Lab 4: Motor Drive Circuit

In this lab you will learn and tinker with:

- General function of capacitors
- Behavior and functionality of flip-flops
- Building finite state machines
- Building a prototype for a motor drive circuit of the DEB

1.1: The Problem to be Solved

DC motors are essential additions to implementations that require a high starting torque and controllable speeds. However, most motors require high currents that are hard for IC chips to supply. If these motors are connected directly to ICs, they might get damaged. Consequently, motor drivers are an effective solution to this common problem. These circuits act as a bridge between the ICs and the motor itself, driving the motor with an amplified current and high voltage. Additionally, motor drive circuits protect motors by preventing failures in extreme circumstances, which include various temperature conditions.

Motors and their drive circuits are tested in extreme conditions such as harsh temperatures to determine their reliability and durability. During this process, the motor goes through several forward and reverse cycles to verify that all functions work correctly and efficiently. Your job, therefore, is to build a digital circuit prototype that simulates this testing process. This will be done by using a powerful digital design tool known as finite state machines, which will be covered in the following sections of this lab.

1.2: Learning about Motor Drive Circuits

The diagram in figure 1 shows the driver circuit for a specific motor. As can be seen, the circuit consists of four switches labeled S1-S4. Switches S3 and S4 are active-low switches, meaning that they are true when they are not pressed. On the other hand, S1 and S2 are active-high, meaning that they are true when they are pressed. Additionally, the driver has a charging circuit that supplies a capacitor with a large storage capacity. A capacitor is an electric component that stores energy (electrical charge) and resists sudden changes in voltage. In a motor drive circuit, the capacitor's purpose is mainly to provide power to the motor when moving and to recharge when the motor is not moving.

Now, let's consider the two motion stages of the motor circuit in Figure 1. If we desire to make the motor move in a forward fashion, switches S1 and S3 should be on while S2 and S4 are off since this combination will provide a positive voltage to the motor. On the contrary, for the motor to move in reverse fashion, switches S2 and S4 should be on while S1 and S3 are off, which will provide a negative voltage to the motor. Finally, if all the switches are off, this will prevent the motion of the motor and, therefore, the charging circuit will recharge the capacitor with a DC source.

Moreover, inputs X and Y determine the state of the switches and when the charging circuit will be on or off. For example, when both inputs are 0, all of the

switches will be off, but the charging circuit will be on, meaning that the capacitor is supplied with the DC source and recharged.

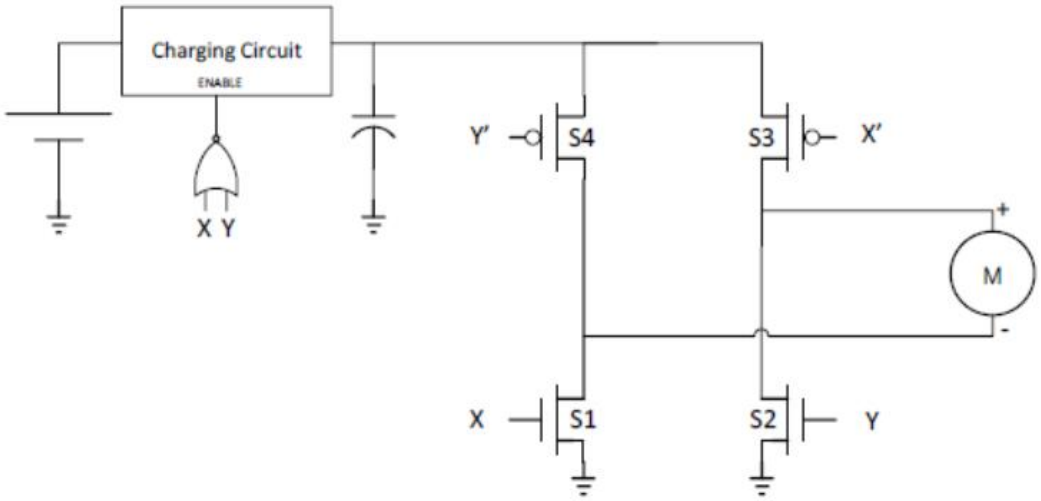


Figure 1: *Motor Drive Circuit.*

a. For the first lab activity, draw diagrams for each of the four combinations of the X and Y inputs. In your diagrams, indicate which switches are open and which are closed. For example, in the scenario when both X and Y are off, the charging circuit will be on and, therefore, the battery and capacitor are shorted together.

   i. Consider when X and Y are both equal to 1. Is there an issue with this specific combination? Explain.

b. Translate your diagrams from part a to the truth table shown below. Similar to part a, specify the input combinations for X and Y, the corresponding state of each of the switches (ON/OFF) for each combination, and whether the charging circuit is enabled or not. The first row has been completed for you.

Inputs		**Switch Outputs**			
X	Y	S1	S2	S3	S4
0	0	OFF (0)	OFF (0)	OFF (1)	OFF (1)

1.3: Finite State Machines

So far, you have learned about and constructed circuits that use combinational logic. This means that the outputs are dependent on the current state of the inputs and the previous state does not affect the output. However, some implementations need our circuit to remember the previous state of inputs in order to produce the outputs. Circuits that require this kind of logic are known as ***sequential logic*** circuits, which, as the name implies, consists of a sequence of output states.

In order to design such circuits, it is important to first consider the sequence of states that the logic will follow. Such sequence can be illustrated using what is known as a ***state diagram***, as shown in Figure 2. A state diagram consists of nodes (circles) each representing a state and arrows that indicate the next state in the logic. Each node is given a meaningful name that specifies its function.

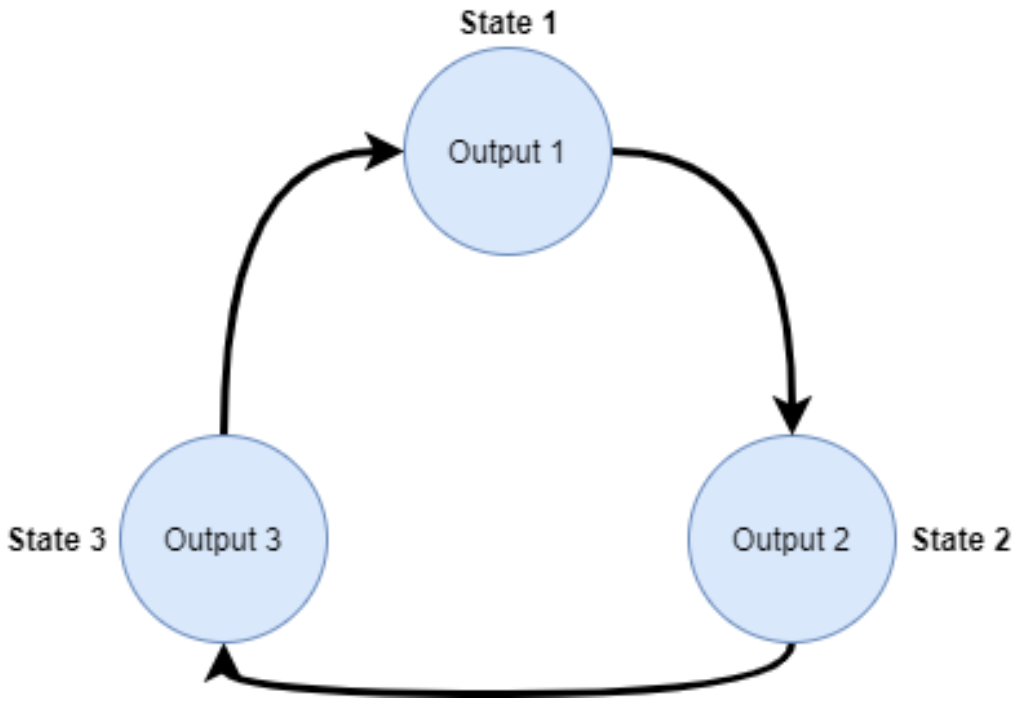


Figure 2: *Example of a state diagram showing nodes and transitions.*

Translating a state diagram to logic circuitry uses a combination of both combinational and sequential logic. The states, for example, are held in memory using a sequential logic component known as a ***flip-flop*** (see appendix for more information on flip-flops). Transitions between one state to another as well as producing outputs requires combinational logic that uses the state held in memory by the flip-flops. Altogether, the overall design of any finite state machine is summarized by the block diagram shown in Figure 3.

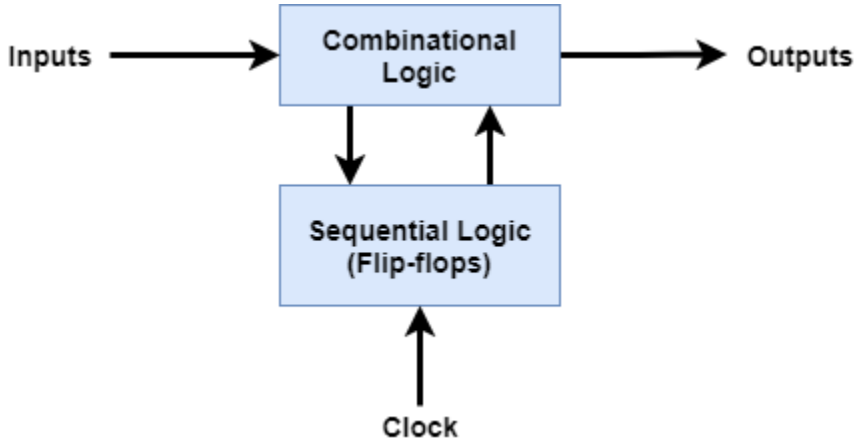


Figure 3: *General design of a finite state machine*

As an example, let's say we want to design a counter that counts a sequence of numbers within the range 0-7. Since we need to account for numbers up to 7, we will use a 3-bit counter to properly represent all the numbers. Therefore, for this design the sequence of states will be 0→1→2→3→4→5→6→7 and then repeating the sequence after 7, each number represented as a 3-bit binary number.

First, we need to draw a state diagram illustrating the sequence of states and the transitions. In Figure 4, you can see a completed state diagram for this specific example, showing each node with a specific label and arrows to represent the transitions.

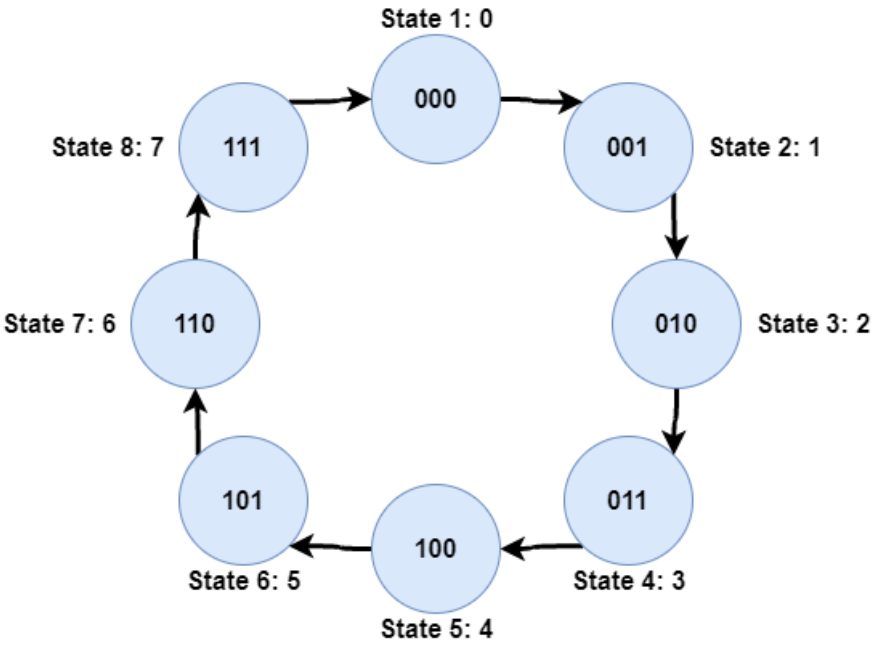


Figure 4: *State diagram for 3-bit counter.*

Next, we will create a ***state transition table***, which simply consists of the outputs of the flip-flops, which represent the current states, and the inputs to the flip-flops, which represent the next states. This table will consequently help us build the combinational logic portion of our design. In order to know how many flip-flops we will need, we must consider the number of states that our implementation needs. If our design requires N states, then the number of flip-flops we need is equal to ceil($\log_2$N), ceil meaning that, in the case we get a decimal number, we need to round it up to the nearest whole number that is greater than or equal to the given number. In our case, for example, we have a total number of 8 states, therefore, the number of flip-flops we need is ceil($\log_2$8) = 3 flip-flops.

In table 1, the state transition table is shown. In this case, the Q columns represent the current states and the D columns represent the next states. Furthermore, we can write down the logic equations for the D signals based on the Q signals.

Current State			**Next State**		
Q2	**Q1**	**Q0**	**D2**	**D1**	**D0**
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Table 1: *State transition table for 3-bit counter.*

$$D2 = Q2 * /Q1 + Q2 * /Q0 + /Q2 * Q1 * Q0$$

$$D1 = /Q1 * Q0 + Q1 * /Q0$$

$$D0 = /Q0$$

These logic equations will now help us design the combinational logic portion of our counter design. Figure 5 shows the translation of the logic equations into their logic gate equivalents. Furthermore, we can combine this design with our three flip-flops as well as a clock signal to form the final product, as shown in Figure 6.

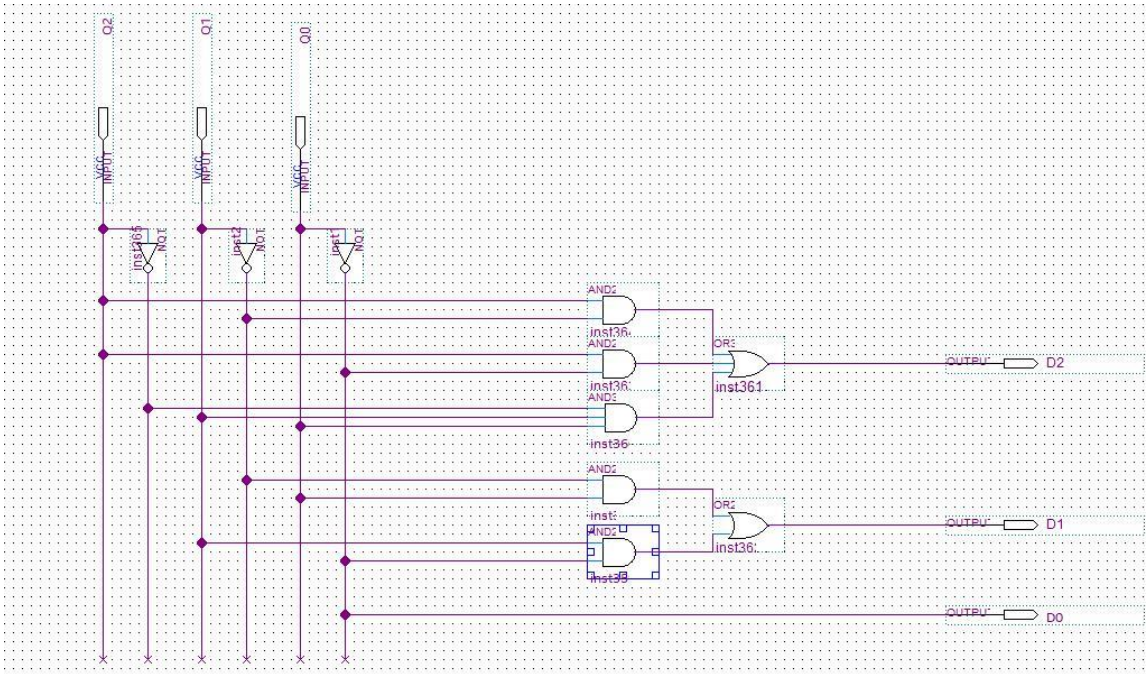


Figure 5: *Combinational logic portion of counter design*

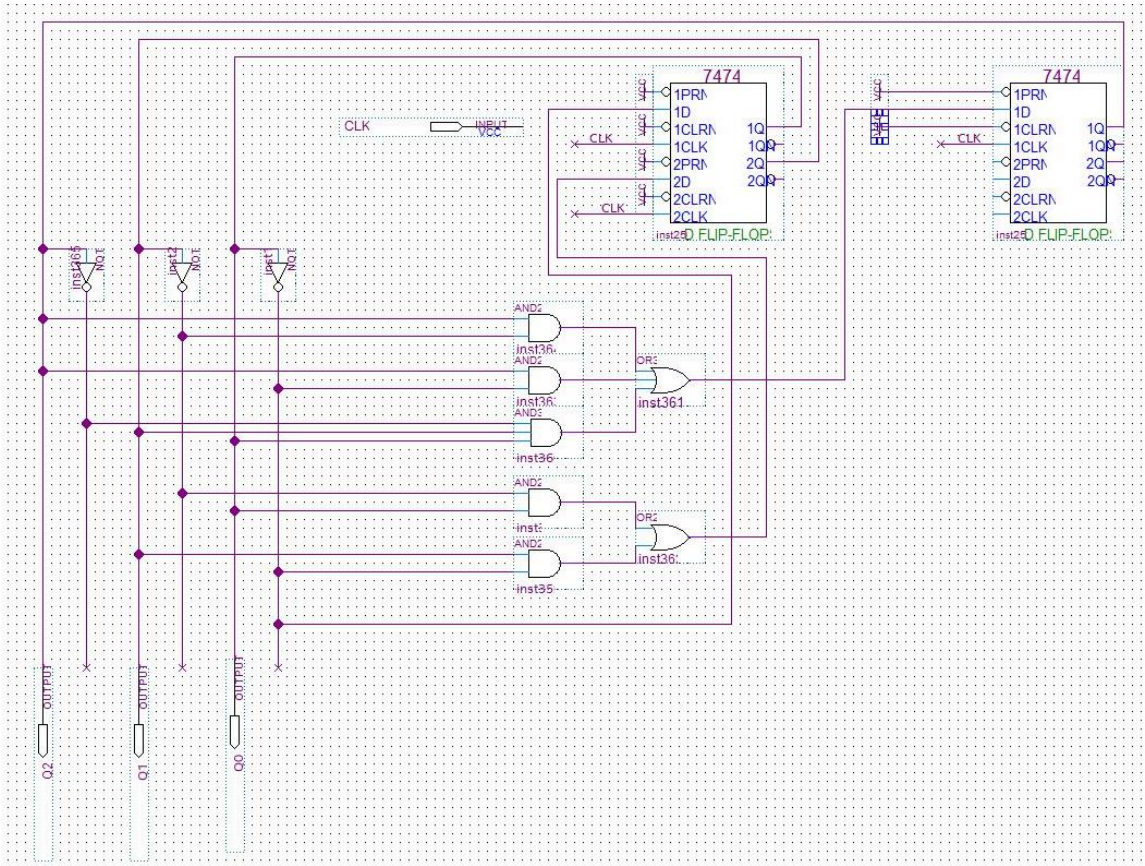


Figure 6: *Full design of counter design including combinational logic and memory portion.*

Now that we know how to construct finite state machines, we can design our motor drive circuit. As discussed previously, our goal is to automate the testing process for a motor and its drive circuit. To do this, we need to construct a finite state machine that produces a simple sequence determined by the value of outputs X and Y. This sequence is as follows: charge, forward motion, charge, reverse motion. For this final portion of the lab refer to your truth table from section 1.2 part b to determine which combination of X and Y we need to output the desired sequence of actions.

a. Draw, on paper, a state diagram like the ones shown in figures 4 and 2 for your motor drive state machine to follow the sequence charge→forward→reverse and repeat the cycle. Remember to label each node/state in your diagram with an appropriate name describing its function and include the binary data corresponding to each state.
b. How many flip-flops will you need for this state machine?
c. Complete a state transition table like the one in table 1 to help you design the combinational logic for your motor drive state machine. After this, write down the logic equations for the X and Y outputs in your transition table.
d. Design the combinational logic portion of your state machine based on your logic equations from part c.
e. Incorporate flip-flop(s) to your combinational logic to complete your motor drive state machine design. Remember to include an input clock signal for your flip-flop(s) and to connect preset and clear signals to VCC to deactivate them.
f. Build your state machine on your DEB using appropriate switches for your inputs. For outputs, use OUT9 for X and OUT8 for Y. Check to yourself that your state machine correctly outputs the desired sequence.
g. After designing your motor controller, consider the possibility of X=1 and Y=1 occurring simultaneously. Why won't this combination occur? Or, if it occurs in your design, what modification would you make in your design to prevent it from happening? Explain in words, no need for a prototype.

Appendix:

Flip-Flops

In digital electronics, flip-flops are memory devices that store data utilizing sequential logic, as discussed in section 1.3. Flip-flops have an input control signal called a clock. This clock's signal continuously alternates between high and low, which is known as the clock pulse. For a flip-flop, whenever a rising edge or a falling edge (depending on the configuration of the clock) is encountered as seen in figure 7, the output of the flip-flop will change, meaning that as long as the clock does not encounter an edge, the state of the flip-flop will be preserved, which is why they are considered memory devices.

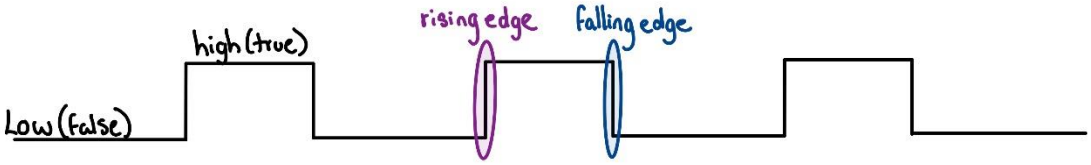


Figure 7: *Clock pulse depicting rising and falling edges as well as true and false states.*

The four most common types of flip-flops are SR, D, JK, and T flip-flop. The DEB kit contains two of these four types: the D flip-flop and the JK flip-flop. For now, however, our primary focus will be to learn about the D flip-flop since it is the one you will be using in this lab. A block diagram of a typical D flip-flop is shown in Figure 8.

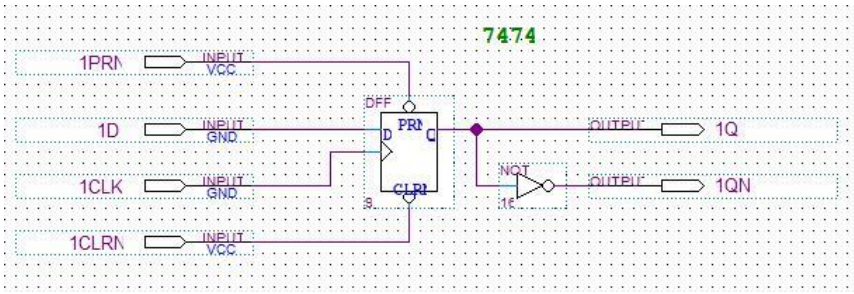


Figure 8: *Block diagram of D flip-flop*

As you can see from Figure 8, the D flip-flop has four input signals and 1 output signal. The active-high D input signal sends the data that is to be preserved by the flip-flop. The clock, as discussed previously, tells the flip-flop when its output should change, which often occurs at a rising or falling edge. The active-low input signals PRN and CLR are known as preset and clear signals, respectively. These signals control the output, Q, of the flip-flop regardless of the state of the clock. When the PRN signal is

set low (true), the Q output signal will be set to 1 or high. On the other hand, when the CLR signal is set low (true), the Q output signal will be reset to 0 or low.

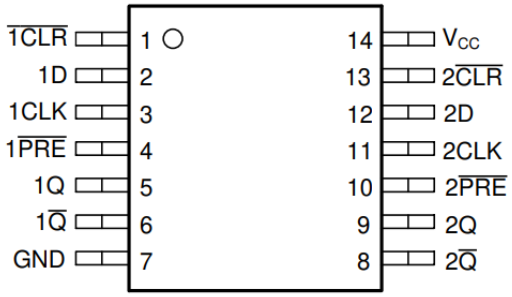
Table 2 shows the truth table for the D flip-flop. In this table, D is the input, Q represents the current state of the flip-flop, and Q(t+1) represents the next state of the flip-flop. You may notice that the Q column is filled with Xs, which are don't cares. This is because the current state of the input does not matter for the next state, meaning that the value of the D input will be changed to the value of the output of the flip-flop regardless of the previous value of the output.

Input	Current State	Next State
D	**Q**	**Q(t+1)**
0	X	0
1	X	1

Table 2: *Truth table for D flip-flop.*

Finally, Figure 9 shows us the pin diagram for the 74HC74 IC D flip-flop chip that is available in the DEB kit. This chip has access to two D flip-flops, each having its own set of inputs and outputs. More specifically, D flip-flop 1 can be accessed using the pins on the left side while D flip-flop 2 can be accessed using the pins on the right side. In this case, CLR is the clear input signal, PRE is preset, CLK is the clock, Q is the output of the flip-flop, and D is the data input. One important thing to note is that whenever several flip-flops are used simultaneously for a unique purpose, they all must share the same clock signal so that their behavior is synchronized.

5 Pin Configuration and Functions



D, DB, N, NS, PW, J, or W Package
14-Pin SOIC, SSOP, PDIP, SO, TSSOP, CDIP, or CFP
Top View