

## 6 Counters, Registers, and State Machines II

The general scheme for a *state machine* is given in Fig. 33. It has  $n$  bits of memory,  $k$  inputs, and  $m$  outputs. It consists of a synchronous data register (lower box) which stores the machine's *present state*. A set of separate flip-flops can be used for this, as long as they are clocked synchronously. The logic in the upper box acts upon the current state, plus any inputs, to produce the machine's next state, as well as any outputs. Upon each pulse of the clock input **CLK**, the machine is moved from the present state to the next state. We will introduce this topic using counters as examples, then moving to more general applications. We will see, in fact, that the state machine prepresents a simple *processor*: The inputs can be generalized to be the processor program and the logic might be replaced by a random-access memory (RAM).

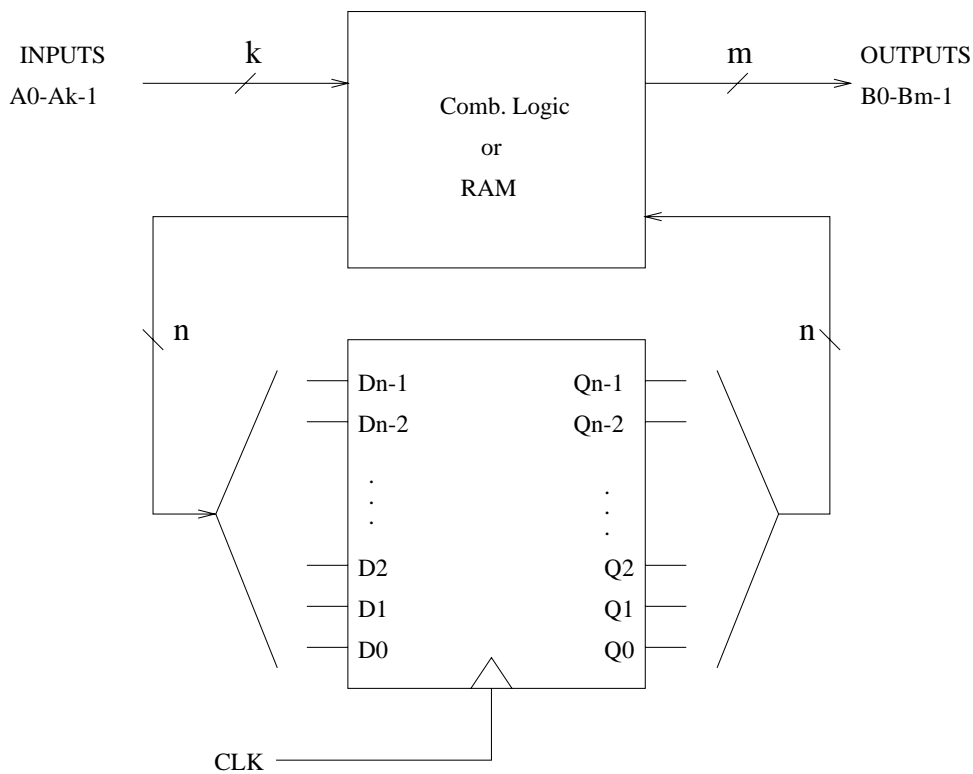


Figure 33: General scheme for state machine.

The strategy for applying this scheme to a given problem consists of the following:

1. Identify the number of required states,  $\ell$ . The number of bits of memory (*e.g.* number of flip-flops) required to specify the  $m$  states is at minimum  $n = \log_2(m)$ .
2. Make a *state diagram* which shows all states, inputs, and outputs.
3. Make a truth table for the logic section. The table will have  $n + k$  inputs and  $n + m$  outputs.
4. Implement the truth table using our combinational logic techniques.

## 6.1 State Machine Introduction: Synchronous Counters

Counters implemented as state machines are always *synchronous*, that is the entire circuit is in phase with the clock. Recall that our previous “ripple” counters were asynchronous — logic was initiated at different times throughout the circuit. Synchronous systems are essential whenever a sequential system requires more than a very modest speed or complexity.

### 6.1.1 Example: Up/down 2-bit Synchronous Counter

A 2-bit counter requires 4 states, with each state corresponding to one of the 4 possible 2-bit numbers. Hence, 2 bits of memory are required. We will use 2 flip-flops (D-type) to implement this. The state diagram is given in Fig. 34. Each circle represents one of the states, and the arrows represent a clock pulse which offers a transition to another state (or possibly to remain at the present state). The 4 states are specified by the 2 bits of memory:  $A = 00$ ,  $B = 01$ ,  $C = 10$ ,  $D = 11$ . Note that we are free to label the states as we choose, as long as they are uniquely specified. However, in this case it is easiest to choose labels which correspond to our desired outputs, that is the 2-bit binary sequence 00, 01, 10, and 11. Hence, these labels are equivalent to our desired outputs, call them  $Q_1Q_0$ , which are available at each state. (Note that the lettered labels  $A$ – $D$  are superfluous; they could be omitted.)

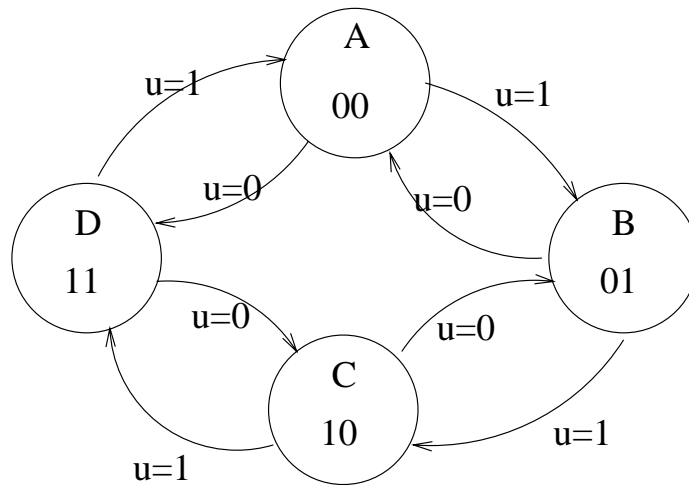


Figure 34: State diagram for 2-bit up/down synchronous counter.

Our processor has one input bit  $u$ , which programs the up-counting ( $u = 1$ ) or down-counting ( $u = 0$ ) functions. In this case, the state machine outputs are the two bits of the present state,  $Q_1Q_0$ , so we do not reproduce them in our truth table. The truth table for the logic is below.

$u$	Present State		Next State	
		$Q_1Q_0$		$D_1D_0$
1	A	00	B	01
1	B	01	C	10
1	C	10	D	11
1	D	11	A	00
0	A	00	D	11
0	D	11	C	10
0	C	10	B	01
0	B	01	A	00

We can now invoke the logic as usual. We have 2 “outputs”,  $D_0$  and  $D_1$ , which are to be evaluated separately. From the truth table, or using a K-map, we see that

$$D_1 = \overline{u \oplus (Q_0 \oplus Q_1)}; \quad D_0 = \overline{Q_0}$$

### 6.1.2 Example: Divide-by-Three Synchronous Counter

Our state machine is supposed to count input pulses (input at the CLK) and set an output bit HIGH on every 3<sup>rd</sup> input pulse. Note that this could represent either a 2-bit (total) counter, or more generally the 2 least-significant bits of a many-bit counter.

We require 3 states, therefore we need 2 bits of memory (2 D-type flip-flops, for example). These 2 flip-flops can describe 4 states, so we will have one “unused” state. A state diagram is shown in Fig. 35, with one way of labelling the states and output bit (called  $p$ ) given.

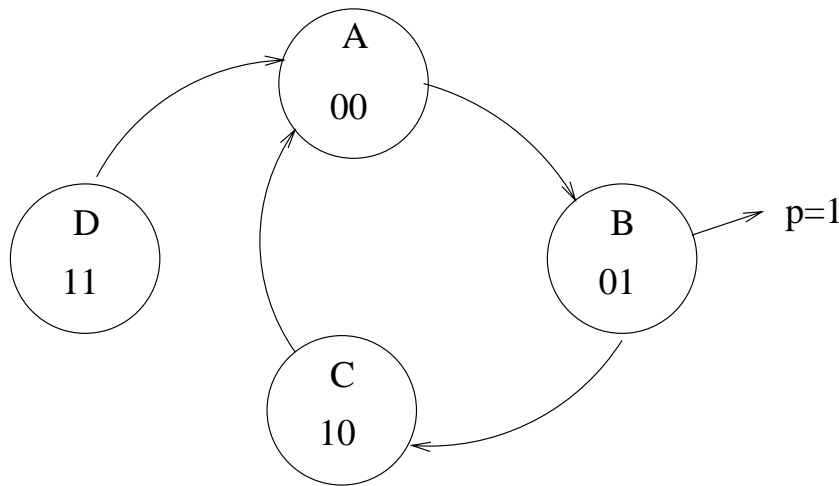


Figure 35: State diagram for a divide-by-3 synchronous counter.

The truth table for the combinational logic is below. It is important that the “extra state”  $D = 11$  be given an exit path, otherwise your processor may end up there upon power-up and remain stuck. (This effect has probably come to your attention with the “frozen” computer, which may require a reboot.) Also, note that we could have taken the output  $p$  from any of the states  $A$ ,  $B$ , or  $C$ .

Present State		Next State		Output
$Q_1Q_0$		$D_1D_0$		$p$
A	00	B	01	0
B	01	C	10	1
C	10	A	00	0
D	11	A	00	0

What are the logic expressions for our 3 “outputs” of this truth table ( $D_1$ ,  $D_2$ , and  $p$ )? How would this be implemented with D-type flip-flops and logic gates? With JK flip-flops replacing the D-type?