Chapter 6: State Machines

In this chapter we describe the operation of state machines, which are devices that form the basis for microprocessors and computers.

I. State Machine Theory

A state machine is a machine that makes predictable transitions through a sequence of states, based on external inputs and the current state of the machine. In digital electronics, the timing of the transition of the machine from one state to another is controlled by a register and a system clock, while the next state of the machine is determined by a combination of logic gates and embedded memory. A schematic representation of a state machine is shown in figure 1 below.

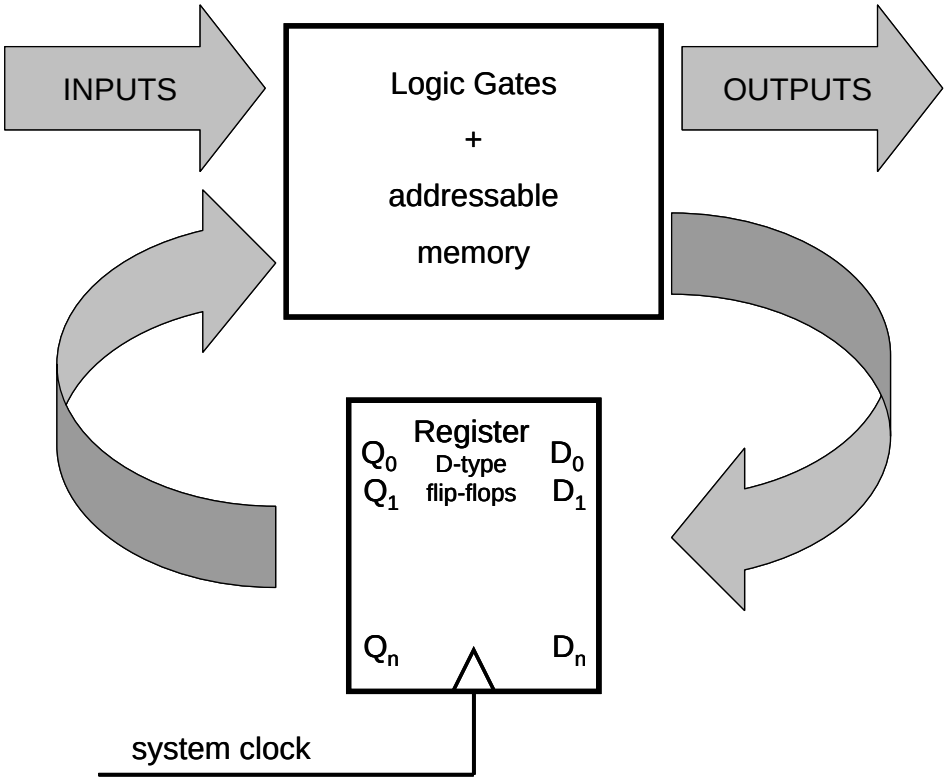

Figure 1: Block diagram representation of a digital electronics state machine.

A counter is one of the most basic state machines on can make. It simply makes a transition from one binary number (i.e. a state) to the next on each clock cycle. A two-bit counter would make this sequence of transitions (each number is a particular state):

$$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$$

State machines are the essential components that make a computer possible. Most computers are controller by one counter (called the program counter) that steps through a simple numerical sequence. It becomes a computer when each number in that sequence is assigned an action. Any device that has this function is called *sequencer*.

External Input

Some state machines simply plod through from one state to the next. Others make transitions that are conditional upon some external input. In a synchronous counter, the "count enable" input produces this type of more sophisticated state machine. At each clock pulse, it either advanced to a new state or stayed at the same state depending on the condition of the other bit. This is an example of a state machine that changes state under the influence of an external control line. Note that the *next state* depends on *both* the external control and on the internal present state. This is what makes it sophisticated.

With this counter, you have the essence of all computers. The system steps through a sequence that depends on its current state, and on external input. This is fundamentally no different than your PC displaying a view on your monitor and knowing to update the view periodically but also knowing to respond to any movements of the mouse.

II. Examples

In this section we will show how to implement various types of state machines using D-type registers and logic gates. If the number of states is small, then the logic gate circuitry can be determined using Karnaugh maps.

T-type flip-flop

We can consider a T-type flip-flop as a state machine. A T-type flip-flop has two states, 0 and 1, so only one D-type flip-flop is required. The block diagram for the state machine is shown in figure 2 on the right.

In order to determine what circuitry to put into the "logic gates" block, we construct the logic table for the state machine sequence, which is shown below.

Input	Q_n	Q_{n+1} & D
1	1	0
1	0	1
0	1	1
0	0	0

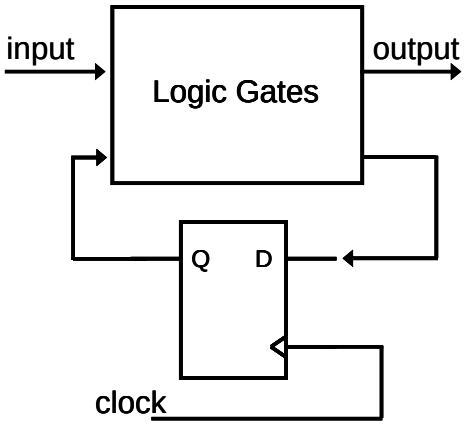

Figure 2: Block diagram for the state machine representation of T-type flip-flop.

From the logic table, we infer the associated Karnaugh map shown below in figure 3. The Karnaugh map is relatively straightforward and is equivalent to a XOR gate. Consquently, the circuit for a T-type flip-flop consists of a D-type flip-flop and a single XOR gate as shown in figure 3 below.

T_{in} \ Q_n	**0**	**1**
0	0	1
1	1	0

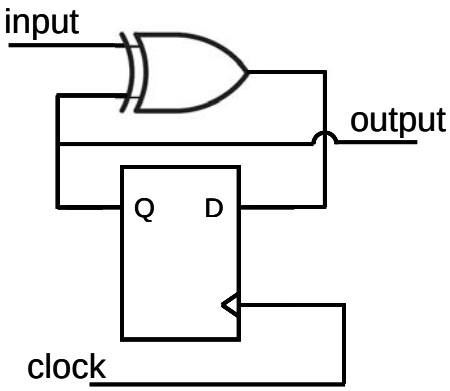

Figure 3: Left: The Karnaugh map for the T-type flip-flop state machine. Right: Circuit diagram for a T-type flip-flop.

Sequencers

If you have a very simple state machine to design that just steps from one state to the next it can be based on one or more counters with some external logic. The output logic for this device will decode the counter value and put a combination of output lines into operation. This circuit is called a *sequencer*, because it steps through a simple sequence of operations.

A real world example of such a sequencer is the control unit of a dish washer. The desired sequence might be the following:

1. Turn on the faucet for 5 minutes;
2. Turn on the washer heads for 10 minutes;
3. Pump out the water for 5 minutes;
4. Turn on the faucet for 5 minute;
5. Turn on the washer heads for 5 minutes;
6. Pump out the water for 5 minutes;
7. Turn on the heater for 15 minutes.

This sequence should start with the push of a button, run once, and stop.

There are two common ways to implement such a sequence.

- A single master-counter that counts over the entire 50 minutes and uses logic on the outputs to decode which device should be on at any given count;
- A series of seven counters with predetermined timing (i.e. number of counts) and

logic to insure that the counters start and stop counting in the proper order.

The optimal method depends on the details and complexity of the sequence to be generated. In the single counter method, we might select a clock with a 5 minute period. We can then lay out a functional table for our sequencer like the following:

Counter Value	***Output Asserted***
10	Faucet
9	Washer heads
8	Washer heads
7	Pump
6	Faucet
5	Washer heads
4	Pump
3	Heater
2	Heater
1	Heater
0	Nothing!

With a small number of steps one can just use simple logic based on the counter's bits to determine how to turn on which output line.

Counter Value	***Output Asserted***
10 & 6	Faucet
9 & 8 & 5	Washer heads
7 &4	Pump
3 & 2 & 1	Heater

From a table like this one can then take the output bits from the counter and design an array of logic gates that will ensure the control lines are asserted on the correct count values.

Since decoding a sequence is so common, there are chips, such as the 74LS138, which perform these sorts of operations. They will take, say, a 3-bit binary number and turn on one of eight output lines based on the binary number encoded in the inputs.

Design Exercises

Design Exercise 6-1: Design a sequencer circuit to turn on four output lines in the following sequence:

(a) Bit 1 on for 5 seconds
(b) Bit 2 on for 10 seconds
(c) Bit 3 on for 10 seconds
(d) Bit 4 on for 5 seconds

This sequence should start after you push a start button and only go through the cycle once. You may use the function generator as your clock.

Design Exercise 6-2: Design a circuit for detecting the sequence 1010 on a serial input line. The circuit should output a HIGH pulse (1 clock duration) when it detects the sequence and output LOW otherwise. The clock frequency should be 0.1-1 Hz.

Design Exercise 6-3: Design a state machine circuit for a JK-type flip-flop.