11

Counters and Registers

Counters and *registers* belong to the category of MSI sequential logic circuits. They have similar architecture, as both counters and registers comprise a cascaded arrangement of more than one flip-flop with or without combinational logic devices. Both constitute very important building blocks of sequential logic, and different types of counter and register available in integrated circuit (IC) form are used in a wide range of digital systems. While counters are mainly used in counting applications, where they either measure the time interval between two unknown time instants or measure the frequency of a given signal, registers are primarily used for the temporary storage of data present at the output of a digital circuit before they are fed to another digital circuit. We are all familiar with the role of different types of register used inside a microprocessor, and also their use in microprocessor-based applications. Because of the very nature of operation of registers, they form the basis of a very important class of counters called *shift counters*. In this chapter, we will discuss different types of counter and register as regards their operational basics, design methodology and application-relevant aspects. Design aspects have been adequately illustrated with the help of a large number of solved examples. A comprehensive functional index of a large number of integrated circuit counters and registers is given towards the end of the chapter.

11.1 Ripple (Asynchronous) Counter

A *ripple counter* is a cascaded arrangement of flip-flops where the output of one flip-flop drives the clock input of the following flip-flop. The number of flip-flops in the cascaded arrangement depends upon the number of different logic states that it goes through before it repeats the sequence, a parameter known as the modulus of the counter.

In a ripple counter, also called an *asynchronous counter* or a *serial counter*, the clock input is applied only to the first flip-flop, also called the input flip-flop, in the cascaded arrangement. The clock input to any subsequent flip-flop comes from the output of its immediately preceding flip-flop. For instance, the output of the first flip-flop acts as the clock input to the second flip-flop, the output of the second flip-flop feeds the clock input of the third flip-flop and so on. In general, in an arrangement of n

Digital Electronics: Principles, Devices and Applications Anil K. Maini
 ISBN: 978-0-470-03214-5

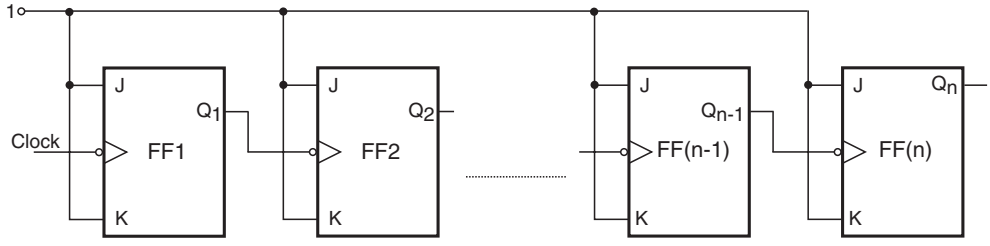


Figure 11.1 Generalized block schematic of n-bit binary ripple counter.

flip-flops, the clock input to the nth flip-flop comes from the output of the $(n-1)$th flip-flop for $n > 1$. Figure 11.1 shows the generalized block schematic arrangement of an n-bit binary ripple counter.

As a natural consequence of this, not all flip-flops change state at the same time. The second flip-flop can change state only after the output of the first flip-flop has changed its state. That is, the second flip-flop would change state a certain time delay after the occurrence of the input clock pulse owing to the fact that it gets its own clock input from the output of the first flip-flop and not from the input clock. This time delay here equals the sum of propagation delays of two flip-flops, the first and the second flip-flops. In general, the nth flip-flop will change state only after a delay equal to n times the propagation delay of one flip-flop. The term 'ripple counter' comes from the mode in which the clock information ripples through the counter. It is also called an 'asynchronous counter' as different flip-flops comprising the counter do not change state in synchronization with the input clock.

In a counter like this, after the occurrence of each clock input pulse, the counter has to wait for a time period equal to the sum of propagation delays of all flip-flops before the next clock pulse can be applied. The propagation delay of each flip-flop, of course, will depend upon the logic family to which it belongs.

11.1.1 Propagation Delay in Ripple Counters

A major problem with ripple counters arises from the propagation delay of the flip-flops constituting the counter. As mentioned in the preceding paragraphs, the effective propagation delay in a ripple counter is equal to the sum of propagation delays due to different flip-flops. The situation becomes worse with increase in the number of flip-flops used to construct the counter, which is the case in larger bit counters. Coming back to the ripple counter, an increased propagation delay puts a limit on the maximum frequency used as clock input to the counter. We can appreciate that the clock signal time period must be equal to or greater than the total propagation delay. The maximum clock frequency therefore corresponds to a time period that equals the total propagation delay. If t_{pd} is the propagation delay in each flip-flop, then, in a counter with N flip-flops having a modulus of less than or equal to 2^N, the maximum usable clock frequency is given by $f_{max} = 1/(N \times t_{pd})$. Often, two propagation delay times are specified in the case of flip-flops, one for LOW-to-HIGH transition (t_{pLH}) and the other for HIGH-to-LOW transition (t_{pHL}) at the output. In such a case, the larger of the two should be considered for computing the maximum clock frequency.

As an example, in the case of a ripple counter IC belonging to the low-power Schottky TTL (LSTTL) family, the propagation delay per flip-flop typically is of the order of 25 ns. This implies that a four-bit

ripple counter from this logic family can not be clocked faster than 10 MHz. The upper limit on the clock frequency further decreases with increase in the number of bits to be handled by the counter.

11.2 Synchronous Counter

In a *synchronous counter*, also known as a *parallel counter*, all the flip-flops in the counter change state at the same time in synchronism with the input clock signal. The clock signal in this case is simultaneously applied to the clock inputs of all the flip-flops. The delay involved in this case is equal to the propagation delay of one flip-flop only, irrespective of the number of flip-flops used to construct the counter. In other words, the delay is independent of the size of the counter.

11.3 Modulus of a Counter

The *modulus* (MOD number) of a counter is the number of different logic states it goes through before it comes back to the initial state to repeat the count sequence. An n-bit counter that counts through all its natural states and does not skip any of the states has a modulus of 2^n. We can see that such counters have a modulus that is an integral power of 2, that is, 2, 4, 8, 16 and so on. These can be modified with the help of additional combinational logic to get a modulus of less than 2^n.

To determine the number of flip-flops required to build a counter having a given modulus, identify the smallest integer m that is either equal to or greater than the desired modulus and is also equal to an integral power of 2. For instance, if the desired modulus is 10, which is the case in a decade counter, the smallest integer greater than or equal to 10 and which is also an integral power of 2 is 16. The number of flip-flops in this case would be 4, as $16 = 2^4$. On the same lines, the number of flip-flops required to construct counters with MOD numbers of 3, 6, 14, 28 and 63 would be 2, 3, 4, 5 and 6 respectively. In general, the arrangement of a minimum number of N flip-flops can be used to construct any counter with a modulus given by the equation

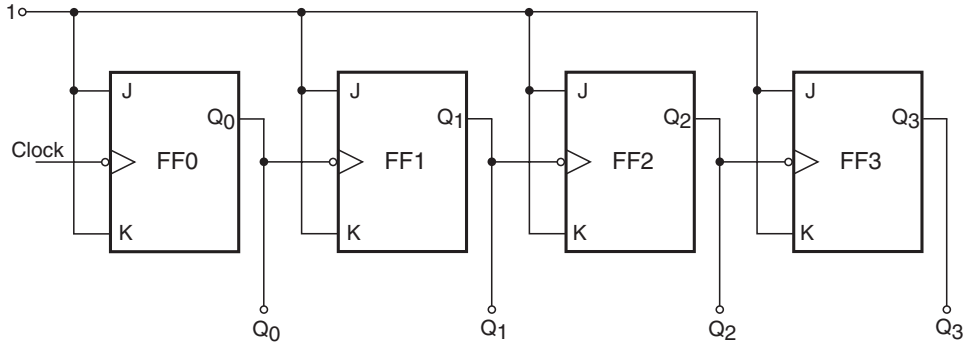
$$(2^{N-1} + 1) \leq modulus \leq 2^N \tag{11.1}$$

11.4 Binary Ripple Counter – Operational Basics

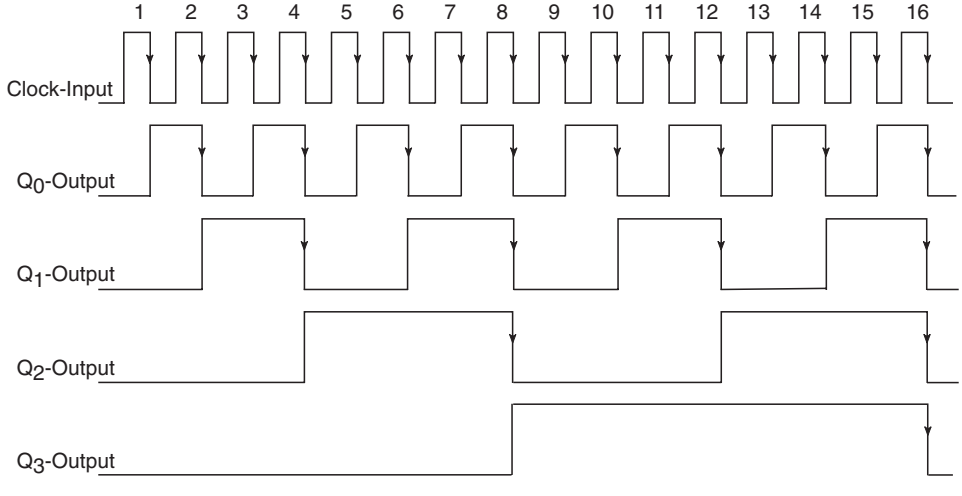
The operation of a binary ripple counter can be best explained with the help of a typical counter of this type. Figure 11.2(a) shows a four-bit ripple counter implemented with negative edge-triggered *J-K* flip-flops wired as toggle flip-flops. The output of the first flip-flop feeds the clock input of the second, and the output of the second flip-flop feeds the clock input of the third, the output of which in turn feeds the clock input of the fourth flip-flop. The outputs of the four flip-flops are designated as Q_0 (LSB flip-flop), Q_1, Q_2 and Q_3 (MSB flip-flop). Figure 11.2(b) shows the waveforms appearing at Q_0, Q_1, Q_2 and Q_3 outputs as the clock signal goes through successive cycles of trigger pulses. The counter functions as follows.

Let us assume that all the flip-flops are initially cleared to the '0' state. On HIGH-to-LOW transition of the first clock pulse, Q_0 goes from '0' to '1' owing to the toggling action. As the flip-flops used are negative edge-triggered ones, the '0' to '1' transition of Q_0 does not trigger flip-flop FF1. FF1, along with FF2 and FF3, remains in its '0' state. So, on the occurrence of the first negative-going clock transition, $Q_0 = 1$, $Q_1 = 0$, $Q_2 = 0$ and $Q_3 = 0$.

On the HIGH-to-LOW transition of the second clock pulse, Q_0 toggles again. That is, it goes from '1' to '0'. This '1' to '0' transition at the Q_0 output triggers FF1, the output Q_1 of which goes from '0'



(a)



(b)

Figure 11.2 Four-bit binary ripple counter.

to '1'. The Q_2 and Q_3 outputs remain unaffected. Therefore, immediately after the occurrence of the second HIGH-to-LOW transition of the clock signal, $Q_0 = 0$, $Q_1 = 1$, $Q_2 = 0$ and $Q_3 = 0$. On similar lines, we can explain the logic status of Q_0, Q_1, Q_2 and Q_3 outputs immediately after subsequent clock transitions. The logic status of outputs for the first 16 relevant (HIGH-to-LOW in the present case) clock signal transitions is summarized in Table 11.1.

Thus, we see that the counter goes through 16 distinct states from 0000 to 1111 and then, on the occurrence of the desired transition of the sixteenth clock pulse, it resets to the original state of 0000 from where it had started. In general, if we had N flip-flops, we could count up to 2^N pulses before the counter resets to the initial state. We can also see from the Q_0, Q_1, Q_2 and Q_3 waveforms, as shown

Table 11.1 Output logic states for different clock signal transitions for a four-bit binary ripple counter.

Clock signal transition number	Q_0	Q_1	Q_2	Q_3
After first clock transition	1	0	0	0
After second clock transition	0	1	0	0
After third clock transition	1	1	0	0
After fourth clock transition	0	0	1	0
After fifth clock transition	1	0	1	0
After sixth clock transition	0	1	1	0
After seventh clock transition	1	1	1	0
After eighth clock transition	0	0	0	1
After ninth clock transition	1	0	0	1
After tenth clock transition	0	1	0	1
After eleventh clock transition	1	1	0	1
After twelfth clock transition	0	0	1	1
After thirteenth clock transition	1	0	1	1
After fourteenth clock transition	0	1	1	1
After fifteenth clock transition	1	1	1	1
After sixteenth clock transition	0	0	0	0

in Fig. 11.2(b), that the frequencies of the Q_0, Q_1, Q_2 and Q_3 waveforms are $f/2$,$f/4$, $f/8$ and $f/16$ respectively. Here, f is the frequency of the clock input. This implies that a counter of this type can be used as a divide-by-2^N circuit, where N is the number of flip-flops in the counter chain. In fact, such a counter provides frequency-divided outputs of $f/2^N$, $f/2^{N-1}$, $f/2^{N-2}$, $f/2^{N-3}$, . . . , $f/2$ at the outputs of the Nth, $(N-1)$th, $(N-2)$th, $(N-3)$th, . . . , first flip-flops. In the case of a four-bit counter of the type shown in Fig. 11.2(a), outputs are available at $f/2$ from the Q_0 output, at $f/4$ from the Q_1 output, at $f/8$ from the Q_2 output and at $f/16$ from the Q_3 output. It may be noted that frequency division is one of the major applications of counters.

Example 11.1

A four-bit binary ripple counter of the type shown in Fig. 11.2(a) is initially in the 0000 state before the clock input is applied to the counter. The clock pulses are applied to the counter at some time instant t_1 and then again removed some time later at another time instant t_2. The counter is observed to read 0011. How many negative-going clock transitions have occurred during the time the clock was active at the counter input?

Solution

It is not possible to determine the number of clock edges – it could have been 3, 19, 35, 51, 67, 83 . . . – as there is no means of finding out whether the counter has recycled or not from the given data. Remember that this counter would come back to the 0000 state after every 16 clock pulses.

Example 11.2

It is desired to design a binary ripple counter of the type shown in Fig. 11.1 that is capable of counting the number of items passing on a conveyor belt. Each time an item passes a given point, a pulse is generated that can be used as a clock input. If the maximum number of items to be counted is 6000, determine the number of flip-flops required.

Solution

- The counter should be able to count a maximum of 6000 items.
- An N-flip-flop would be able to count up to a maximum of $2^N - 1$ counts.
- On the 2^Nth clock pulse, it will get reset to all 0s.
- Now, $2^N - 1$ should be greater than or equal to 6000.
- That is, $2^N - 1 \geq 6000$, which gives $N \geq \log 6001/\log 2 \geq 3.778/0.3010 \geq 12.55$.
- The smallest integer that satisfies this condition is 13.
- Therefore, the minimum number of flip-flops required = 13

11.4.1 Binary Ripple Counters with a Modulus of Less than 2^N

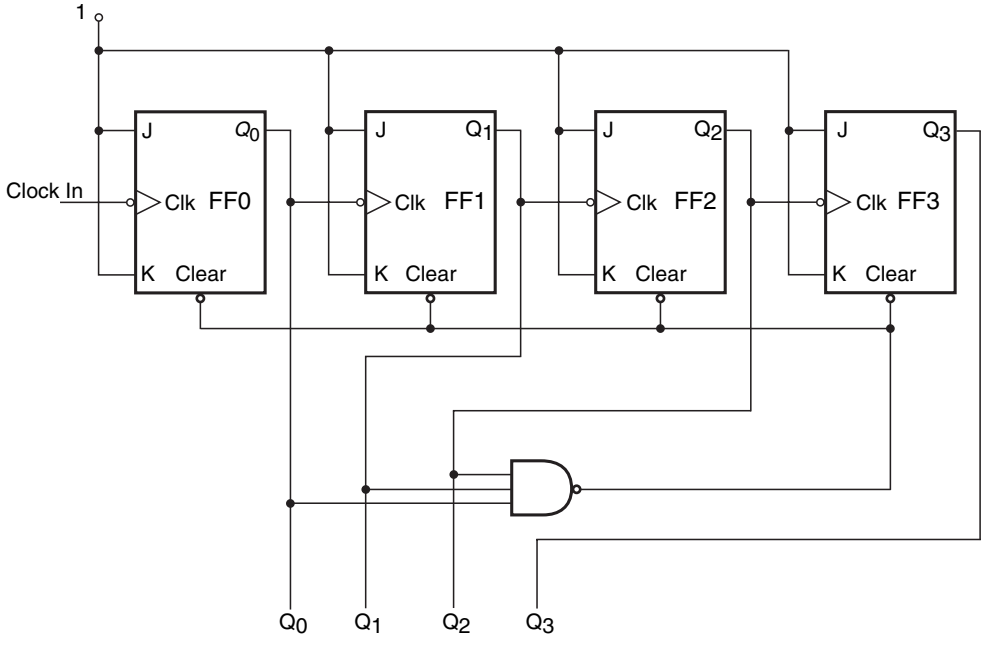
An N-flip-flop binary ripple counter can be modified, as we will see in the following paragraphs, to have any other modulus less than 2^N with the help of simple externally connected combinational logic. We will illustrate this simple concept with the help of an example.

Consider the four-flip-flop binary ripple counter arrangement of Fig. 11.3(a). It uses *J-K* flip-flops with an active LOW asynchronous CLEAR input. The NAND gate in the figure has its output connected to the CLEAR inputs of all four flip-flops. The inputs to this three-input NAND gate are from the Q outputs of flip-flops FF0, FF1 and FF2. If we disregard the NAND gate for some time, this counter will go through its natural binary sequence from 0000 to 1111. But that is not to happen in the present arrangement. The counter does start counting from 0000 towards its final count of 1111. The counter keeps counting as long as the asynchronous CLEAR inputs of the different flip-flops are inactive. That is, the NAND gate output is HIGH. This is the case until the counter reaches 0110. With the seventh clock pulse it tends to go to 0111, which makes all NAND gate inputs HIGH, forcing its output to LOW. This HIGH-to-LOW transition at the NAND gate output clears all flip-flop outputs to the logic '0' state, thus disallowing the counter to settle at 0111. From the eighth clock pulse onwards, the counter repeats the sequence. The counter thus always counts from 0000 to 0110 and resets back to 0000. The remaining nine states, which include 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 1111, are skipped, with the result that we get an MOD-7 counter.

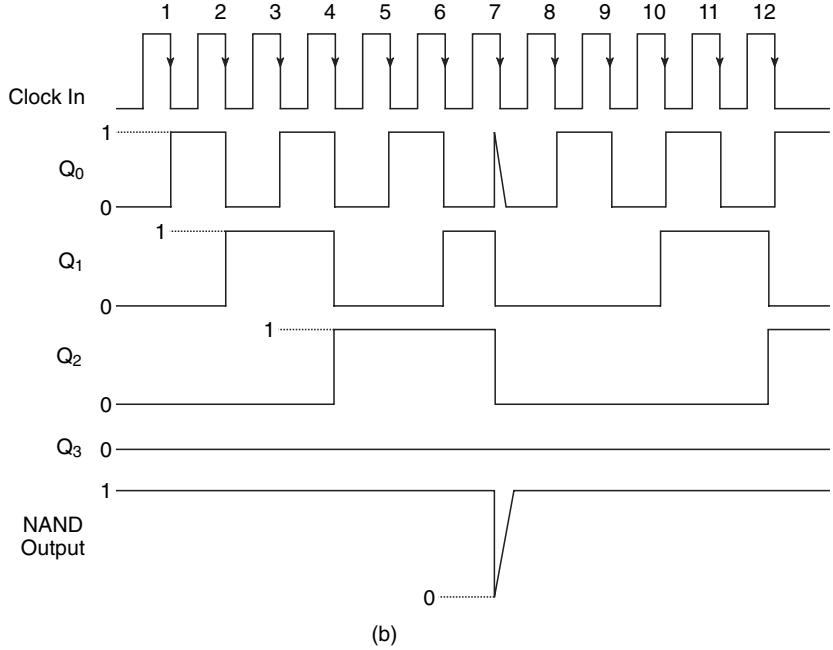
Figure 11.3(b) shows the timing waveforms for this counter. By suitably choosing NAND inputs, one can get a counter with any MOD number less than 16. Examination of timing waveforms also reveals that the frequency of the Q_2 output is one-seventh of the input clock frequency.

The waveform at the Q_2 output is, however, not symmetrical as it would be if the counter were to go through its full binary sequence. The Q_3 output stays in the logic LOW state. It is expected to be so because an MOD-7 counter needs a minimum of three flip-flops. That is why the fourth flip-flop, which was supposed to toggle on the HIGH-to-LOW transition of the eighth clock pulse, and on every successive eighth pulse thereafter, never gets to that stage. The counter is cleared on the seventh clock pulse and every successive seventh clock pulse thereafter.

As another illustration, if the NAND gate used in the counter arrangement of Fig. 11.3(a) is a two-input NAND and its inputs are from the Q_1 and Q_3 outputs, the counter will go through 0000 to 1001 and then reset to 0000 again, as, the moment the counter tends to switch from the 1001 to the 1010 state, the NAND gate goes from the '1' to the '0' state, clearing all flip-flops to the '0' state.



(a)



(b)

Figure 11.3 Binary ripple counter with a modulus of less than 2^N.

Steps to be followed to design any binary ripple counter that starts from 0000 and has a modulus of X are summarized as follows:

1. Determine the minimum number of flip-flops N so that $2^N \geq X$. Connect these flip-flops as a binary ripple counter. If $2^N = X$, do not go to steps 2 and 3.
2. Identify the flip-flops that will be in the logic HIGH state at the count whose decimal equivalent is X. Choose a NAND gate with the number of inputs equal to the number of flip-flops that would be in the logic HIGH state. As an example, if the objective were to design an MOD-12 counter, then, in the corresponding count, that is, 1100, two flip-flops would be in the logic HIGH state. The desired NAND gate would therefore be a two-input gate.
3. Connect the Q outputs of the identified flip-flops to the inputs of the NAND gate and the NAND gate output to asynchronous clear inputs of all flip-flops.

11.4.2 Ripple Counters in IC Form

In this section, we will look at the internal logic diagram of a typical binary ripple counter and see how close its architecture is to the ripple counter described in the previous section. Let us consider binary ripple counter type number 74293. It is a four-bit binary ripple counter containing four master–slave-type *J-K* flip-flops with additional gating to provide a divide-by-2 counter and a three-stage MOD-8 counter. Figure 11.4 shows the internal logic diagram of this counter. To get the full binary sequence of 16 states, the Q output of the LSB flip-flop is connected to the B input, which is the clock input of the next higher flip-flop. The arrangement then becomes the same as that shown in Fig. 11.2(a), with the exception of the two-input NAND gate of Fig. 11.4, which has been included here for providing the clearing features. The counter can be cleared to the 0000 logic state by driving both RESET inputs to the logic HIGH state. Tables 11.2 and 11.3 respectively give the functional table and the count sequence.

Example 11.3

Refer to the binary ripple counter of Fig. 11.5. Determine the modulus of the counter and also the frequency of the flip-flop Q_3 output.

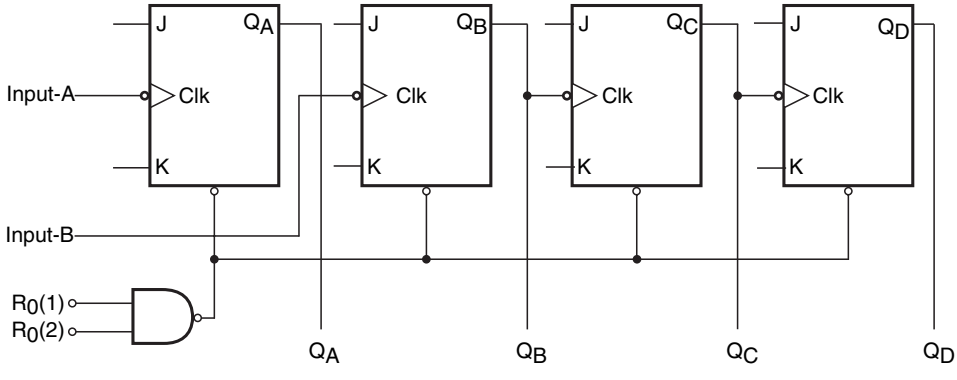


Figure 11.4 Logic diagram of IC 74293.

Table 11.2 Functional table for binary ripple counter, type number 74293.

RESET inputs		Outputs			
$R_0(1)$	$R_0(2)$	Q_D	Q_C	Q_B	Q_A
H	H	L	L	L	L
L	X		Count		
X	L		Count		

Table 11.3 Count sequence for binary ripple counter, type number 74293.

Count	Outputs			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

Solution

- The counter counts in the natural sequence from 0000 to 1011.
- The moment the counter goes to 1100, the NAND output goes to the logic '0' state and immediately clears the counter to the 0000 state.
- Thus, the counter is not able to stay in the 1100 state. It has only 12 stable states from 0000 to 1011.
- Therefore, the modulus of the counter = 12.
- The Q_3 output is the input clock frequency divided by 12.
- Therefore, the frequency of the Q_3 output waveform $= 1.2 \times 10^3/12 = 100$ kHz.

Example 11.4

Design a binary ripple counter that counts 000 and 111 and skips the remaining six states, that is, 001, 010, 011, 100, 101 and 110. Use presentable, clearable negative edge-triggered J-K flip-flops with active LOW PRESET and CLEAR inputs. Also, draw the timing waveforms and determine the frequency of different flip-flop outputs for a given clock frequency, f_c.

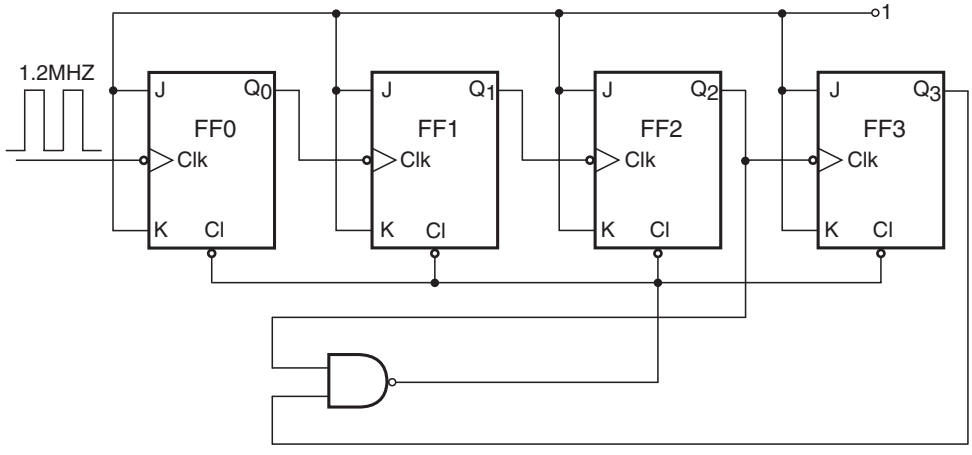


Figure 11.5 Example 11.3.

Solution

The counter is required to go to the 111 state from the 000 state with the first relevant clock transition. The second transition brings it back to the 000 state. That is, the three flip-flops toggle from logic '0' state to logic '1' state with every odd-numbered clock transition, and also the three flip-flops toggle from logic '1' state to logic '0' state with every even-numbered clock transition. Figure 11.6(a) shows the arrangement. The PRESET inputs of the three flip-flops have been tied to the NAND output whose inputs are $Q_A, \overline{Q_B}$ and $\overline{Q_C}$. Every time the counter is in the 000 state and is clocked, the NAND output momentarily goes from logic '1' state to logic '0' state, thus presetting the Q_A, Q_B and Q_C outputs to the logic '1' state. The timing waveforms as shown in Fig. 11.6(b) are self-explanatory.

The Q_A, $Q_B$and $Q_C$waveforms are identical, and each of them has a frequency of $f_c/2$, where f_c is the clock frequency.

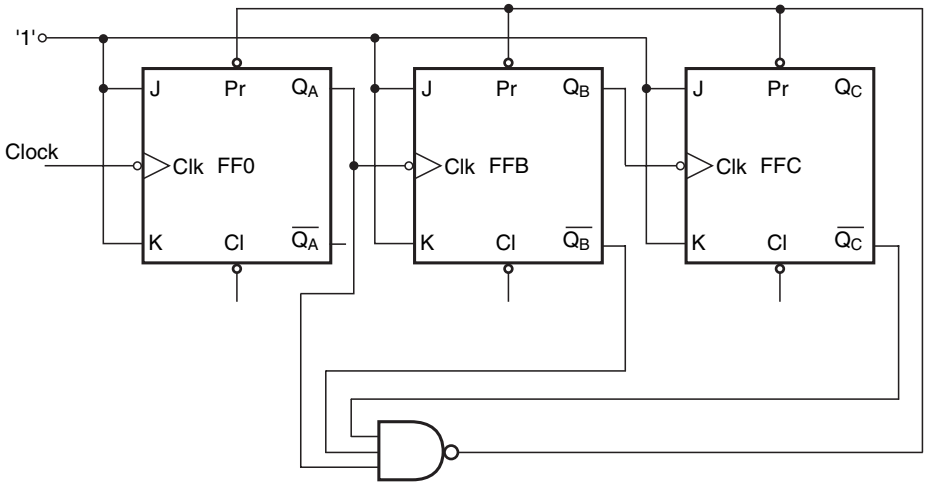
Example 11.5

Refer to the binary ripple counter arrangement of Fig. 11.7. Write its count sequence if it is initially in the 0000 state. Also draw the timing waveforms.

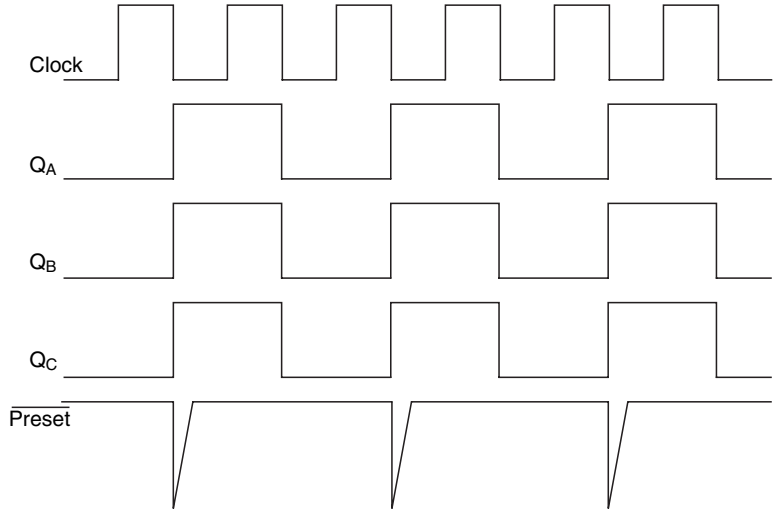
Solution

The counter is initially in the 0000 state. With the first clock pulse, Q_0 toggles from the '0' to the '1' state, which means $\overline{Q_0}$ toggles from '1' to '0'. Since $\overline{Q_0}$ here feeds the clock input of next flip-flop, flip-flop FF1 also toggles. Thus, Q_1 goes from '0' to '1'. Since flip-flops FF2 and FF3 are also clocked from complementary outputs of their immediately preceding flip-flops, they also toggle. Thus, the counter moves from the 0000 state to the 1111 state with the first clock pulse.

With the second clock pulse, Q_0 toggles again, but the other flip-flops remain unaffected for obvious reasons and the counter is in the 1110 state. With subsequent clock pulses, the counter keeps counting downwards by one LSB at a time until it reaches 0000 again, after which the process repeats. The count sequence is given as 0000, 1111, 1110, 1101,1100, 1011, 1010, 1001, 1000,



(a)



(b)

Figure 11.6 Example 11.4.

0111, 0110, 0101, 0100, 0011, 0010, 0001 and 0000. The timing waveforms are shown in Fig. 11.8. Thus, we have a four-bit counter that counts in the reverse sequence, beginning with the maximum count. This is a DOWN counter. This type of counter is discussed further in the subsequent paragraphs.

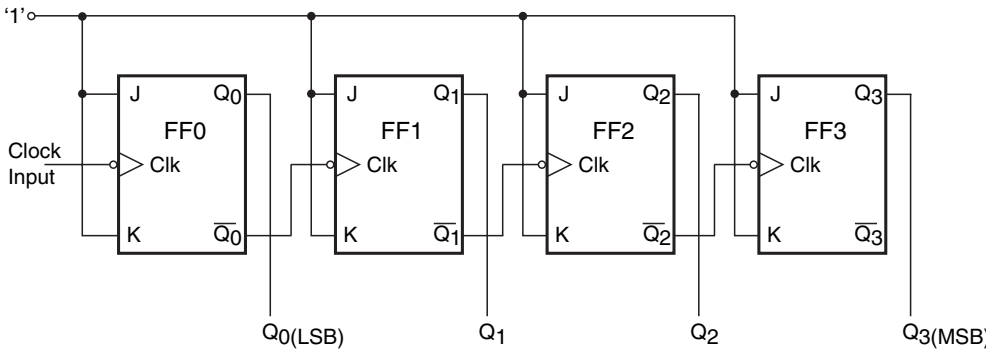


Figure 11.7 Counter schematic, example 11.5.

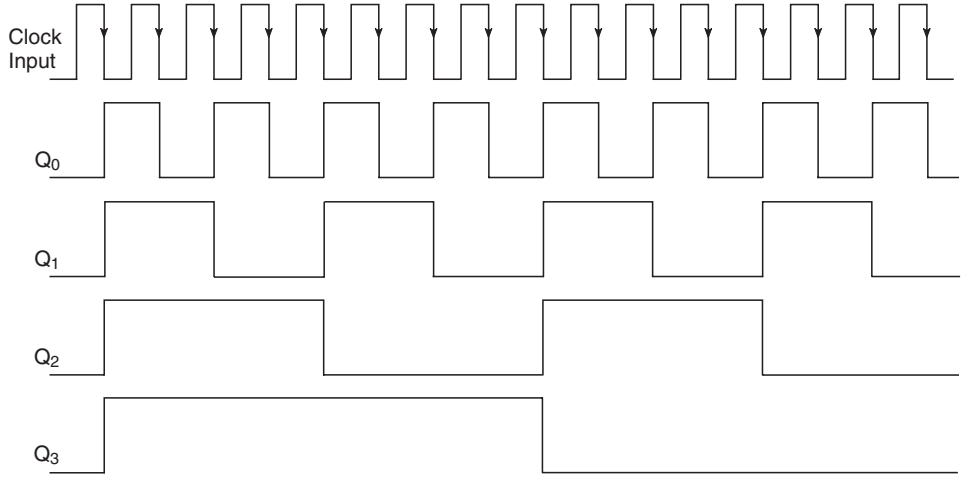


Figure 11.8 Timing waveforms, example 11.5.

From what we have discussed for a binary ripple counter, including the solved examples given to supplement the text, we can make the following observations:

1. If the flip-flops used to construct the counter are negative (HIGH-to-LOW) edge triggered and the clock inputs are fed from Q outputs, the counter counts in the normal upward count sequence.
2. If the flip-flops used to construct the counter are negative edge triggered and the clock inputs are fed from $\overline{Q}$ outputs, the counter counts in the reverse or downward count sequence.
3. If the flip-flops used to construct the counter are positive (LOW-to-HIGH) edge triggered and the clock inputs are fed from Q outputs, the counter counts in the reverse or downward count sequence.
4. If the flip-flops used to construct the counter are positive edge triggered and the clock inputs are fed from the $\overline{Q}$ outputs, the counter counts in the normal upward count sequence.