RTL Overview (excerpts)

"Principles of Digital Design", Daniel D. Gajski

Prentice Hall (1997)

pp. 319 – 322, 328 – 336

CHAPTER

8

Register-Transfer Design

In Chapter 7 we designed various storage components, such as registers, counters, memories, stacks, and queues; in Chapter 5 we designed combinatorial components such as ALUs, comparators, shifters, selectors, buses, ROMs and PLAs. In designing application-specific integrated circuits (ASICs) and standard processors, we group those components into control units and datapaths. Each ASIC or processor consists of at least one control unit and one datapath, although many ASICs contain multiple control units and datapaths. To synthesize register-transfer designs we introduce the model of an FSM with a datapath (FSMD) and present several different ways to specify register-transfer designs,

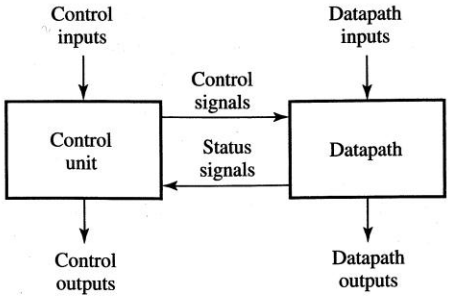
including popular algorithmic-state-machine (ASM) charts. We then explain the techniques for converting such an ASM chart into an design implementation consisting of a control unit and a datapath. We also describe design techniques for cost and performance optimization of these implementations. While register-transfer design is the focus of this chapter, we cover the design of standard processors in Chapter 9 as a special case of register-transfer design. Since the processor instruction set can be described by a restricted type of ASM chart, the processor architecture can also be thought of as a special case of the FSMD model.

8.1 DESIGN MODEL

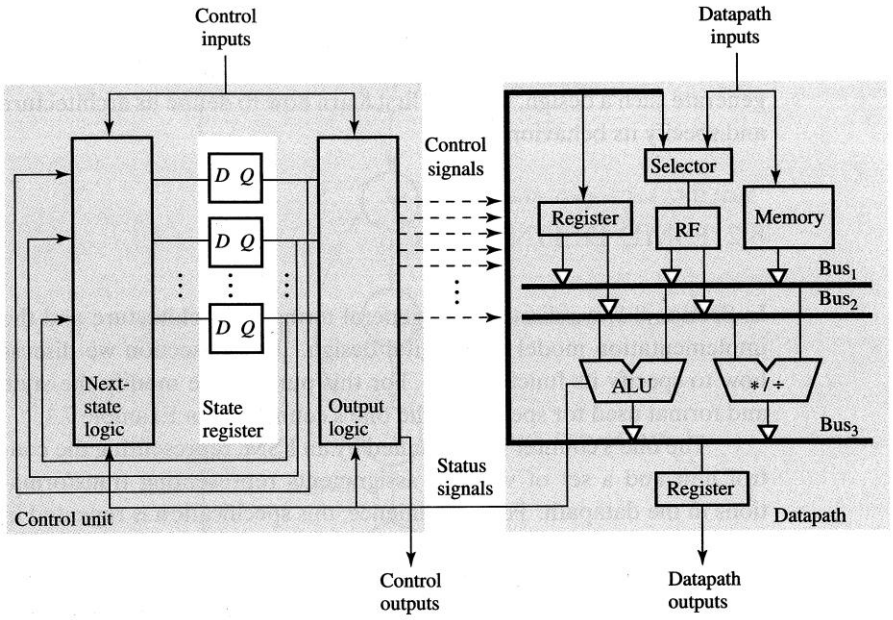
In Example 7.1 we implemented the one's-count algorithm with a standard datapath and a custom control unit. The control unit had eight states, two input signals, and 14 output signals. The input signals were the external signal *Start* and a status signal from the datapath (*Data* = 0). The output signals were the external signal *Done* and datapath control-signals. The datapath contained a register file, an ALU, and a shifter, and had an input port and an output port. The 16-bit operand *Data* was entered into the datapath through the input port at the beginning of the operation, and the result *Ocount* was outputted at the end, through the output port.

Similar to the one's counter, each digital design consists of a control unit and a datapath. As shown in Figure 8.1(a), the datapath has two types of I/O ports. One type of I/O ports are data ports, which are used by the outside environment to send and receive data to and from the microchip. The data could be of type integer, floating point, or characters, and it is usually packed into one or more words. The data ports are usually 8, 16, 32, or 64 bits wide. The other type of I/O ports are control ports, which are used by the control unit to control the operations performed by the datapath and receive information about the status of selected registers in the datapath.

As shown in Figure 8.1(b), the datapath consists of storage units such as registers, register files, and memories, and combinatorial units such as ALUs, multipliers, shifters, and comparators. These units and the input and output ports are connected by buses. The datapath takes the operands from storage units, performs the computation in the combinatorial units, and returns the results to storage units during each state, which is usually equal to one clock cycle. The selection of operands, operations, and the destination for the result is controlled by the control unit by setting proper values of datapath control signals. The datapath also indicates through status signals when a particular value is stored in



(a) High-level block diagram



(b) Register-transfer-level block diagram

FIGURE 8.1
Design model.

a particular storage unit or when a particular relation between two data values stored in the datapath is satisfied.

Similar to the datapath, a control unit has a set of input and a set of output signals. Each signal is a Boolean variable that can take a value of 0 or 1. There are two types of input signals: external signals and status signals. External signals represent the conditions in the external environment on which an ASIC must respond. The *Start* signal in Example 7.1, which starts the one's counter, is such an input signal. On the other

hand, the status signals represent the state of the datapath. Their value is obtained by comparing values of selected variables stored in the datapath. For example, *Data* = 0 in Example 7.1 was such a signal whose value was equal to 1 when the value of Data was equal to 0 and 0 when *Data* value was not equal to 0.

There are also two types of output signals: external signals and datapath control signals. External signals identify to the environment that an ASIC has reached a certain state or finished a particular computation. The datapath controls select the operation for each component in the datapath.

Each ASIC implementation follows this general architecture, although two ASICs may differ in the number of control units and datapaths, the number of components and connections in the datapath, the number of states in the control unit, and the number of I/O ports. To generate such a design, we must first learn how to define its architecture and specify its behavior.

PRESENT STATE	NEXT STATE		CONTROL AND DATAPATH ACTIONS	
	CONDITION,	STATE	CONDITION,	ACTIONS
s_0	$Start = 0$,	s_0		$Done = 0$
	$Start = 1$,	s_1		$Output = Z$
s_1		s_2		$Data = Inport$
s_2		s_3		$Ocount = 0$
s_3		s_4		$Mask = 1$
s_4		s_5		$Temp = Data$ AND $Mask$
s_5		s_6		$Ocount = Ocount + Temp$
s_6	$Data \neq 0$,	s_4		$Data = Data>>1$
	$Data = 0$,	s_7		
s_7		s_0		$Done = 1$
				$Output = Ocount$

(c) State–action table

FIGURE 8.3
Continued.

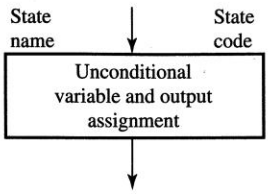
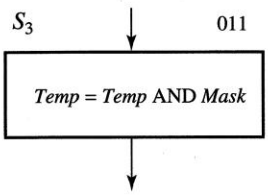
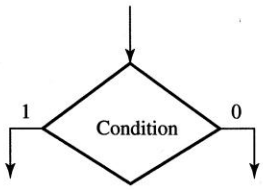
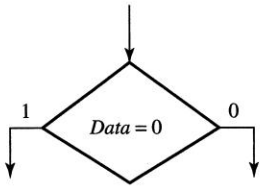
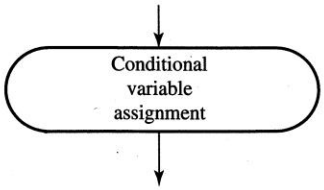
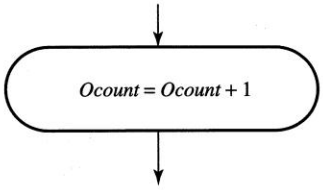
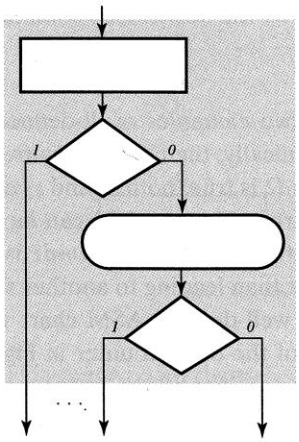
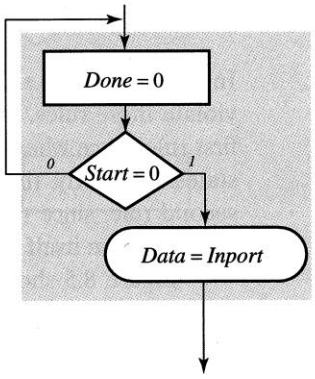
8.3 ALGORITHMIC-STATE-MACHINE CHARTS

In Section 8.2, we defined the FSMD model and explained how to develop a state-action table for specifying FSMDs. In this section we introduce an alternative graphic form for specifying FSMDs, referred to as an **algorithmic-state-machine** (ASM) **chart**. In general, an ASM chart is fully equivalent to the state-action tables described above: that is, for every state-action table, there is at least one ASM chart that describes the same functionality. In many cases, however, ASM charts can have a slight advantage for human consumption, since they explicitly show the paths from one state to another that are less visible in a state-action table. The following definition makes these advantages apparent.

As shown in Table 8.1, the ASM chart represents the FSMD in terms of four basic components: a state box, a decision box, a conditional output box, and the ASM block. Each state in the ASM chart is indicated by a **state box**, which contains the set of unconditional assignments to variables and output ports in the datapath. This state box has its name placed on the top of the box on the left side. If known, the state code can be placed on the top of the box on the right side. However, the state code is not usually known when the ASM chart is first drawn, as it must be added later during the process of state assignment described in Chapter 6.

The **decision box** describes the condition under which the FSMD will execute specific actions in the datapath and select the next state. These conditions can refer either to external control inputs or to status

TABLE 8.1
ASM Symbols

NAME	DEFINITION	EXAMPLE
State box		
Decision box		
Condition box		
ASM block		

signals. Note that each decision box has two exit paths, one path that will be taken when the enclosed condition is true, and the other when it is false. These two paths are usually indicated by a 1 for true and a 0 for false.

The **conditional output box** describes variable or output assignments that are executed under conditions specified by one or more decision boxes. The rounded corners of a condition output box differentiate it from a state box.

Finally, the **ASM block** is the complex structure that incorporated one state box and a serial–parallel network of decision boxes and conditional output boxes. The ASM block is usually represented with dashed lines around it. As you can see in Table 8.1, it has one input and can have any number of output paths that are generated by its particular network of decision boxes.

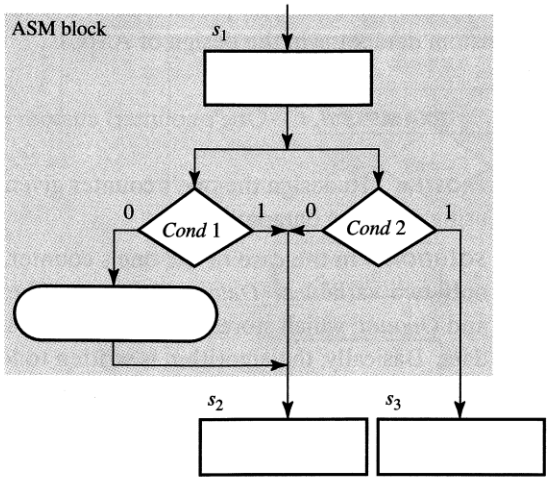
In general, an **ASM chart** will consist of one or more interconnected ASM blocks, arranged so that each output path from an ASM block connects to a single state box. Each ASM block describes the operations executed in one state. In other words, each block is equivalent to a row in the state-action table described in Section 8.2, the only difference being the way they present their conditions: In the state-action table, the conditions for the selection of the next state and the execution of the datapath operations were separated, whereas the ASM block combines them into a binary tree of decision and condition boxes.

When specifying an FSMD with a ASM chart, there are two rules we must follow:

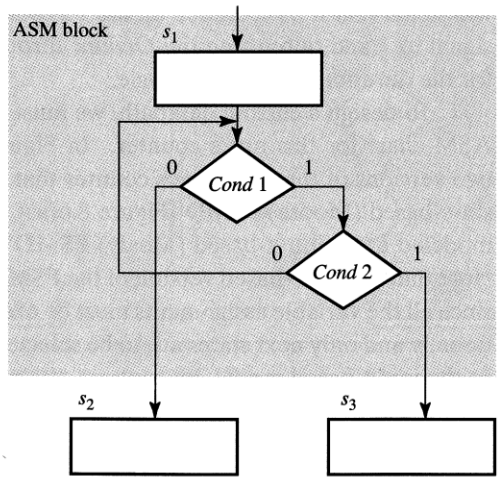
1. The chart must define a unique next state for each state and set of conditions.
2. Every path defined by the network of condition boxes must lead to another state.

In Figure 8.4, we show two examples of ill-defined ASM charts that violate these rules. Specifically, the chart in Figure 8.4(a) violates the first rule, since when *Cond2* is true, both s_2 and s_3 are specified as next states. Similarly, the chart in Figure 8.4(b) can be seen to violate the second rule, since the path defined when *Cond1* is true and *Cond2* is false loops on itself rather than leading to another state.

Figure 8.5 shows a well-defined ASM chart which is equivalent to the state-action table of the one's counter in Figure 8.3(c). As you can see, this chart clearly specifies all the states, next-state transitions, and datapath actions (variable assignments) that were represented in the state-action table. Note also that this ASM chart clearly shows the loops which were less visible in the state-action table, as well as showing all the conditional paths as a tree rather than specifying each path by a conditional expression. We should realize, of course, that an ASM chart may become too bulky when the FSMD in question has a large number of states and conditions.



(a) Undefined next state



(b) Undefined exit path

FIGURE 8.4
Ill-defined ASM charts.

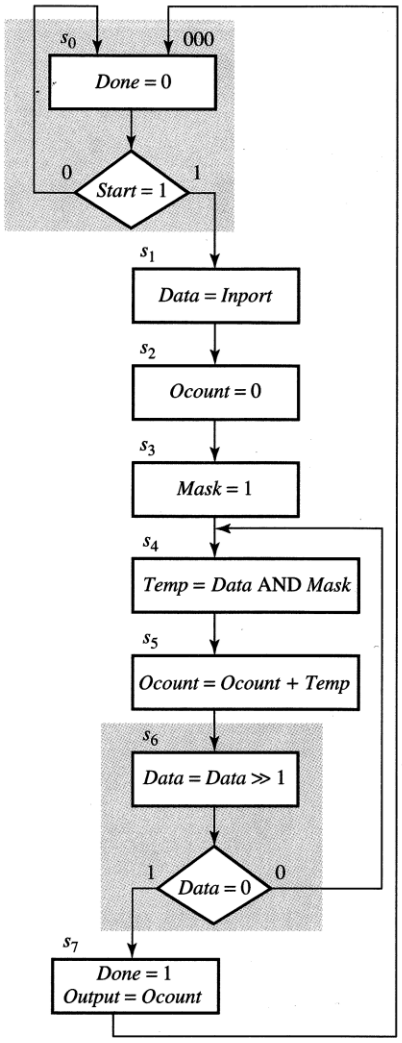


FIGURE 8.5
ASM chart for one's counter.

To understand the role of ASM charts and state-action tables in the design process, we redesign the one's counter given in Example 7.1, this time using a custom datapath instead of the standard one. In comparison with a standard datapath, a custom datapath would be tuned to a particular algorithm and might therefore require fewer components and interconnections. Since it uses fewer components, the custom data-

path may also exhibit higher performance. For these reasons, designers use custom datapaths in the design of ASICs.

EXAMPLE 8.1 One's counter: custom design

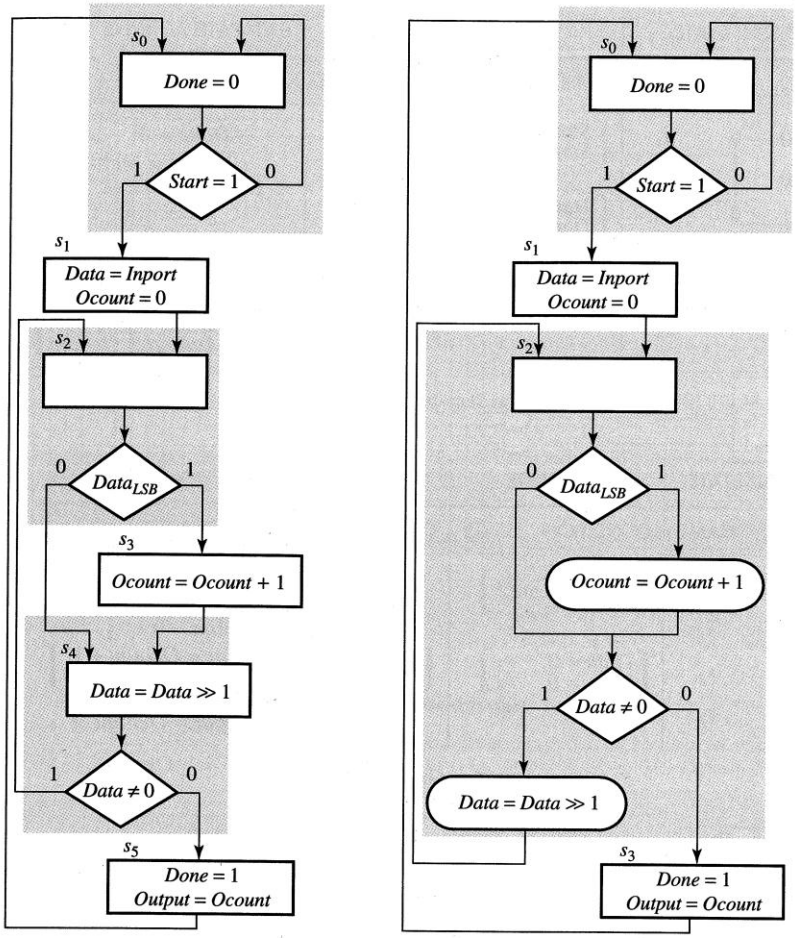
PROBLEM Redesign the one's counter given in Example 7.1 using a custom datapath.

SOLUTION In the case of the one's counter, we really need only two variables: *Data*, which stores the incoming data, and *Ocount*, which stores the number of 1's counted in that data. Basically, the algorithm is written to look at the least-significant bit of data, $Data_{LSB}$, and add 1 to the *Ocount* whenever $Data_{LSB} = 1$. At this point, *Data* will be shifted one position to the right and the same sequence will be repeated as long as $Data \neq 0$. As before, the one's counter waits for *Start* to become 1 to input the data from the input port *Inport*. Whenever $Data = 0$, though, the one's counter will signal that it has counted all the 1's by setting the *Done* signal to 1 and outputting the *Ocount* through the *Outport* for the duration of one clock cycle.

To design a custom datapath, we must first develop an ASM chart for this one's counter. In Figure 8.6 we show two versions of the same one's counter that is modeled as a state-based (Moore) FSMD [Figure 8.6(a)], and one that is modeled as an input-based (Mealy) FSMD [Figure 8.6(b)]. Note that the state-based version of the FSMD has six states, since all the variable assignments must be executed unconditionally and only next states are to be selected conditionally. In the input-based model, by contrast, the number of states has been reduced to four since the variable assignments are to be executed conditionally together with the conditional selection of next states.

Either one of these ASM charts can easily be converted to a state-action table, which enables us to derive Boolean equations for the next-state and output logic in the control unit. In Figure 8.7(a) and (b), we show the state-action tables that correspond to the state-based and input-based versions of the one's counter.

The implementation of this same one's counter is shown in Figure 8.8. Note that the datapath is the same in both versions since both counters store only two variables and perform only two operations—that is, both versions shift the value in the variable *Data* and increment the value in the



(a) State-based (Moore) chart

(b) Input-based (Mealy) chart

FIGURE 8.6
ASM charts for one's counter (custom design).

variable *Ocount*. Given these two operators, the datapath needs only a shift register with parallel load, which loads data from the *Inport* and an up/down-counter with parallel load, which loads 0 at the start of one's counting and increments its content whenever $Data_{LSB}$ is equal to 1.

The control unit of this one's counter consists of a state register and the next-state and output logic. For the state-based version, we would need a 3-bit state register, whereas the input-based version would require a 2-bit register. The encoding of states is also given in the state-action tables,

PRESENT STATE				NEXT STATE		DATAPATH ACTIONS	
Q_2	Q_1	Q_0	NAME	CONDITION,	STATE	CONDITION,	OPERATIONS
0	0	0	s_0	$\begin{bmatrix} Start = 0, \\ Start = 1, \end{bmatrix}$	$\begin{matrix} s_0 \\ s_1 \end{matrix}$		$Done = 0$
0	0	1	s_1		s_2		$\begin{bmatrix} Data = Inport \\ Ocount = 0 \end{bmatrix}$
0	1	0	s_2	$\begin{bmatrix} Data_{LSB} = 1, \\ Data_{LSB} = 0, \end{bmatrix}$	$\begin{matrix} s_3 \\ s_4 \end{matrix}$		
0	1	1	s_3		s_4		$Ocount = Ocount + 1$
1	0	0	s_4	$\begin{bmatrix} Data \neq 0, \\ Data = 0, \end{bmatrix}$	$\begin{matrix} s_2 \\ s_5 \end{matrix}$		$Data = Data >> 1$
1	0	1	s_5		s_0		$\begin{bmatrix} Done = 1 \\ Output = Ocount \end{bmatrix}$

(a) State-based table

PRESENT STATE			NEXT STATE		DATAPATH ACTIONS	
Q_1	Q_0	NAME	CONDITION,	STATE	CONDITION,	OPERATIONS
0	0	s_0	$\begin{bmatrix} Start = 0, \\ Start = 1, \end{bmatrix}$	$\begin{matrix} s_0 \\ s_1 \end{matrix}$		$Done = 0$
0	1	s_1		s_2		$\begin{bmatrix} Data = Inport \\ Ocount = 0 \end{bmatrix}$
1	0	s_2	$\begin{bmatrix} Data \neq 0, \\ Data = 0, \end{bmatrix}$	$\begin{matrix} s_2 \\ s_3 \end{matrix}$	$\begin{bmatrix} Data_{LSB} = 1, \\ Data \neq 0, \end{bmatrix}$	$\begin{bmatrix} Ocount = Ocount + 1 \\ Data = Data >> 1 \end{bmatrix}$
1	1	s_3		s_0		$\begin{bmatrix} Done = 1 \\ Output = Ocount \end{bmatrix}$

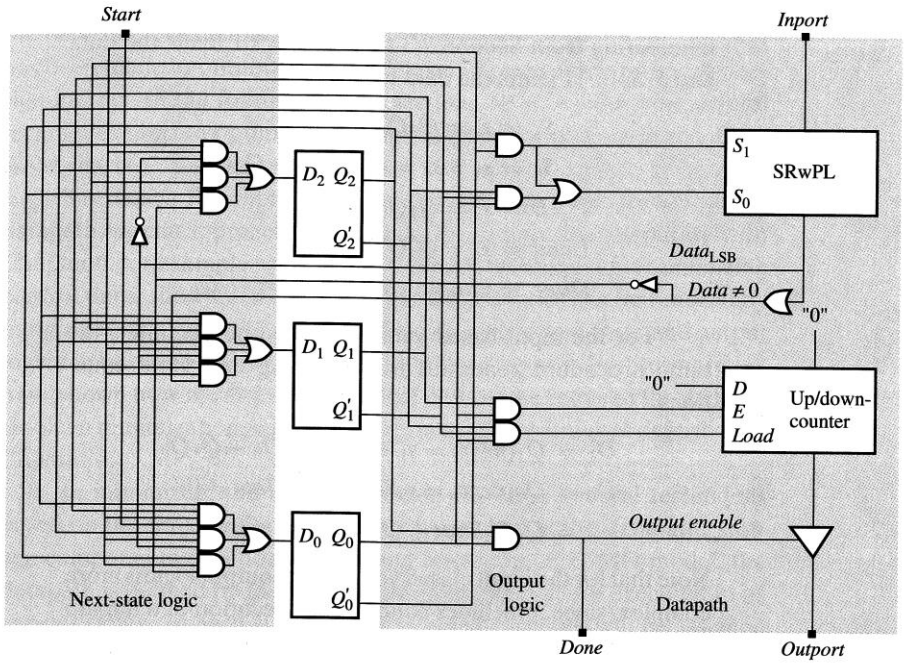
(b) Input-based table

FIGURE 8.7
State-action tables for one's counter.

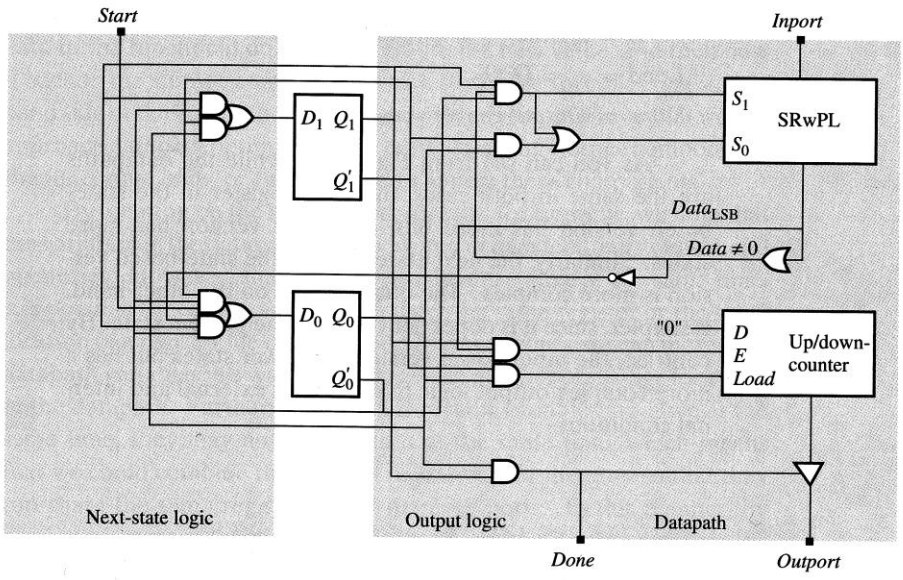
where we have used a natural binary encoding to simplify understanding of the control logic schematic.

When implementing the control logic, we could reduce the number and size of the gates if we note that in the state-based version, states s_2, s_3, s_4, and s_5 can be defined uniquely by Q_1Q_0', Q_1Q_0, Q_2Q_0', and Q_2Q_O. With this in mind we can derive the next-state equations for the state-based version in Figure 8.8(a) directly from the state-action table.

$$\begin{aligned} D_2 &= Q_2(next) = s_2 Data'_{LSB} + s_3 + s_4(Data = 0) \\ &= Q_1Q_0' Data'_{LSB} + Q_1Q_0 + Q_2Q_0'(Data = 0) \\ D_1 &= Q_1(next) = s_1 + s_2 Data_{LSB} + s_4(Data \neq 0) \\ &= Q_2'Q_1'Q_0 + Q_1Q_0' Data_{LSB} + Q_2Q_0'(Data \neq 0) \\ D_0 &= Q_0(next) = s_0 Start + s_2 Data_{LSB} + s_4(Data = 0) \\ &= Q_2'Q_1'Q_0' Start + Q_1Q_0' Data_{LSB} + Q_2Q_0'(Data = 0) \end{aligned}$$



(a) State-based version



(b) Input-based version

FIGURE 8.8
Logic schematics for one's counter.

Similarly, we can derive the output-logic equations remembering from Section 7.3 that $S_1S_0 = 01$ loads the data and $S_1S_0 = 11$ shifts the data to the right:

$$\begin{aligned} S_1 &= s_4 = Q_2Q_0' \\ S_0 &= s_1 + s_4 = Q_2'Q_1'Q_0 + Q_2Q_0' \\ E &= s_3 = Q_1Q_0 \\ Load &= s_1 = Q_2'Q_1'Q_0 \\ Done &= Output\ enable = s_5 = Q_2Q_0 \end{aligned}$$

For the input-based version of the one's counter, the same procedure generates the following next-state equations:

$$\begin{aligned} D_1 &= Q_1(next) = s_1 + s_2 = Q_1'Q_0 + Q_1Q_0' \\ D_0 &= Q_0(next) = s_0Start + s_2(Data \neq 0)' \\ &= Q_1'Q_0'Start + Q_1Q_0'(Data \neq 0)' \end{aligned}$$

Note that for the input-based version, the output logic is more complex, since it includes conditional execution of datapath operations:

$$\begin{aligned} S_1 &= s_2(Data \neq 0) = Q_1Q_0'(Data \neq 0) \\ S_0 &= s_1 + s_2(Data \neq 0) = Q_1'Q_0 + Q_1Q_0'(Data \neq 0) \\ E &= s_2Data_{LSB} = Q_1Q_0'Data_{LSB} \\ Load &= s_1 = Q_1'Q_0 \\ Done &= Output\ enable = s_3 = Q_1Q_0 \end{aligned}$$

As you can see from Figure 8.8, while the datapaths are the same in both cases, the state register in the state-based version has more bits, since this version has more states. Similarly, the next-state logic of the state-based version is more complex. The output logic, on the other hand, is simpler, since it is dependent only on the present state. By contrast, the input-based version has fewer states but has a more complex output logic that includes external and internal conditions.