Introduction to State Machines

1. Synchronous and Asynchronous Circuits

The next few sets of notes are concerned with the design of synchronous sequential logic circuits: those in which changes of states are allowed to occur only at regularly-spaced clock-transition times. Synchronous operation avoids some of the problems that arise in an asynchronous circuit, where changes of state are allowed to occur at arbitrary times, determined by events external to the circuit.

The main virtue of synchronous operation is that, with proper design, each change of state is allowed a definite time to stabilize before the logical variables are called on to help determine the next state. In an asynchronous circuit, on the other hand, there can be occasions when an input variable changes before the previous change of state has stabilized. If such occurrences are not anticipated in its design, an asynchronous circuit will sometimes behave in an unintended way. Of course, such possibilities should be taken into account during design, but it is fairly obvious that they will make the design process more complicated than for a synchronous circuit.

Figure 1: Finite State Machine
2. **FINITE-STATE MACHINES**

Any deterministic digital circuit can be regarded as a finite-state machine. This means that it has a finite number of distinct states, arising as combinations of a finite number of stored variables. A distinction is often drawn between the total state of the machine, which includes the input as well as the stored variables, and the internal state of the machine, which depends on the stored variables alone. When "the state" is referred to without qualification, it is normally the internal state that is meant.

**Question**

(i) The circuit shown in Figure 1 has two binary input variables, three flip-flops and four binary output variables. What is maximum number of distinct total states, and the maximum number of distinct internal states, that this circuit can possess?

The number of outputs is irrelevant to determining the number of states. The point here is that the outputs can depend only on combinations of input and stored variables (one per flip-flop). There are no other mechanisms that can influence the outputs if the circuit is deterministic. Any given combination of input and stored variables must consistently produce the same outputs.

Restricting the discussion now to synchronous finite-state machines, changes of state can occur only at clock-transition instants. This implies that the values of the input variables are significant to the machine only at times just prior to the clock-transitions. For simplicity, it will be assumed (arbitrarily) that only the upwards clock transitions cause changes of state.

The current input and stored variables not only determine the current outputs of the finite-state machine; they also determine the new values of the stored variables following the next (upwards) clock-transition. Once again, the argument is simply that the current inputs to the flip-flops can only be logical combinations of the current stored variables (flip-flop outputs) and the external inputs to the complete circuit. No other influences are possible.

**Question**

(ii) For the circuit of Figure 1, if the current total state is

\[(U_0, U_1, S_0, S_1, S_2) = 01001,\]

how many possibilities are there for the next total state of the machine?
How many possibilities are there for the next internal state?
3. A SIMPLE STATE-MACHINE EXAMPLE

The following example is a synchronous sequential circuit, whose operation is simple enough to be easily understood, but which illustrates the main features of state machines generally. Its function is to drive four light-emitting diodes to simulate rotation, as shown in Figure 2.

![Figure 2: Rotating Light-Emitting Diodes](image)

The circuit has two inputs: one to control the sense of rotation and the other to permit or inhibit rotation. From an external point of view, the circuit's function can be represented by the block diagram of Figure 3.

![Figure 3: Block Diagram Showing Inputs and Outputs](image)

It is obvious that the circuit needs only four internal states, which can be provided by two flip-flops, arranged as some form of counter. There is an element of arbitrariness in the choice of counter, but the number of possibilities (e.g., natural binary or Gray code) is quite small. In order to make the operation easy to follow, a natural binary counter is chosen here. The flip-flop outputs are assigned to the state-labels as follows:

<table>
<thead>
<tr>
<th>Label</th>
<th>( Q_1 )</th>
<th>( Q_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The next-state table, given below, allows the original specification to be converted into logical expressions for the J and K inputs of the flip-flops.
<table>
<thead>
<tr>
<th>Label</th>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stored $Q_1$ $Q_0$</td>
<td>Input $U_1$ $U_0$</td>
</tr>
<tr>
<td>a</td>
<td>0 0 0 0</td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>b</td>
<td>0 1 0 0</td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>c</td>
<td>1 0 0 0</td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>d</td>
<td>1 1 0 0</td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1</td>
</tr>
</tbody>
</table>

Karnaugh map for next $Q_1$:

Karnaugh map for next $Q_0$:

Figure 4: Circuit Implementation by JK Flip-flops
Applying the next-state relationship for the JK flip-flop (write it down:)

gives: \( J_1 = K_1 = U_1 \cdot (Q_0 \oplus U_0) \) and \( J_0 = K_0 = U_1 \).

The remaining task is the decoding of the internal state variables to provide outputs to the LEDs. The complete circuit is given in Figure 4.

4. OTHER REPRESENTATIONS

The behaviour of a sequential circuit can be described in several ways. The circuit of Figure 4 is just one possible implementation of the function originally specified. From the point of view of design, it is desirable that the required function be represented as clearly as possible before any commitment is made to a particular type of circuit. The reasons for this are:

(i) so that a lack of precision or completeness in the specification can be identified before substantial design effort is wasted, and

(ii) to help identify the 'structure' of the problem, which may allow for some simplification of the design.

4.1 State Transition Diagram Representation

One useful way of describing a sequential system is the state-transition diagram, exemplified in Figure 5 for the same specification given earlier.

![State Transition Diagram]

Transition Labelled: \( U_1U_0 \quad U_0 = 1 \) for anti-clockwise \quad \( U_1 = 1 \) for rotate

Figure 5: State Transition Diagram Representation
In Figure 5, the small circles represent internal states, while the transitions between states are labelled with combinations of input variables. This type of flow chart is the first step in the *algorithmic state machines* (ASM) approach to sequential logic design.

![Algorithmic State Machine Flow Chart](image)

**Figure 6: Algorithmic State Machine Flow Chart**

### 4.2 Algorithmic State Machine (ASM) Representation

Figure 6 describes the same sequential system in terms of an *algorithm*, in which the (internal) states are linked by decisions governed by the input variables. This type of flow chart is the first step in the *algorithmic state machine* (ASM) approach to sequential logic design. This approach to digital design will be developed in detail in the forthcoming notes.

### 4.3 General View of State Machines

A state machine, such as the one discussed in these notes, can be regarded quite generally as a combination of memory (to store the variables embodying the current state) and two combinational blocks, as shown in Figure 7.
Each state of the machine includes a partial determination of the next state, in the sense that not all states are necessarily directly accessible from the current one. (Look at Figure 5, for example.) The inputs, which complete the determination of the next state, are called *qualifiers* (in the same sense as an adjective qualifies a noun). The outputs are also determined by the stored variables in conjunction with the inputs. This view of the state machine suggests alternatives to the JK flip-flop implementation developed earlier in these notes.

![Diagram of state machine](image)

*Figure 7: General Representation of State Machines*

5. **Summary of Objectives**

These notes explain the design of sequential circuits, using the concept of finite-state machines. Other descriptions of sequential circuits, which include state-transition diagrams and algorithmic state machines, are also discussed.

Michael C.K. Tse  
Department of Electronic Engineering