State Machines: Brief Introduction to Sequencers

Prof. Andrew J. Mason, Michigan State University

Introduction

A **state machine** models behavior defined by a finite number of *states* (unique configurations), transitions between those states, and actions (outputs) within each state. A *finite* state machine refers to a machine with only a relatively small number of states, though this term is often truncated to simply state machine. The current state is a function of past states, and thus the state machine must have memory of its past. A state machine can be represented by a *state diagram* and/or *state transition tables*.

A counter is a type of state machine. It constantly increases its output value as it sequences through states until it reaches its final value and returns to its initial value. We will now consider a type of state machine referred to as a sequencer, one that cycles through a sequence of states in a predefined order. Consider the state diagram in Figure 1. It cycles through four states in a specific sequence. The states are number 1-4 in binary notation such that the sequence is 00 → 01 → 11 → 10 → 00.

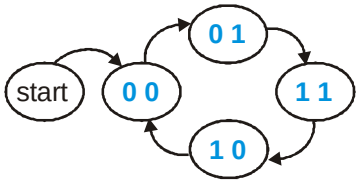

Fig. 1: State diagram for a simple 2-bit state sequencer.

Number of flip flops

To implement state machines in digital hardware, flip flops are often used because they exhibit the necessary memory capabilities. They would be combined with so-called "glue logic" that determines the proper input values in order for the specified state sequence to be realized.

The number of flip flops needed to implement a state machine is a function of the number of states in the state machine. Since a single flip flop has one digital output, it can represent two discrete states, namely 0 and 1. Two flip flops working together provide an output pair (2 signals) that can implement four states, 00, 01, 10, and 11. In general, *n* flip flops can represent 2^n states, much in the same way as an *n*-bit number can represent 2^n values. Thus, the number of binary bits needed to represent all of the states will equal the number of flip flops needed to implement the state machine.

Example 1: How many flip flops are needed to implement a state machine with 12 states?

Answer: 3 flip flops can implement 2^3=8 states and 4 can implement 2^4=16. Since 8 states are too few, 4 flip flops would be needed to cover 12 states. Note this would provide 4 unused states that must be taken care of as described later.

Implementing Glue Logic

Knowing how many flip flops are needed is the starting point, but how can we force the flip flops to produce the correct output to implement the desired state sequence? The answer lies in the *glue logic*, the circuitry that will set the flip flop inputs to the proper values at each state to

ensure that they will generate the desired next-state outputs after a clock transition. Indentifying the glue logic functionality requires developing state transition tables.

Next-state table

A next-state table is simply a map of current state, at time *t*, to the desired next state, at time *t*+1. Because the states are determined by flip flop outputs, a two-bit state can be defined by (Q_1,Q_0), where Q_1 and Q_0 are the outputs of two flip flops. The next-state table for the state diagram in Fig. 1 is shown in Fig. 2. This table is similar to a logic truth table, but here the next-state outputs will only occur at a future time (next clock cycle).

current state		*next state*	
Q_1^t	Q_0^t	Q_1^{t+1}	Q_0^{t+1}
0	0	0	1
0	1	1	1
1	0	0	0
1	1	1	0

Fig. 2. Next-state table for the 2-bit sequencer in Fig. 1.

Next-state excitation table

Now that we have a table of the desired next-state outputs, we need to determine what the flip flops inputs must be to excite the proper flip flop output transitions. We'll call this mapping a *next-state excitation table*. For a flip flop, an excitation table identifies the input values needed to generate all possible transitions. Since a flip flop has two possible output states (0, 1), there are four possible transitions (1→ 1, 1→ 0, etc.) Clearly, a table of which input excitations would generate each output transition will be a direct function of the type of flip flop used; an SRFF would not have the same excitation tables as a JKFF, etc. Let's first look at flip flop excitation tables and then expand them to next-state excitation tables for a given state machine sequence. Fig. 3 shows the truth and excitation tables for an SRFF. Fig. 4 shows the truth table and excitation table for a JKFF.

truth table			
S	R	Q^{t+1}	action
0	0	Q^t	hold
0	1	0	reset
1	0	1	set
1	1	-	n/a

excitation table			
Q^t → Q^{t+1}	action	S	R
0 → 0	hold (00) reset (01)	0	*X*
0 → 1	set (10)	1	0
1 → 0	reset (01)	0	1
1 → 1	hold (00) set (10)	*X*	0

Fig. 3. Truth table and excitation table for an SR flip flop. *X* = don't care.

truth table			
J	K	Q^{t+1}	action
0	0	Q^t	hold
0	1	0	reset
1	0	1	set
1	1	$\overline{Q^t}$	toggle

excitation table			
Q^t → Q^{t+1}	action	J	K
0 → 0	hold (00) reset (01)	0	*X*
0 → 1	set (10) toggle (11)	1	*X*
1 → 0	reset (01) toggle (11)	*X*	1
1 → 1	hold (00) set (10)	*X*	0

Fig. 4. Truth table and excitation table for a JK flip flop.

Exercise 1: Using only the truth table, complete the JKFF excitation table showing what values J and K must be in order to generate each of the output transitions.

We are now in a position to combine the state machine's next-state table with the flip flop's excitation table to form the desired state machine next-state excitation table. This is fairly straightforward so we'll illustrate the process with an example. Fig. 5 shows the next-state excitation table for the state sequence defined by the state diagram in Fig. 1 and the next-state table in Fig. 2 when SR flip flops are used. Notice there are two output bits, Q_1 and Q_0, so two input S-R pairs must be defined in the table, one for each flip flop.

current state		*next state*		*bit 1 inputs*		*bit 0 inputs*	
Q_1^t	Q_0^t	Q_1^{t+1}	Q_0^{t+1}	S_1	R_1	S_0	R_0
0	0	0	1	0	*X*	1	0
0	1	1	1	1	0	*X*	0
1	0	0	0	0	1	0	*X*
1	1	1	0	*X*	0	0	1

Fig. 5. Next-state excitation table for Fig. 1 state machine implemented with SR flip flops.

The first four columns of Fig. 5 are simply a copy of the next-state table in Fig. 2 repeated for convenience. The task of filling in all of the S and R cells requires mapping specific transitions from Table 3 into each S and R cell. For example, the top row of cells sees a Q_1^t =0 → Q_1^{t+1} = 0 transition for bit 1 and a Q_0^t =0 → Q_0^{t+1} = 1 transition for bit 0. Thus, S_1 and R_1 are filled in by looking at the transition 0 → 0. Fig 3 shows that, for 0 → 0, S should be a 0 and R should be *X* (don't care). Similarly, S_0 and R_0 are filled in by looking at the transition 0 → 1, where Fig 3 shows that S should be a 1 and R should be 0. Completion of the second and third rows for Fig. 5 is described by Examples 2 and 3 below. The final row is completed without description, and verification of this row is left as an exercise. Completion of the next-state excitation table when a JKFF is used instead of an SRFF is left as an exercise.

Example 2: What values for S and R are needed to complete the *second* row of the next-state excitation table in Fig. 5?

Answer: The second row of cells sees

Q_1^t =0 → Q_1^{t+1} = 1, thus 0→1 transition for bit 1

Q_0^t =1 → Q_0^{t+1} = 1, thus 1→1 transition for bit 0.

S_1 and R_1 are filled in by looking at the transition 0 → 1, where Fig 3 shows that S should be a 1 and R should be 0.

S_0 and R_0 are filled in by looking at the transition 1 → 1, where Fig 3 shows that S should be *X* and R should be 0. These results are included in Fig. 5.

Example 3: What values for S and R are needed to complete the *third* row of the next-state excitation table in Fig. 5?

Answer: The third row of cells sees

Bit 1: Q_1^t =1 → Q_1^{t+1} = 0. Thus S_1 = 0 and R_1 = 1.

Bit 0: Q_1^t =0 → Q_1^{t+1} = 0. Thus S_0 = 0 and R_0 = *X*.

These results are included in Fig. 5.

Exercise 2: Following examples 2 and 3, verify that row 4 of the table in Fig. 5 is correct.

Exercise 3: Construct a next-state excitation table like Fig. 5 when a JKFF is used rather than an SRFF.

Input logic

The next-state excitation table provides all of the information necessary to implement the logic for the inputs of each flip flop used to create the state machine. For example, Fig. 5 shows what all S and R inputs would be needed to implement the state machine from Fig. 1 using SR flip flops. These inputs are determined entirely by the *current state* flip flop outputs, namely Q_0 and Q_1 for our example 2-bit sequencer state machine. The inputs must be determined by the current-state output values so that the inputs are set before the next clock transition, when the flip flop outputs will change to their next-state values. To more clearly illustrate the logic needed to set each of the S and R inputs, Fig. 6 shows the truth table for each of the inputs based on the current state output values. This information is already available in Fig. 5 but simply reformatted in Fig. 6 to highlight the necessary input logic.

Q_1^t	Q_0^t	S_1
0	0	0
0	1	1
1	0	0
1	1	*X*

Q_1^t	Q_0^t	R_1
0	0	*X*
0	1	0
1	0	1
1	1	0

Q_1^t	Q_0^t	S_0
0	0	1
0	1	*X*
1	0	0
1	1	0

Q_1^t	Q_0^t	R_0
0	0	0
0	1	0
1	0	*X*
1	1	1

Fig. 6. Input truth tables for the next-state excitation table in Fig. 5.

With the information in Fig. 6, we could directly generate the following logic expressions for the inputs:

$$S_1 = \overline{Q_1} \cdot Q_0, \qquad R_1 = Q_1 \cdot \overline{Q_0}, \qquad S_0 = \overline{Q_1} \cdot \overline{Q_0}, \qquad R_0 = Q_1 \cdot Q_0$$

However, it is better (less complex circuitry) to reduce the truth table and determine the minimized logic expression for each input. This is easily accomplished using Karnaugh maps. The k-maps for each input are shown in Fig. 7. From these k-maps we can determine the following minimized logic expression for each input as a function of only flip flop outputs:

$$S_1 = Q_0, \qquad R_1 = \overline{Q_0}, \qquad S_0 = \overline{Q_1}, \qquad R_0 = Q_1$$

S_1

Q_1 \ Q_0	**0**	**1**
0	0	1
1	0	*X*

R_1

Q_1 \ Q_0	**0**	**1**
0	*X*	0
1	1	0

S_0

Q_1 \ Q_0	**0**	**1**
0	1	*X*
1	0	0

R_0

Q_1 \ Q_0	**0**	**1**
0	0	0
1	*X*	1

Fig. 6. Input signal Karnaugh maps for the Fig. 1 state machine using SR flip flops.

Interestingly, the inputs to FF1 are both determined by outputs from FF0, and visa versa. This is purely a consequence of the specific state sequence chosen for this example. Other state sequences would generate different minimized input functions.

Implementation of State Machine using SR Flip Flops

The final step to implementing a state machine involves realizing the input "glue logic" in digital gates and connecting this circuitry to the flip flops that from the core state machine. For simple state machines such as sequencers, the core circuit maintains a regular structure that can be extended to any number of states (output bits). Fig. 7 shows the core structure of a 2-bit

sequencer state machine implemented with SR flip flops. The outputs are Q_1 and Q_0. The external clock input goes to all flip flops simultaneously. There are no other external inputs in this simple example, although in general external inputs could be used to set flip flop inputs (this would require incorporating those external inputs into the next-state excitation tables and complicate design beyond the scope of this introduction). In our simple example, the flip flop S and R inputs are determined by logical manipulations of internal signals, as represented by the "?" boxes in the schematic. When we substitute the non-minimized S and R logic expressions (shown above) derived from the Fig. 6 truth tables, our example state machine of Fig. 1 is realized by the schematic shown in Fig. 8, where only the Q output is used (not Q'). If, instead, we use the minimized logic expressions and use both Q and Q' outputs, the circuit reduces to that shown in Fig. 9. Notice that no glue logic is actually needed for this state sequence. The circuit could be readily expanded to include an asynchronous reset function using flip flops with reset inputs. A similar schematic for this 2-bit example sequencer can be realized using JK flip flops, although the glue logic (or connections) would be different because the JKFF excitation table is different than the SRFF. The JKFF implementation is left as an exercise for the reader.

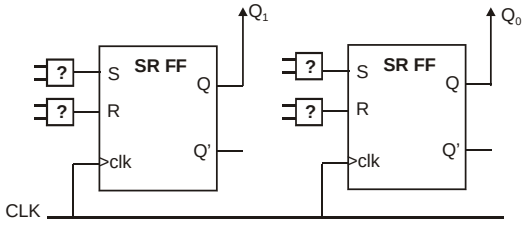

Fig. 7. Core structure of a 2-bit SRFF state machine.

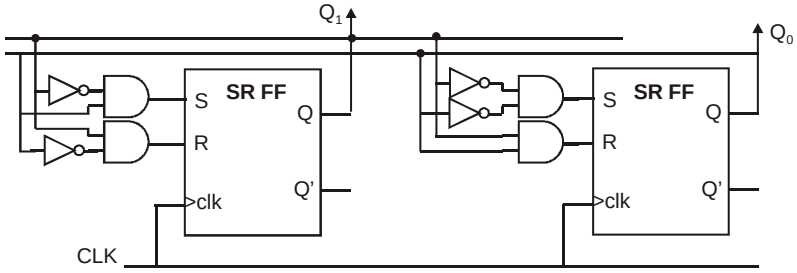

Fig. 8. Overly complex SRFF circuit implementation of the state machine defined by Fig. 1.

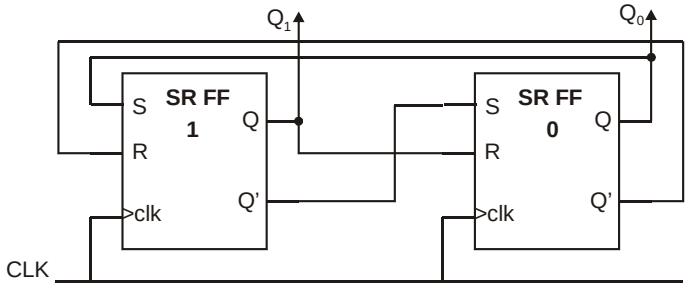

Fig. 9. Minimized SRFF circuit implementation of the state machine defined by Fig. 1.

Exercise 4: Redraw the Fig. 9 schematic using flip flops with reset inputs to implement an asynchronous reset function.

Exercise 5: Using results from Exercise 3, determine the glue logic needed for a JKFF implementation of the Fig. 1 2-bit sequencer and construct the schematic to realize this circuit.