If the T input is in 0 state (*i.e.*, J = K = 0) prior to a clock pulse, the Q output will not change with the clock pulse. On the other hand, if the T input is in 1 state (*i.e.*, J = K = 1) prior to a clock pulse, the Q output will change to Q' with the clock pulse. In other words, we may say that, if T = 1 and the device is clocked, then the output toggles its state.

The truth table shows that when T = 0, then Q*n*+1 = Q*n*, *i.e.*, the next state is the same as the present state and no change occurs. When T = 1, then Q*n*+1 = Q'*n*, *i.e.*, the state of the flip-flop is complemented. The circuit diagram of a T flip-flop is shown in Figure 2.4(f) and the block diagram of the flip-flop is shown in Figure 7.41.

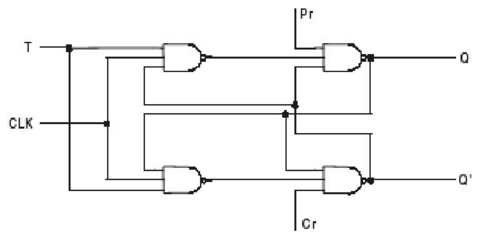


Figure 2.4(f): A T flip flop

The T flip-flop has one input, *T* (which stands for toggle), in addition to the clock. When *T* is asserted (*T* = 1), the flip-flop state toggles back and forth at each active edge of the clock, and when *T* is de-asserted, the flip-flop keeps its current state. The characteristic table, characteristic equation, state diagram, circuit, logic symbol, and excitation table for the T flip-flop are shown in Figure 2.3(f).

4.0 Conclusion

When latches are used for the memory elements in sequential circuits, a serious difficulty arises. Recall that latches have the property of immediate output responses (i.e., transparency). Because of this the output of a latch cannot be applied directly (or through logic) to the input of the same or another latch when all the latches are triggered by a common clock source. Flip-flops are used to overcome this difficulty.

5.0 Summary

In this unit, you learnt about:

- J-K Flip-Flop
- Master-Slave J-K Flip-flop
- T Flip-Flop

6.0 Tutor Marked Assignment

1. What is the difference between a latch and a flip-flop?
2. Draw the circuit diagram for the J-K flip-flop with enable.
3. Draw the circuit diagram for the T flip-flop with enable.
4. *True or False:* The slave will respond to changes in *J* and *K* while *CLK* = 0.

7.0 Further Reading and Other Resources

1. Ronald J. Tocci (1988). “Digital Systems: Priciples and Applications”, 4th Edition Prentice-Hall International edition.
2. http://en.wikipedia.org/wiki/Logic_gate
3. http://www.discovercircuits.com/D/digital.htm
4. http://www.encyclopedia.com/doc/1G1-168332407.html
5. http://www.logiccircuit.org/

MODULE 3 - Counters

UNIT 1: Introduction to Counters

Contents **Pages**

1.0 Introduction

Counters are one of the simplest types of sequential networks. A counter is usually constructed from one or more flip-flops that change state in a prescribed sequence when input pulses are received. A counter driven by a clock can be used to count the number of clock cycles. Since the clock pulses occur at known intervals, the counter can be used as an instrument for measuring time and therefore period of frequency.

2.0 Objectives

Upon completion of this unit, you will be able to:

- Understand Counters and types of counters
- Discuss the Binary Up Counter
- Discuss the Binary Up-down counter
- Understand the BCD Counter
- Discuss the BCD Up-down counter

3.0 TYPES OF COUNTERS

Counters, as the name suggests, are for counting a sequence of values. However, there are many different types of counters depending on the total number of count values, the sequence of values that it outputs, whether it counts up or down, and so on. The simplest is a modulo-*n* counter that counts the decimal sequence 0, 1, 2, …, up to *n*-1 and back to 0. Some typical counters are described next.

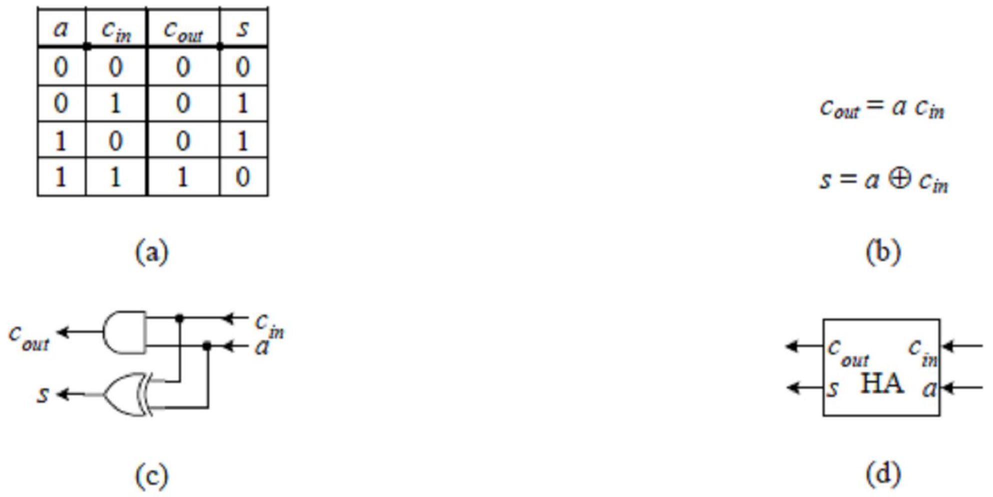
- **Modulo-*n* counter**: Counts from decimal 0 to *n* – 1 and back to 0. For example, a modulo-5 counter sequence in decimal is 0, 1, 2, 3, and 4.
- **Binary coded decimal (BCD) counter**: Just like a modulo-*n* counter, except that *n* is fixed at 10. Thus, the sequence is always from 0 to 9.
- ***n*-bit binary counter**: Similar to modulo-*n* counter, but the range is from 0 to 2*n* – 1 and back to 0, where *n* is the number of bits used in the counter. For example, a 3-bit binary counter sequence in decimal is 0, 1, 2, 3,4, 5, 6, and 7.
- **Gray-code counter**: The sequence is coded so that any two consecutive values must differ in only one bit. For example, one possible 3-bit gray-code counter sequence is 000, 001, 011, 010, 110, 111, 101, and 100.
- **Ring counter**: The sequence starts with a string of 0 bits followed by one 1 bit, as in 0001. This counter simply rotates the bits to the left on each count. For example, a 4-bit ring counter sequence is 0001, 0010, 0100, 1000, and back to 0001.

We will now look at the design of several counters.

3.1 Binary Up Counter

An *n*-bit binary counter can be constructed using a modified *n*-bit register where the data inputs for the register come from an incrementer (adder) for an up counter, and a decrementer

(subtractor) for a down counter. To get to the next up-count sequence from the value that is stored in a register, we simply have to add a 1 to it. The full adder adds two operands plus the carry. But what we want is just to add a 1, so the second operand to the full adder is always a 1. Since the 1 can also be added in via the carry-in signal of the adder, we really do not need the second operand input.
This modified adder that only adds one operand with the carry-in is called a **half adder** (HA). Its truth table is shown in (a) of Figure 3.1(a). We have *a* as the only input operand, c_{in} and c_{out} are the carry-in and carry-out signals, respectively, and *s* is the sum of the addition. In the truth table, we are simply adding *a* plus c_{in} to give the sum *s* and possibly a carry-out, c_{out}. From the truth table, we obtain the two equations for c_{out} and *s* shown in (b) of Figure 3.1(a). The HA circuit is shown in (c) of Figure 3.1(a) and its logic symbol in (d).



a	c_{in}	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a)

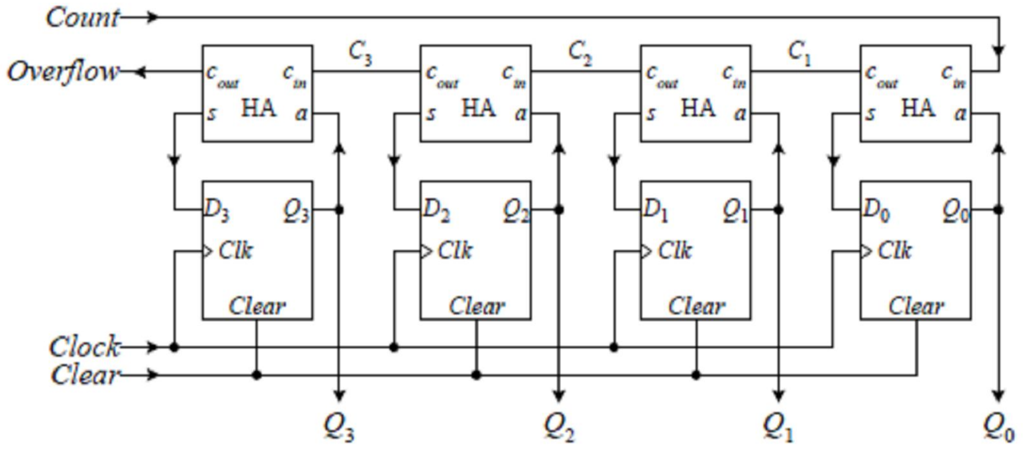
$$c_{out} = a\, c_{in}$$

$$s = a \oplus c_{in}$$

(b)

Figure 3.1(a): Half adder: (a) truth table; (b) equations; (c) circuit; (d) logic symbol.

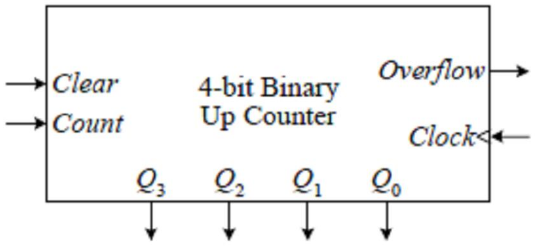
Several half adders can be daisy-chained together, just like with the full adders to form an *n*-bit adder. The single operand input *a* comes from the register. The initial carry-in signal, *c*0, is used as the count enable signal, since a 1 on *c*0 will result in incrementing a 1 to the register value, and a 0 will not. The resulting 4-bit binary up counter circuit is shown in (a) of Figure 3.1(b), along with its operation table and logic symbol in (b) and (c). As long as *Count* is asserted, the counter will increment by 1 on each clock pulse until *Count* is de-asserted. When the count reaches 2*n* – 1 (which is equivalent to the binary number with all 1's) the next count will revert back to 0, because adding a 1 to a binary number with all 1's will result in an overflow on the *Overflow* bit, and all of the original bits will reset to 0. The *Clear* signal allows an asynchronous reset of the counter to 0.



(a)

Clear	*Count*	Operation
1	×	Reset counter to zero
0	0	No change
0	1	Count up

(b)



(c)

Figure 3.1(b): A 4-bit binary up counter with asynchronous clear: (a) circuit; (b) operation table; (c) logic symbol.

3.2 Binary Up-down Counter

We can design an *n*-bit binary up-down counter just like the up counter, except that we need both an adder and a subtractor for the data input to the register. The **half adder-subtractor** (HAS) truth table is shown in (a) of Figure 3.1(c).

The *Down* signal is to select whether we want to count up or down. Asserting *Down* (setting to 1) will count down. The top half of the table is exactly the same as the HA truth table. For the bottom half, we are performing a subtraction of *a* – *cin*, where *s* is the difference of the subtraction, and *cout* is a 1 if we need to borrow. For example, for 0 – 1, we need to borrow, so *cout* is a 1. When we borrow, we get a 2; and 2 – 1 = 1, so *s* is also a 1. The two resulting equations for *cout* and *s* are shown in (b) of Figure 3.1(c). The circuit and logic symbol for the half adder subtractor are shown in (c) and (d) of Figure 3.1(c).

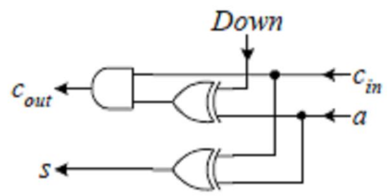
Down	*a*	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

(a)

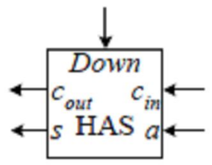
$$c_{out} = Down'\, a\, c_{in} + Down\, a'\, c_{in} = (Down \oplus a)\, c_{in}$$

$$s = Down'\,(a \oplus c_{in}) + Down\,(a \oplus c_{in}) = a \oplus c_{in}$$

(b)



(c)



(d)

Figure 3.1(c): Half Adder-Subtractor (HAS): (a) truth table; (b) equations; (c) circuit; (d) logic symbol.

We can simply replace the HAs with the HASs in the up counter circuit to get the up-down counter circuit, as shown in Figure 8.15(a). Its operation table and logic symbol are shown in Figure 8.15(b) and (c). Again, the *Overflow* signal is asserted each time the counter rolls over from 1111 back to 0000.

3.3 BCD Up Counter

A limitation with the binary up-down counter with parallel load is that it always counts up to 2*n* – 1 for an *n* bit register and then cycles back to zero. If we want the count sequence to end at a number less than 2*n* – 1, we need to use an equality comparator to test for this new ending number. The comparator compares the current count value that is in the register with this new ending number. When the counter reaches this new ending number, the comparator asserts its output.

The counter can start from a number that is initially loaded in. However, if we want the count sequence to cycle back to this new starting number each time, we need to assert the *Load* signal at the end of each count sequence and reload this new starting number. The output of the comparator is connected to the *Load* line, so that when the counter reaches the ending number, it will assert the *Load* line and loads in the starting number. Hence, the counter can end at a new ending number and cycles back to a new starting number.

The binary coded decimal (BCD) up counter counts from 0 to 9 and then cycles back to 0. The circuit for it is shown in Figure 3.1(d). The heart of the circuit is just the 4-bit binary up-down counter with parallel load. A 4-input AND gate is used to compare the count value with the number 9. When the count value is 9, the AND gate comparator outputs a 1 to assert the *Load* line. Once the *Load* line is asserted, the next counter value will be the value loaded in from the counter input *D*. Since *D* is connected to all 0's, the counter will cycle back to 0 at the next rising clock edge. The *Down* line is connected to a 0, since we only want to count up.

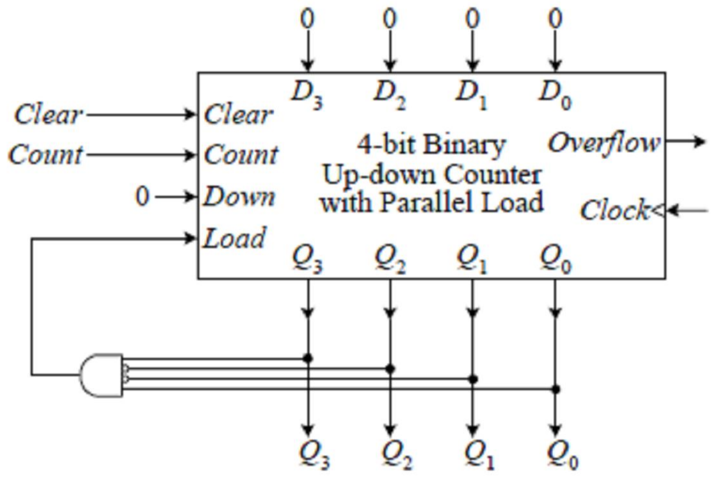


Figure 3.1(d): BCD up counter.

In order for the timing of each count to be the same, we must use the *Load* operation to load in the value 0, rather than using the *Clear* operation. If we connect the output of the AND gate to the *Clear* input instead of the *Load* input, we will still get the correct count sequence. However, when the count reaches 9, it will change to a 0 almost immediately, because when the output of the AND gate asserts the asynchronous *Clear* signal, the counter is reset to 0 right away and not at the next rising clock edge.

Example 3.1(a): Constructing an up counter circuit
This example uses the 4-bit binary up-down counter with parallel load to construct an up counter circuit that counts from 3 to 8 (in decimal), and back to 3.
The circuit for this counter, shown in Figure 3.1(e), is almost identical to the BCD up counter circuit. The only difference is that we need to test for the number 8 instead of 9 as the last number in the sequence, and the first number to load in is a 3 instead of a 0. Hence, the inputs to the AND gate for comparing with the binary counter output is 1000, and the number for loading in is 0011.

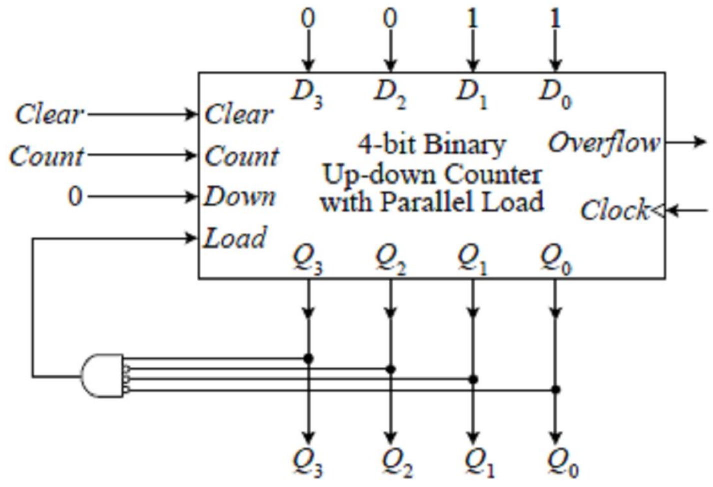


Figure 3.1(e): Counter for Example 3.1(a)

3.4 BCD Up-down Counter

We can get a BCD up-down counter by modifying the BCD up counter circuit slightly. The counter counts from 0 to 9 for the up sequence and 9 down to 0 for the down sequence. For the up sequence, when the count reaches 9, the *Load* line is asserted to load in a 0 (0000 in binary). For the down sequence, when the count reaches 0, the *Load* line is asserted to load in a 9 (1001 in binary).

The BCD up-down counter circuit is shown in Figure 3.1(f). Two 5-input AND gates acting as comparators are used. The one labeled “Up” will output a 1 when *Down* is de-asserted (i.e., counting up), and the count is 9. The one label “Dn” will output a 1 when *Down* is asserted, and the count is 0. The *Load* signal is asserted by either one of these two AND gates. Four 2-to-1 multiplexers are used to select which of the two starting values, 0000 or 1001, is to be loaded in when the *Load* line is asserted. The select lines for these four multiplexers are connected in common to the *Down* signal, so that when the counter is counting up, 0000 is loaded in when the counter wraps around, and 1001 is loaded in when the counter wraps around while counting down. It should be obvious that the two values, 0000 and 1001, can also be loaded in without the use of the four multiplexers.

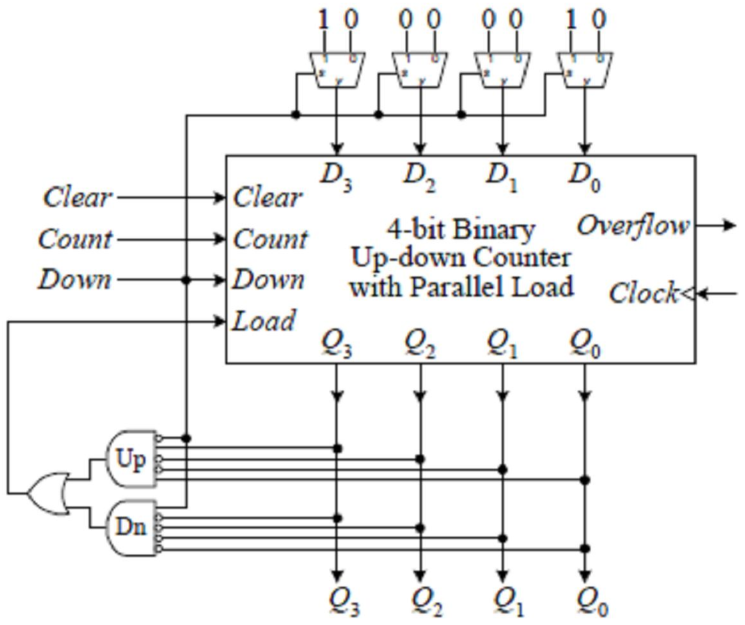


Figure 3.1(f): BCD up-down counter.

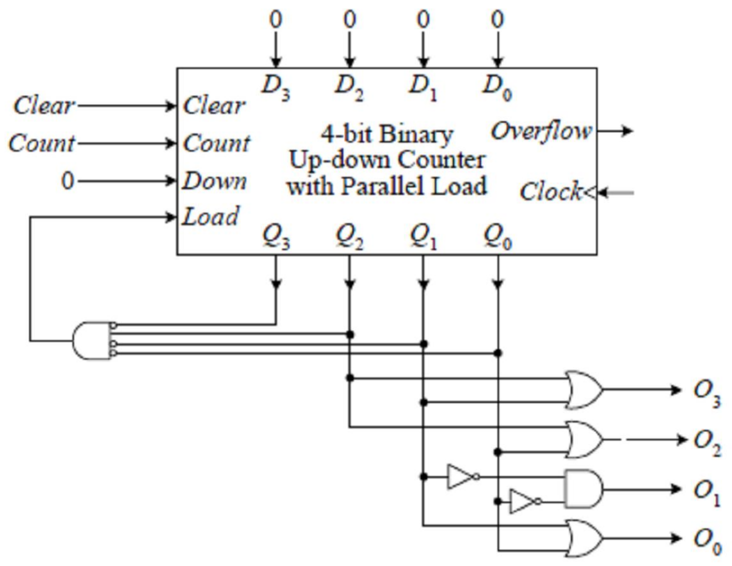
Example 3.1(b): Constructing an up-down counter circuit
This example uses the 4-bit binary up-down counter with parallel load to construct an up-down counter circuit that outputs the sequence, 2, 5, 9, 13, and 14, repeatedly.
The 4-bit binary counter can only count numbers consecutively. In order to output numbers that are not consecutive, we need to design an output circuit that maps from one number to another number. The required sequence has five numbers, so we will first design a counter to count from 0 to 4. The output circuit will then map the numbers, 0, 1, 2, 3, and 4 to the required output numbers, 2, 5, 9, 13, and 14, respectively.
The inputs to the output circuit are the four output bits of the counter, Q_3, Q_2, Q_1, and Q_0. The outputs from this circuit are the modified four bits, O_3, O_2, O_1, and O_0, for representing the five output numbers. The truth table and the resulting output equations for the output circuit are shown in (a) and (b) of Figure 3.1(g), respectively. The easiest way to see how the output equations are obtained is to use a K-map and put in all of the don’t-cares. The complete counter circuit is shown in (c) of Figure 3.1(g).

Decimal Input	Q_3	Q_2	Q_1	Q_0	Decimal Output	O_3	O_2	O_1	O_0
0	0	0	0	0	2	0	0	1	0
1	0	0	0	1	5	0	1	0	1
2	0	0	1	0	9	1	0	0	1
3	0	0	1	1	13	1	1	0	1
4	0	1	0	0	14	1	1	1	0
Rest of the Combinations						×	×	×	×

(a)

$$O_0 = Q_1 + Q_0$$
$$O_1 = Q_1'Q_0'$$
$$O_2 = Q_2 + Q_0$$
$$O_3 = Q_2 + Q_1$$

(b)



(c)

Figure 3.1(g): Counter for Example 3.1

4.0 Conclusion

Counter is a fundamental part of most digital logic applications. It is used in timing units, control circuits, signal generators and numerous other devices. Down counters are not as widely used as up counters. Their major application is in situations where it must be known when a desired number of input pulses has occurred. In these situations, the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses has occurred.

5.0 Summary

In this unit, you learnt about:

- Counters and types of counters including:
  - Binary Up Counter

- Binary Up-down Counter
- BCD Up Counter
- BCD Up-down Counter

6.0 Tutor Marked Assignment

1) How can you convert an up-counter into a down-counter?
2) Which flip-flop is best suited for designing a counter and why?
3) What is meant when we say that a counter is presettable?
4) Design a 4-bit counter with one control signals S. The counter should operate as
   (a) Binary down-counter when S = 0,
   (b) Binary up-counter when S = 1.

7.0 Further Reading and Other Resources

1. Ronald J. Tocci (1988). “Digital Systems: Priciples and Applications”, 4th Edition Prentice-Hall International edition.
2. http://en.wikipedia.org/wiki/Logic_gate
3. http://www.discovercircuits.com/D/digital.htm
4. http://www.encyclopedia.com/doc/1G1-168332407.html
5. http://www.logiccircuit.org/

MODULE 3 - Counters

UNIT 2: Asynchronous Counter

Contents **Pages**

1.0 Introduction

Asynchronous or *ripple counters* are counter circuits made from cascaded J-K flip-flops where each clock input receives its pulses from the output of the previous flip-flop invariably exhibit a *ripple effect*, where false output counts are generated between some steps of the count sequence.

2.0 Objectives

Upon completion of this unit, you will be able to:

- Understand Asynchronous Counters

3.0 About Asynchronous (Serial or Ripple) Counters

The simplest counter circuit can be built using T flip-flops because the toggle feature is naturally suited for the implementation of the counting operation. J-K flip-flops can also be used with the *toggle* property in hand. Other flip-flops like D or S-R can also be used, but they may lead to more complex designs.

In this counter all the flip-flops are not driven by the same clock pulse. Here, the clock pulse is applied to the first flip-flop; *i.e.*, the least significant bit state of the counter and the successive flip-flop is triggered by the output of the previous flip-flop. Hence the counter has cumulative settling time, which limits its speed of operation. The first stage of the counter changes its state first with the application of the clock pulse to the flip-flop and the successive flip-flops change their states in turn causing a *ripple-through* effect of the clock pluses. As the signal propagates through the counter in a *ripple* fashion, it is called a *ripple counter*.

3.1 Asynchronous (or Ripple) Up-counter

Figure 3.2(a) shows a 3-bit counter capable of counting from 0 to 7. The clock inputs of the three flip-flops are connected in cascade. The T input of each flip-flop is connected to a constant 1, which means that the state of the flip-flop will toggle (reverse) at each negative edge of its clock. We are assuming that the purpose of this circuit is to count the number of pulses that occur on the primary input called CLK (Clock). Thus, the clock input of the first flip-flop is connected to the *Clock* line. The other two flip-flops have their clock inputs driven by the Q output of the preceding flip-flop. Therefore, they toggle their state whenever the preceding flip-flop changes its state from Q = 1 to Q = 0, which results in a negative edge of the Q signal.

Figure 3.2(*b*) shows a timing diagram for the counter. The value of Q0 toggles once each clock cycle. The change takes place shortly after the negative edge of the *Clock* signal.

The delay is caused by the propagation delay through the flip-flop. Since the second flip-flop is clocked by Q_0, the value of Q_1 changes shortly after the negative edge of the Q0 signal.

Similarly, the value of Q_2 changes shortly after the negative edge of the Q_1 signal. If we look at the values Q_2 Q_1 Q_0 as the count, then the timing diagram indicates that the counting sequence is 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, and so on. This circuit is a modulo-8 counter. Since it counts in the upward direction, we call the circuit an *up-counter*.

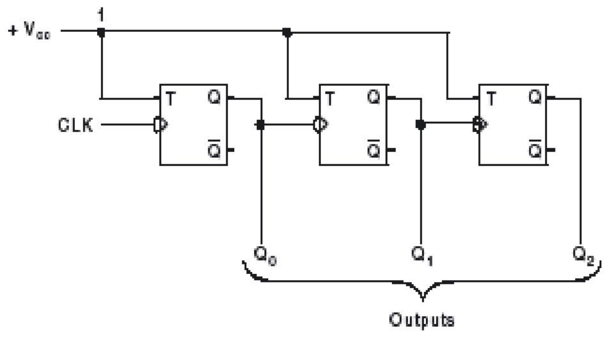


Figure 3.2(a): Logic circuit diagram of a 3-bit asynchronous up-counter.

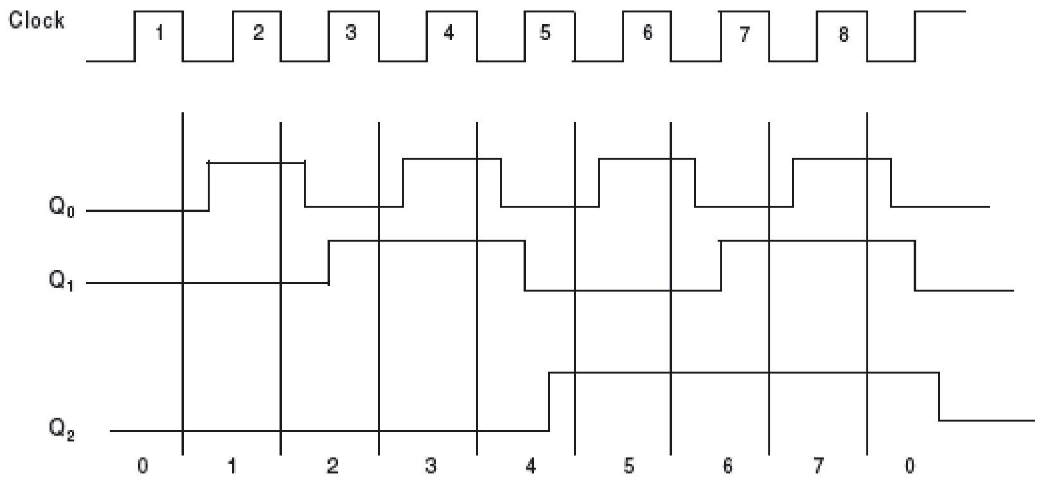


Figure 3.2(b): Timing diagram

The counter in Figure 3.2(a) has three stages, each comprising of a single flip-flop. Only the first stage responds directly to the Clock signal. Hence we may say that this stage is synchronized to the clock. The other two stages respond after an additional delay. For example, when count = 3, the next clock pulse will change the count to 4. Now this change requires all three flip-flops to toggle their states. The change in Q_0 is observed only after a propagation delay from the negative edge of the clock pulse. The Q_1 and Q_2 flip-flops have not changed their states yet. Hence, for a brief period, the count will be $Q_2Q_1Q_0 = 010$.
The change in Q1 appears after a second propagation delay, and at that point the count is $Q_2Q_1Q_0 = 000$. Finally, the change in Q_2 occurs after a third delay, and hence the stable state of the circuit is reached and the count is $Q_2Q_1Q_0 = 100$.
Table 3.2(a) shows the sequence of binary states that the flip-flops will follow as clock pulses are applied continuously. An *n*-bit binary counter repeats the counting sequence for every 2*n* (*n* = number of flip-flops) clock pulses and has discrete states from 0 to 2^n-1.

Table 3.2(a): Count sequence of a 3-bit binary ripple up-counter

Counter State	Q_2	Q_1	Q_0
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

Figure 3.2(c) shows the 3-bit binary ripple counter with decoded outputs. It consists of the same circuit as shown in Figure 3.2(a) with additional decoding circuitry. In decoding the states of a ripple counter, pulses of one clock duration will occur at the decoding gate outputs as the flip-flops change their state when the counter content is equal to the given state. For example, a decoding gate Q_6 connected in the circuit will decode state 6 (*i.e.*, $Q_AQ_BQ_C = 110$). Thus the gate output will be high only when $Q_A = 1$, $Q_B = 1$, and $Q_C = 0$. The remaining seven states of the 3-bit counter can be decoded in a similar manner using AND gates as Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, and Q_7.

Now, theoretically each decoding output will be high only when the counter content is equal to a given state, and this state occurs only once during a cycle of 2*n* states of the counter, where *n* is the number of flip-flops in the counter. But practically in an asynchronous counter, the decoding gate produces a high output more than once during the cycle of 2*n* states. Such undesired high or low pulses of short duration, that appear at the decoding gate output at undesired time instants are known as *spikes* or *glitches*. The reason for these spikes is the cumulative propagation delay in the synchronous counter.

As TTL circuits are very fast, they will respond to even glitches of very small duration (a few nanoseconds). Therefore, these glitches should be eliminated. These can be eliminated by using any one of the following methods: (*i*) clock input to strobe the decoding gates, or (*ii*) using synchronous counters.

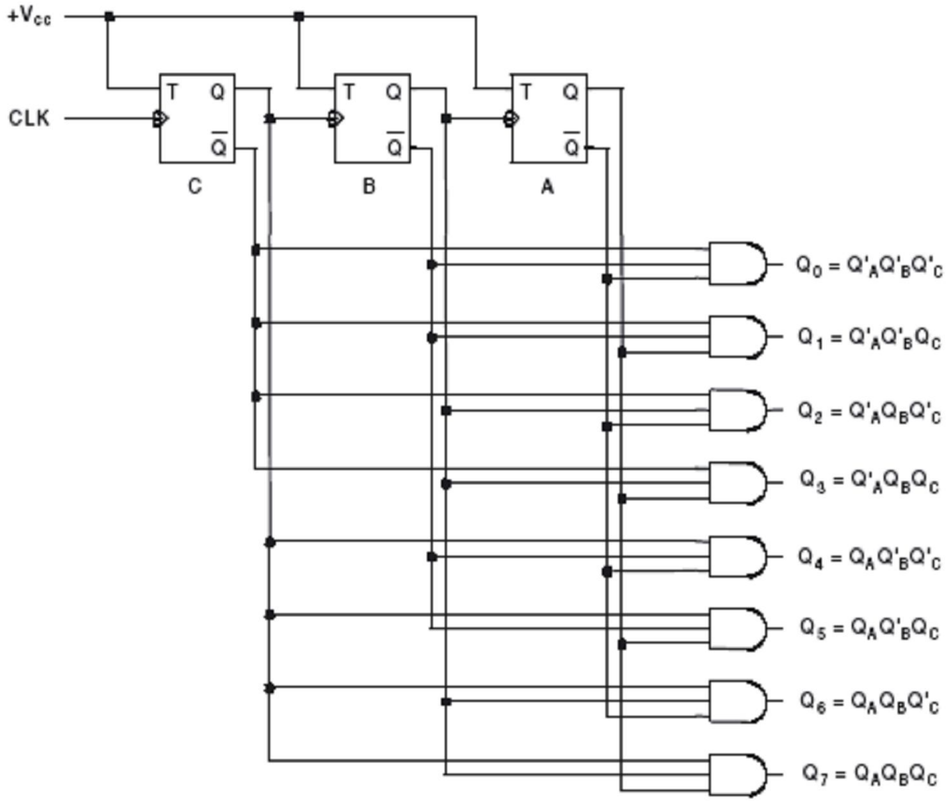


Figure 3.2(c): 3-bit binary asynchronous counter with decoded outputs

3.2 Asynchronous (or Ripple) Down-counter

Figure 3.2(a) is an up-counter because it counts upward from zero. It is a relatively simple matter to construct asynchronous down counters, which will count downward fro a maximum count to zero. Before looking at a ripple down counter, let us examine the count-down sequence for a 3-bit down counter:

	CBA	CBA	CBA
(7)	1 1 1	1 1 1	1 1 1
(6)	1 1 0	1 1 0	1 1 0
(5)	1 0 1	1 0 1	. . .
(4)	1 0 0	1 0 0	. . .
(3)	0 1 1	0 1 1	etc
(2)	0 1 0	0 1 0	etc
(1)	0 0 1	0 0 1	
(0)	0 0 0	0 0 0	etc.

A, *B,* and *C* represent the flip-flop output states as the counter goes through its sequence.

It can be seen that the A flip-flop (LSB) changes states (toggles) at each in the sequence just as it does in the up counter. The B flip-flop changes states each time *A* goes from LOW to HIGH; *C* changes states each time *B* goes from LOW to HIGH. Thus, in a down counter each flip-flop, except the first, must toggle when the preceding flip-flop goes from LOW to HIGH. If the flip-flops have *CLK* inputs that respond to negative transitions (HIGH to LOW), then an inverter can be placed in front of each *CLK* input; however, the same effect can be accomplished by driving each flip-flop *CLK* input from the inverted output of the preceding flip-flop. This is illustrated in Figure 3.2(d) for a MOD-8 down counter.

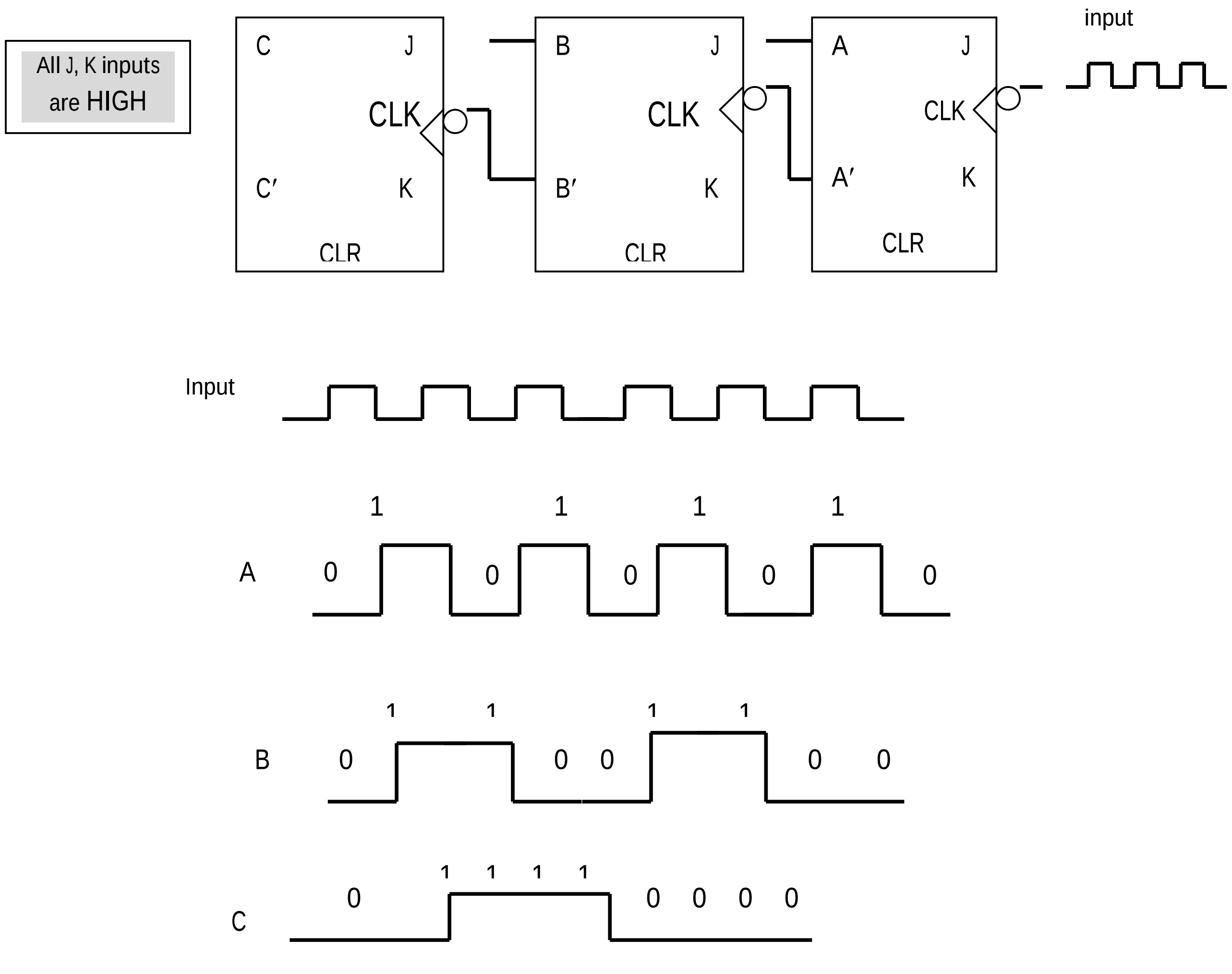


Figure 3.2(d): MOD-8 down counter

The input pulses are applied to the A flip-flop; the A′ output serves as the *CLK* input for the B flip-flop; the B′ output serves as the *CLK* input for the *C* flip-flop. The waveforms at A, B, and C show that B toggles whenever B goes LOW to HIGH. This results in the desired down-counting sequence at the *C*, *B,* and *A* outputs.

3.3 MOD Number

The counter in Figure 3.2(c) has 16 distinctly different states (0000 through 1111). Thus, it is a MOD-16 ripple counter. Recall that the MOD number is always equal to the number of states which the counter goes through in each complete cycle before it recycles back to its starting state. The MOD number can be increased simply by adding more Flip-Flops to the counter. That is, MOD number = 2^N where N is the number of Flip-Flops connected in the arrangement.

Example 3.1(a): A counter is needed that will count the number of items passing on a conveyor belt. A photocell and light source combination is used to generate a single pulse each time an item crosses its path. The counter has to be able to count as many as one thousand items. How many flip-flops are required?

Solution: It is a simple matter to determine what value of N is needed so that $2^N \geq 1000$. Since $2^9 = 512$, 9 Flip-Flops will not be enough. $2^{10} = 1024$, so 10 flip-flops would produce a counter that could count as high as $1111111111_2 = 1023_{10}$. Therefore, we should use 10 flip-flops. We could use more than 10, but it would be a waste of flip-flops, since any flip-flops past the 10th one will never be toggled.

3.3.1 Counters with MOD numbers < 2^N

The basic ripple counter of Figure 3.2(a) is limited to MOD numbers that are equal to 2^N, where N is the number of flip-flops. This value is actually the maximum MOD number that can be obtained using N flip-flops. The basic counter can be modified to produce MOD numbers less than 2^N by allowing the counter to *skip states* that are normally part of the counting sequence. One of the most common methods for doing this is illustrated in Figure 3.2(e) where a 3-bit ripple counter is shown.

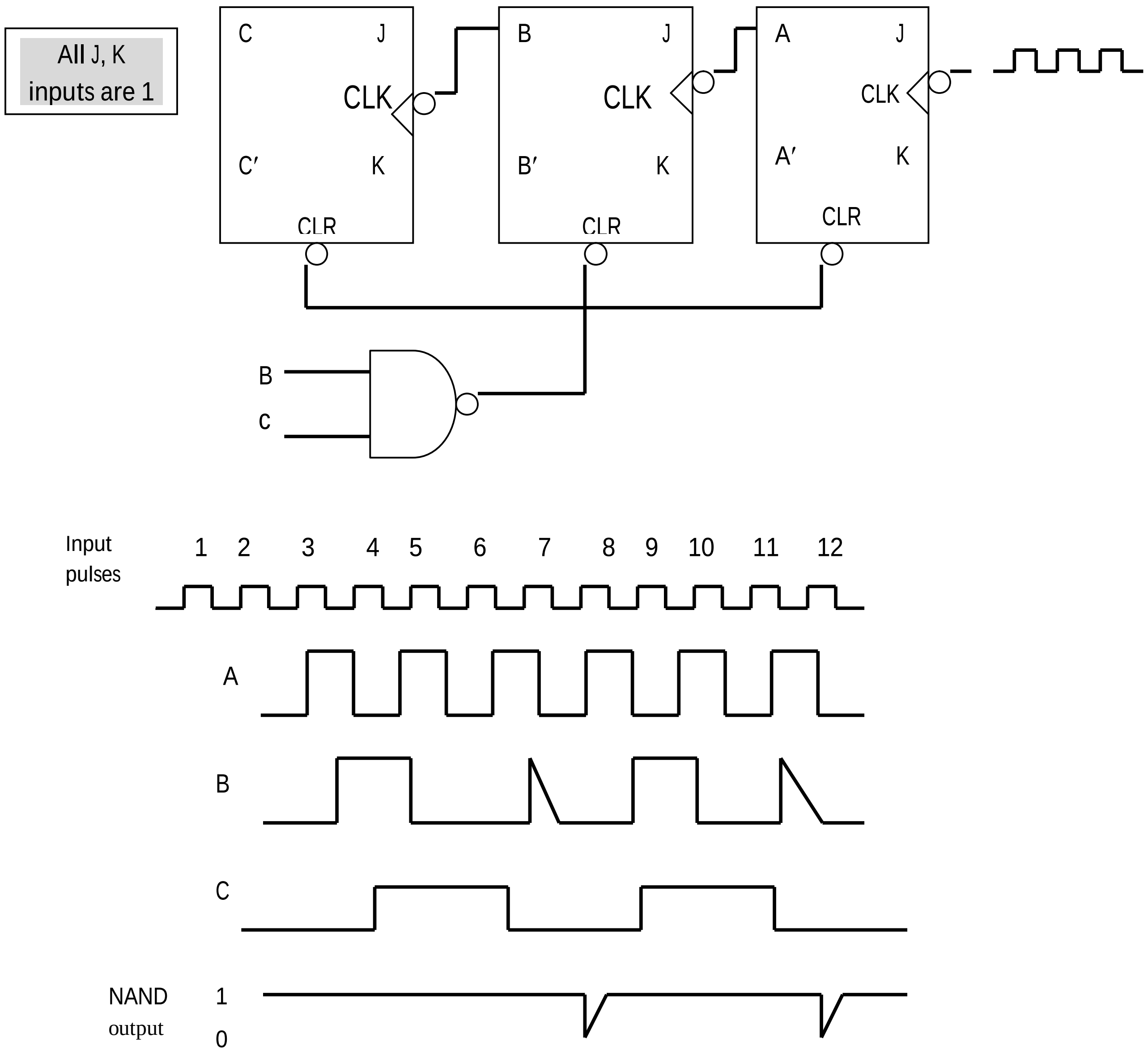


Figure 3.2(e): MOD-6 produced by clearing s MOD-8 counter when count of six (110) occurs

Disregarding the NAND gate for a moment we can see that the counter is a MOD-8 binary counter which will count in sequence from 000 to 111. However, the presence of the NAND gate will alter this sequence as follows:

1. The NAND output is connected to the asynchronous CLEAR inputs of each Flip-flop. As long as the NAND output is HIGH, it will have no effect on the counter. When it goes LOW, however, it will clear all the flip-flops so that the counter immediately goes to the 000 state.

2. The inputs to the NAND gate are the outputs of the *B* and *C* flip-flops, so the NAND output will go LOW whenever *B* = *C* = 1. This condition will occur when the counter goes from the 101 state to the 110 state (input pulse 6 on waveforms). The LOW at the NAND output will immediately (generally within a few nanoseconds) clear the counter to the 000 state. Once the flip-flops have been cleared, the NAND output goes back HIGH, since the *B* = *C* = 1 condition no longer exists.

3. The counting sequence is therefore,

C	B	A	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	(temporary state needed to clear counter)

Although the counter does go to the 110 state, it remains there for only a few nanoseconds before it recycles to 000. Thus, we can essentially say that this counter counts from 000 (zero) to 101 (five) and then recycles to 000. It essentially skips 110 and 111 so that it only goes through six different states; thus, it is a MOD-6 counter.

3.3.2 Changing the MOD number

The counter of Figure 3.2(d) is a MOD-6 because of the choice of inputs to the NAND gate. Any desired MOD number can be obtained by changing these inputs. For example, using a three-input NAND gate with inputs *A*, *B*, and *C*, the counter would function normally until the 111 condition was reached, at which point it would immediately reset to the 000 state. Ignoring the temporary excursion into the 111 state, the counter would go from 000 through 110 and then recycle back to 000, resulting in a MOD-7 counter (seven states).

Self Assessment Exercise:

1. How many Flip-flops are required for a counter that will count 0 to 255_{10}?

2. What is the MOD number of this counter?

3. What is the difference between the counting sequence of an up counter and a down counter?

4.0 Conclusion

The ripple or asynchronous counter is simple and straightforward in operation and construction and usually requires a minimum amount of hardware. In asynchronous counters, each flip-flop is triggered by the previous flip-flop, and hence the speed of operation is limited. However, the asynchronous counter highest operating frequency is limited because of ripple action. This problem can be overcome, if all flip-flops are clocked synchronously. The resulting circuit is known as a synchronous counter. Down counters are not as widely used as up counters. Their major application is in situations where it must be known when a desired number of pulses has occurred. In these situations the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses has occurred.

5.0 Summary

In this unit, you learnt about:

- Asynchronous (Serial or Ripple) Counters
- How NAND gate can be used to construct counters

6.0 Tutor Marked Assignment

1) Construct a MOD-10 counter that will count from 0000 through 1001 (decimal 9).
2) Compare between a ripple and a synchronous counter.
3) Discuss the procedure for designing synchronous counters.
4) In an asynchronous counter, all Flip-Flops change states at the same time.

7.0 Further Reading and Other Resources

1. Ronald J. Tocci (1988). "Digital Systems: Priciples and Applications", 4th Edition Prentice-Hall International edition.
2. http://en.wikipedia.org/wiki/Logic_gate
3. http://www.discovercircuits.com/D/digital.htm
4. http://www.encyclopedia.com/doc/1G1-168332407.html
5. http://www.logiccircuit.org/

MODULE 3 - Counters

UNIT 3: Synchronous Counter

Contents **Pages**

1.0 Introduction

A *synchronous counter*, in contrast to an *asynchronous counter,* is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time.

2.0 Objectives

Upon completion of this unit, you will be able to:

- Understand Synchronous Counter with Ripple Carry
- Understand Synchronous Down-Counter
- Understand Synchronous Up-Down Counter
- Application of Counters

3.0 Why Synchronous (Parallel) Counters?

The problems encountered with ripple counters are caused by the accumulated Flip-flop propagation delays; stated another way, the Flip-flops do not all change states simultaneously in synchronism with the input pulses. These limitations can be overcome with the use of synchronous or parallel counters in which all the Flip-flops are triggered simultaneously (in parallel) by the clock input pulses. Since the input pulses are applied to all the Flip-flops, some means must be used to control when a Flip-flop is to toggle and when it is to remain unaffected by a clock pulse. This is accomplished by using the *J* and *K* inputs.

3.1 Synchronous (Parallel) Counters

The ripple or asynchronous counter is the simplest to build, but its highest operating frequency is limited because of ripple action. Each flip-flop has a delay time. In ripple counters these delay times are additive and the total "settling" time for the counter is approximately the product of the delay time of a single flip-flop and the total number of flip-flops. Again, there is the possibility of glitches occurring at the output of decoding gates used with a ripple counter.

Both of these problems can be overcome, if all the flip-flops are clocked synchronously. The resulting circuit is known as a *synchronous counter*. Synchronous counters can be designed for any count sequence (need not be straight binary). These can be designed following a systematic approach. Before we discuss the formal method of design for such counters, we shall consider an intuitive method.

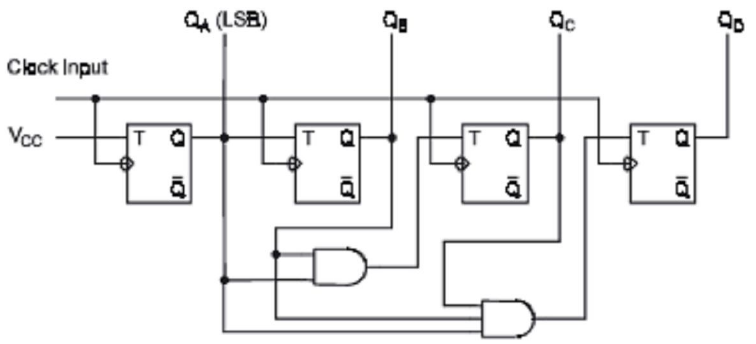


Figure 3.3(a): A 4-bit (MOD-16) synchronous counter.

A 4-bit synchronous counter with parallel carry is shown in Figure 3.3(a). In this circuit the clock inputs of all the flip-flops are tied together so that the input clock signal may be applied simultaneously to each flip-flop. Only the LSB flip-flop A has its T input connected permanently to logic 1 (*i.e.*, VCC), while the T inputs of the other flip-flops are driven by some combination of flip-flop outputs. The T input of flip-flop B is connected to the output Q_A of flip-flop A; the T input of flip-flop C is connected with the AND-operated output of Q_A and Q_B. Similarly, the T input of D flip-flop is connected with the AND-operated output of Q_A, Q_B, and Q_C.
From the circuit, we can see that flip-flop A changes its state with the negative transition of each clock pulse. Flip-flop B changes its state only when the value of QA is 1 and a negative transition of the clock pulse takes place. Similarly, flip-flop C changes its state only when both Q_A and Q_B are 1 and a negative edge transition of the clock pulse takes place. In the same manner, the flip-flop D changes its state when $Q_A = Q_B = Q_C = 1$ and when there is a negative transition at clock input. The count sequence of the counter is given in Table 3.3(a).

Table 3.3(a): Count Sequence of a 4-bit binary synchronous counter

State	Q_D	Q_C	Q_B	Q_A
0	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	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

3.2 Synchronous Counter with Ripple Carry

The 4-bit synchronous counter discussed in the previous unit is said to be a s*ynchronous counter with parallel carry*. Moreover, in this type of counter, as the number of stages increases, the number of AND gates also increases, along with the number of inputs for each of those AND gates. This is a certain disadvantage for such type of circuits. Now this problem can be eliminated if we use the *synchronous counter with ripple carry* shown in Figure 3.3(b).

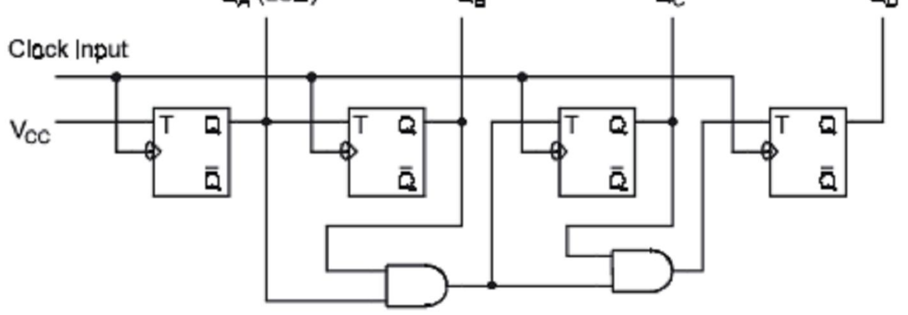


Figure 3.3(b): A 4-bit synchronous counter with ripple carry

But in such circuits the maximum clock frequency of the counter is reduced. This reduction of the maximum clock frequency is due to the delay through control logic which is now 2_{tg} instead of t_g which was achieved with parallel carry. The maximum clock frequency for an *n*-bit synchronous counter with ripple carry is given by

$$f_{\max} = \frac{1}{t_p + (n - 2)t_g}$$

where n = number of flip-flop stages.

3.3 Synchronous Down-Counter

A parallel down-counter can be made to count down by using the inverted outputs of flip-flops to feed the various logic gates. Even the same circuit may be retained and the outputs may be taken from the complement outputs of each flip-flop.

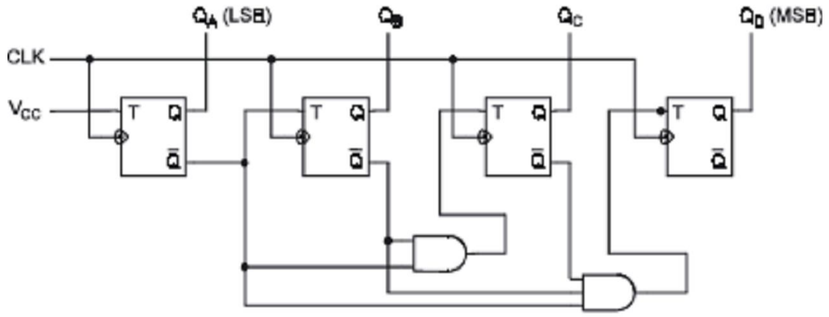


Figure 3.3(c): A 4-bit synchronous down-counter.

The parallel counter shown in Figure 3.3(b) can be converted to a down-counter by connecting the Q′A, Q′B, and Q′C outputs to the AND gates in place of QA, QB, and QC respectively as shown in Figure 3.3(c). In this case the count sequences through which the counter proceeds will be as shown in Table 3.3(b).

Table 3.3(b): Count Sequence of a 4-bit synchronous down-counter

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

3.4 Synchronous Up-Down Counter

Combining both the functions of up- and down-counting in a single counter, we can make a synchronous up-down counter as shown in Figure 3.3(d).

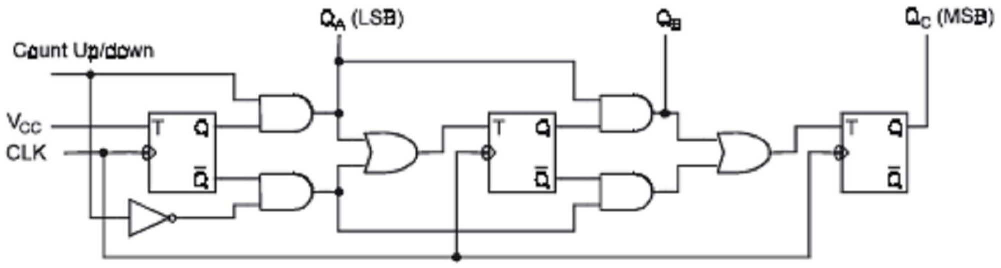


Figure 3.3(d): A MOD-8 synchronous up-down-counter

Here the control input (count-up/down) is used to allow either the normal output or the inverted output of one flip-flop to the T input of the following flip-flop. Two separate control lines (count-Up and count-down) could have been used but in such case we have to be careful that both of the lines cannot be simultaneously in the high state. When the count-up/down line is high, then the upper AND gates will be active and the lower AND gates will remain inactive and hence the normal output of each flip-flop is carried forward to the following flip-flop. In such case, the counter will count from 000 to 111. On the other hand, if the control line is low, then the upper AND gates remain inactive, while the lower AND gates will become active. So the inverted output comes into operation and the counter counts from 111 to 000.

3.5 Application of Counters Digital Clock

A digital clock, which displays the time of day in hours, minutes, and seconds, is one of the most common applications of counters. To construct an accurate digital clock, a very highly controlled basic clock frequency is required. For battery-operated digital clocks (or watches) the basic frequency can be obtained from a quartz-crystal oscillator. Digital clocks operated from the AC power line can use the 50 Hz power frequency as the basic clock frequency. In either case, the basic frequency has to be divided down to a frequency of 1 Hz or pulse of 1 second (pps). The basic block diagram for a digital clock operating from 50 Hz is shown in Figure 3.3(e).

The 50 Hz signal is sent through a Schmitt trigger circuit to produce square pulses at the rate of 50 pps. The 50 pps waveform is fed into a MOD-50 counter, which is used to divide the 50 pps down to 1 pps. The 1-pps signal is then fed into the SECONDS section.

This section is used to count and display seconds from 0 through 59. The BCD counter advances one count per second. After 9 seconds the BCD counter recycles to 0. This triggers the MOD-6 counter and causes it to advance one count. This continues for 59 seconds. At this point, the BCD counter is at 1001 (9) count and the MOD-6 counter is at 101 (5).

Hence, the display reads 59 seconds. The next pulse recycles the BCD counter to 0. This, in turn, recycles the MOD-6 counter to 0.

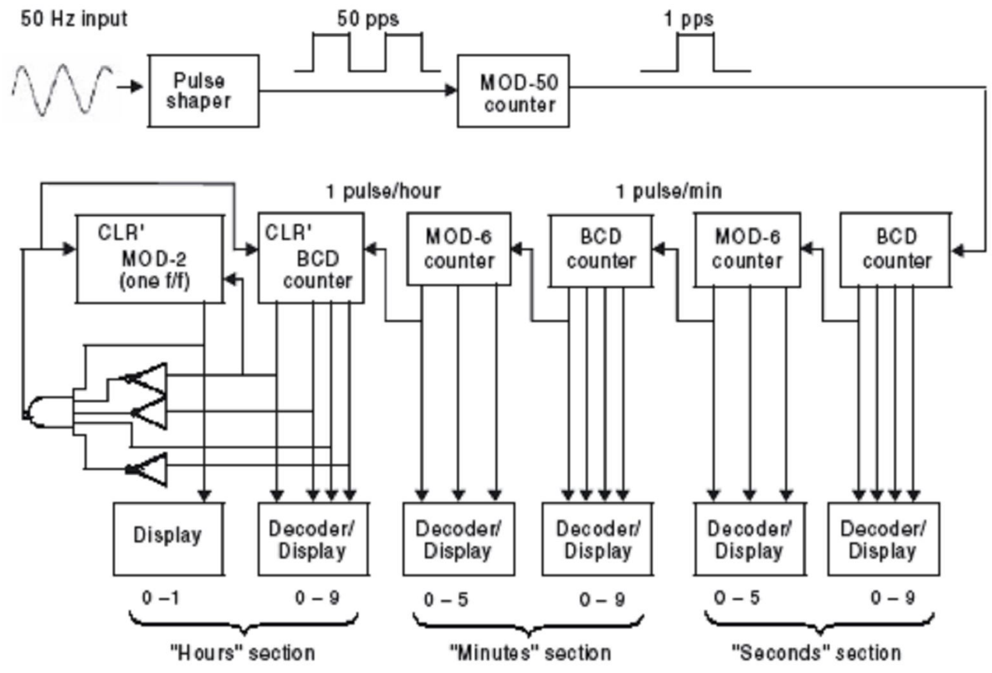


Figure 3.3(e): Block diagram for a digital clock.

The output of the MOD-6 counter in the SECONDS section has a frequency of 1 pulse per minute. This signal is fed to the MINUTES section, which counts and displays minutes from 0 through 59. The MINUTES section is identical to the SECONDS section and operates in exactly the same manner.
The output of the MOD-6 counter in the MINUTES section has a frequency of 1 pulse per hour. This signal is fed to the HOURS section, which counts and displays hours from 1 through 12. The HOURS section is different from the MINUTES and SECONDS section in that it never goes to the zero state. The circuitry in this section is different. When the hours counter reaches 12, it will be reset to zero by the NAND gate.

3.6 Advantage of Synchronous Counters over Asynchronous

In a parallel counter all the Flip-flops will change states simultaneously; that is, they are all synched to the NGTs (see Module 2, Unit 3) of the input clock pulses. Thus, unlike the asynchronous counters, the propagation delays of the flip-flops do not add together to produce the overall delay. Instead, the total response time of a synchronous counter is the time it takes one flip-flop to toggle plus the time for the new logic levels to propagate through a single AND gate to reach the J, K inputs. That is,

$$\text{Total delay} = \text{Flip-Flop } t_{\text{pd}} + \text{AND gate } t_{\text{pd}}$$

This means that a synchronous counter can operate at a much higher input frequency than an asynchronous counter with the same number of flip-flops. Of course, the synchronous counter requires more circuitry than the asynchronous counter. This ability to operate at higher frequencies is the major advantage of synchronous counters.

Self Assessment Exercise

1. What is the advantage of a synchronous counter over an asynchronous counter? What is the disadvantage?

4.0 Conclusion

The basic principle of operation of the synchronous counter is this:
The J and K inputs of the flip-flops are connected so that only those flip-flops that are supposed to toggle on a given NGT will have J = K = 1 when that NGT occurs.

5.0 Summary

In this unit, you learnt about:

- Synchronous Counter with Ripple Carry
- Synchronous Down-Counter
- Synchronous Up-Down Counter
- Application of Counters
- The advantages of synchronous counter over asynchronous counter

6.0 Tutor Marked Assignment

1) Design a 4-bit synchronous counter with ripple carry
2) Design a 4-bit synchronous down-counter.
3) Explain two other applications of Counters

7.0 Further Reading and Other Resources

1. Ronald J. Tocci (1988). “Digital Systems: Priciples and Applications”, 4th Edition Prentice-Hall International edition.
2. http://en.wikipedia.org/wiki/Logic_gate
3. http://www.discovercircuits.com/D/digital.htm
4. http://www.encyclopedia.com/doc/1G1-168332407.html
5. http://www.logiccircuit.org/