output from the OR gate is the SOP expression(as shown in fig3-2).

$$X = AB + A\bar{B} + \bar{A}\bar{B}$$

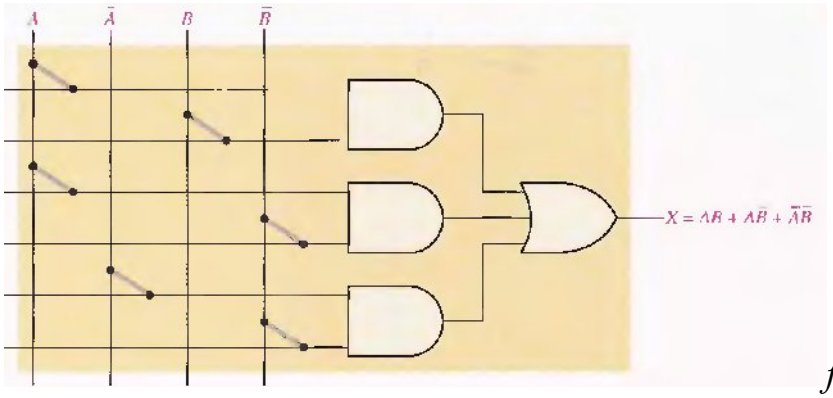


fig3-2 PAL implementation of SOP

SPLD: The GAL

The GAL is essentially a PAL that can be reprogrammed. It has the same type of AND/OR organization that the PAL does. The basic difference is that a GAL uses a reprogrammable process technology, such as EEPROM (E^2CMOS), instead of fuses, as shown in fig3-3.

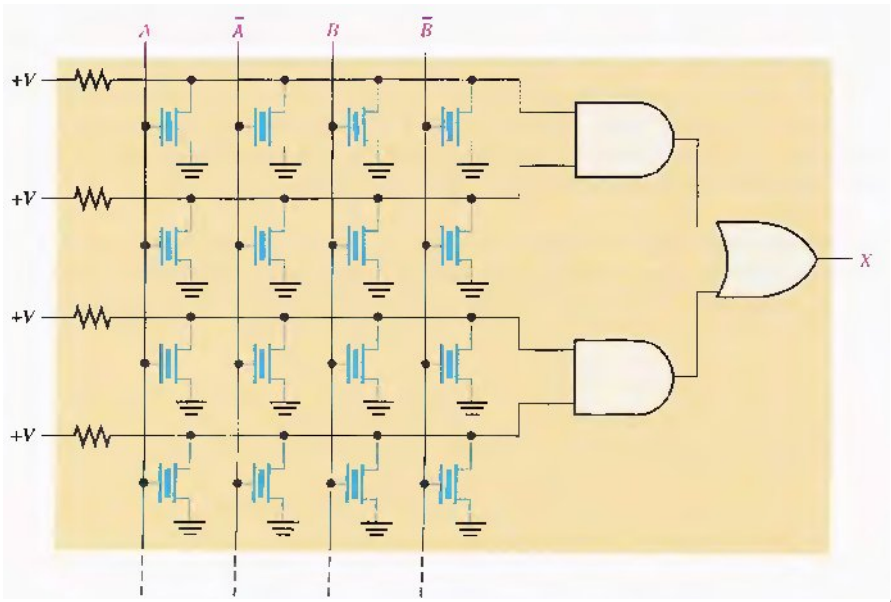


fig3-3 simplified GAL array

Simplified Notation for PAL/GAL Diagrams

PAL and GAL devices have many AND & OR gates in addition to other elements and are capable of handling many variables and their complements. The input variables to a PAL or GAL are usually buffered to prevent loading by a large number of AND gate inputs to which they are connected. a buffer that produces both the variable and its complement. PALs and GALs have a large number of programmable interconnection lines. and each AND gate has multiple inputs. PAL and GAL logic diagrams represent a multiple-input AND gate with an AND gate symbol having a single input line with a slash and a digit representing the actual number of inputs. Programmable links in an array are indicated in a diagram by X at the cross point for an intact fuse or other type of link and the absence of an X for an open fuse or other type of link. In fig3-4, the 2-variable logic function:

$AB + A\bar{B} + \bar{A}\bar{B}$ *is programmed.*

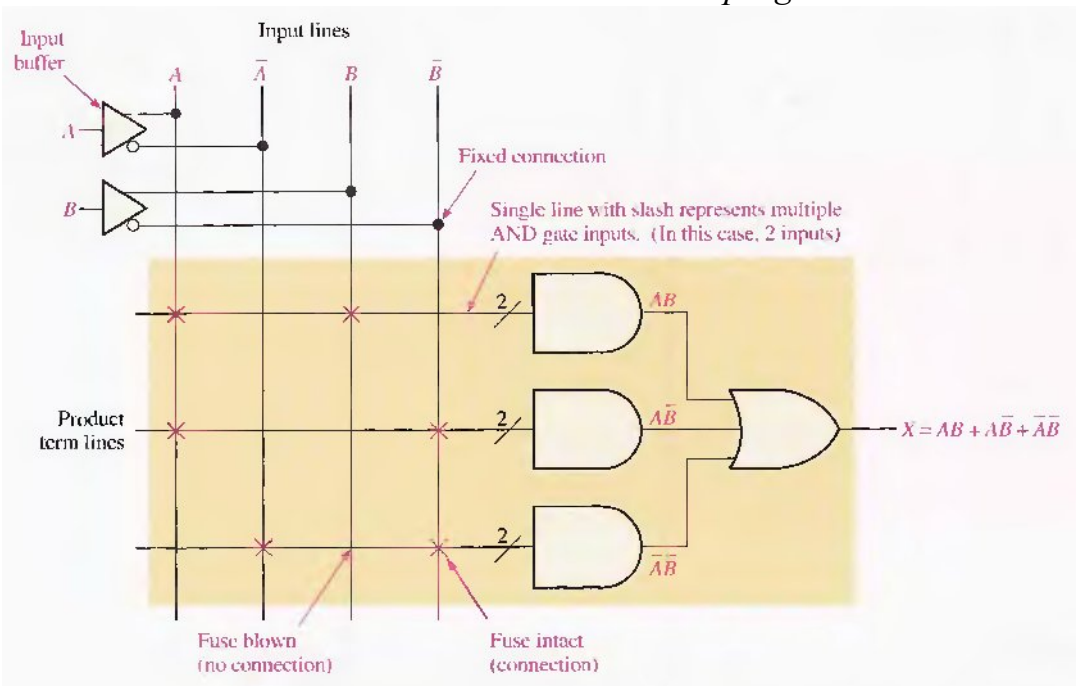


fig3- 4 A portion of a programmed PAL/GAL

Example3-1: *Show how a PAL is programmed for the following 3-variable logic function:*

$$X = A\bar{B}C + \bar{A}B\bar{C} + \bar{A}\bar{B} + AC$$

Solution :*The programmed array is shown in fig3-5. The intact fusible link, are indicated by small Xs. The absence of an X means that the fuse is open.*

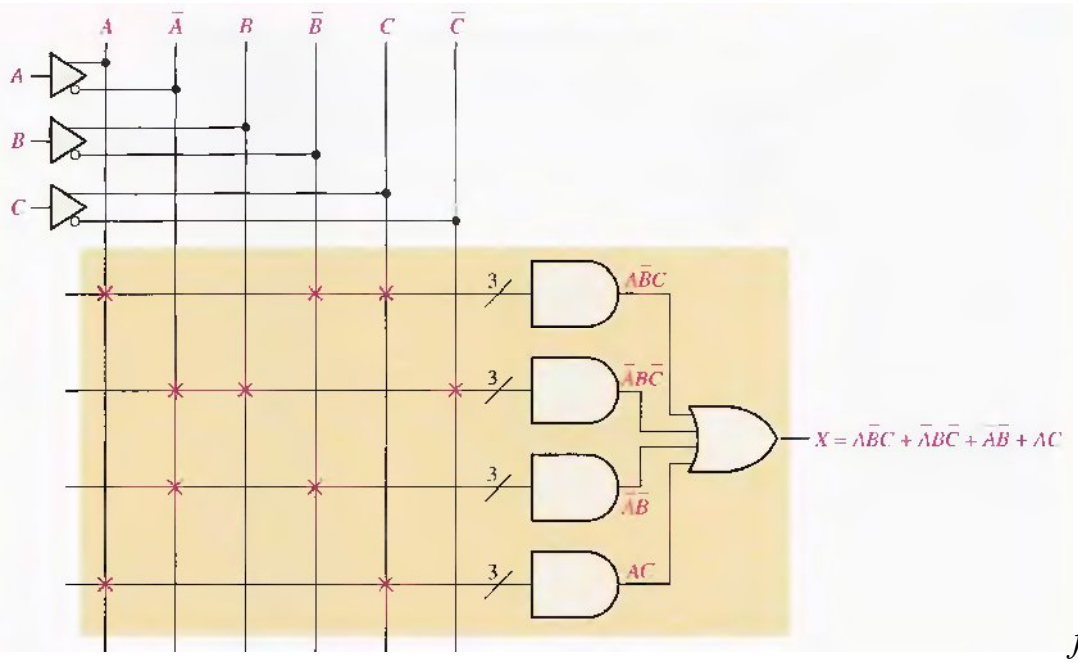


fig3-5

PAL/GAL General Block Diagram

A block diagram of a PAL or GAL is shown in fig3-6. the basic difference is that a GAL has a reprogrammable array and the PAL is one-time programmable. The programmable AND array outputs go to fixed OR gates that are connected to additional output logic. An OR gate combined with its associated output logic is typically called a ***macrocell*** *. The complexity of the macrocell depends on the particular device, and in GALs it is often reprogrammable.*

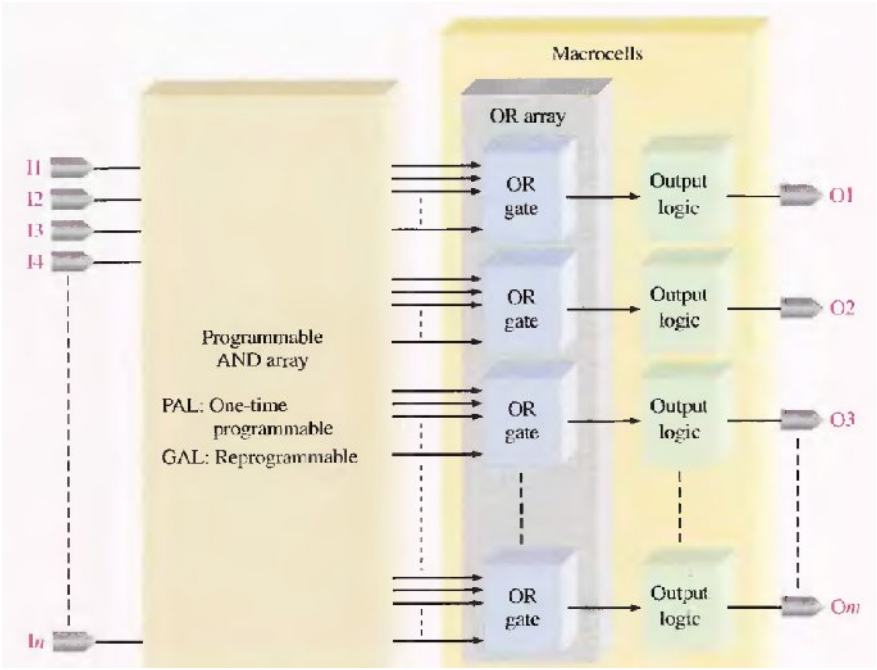
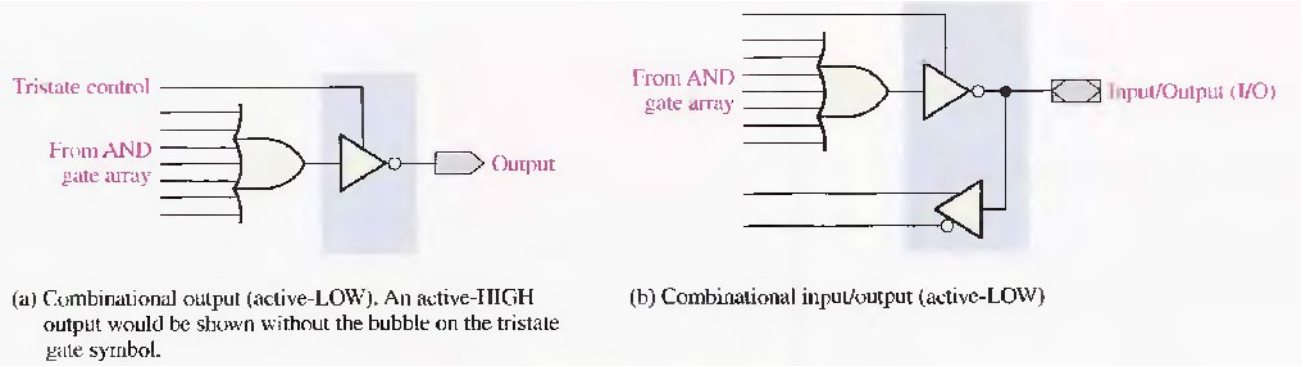


fig3-6 General block diagram of a PAL or GAL

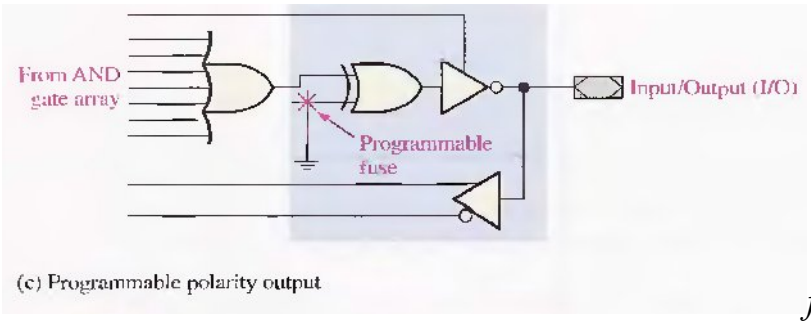
Macrocells

A macrocell generally consists of one OR gate and some associated output logic. The macrocells vary in complexity, depending on the particular type of PAL or GAL. A macrocell can be configured for combinational logic, registered logic, or a combination of both. Registered logic means that there is a flip-flop in the macrocell to provide for sequential logic functions. Fig3-7 (a) shows a simple macrocell with the OR gate and an inverter with a tri state control that can make the inverter like an open circuit to completely disconnect the output. The output of the tri state inverter can be either LOW, HIGH, or disconnected. Part (b) is a macrocell that can be either an input or an output. When the output is used as an input, the tri state inverter is disconnected, and the input goes to the buffer that is connected to the AND array. Part (c) is a macrocell that can be programmed to have either an active-HIGH or an active-LOW output, or it can be used as an input. One input to the exclusive-OR(XOR) gate can be programmed to be either HIGH or LOW. When the programmable XOR input is HIGH, the OR gate output is inverted because 0 +1 =1& 1 + 1 = 0. when the programmable XOR input is LOW. the OR gate output is not inverted because 0 + 0 = 0 & 1+0 = 1.



(a) Combinational output (active-LOW). An active-HIGH output would be shown without the bubble on the tristate gate symbol.

(b) Combinational input/output (active-LOW)



(c) Programmable polarity output

fig3- 7 PAL/GAL macrocells

Specific SPLDs

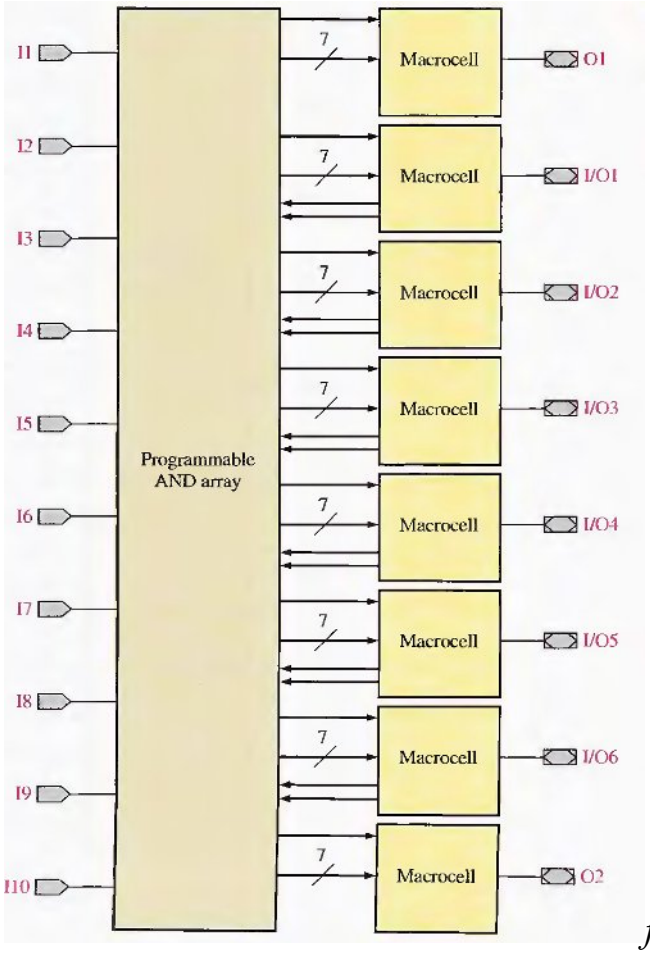


fig3-8 logic block diagram of A PAL 16V8

SPLD package configurations range from 20 pins to 28 pins. Two factors that you can use to help determine whether a certain PAL or GAL is adequate for a given logic design are the number of inputs and outputs and the number of equivalent gates or density. The 16V8 and 22V10 are common types of PALs and GALs. 16V8(fig3-8) means that the device has 16 inputs, 8 outputs, and the outputs are variable (V). each macrocell has 8 inputs from the AND gate array . There are 10 dedicated inputs (I), 2 dedicated outputs (O), and 6 pins that can be used as either inputs or outputs (I/O). Each output is active-LOW. The PAL16V8 has a density of approximately 300 equivalent gates. A block diagram for a GAL22V10 shown in fig3-9. This device has 12 dedicated inputs and 10 pins that can be either inputs or outputs. The macrocellIs have inputs from the AND array that vary from 8 to 16. The GAL22V10 has a density of approximately 500 equivalent gates.

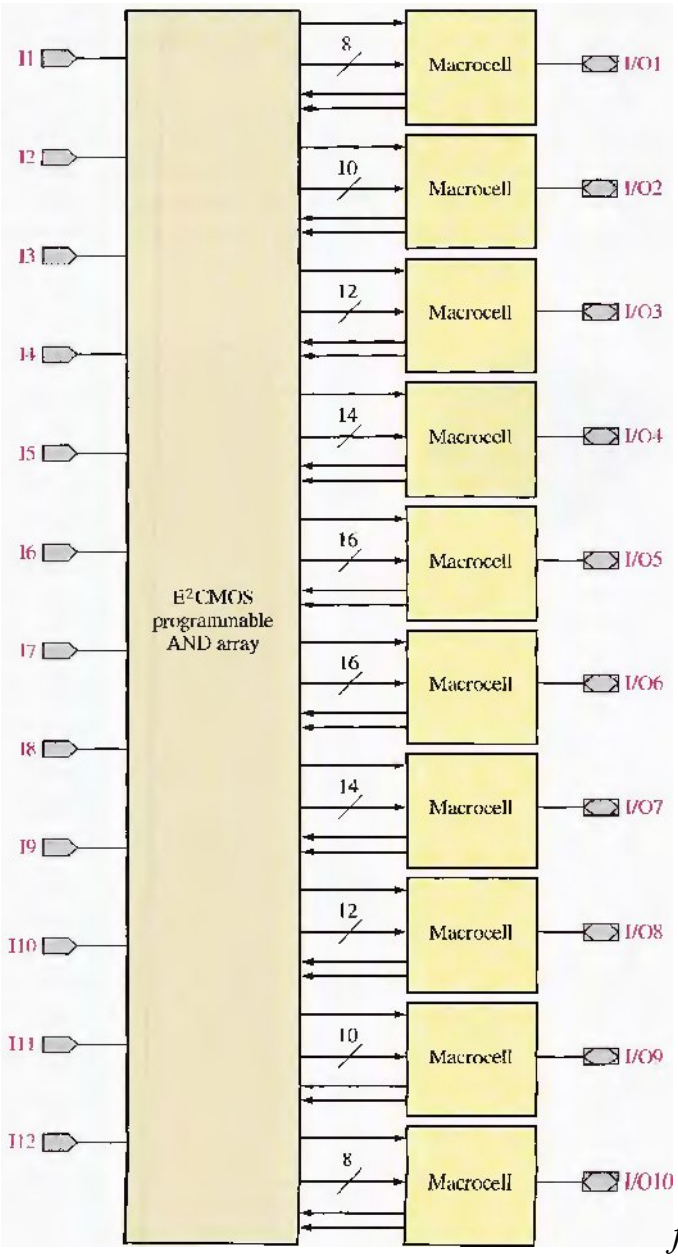


fig3-9 Block diagram of the GAU2V10

CPLD

A CPLD (complex programmable logic device) consists basically of multiple SPLD arrays with programmable interconnections, fig3-10 represents a generic CPLD. We will refer to each SPLD array in a CPLD as ***a LAB*** *(logic array block). Other designations are sometimes used, such as function block, logic block, or generic block. The programmable interconnections are generally called the PIA (programmable interconnect array).The LABs and the interconnections between LABs are programmed using software. A CPLD can be programmed for complex logic functions based on the SOP structure of the individual LABs (actually SPLDs). In- puts can be connected to any of the LABs, and their outputs can be interconnected to any other LABs via the PIA.*

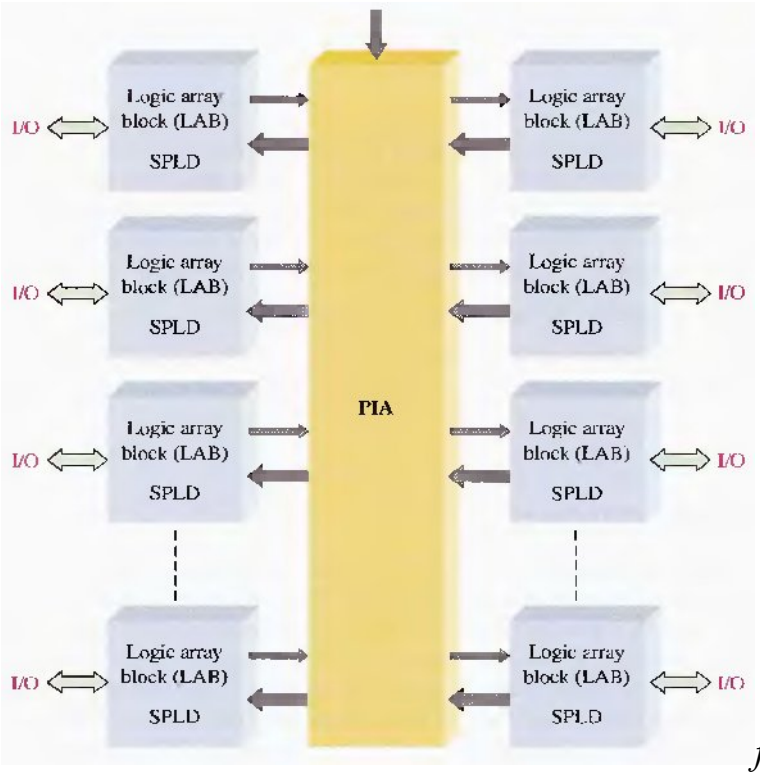


fig3-10 Basic block diagram of CPLD

Densities can range from tens of macrocells to over 2000 macrocells in packages with up to several hundred pins. As PLDs become more complex, maximum densities will increase. Most CPLDs are reprogrammable and use EEPROM or SRAM process technology for the programmable links.

PLA (Programmable logic Array)

As you have learned. the architecture of a CPLD is the way in which the internal elements are organized and arranged. fig 3-11 compares a simple PAL structure with a simple PLA structure. As you know, the PAL has a programmable AND array followed by a fixed OR array and produces an SOP expression, as shown by the example in fig3-11(a). The PLA has a programmable AND array followed by a programmable OR array, as shown by the example in fig3-11(b).

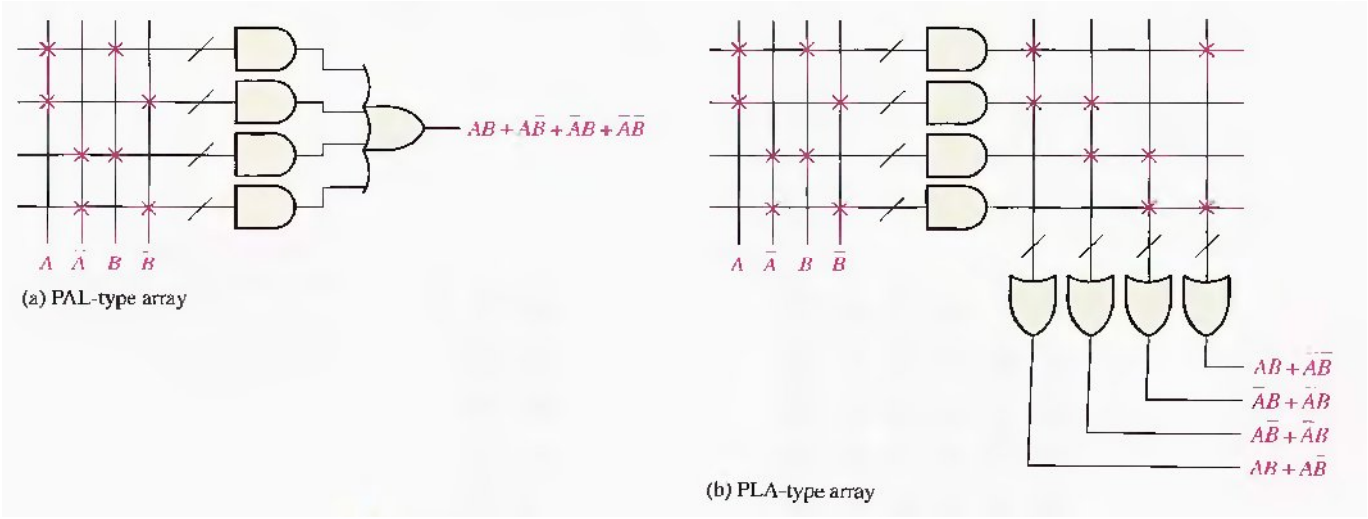


fig3-11 Comparison of a basic PLA to a basic PAL

***Complex PLD (CPLD)**: A digital device consisting of several programmable sections with internal interconnections between the sections. One kind of programmable device is FPGA. What is an **FPGA** Chip? Field Programmable Gate Array, A chip that can be configured by user to implement different digital hardware.*

*FPGAs are a form of **programmable logic (PL)**, which offer flexibility in design like software, but with performance speeds closer to Application Specific Integrated Circuits (ASICs). With the ability to be reconfigured an endless amount of times after it has already been manufactured, FPGAs are re-programmable digital ICs that were developed in the mid 1980s. FPGAs contain an internal array of logic elements, surrounded by a ring of programmable input/output blocks; all connected together via programmable interconnects. A personal computer can be used to design a digital circuit which is then compiled into a special programming file that will realize the circuit once it is downloaded to the FPGA.*

Field Programmable Gate Array Architecture.

***1-**Programmable Logic Cells which provide the functional elements for constructing the user's logic.*

***2-**Programmable Input/output (I/O) blocks which provide the interface between the package pins and logic cell.*

***3-**Programmable interconnects which provide routing paths to connect the inputs and outputs of the logic cell and I/O blocks.*

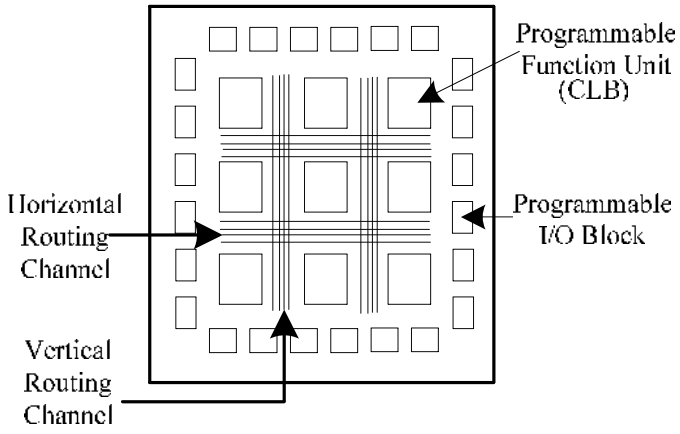


fig3-12 Conceptual structure of FPGAs

- Field Programmable Gate Array
- A chip that can be configured by user to implement different digital hardware
- Configurable Logic Blocks (CLB) and Programmable Switch Matrices
- Bitstream to configure: function of each block & the interconnection between logic blocks

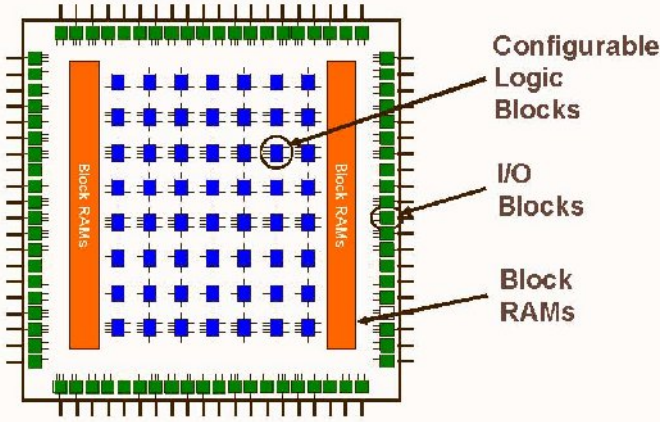


fig3-13 structure FPGA

***1- CLB** has:**4** slices **Each** slices has **2** logic cells **Each** logic cell has **2** LUTs + other logic (carry & control) + a flip-flop/latch*

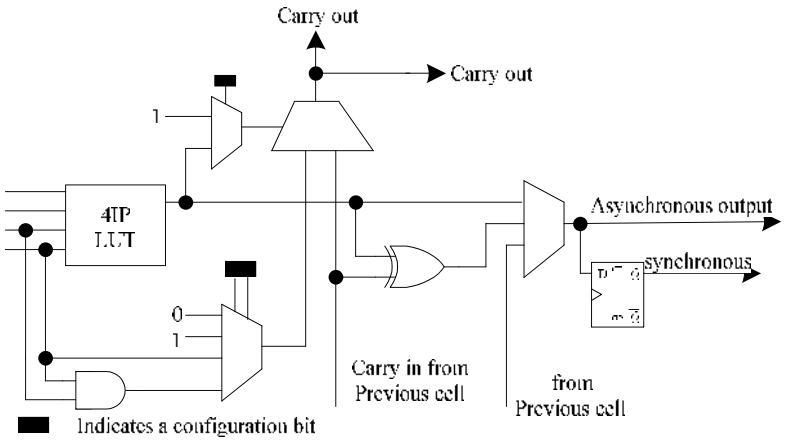


fig 3-14 Detailed view of Slice

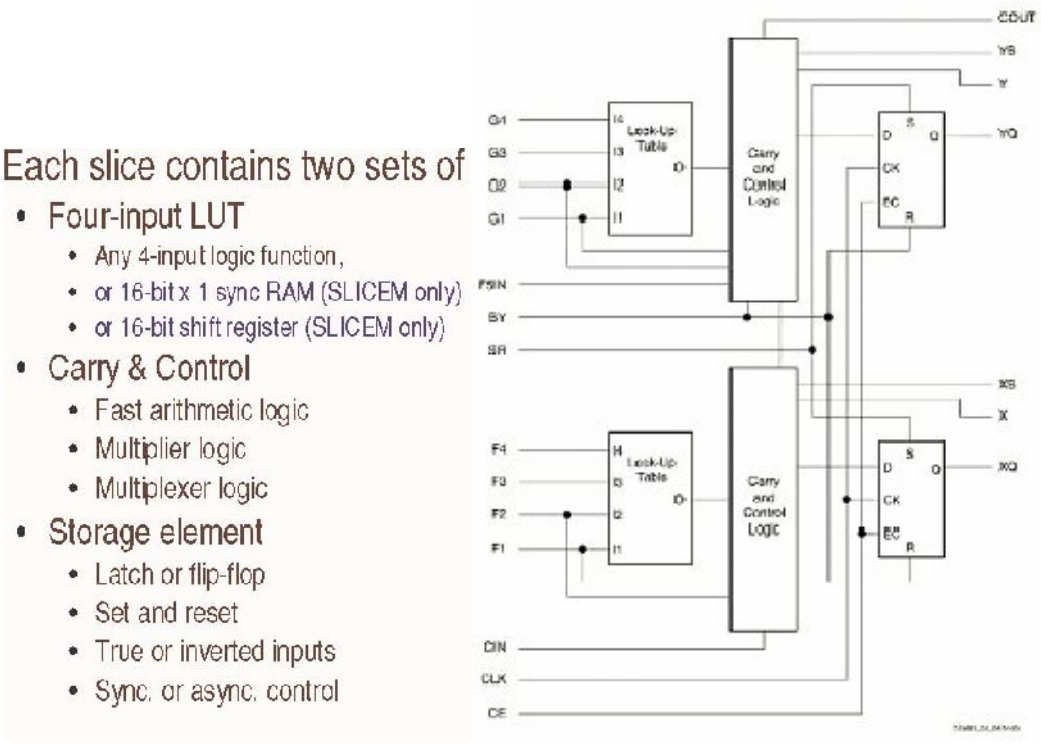


fig3-15 Logic Cell

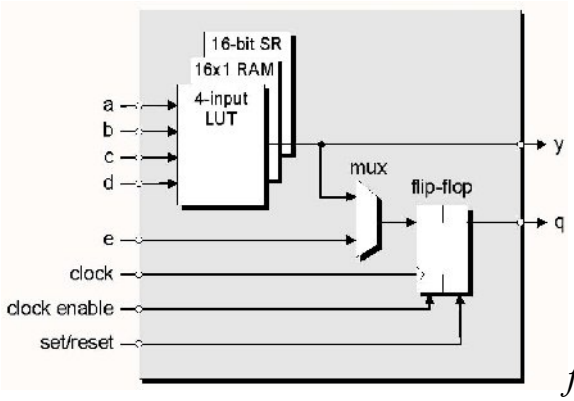


fig 3-16 view of logic cell

LUT (Look-Up Table) Functionality: *The Look Up-Tables can be programmed with truth tables of 4 input or 3 input logic functions*

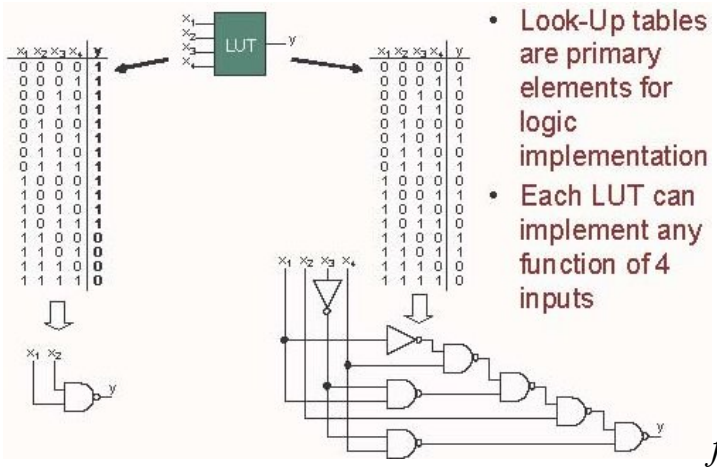


fig3-17 functionality of LUT

-Input Functions implemented using two LUTs

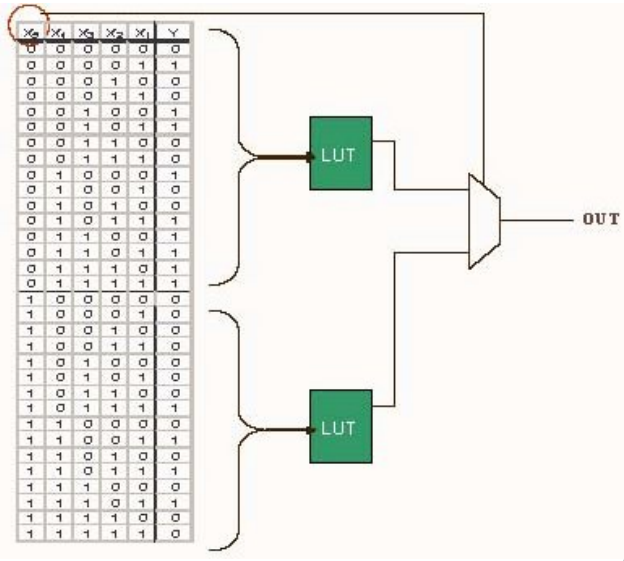


fig3-18 functions implemented using LUT

Distributed RAM

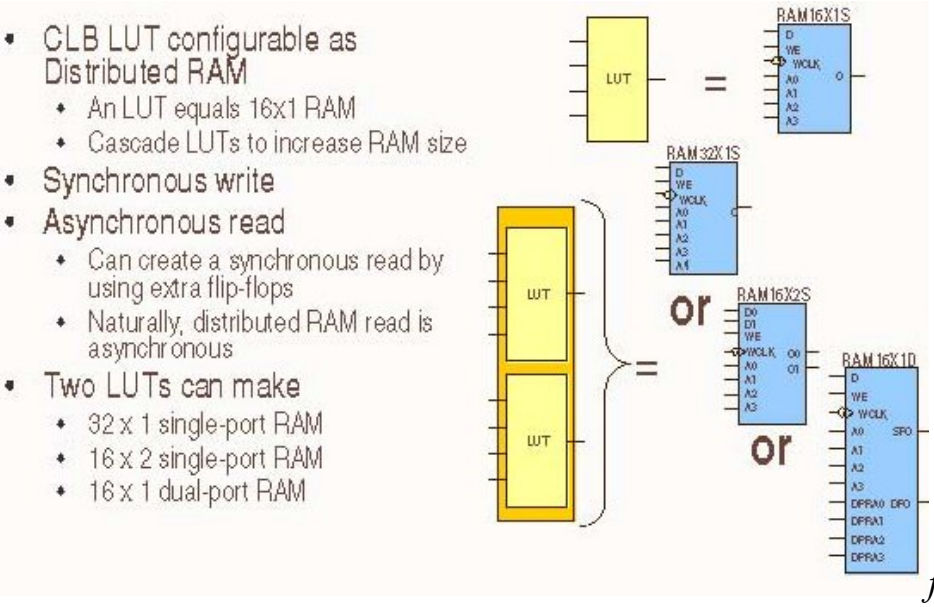


fig3-19LUT (distributed RAM)

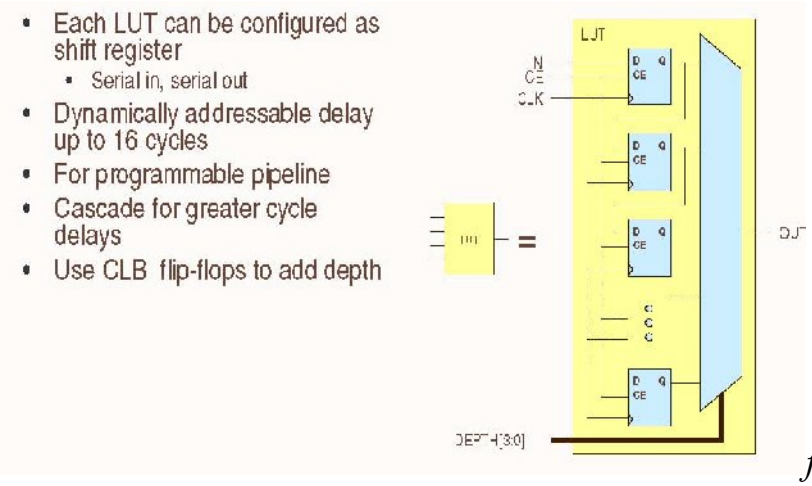


fig 3-20 LUT (Shift Register)

2- Input/output Blocks (IOBs)

The IOBs provide the interface between the package pins and internal signal lines Input-Output Blocks. They can be synchronous or asynchronous, and they can be input, output, tri-state output, or bi-directional pads.

3- Programmable Inter-Connections.

The programmer may need more than one unit to accomplish the designed system, so an inter connection is needed between these individual units to mange their individual or multiplexed tasks. Programmable interconnect paths do the required interconnection between different Programmable Function Units, In addition to CLBs, these FPGAs have horizontal and vertical interconnects and switches (routing resources) to achieve connectivity between different ports of different CLBs.

VHDL : *is a language for describing digital hardware used by industry worldwide. VHDL is an acronym for* ***VHSIC (Very High Speed Integrated Circuit) Hardware Description Language***

VHDL Source Code

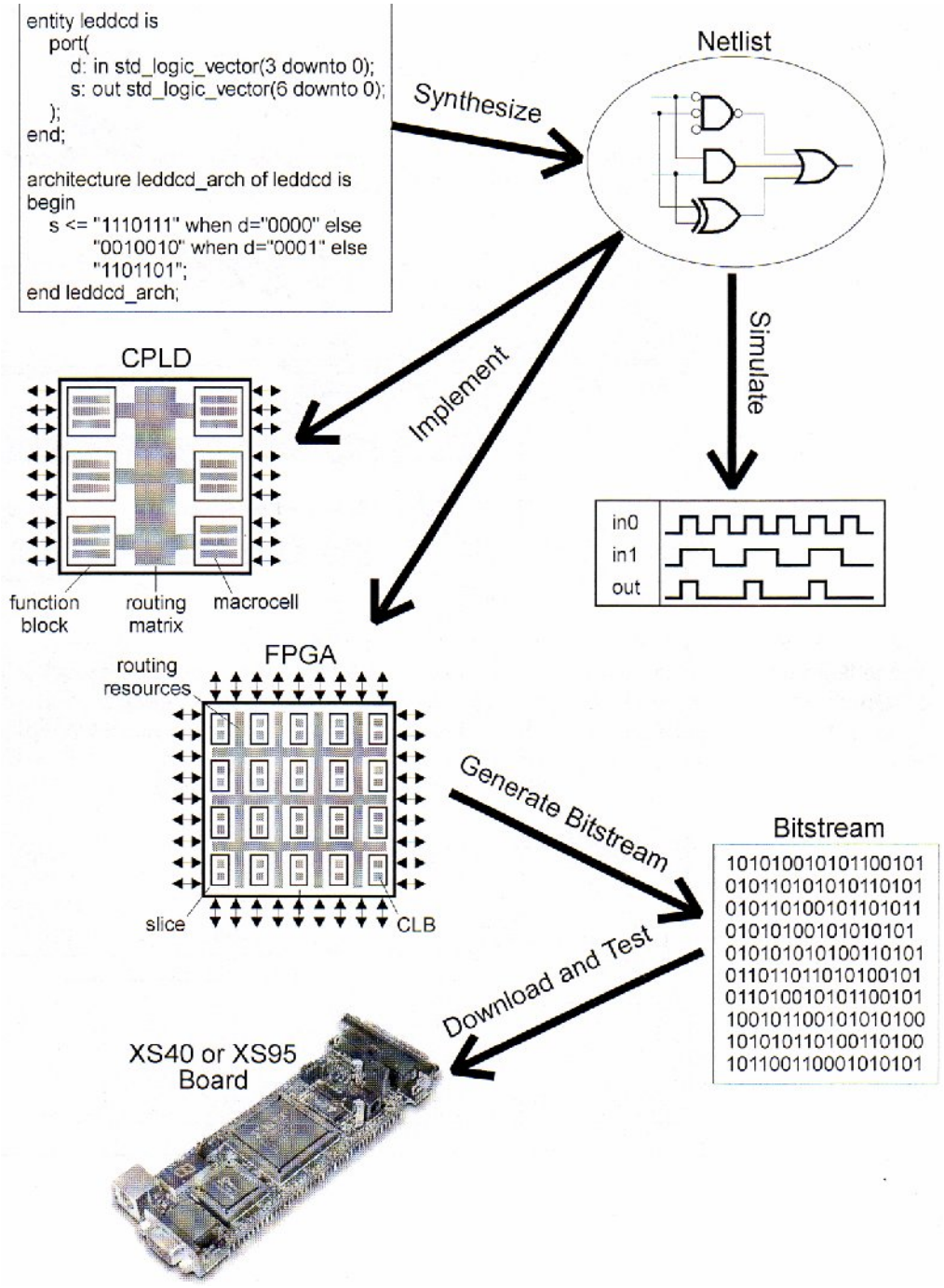


fig 3-20 Steps in creating and testing an FPGA or CPLD-based design