Lecture 9: Finite State representation of digital circuits

We noted that any member of the class of circuits called flip flops can be conveniently be represented by a finite state machine, and in general any synchronous circuit can be modelled by a finite state machine, and *vice versa*. A finite state machine in its most general form can be represented by a pair of equations:

$$S(t+1) = f(S(t), I(t))$$
$$O(t+1) = g(S(t), I(t))$$

Where S(t) represents the state at time t and can take only a finite set of values, normally thought of as integers. I(t) and O(t) are the inputs and outputs which are also normally discrete bounded variables and f and g are functions. Synchronous circuits which conform equation 9.1 are called Mealy machines, and are represented by the block Diagram 9.1. There is a simpler form of finite state machine which is sometimes used where the output is only a function of the state, and the equation pair becomes:

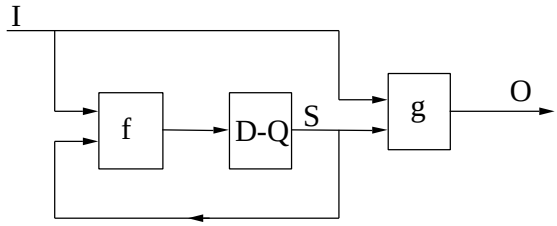

Diagram 9.1 The Mealy Machine

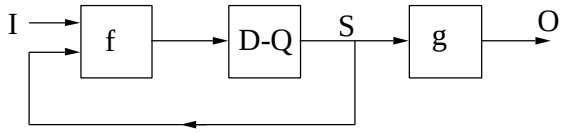

Diagram 9.2 The Moore Machine

$$S(t+1) = f(S(t), I(t))$$
$$O(t) = g(S(t))$$

Circuits of this form are called Moore machines, and are represented Diagram 9.2. In both cases the blocks f and g compute the finite state machine functions and are implemented by combinational logic. We will call block g the decode logic and block f the state control (or state sequencing) logic. The blocks marked *d-q* are called state registers and consist of a bank of flip flops.

The fact that synchronous circuits can be represented by finite state machines gives us the basis of a design methodology. As with programme design (or indeed any engineering design task) we can most easily solve a problem at the highest functional level, hence it is better to design the finite state machine before considering how to construct it out of gates. The methodology for circuit design can be stated as follows:

1. Determine the number of states required by the system
2. Determine the state transitions and outputs and draw the finite state machine
3. Choose the way in which the states will be represented (State Assignment)
4. Express the state sequence logic as a set of Boolean equations combining the states and the inputs, using the existing methodology based on finding minterms. Minimise using the Karnaugh map method.
5. Express the required outputs as a Boolean function of the states, and minimize using Karnaugh maps if possible.
7. Draw the circuit.

We will now consider a simple example which will illustrate some of the issues concerned with this methodology. It is a counting circuit, similar to that designed in the previous lecture. The input will be taken from a push button, and the output will be a digit in the range 0 to 5 displayed on a seven segment display (see diagram 9.3). The seven segment display is a device with seven inputs. A logical 1 will cause the appropriate segment to be illuminated. Such a circuit might used to set the

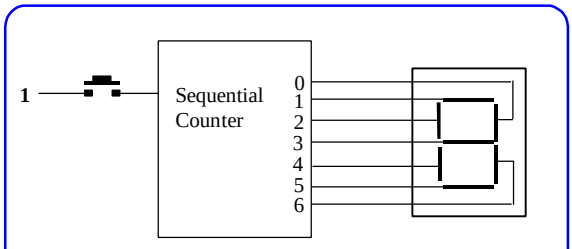

Diagram 9.3: A Sequential Design Problem

time in a digital clock.
The design of the finite state machine is quite straight forward. Clearly there are six states, which we can label 0 to 5 representing the digit that will be displayed. If we assume that the circuit operates on a slow clock, advances when the input button is pushed (one) then the state transitions are also straight forward. The outputs pose more of a problem since we must decide which of the seven segments to illuminate for each state. For example in state 0 we require segments 0,1,2,4,5 and 6 to be illuminated. Thus we might write the corresponding output *A* as {1,1,1,0,1,1,1}. When we have specified all the outputs we have completed the finite state machine design, which is given in Diagram 9.4. Note the convention that in a Moore machine the outputs are written in the nodes, (below the state name). In the Mealy machine they are written on the arcs.

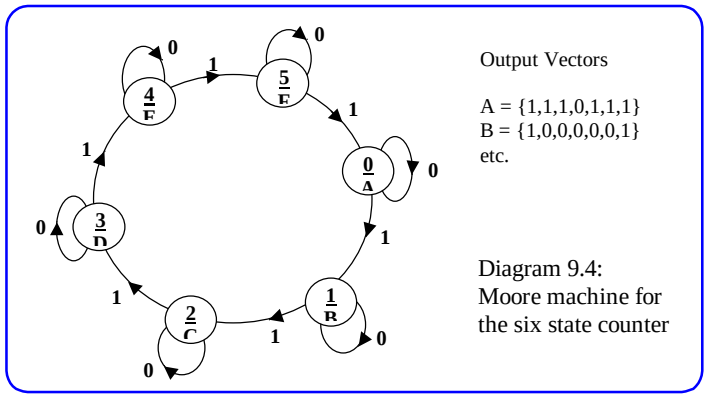

Diagram 9.4: Moore machine for the six state counter

In order to translate a finite state machine into a circuit, we must first decide how the states are to be represented. The simplest possibility is that we allocate one flip flop per state, so that our *d-q* box in diagram 9.2 is made up of six flip flops. The consequence of this is that the decode logic is very simple. It is specified entirely by the OR function. For example, consider output number 1 (which illuminates the top segment). It should be one in states 0,2,3 and 5. Thus we can write:

$$O_1 = Q_0 + Q_2 + Q_3 + Q_5$$

However, this arrangement is likely to use many more components than we need, since the states could be encoded on just three flip flops rather than six. But here again is a problem, since three flip flops can represent eight states, that is to say, if we treat the outputs of the flip flops as a three bit number, we have eight possibilities, and we need to decide which number encodes (or represents) each state. This is the state assignment problem, and the choices that we make will effect both the output and the state sequencing logic.

If we maintain the notation that Si represents the ith state, we can note immediately that each state has a corresponding minterm with the three flip flop outputs as its variables, so we could make a provisional assignment for example:

$$S_0 = Q_2' \bullet Q_1' \bullet Q_0' \qquad S_1 = Q_2' \bullet Q_1' \bullet Q_0 \quad \text{etc}$$

and immediately we can write our decode logic in the canonical form as

$$O_1 = Q_2' \bullet Q_1' \bullet Q_0' + Q_2' \bullet Q_1 \bullet Q_0' + Q_2' \bullet Q_1 \bullet Q_0 + Q_2 \bullet Q_1' \bullet Q_0$$

and apply the Karnaugh map method to determine the minimum circuit for each output. The input logic we can now construct by considering each state in turn. The design decomposes into a separate problem for each state. The equations for a state can be derived simply by looking at the arrows that point to that state in the finite state machine diagram. We can obtain an equation directly in the canonical form. For example:

$$N_0 = S_5 \bullet I + S_0 \bullet I'$$

where N_0 means the next state is to be S_0. Here again we have a problem that we can minimise using the standard Karnaugh map method. However, the new state needs to be presented to the D input of the flip flops, and thus must be encoded. This encoding is simply a set of OR gates, so following the state assignment used above we have that:

$$D_0 = N_1 + N_3 + N_5$$

That is to say, the D0 value should be set to 1 when the next state is S1 (001) or S3 (011) or S5 (101). D0 is the bottom bit of the number that defines the state. Similarly:

$$D_1 = N_2 + N_3 \quad \text{and} \quad D_2 = N_4 + N_5$$

We can substitute for the N values to obtain for example

$D_1 = S_1 \bullet I + S_2 \bullet I' + S_2 \bullet I + S_3 \bullet I'$

$= Q_2' \bullet Q_1' \bullet Q_0 \bullet I + Q_2' \bullet Q_1 \bullet Q_0' \bullet I' + Q_2' \bullet Q_1 \bullet Q_0' \bullet I + Q_2' \bullet Q_1 \bullet Q_0 \bullet I'$

which is now the Boolean equation of the whole state sequencing logic in canonical form ready for minimisation. Once the state assignment has been made, the unused states become don't care states, in that we expect that they will never appear when the circuit is in correct operation. Remembering that don't cares may be treated as either zero or one, and, when using the minterm canonical form we can benefit if we treat them as ones, we add them to the Karnaugh maps for simplification of the resulting circuit.

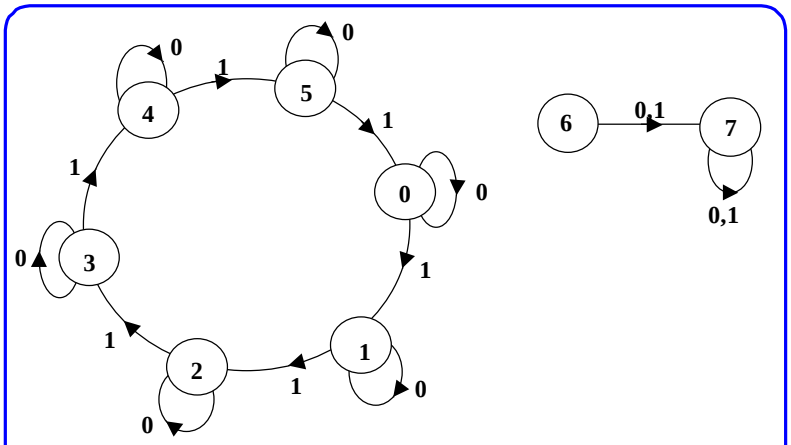

Diagram 9.5: Unallocated States

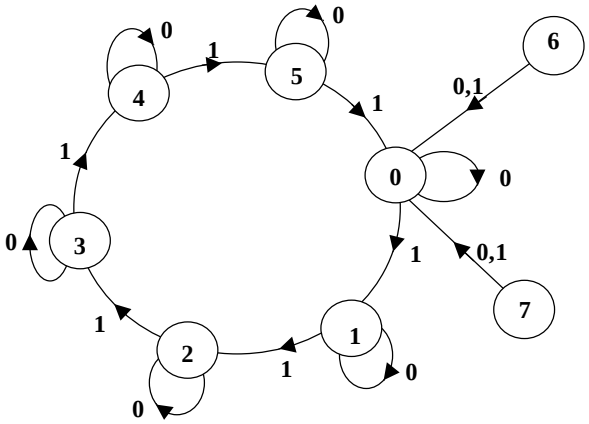

Diagram 9.6: Explicitly allocated unused states

When a synchronous circuit has been designed in this way it is necessary to check that it will start up correctly. The problem is that, since we have treated the unassigned states as don't cares, we cannot define how they will behave. For example, it is possible that the unassigned states form a closed finite state machine as shown in Diagram 9.5, and on start up the circuit never enters any of the required states and therefore does not perform correctly. We could avoid this problem by explicitly including the unassigned states in the design, for example making them all lead to a known state, as shown in Diagram 9.6, but this would have the disadvantage that the resulting circuit may not be the smallest possible. Another strategy would be to check to see what state transitions are implied when the don't cares have been allocated particular values, and modify the circuit if it is unsafe. A third possibility is to use the set and reset features on standard flip flops to force the circuit into an assigned state either on start up, or on the press of a reset button.

Unfortunately, in general, there is no easy way to see which state assignment will result in the best simplification of the whole circuit. One strategy is to try out all possibilities, and determine which gives the best overall result. At first sight this looks like a daunting task, since the number of ways we could allocate the eight possibilities to the six states is 56. However fortunately many of them can be eliminated. For example, consider one possible state assignment shown in Diagram 9.7. We note that flip flops usually always have complementary outputs, and thus if we negate any column this

State	Assignment	Isomorphs
0	000	100 010 110
1	001	101 011 111
2	010	110 000 100
3	011	111 001 101
4	100	000 110 010
5	101	001 111 011

These isomorphs are created by:

Invert Column 1
Invert Column 2
Invert Columns 1 and 2

Diagram 9.7: Isomorphic assignments

will not result in a different circuit, since all we do is exchange the Q and Q' outputs of the flip flop. Applying this principal we can generate a further seven isomorphic assignments by negating the different combinations of columns.

A small refinement of our specification is to make the digit change on the 0 to 1 transition of the input button. We could do this by buffering the input through a T type flip flop, but an alternative strategy is to use the design of diagram 9.8. We have here a twelve state machine, so the strategy of using one flip flop per state will be very wasteful. We can choose to use four flip flops, and make an assignment of the sixteen possibilities to the twelve states, but now the number of possible assignments increases to 1820, and even allowing for the symmetries we have over a hundred possibilities to choose from. Thus to determine the optimal state assignment it will not be possible to evaluate all possibilities. Consequently we need to rely on heuristic rules of the form:

- All those states that have the same next state for the same input should be given adjacent state assignments
- The next states of a state produced by applying adjacent input conditions should be given adjacent state assignments.

In our case the second rule suggests that the neighbouring states in our diagram should be given adjacent state assignments, ie going round the ring in diagram 9.4 we would use 000 001 011 111 110 100 and back to 000. There are many such rules quoted but it is beyond the scope of this course to discuss them.

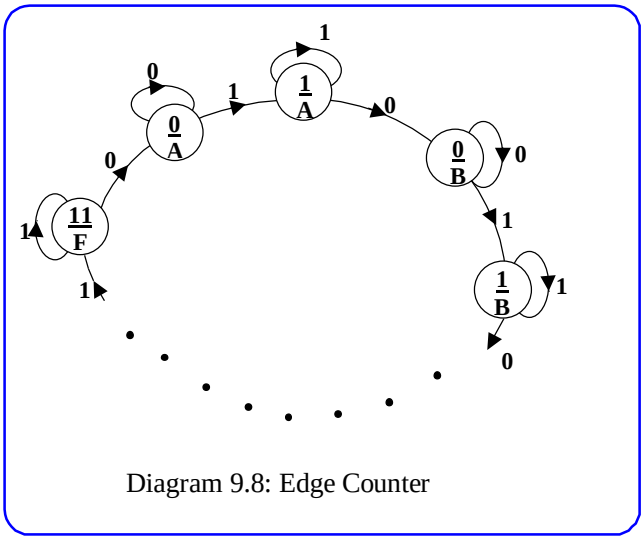

Diagram 9.8: Edge Counter