Example 5.38

The Verilog model *arithmetic_unit* uses functions with descriptive names to make the source code more readable. The combinational circuit synthesized from *arithmetic_unit* is shown in Figure 5-22.

```
module arithmetic_unit (result_1, result_2, operand_1, operand_2,);
 output          [4: 0] result_1;
 output          [3: 0] result_2;
 input           [3: 0] operand_1, operand_2;

 assign result_1 = sum_of_operands (operand_1, operand_2);
 assign result_2 = largest_operand (operand_1, operand_2);

 function [4: 0] sum_of_operands;
 input [3: 0] operand_1, operand_2;

 sum_of_operands = operand_1 + operand_2;
 endfunction

 function [3: 0] largest_operand;
 input [3: 0] operand_1, operand_2;

  largest_operand = (operand_1 >= operand_2) ? operand_1 : operand_2;
 endfunction
endmodule
```

End of Example 5.38

Functions and tasks are both used to improve the readability of a Verilog model and to exploit re-usable code. Funotions are equivalent to combinational logic, and cannot be used to replace code that contains event control (@) or delay control (#) operators. Tasks are more general than functions, and may contain timing controls. Tasks that are to be synthesized may contain event-control operators, but not delay-control operators.

5.14 Algorithmic State Machine Charts for Behavioral Modeling

Many sequential machines implement algorithms (i.e., multistep sequential computations) in hardware. A machine's activity consists of a synchronous sequence of operations on the registers of its datapaths, usually under the direction of a controlling state machine. State-transition graphs (STGs) indicate the transitions that result from inputs that are applied when a state machine is in a particular state, but STGs do not directly

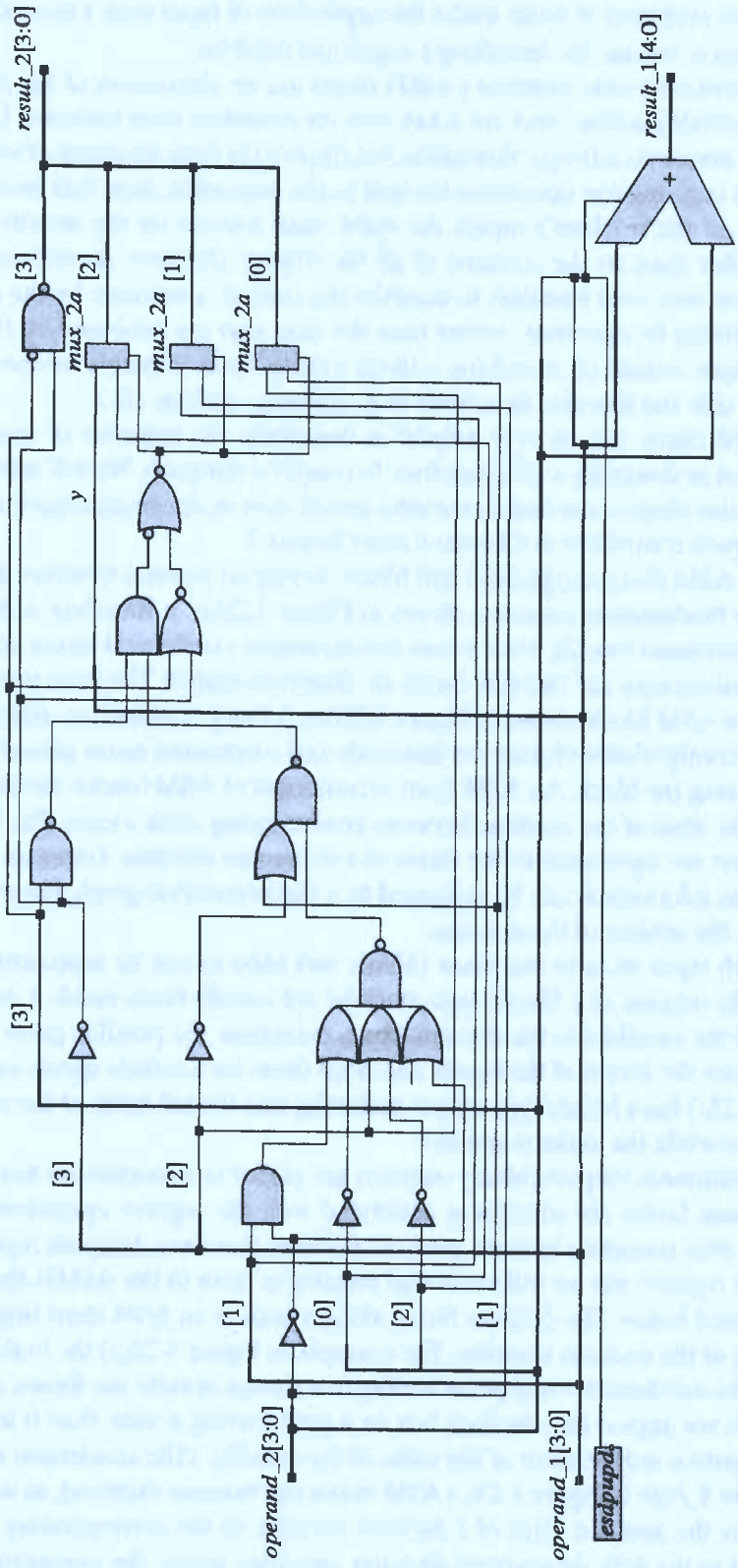


FIGURE 5-22 Circuit synthesized from *arithmetic_unit*.

display the evolution of states under the application of input data. Fortunately, there is an alternative format for describing a sequential machine.

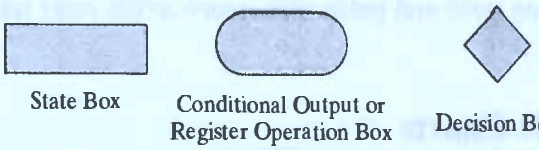
Algorithmic state machine (ASM) charts are an abstraction of the functionality of a sequential machine, and are a key tool for modeling their behavior [1, 2, 6, 7, 8]. They are similar to software flowcharts, but display the time sequence of computational activity (e.g., register operations) as well as the sequential steps that occur under the influence of the machine's inputs. An ASM chart focuses on the activity of the machine, rather than on the contents of all the storage elements. Sometimes it is more convenient, and even essential, to describe the state of a machine by the activity that unfolds during its operation, rather than the data that are produced by the machine. For example, instead of describing a 16-bit counter by its contents we can view it as a datapath unit and describe its activity (e.g., counting, waiting, etc.).

ASM charts can be very helpful in describing the behavior of sequential machines, and in designing a state machine to control a datapath. We will introduce ASM charts in this chapter and make extensive use of them in designing sequential machines and datapath controllers in Chapter 6 and Chapter 7.

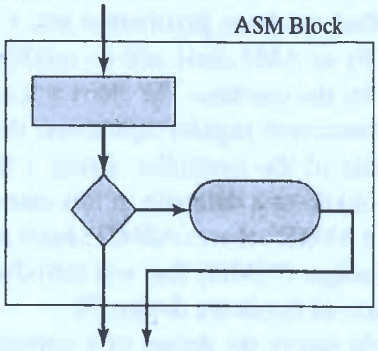
An ASM chart is organized into blocks having an internal structure formed from the three fundamental elements shown in Figure 5.23(a): a state box, a decision box, and a conditional box [2]. State boxes are rectangles, conditional boxes are rectangles with round corners, and decision boxes are diamond-shaped. The basic unit of an ASM chart is an ASM block, shown in Figure 5.23(b). A block contains one state box and an optional configuration of decision diamonds and conditional boxes placed on directed paths leaving the block. An ASM chart is composed of ASM blocks; the state box represents the state of the machine between synchronizing clock events. The blocks of an ASM chart are equivalent to the states of a sequential machine. Given an ASM chart, equivalent information can be expressed by a state-transition graph, but with less clarity about the activity of the machine.

Both types of state machines (Mealy and Moore) can be represented by ASM charts. The outputs of a Moore-type machine are usually listed inside a state box. The values of the variables in the decision boxes determine the possible paths through the block under the action of the inputs. The ASM chart for a vehicle speed controller (see Figure 5-23c) has a Mealy-type output indicating that the tail lights of the vehicle are illuminated while the brake is applied.

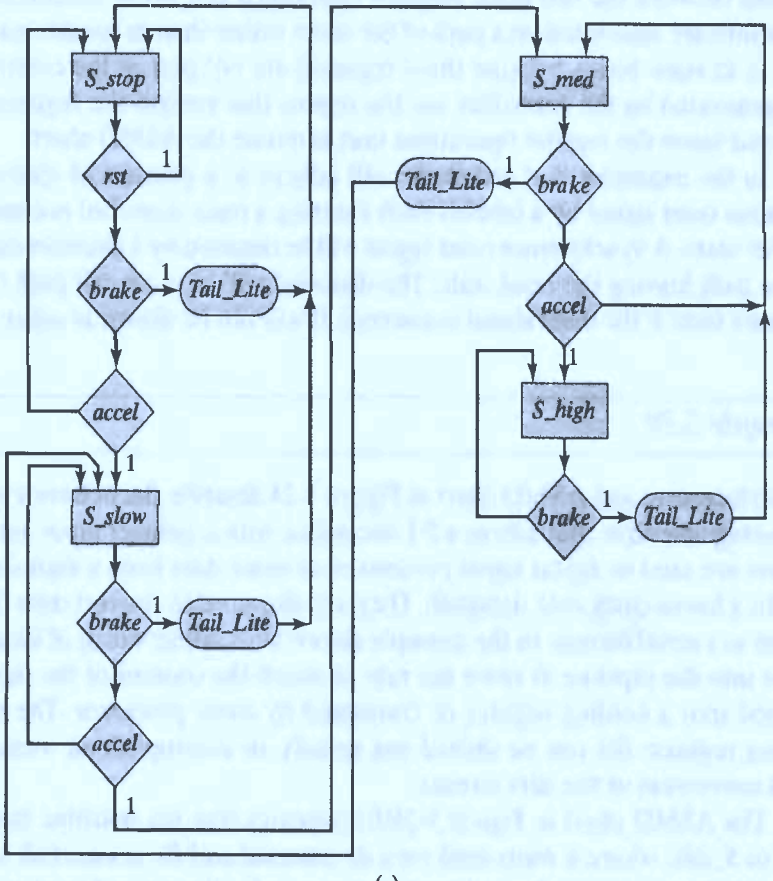
Conditional outputs (Mealy outputs) are placed in a conditional box on an ASM chart. These boxes are sometimes annotated with the register operations that occur with the state transition in more general machines that have datapath registers as well as a state register, but we will avoid that practice in favor of the ASMD charts that will be discussed below. The decision boxes along a path in an ASM chart imply a priority decoding of the decision variables. For example, in Figure 5-23(c) the brake has priority over the accelerator. Only paths leading to a change in state are shown, and if a variable does not appear in a decision box on a path leaving a state, then it is understood that the path is independent of the value of the variable. (The accelerator is not decoded in state *S_high* in Figure 5-23c.) ASM charts can become cluttered, so we sometimes place only the asserted value of a decision variable on the corresponding path and do not label paths with de-asserted decision variables unless the omission would lead to confusion. We also may omit showing default transitions and assertions that



(a)



(b)



(c)

FIGURE 5-23 Algorithmic state machine charts: (a) symbols, (b) an ASM block, and (c) an ASM chart for a vehicle speed controller.

return to the same state and paths that return to the reset state when a reset signal is asserted.

5.15 ASMD Charts

One important use of a state machine is to control register operations on a datapath in a sequential machine that has been partitioned into a controller and a datapath. The controller is described by an ASM chart, and we modify the chart to link it to the datapath that is controlled by the machine. The chart is modified by annotating each of its paths to indicate the concurrent register operations that occur in the associated datapath unit when the state of the controller makes a transition along the path. ASM charts that have been linked to a datapath in this manner are called algorithmic state machine and datapath (ASMD) charts. ASMD charts are motivated by the finite-state machine–datapath paradigm (FSMD) that was introduced in other works as a universal model that represents all hardware design [9].

ASMD charts help clarify the design of a sequential machine by separating the design of its datapath from the design of the controller, while maintaining a clear relationship between the two units. Register operations that occur concurrently with state transitions are annotated on a path of the chart, rather than in conditional boxes on the path, or in state boxes, because these registers are not part of the controller. The outputs generated by the controller are the signals that control the registers of the datapath and cause the register operations that annotate the ASMD chart.

In the examples that follow, we will adhere to a practice of indicating an asynchronous reset signal by a labeled path entering a reset state, but not emanating from another state. A synchronous reset signal will be denoted by a decision diamond placed on the path leaving the reset state. The diamond will have an exit path that returns to the reset state if the reset signal is asserted. It will not be shown at other states.

Example 5.39

The architecture and ASMD chart in Figure 5-24 describe the behavior of *pipe_2stage*, a two-stage pipeline that acts as a 2:1 decimator with a parallel input and output. Decimators are used in digital signal processors to move data from a high-clock-rate datapath to a lower-clock-rate datapath. They are also used to convert data from a parallel format to a serial format. In the example shown here, entire words of data can be transferred into the pipeline at twice the rate at which the content of the pipeline must be dumped into a holding register or consumed by some processor. The content of the holding register, *R0*, can be shifted out serially, to accomplish an overall parallel-to-serial conversion of the data stream.

The ASMD chart in Figure 5-24(b) indicates that the machine has synchronous reset to *S_idle*, where it waits until *rst* is de-asserted and *En* is asserted. Note that transitions that would occur from the other states to *S_idle* under the action of *rst* are not

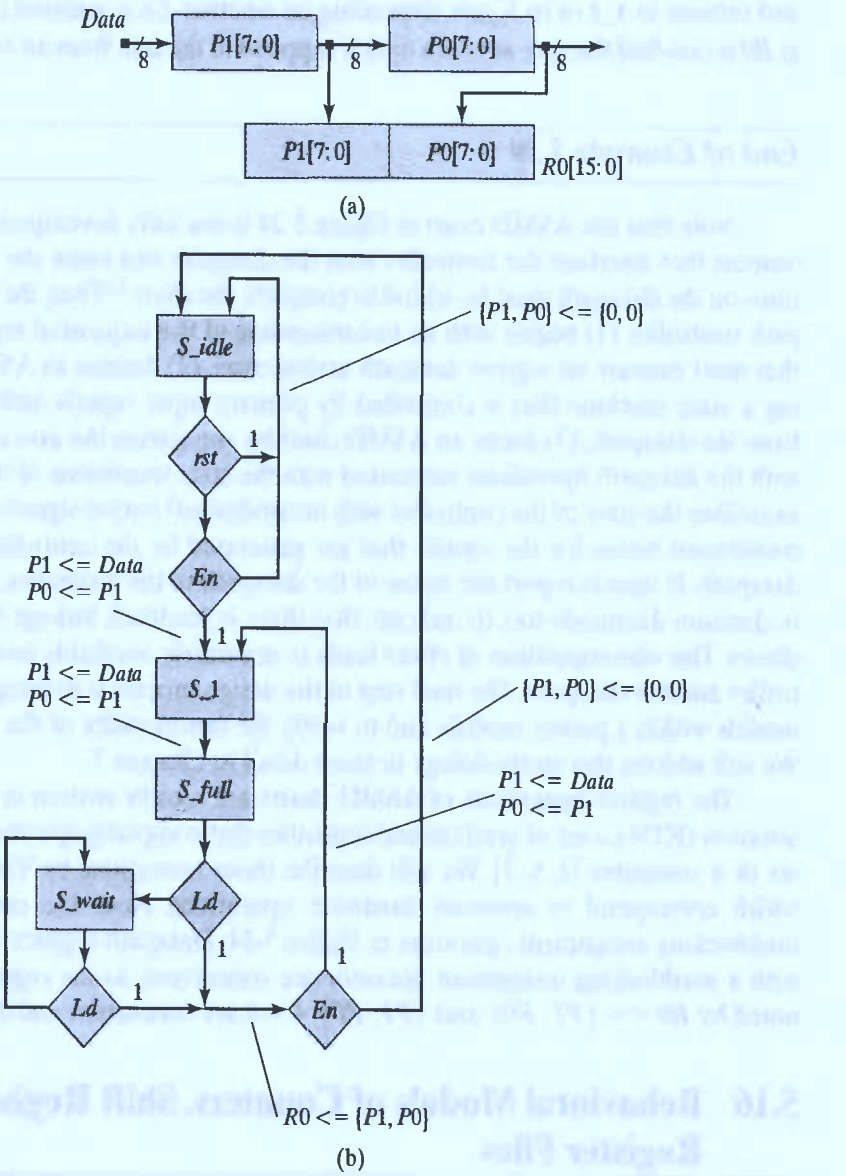


FIGURE 5-24 Two-stage pipeline register: (a) pipeline architecture, and (b) ASMD chart.

shown. With *En* asserted, the machine transitions from *S_idle* to *S_1*, accompanied by concurrent register operations that load the MSByte of the pipe with *Data* and move the content of *P1* to the LSByte (*P0*). At the next clock the state goes to *S_full*, and now the pipe is full. If *Ld* is asserted at the next clock, the machine moves to *S_1* while dumping the pipe into a holding register *R0*. If *Ld* is not asserted, the machine enters *S_wait* and remains there until *Ld* is asserted, at which time it dumps the pipe

and returns to *S_1* or to *S_idle*, depending on whether *En* is asserted too. The data rate at *R0* is one-half the rate at which data is supplied to the unit from an external datapath.

End of Example 5.39

Note that the ASMD chart in Figure 5-24 is not fully developed. The conditional outputs that interface the controller with the datapath and cause the indicated operations on the datapath must be added to complete the chart.[24] Thus, the design of a datapath controller (1) begins with an understanding of the sequential register operations that must execute on a given datapath architecture, (2) defines an ASM chart describing a state machine that is controlled by primary input signals and/or status signals from the datapath, (3) forms an ASMD chart by annotating the arcs of the ASM chart with the datapath operations associated with the state transitions of the controller, (4) annotates the state of the controller with unconditional output signals, and (5) includes conditional boxes for the signals that are generated by the controller to control the datapath. If signals report the status of the datapath to the controller, these are placed in decision diamonds too, to indicate that there is feedback linkage between the machines. This decomposition of effort leads to separately verifiable models for the controller and the datapath. The final step in the design process is to integrate the verified models within a parent module and to verify the functionality of the overall machine. We will address this methodology in more detail in Chapter 7.

The register operations of ASMD charts are usually written in register transfer notation (RTN), a set of symbols and semantics that compactly specifies the instruction set of a computer [2, 6, 7]. We will describe those operations by Verilog's operators, which correspond to common hardware operations. Note the concatenation and nonblocking assignment operators in Figure 5-24. Datapath register operations made with a nonblocking assignment operator are concurrent, so the register transfers denoted by *R0* <= {*P1*, *P0*} and {*P1*, *P0*} <= *0* are concurrent and do not race.

5.16 Behavioral Models of Counters, Shift Registers, and Register Files

The storage elements of counters and registers usually have the same synchronizing and control signals.[25] A counter generates a sequence of related binary words; a register stores data that can be retrieved and/or overwritten under the control of a host

[24] See Problem 29 at the end of the chapter.

[25] An exception is a ripple counter, which connects the output of a stage to the clock input of an adjacent stage [5].

processor. The cells of a shift register exchange contents in a systematic and synchronous manner. Register files are a collection of registers that share the same synchronizing and control signals. Behavioral descriptions of a wide variety of counters, shift registers, and register files are routinely synthesized by modern synthesis tools.

5.16.1 Counters

Example 5.40

Suppose a 3-bit counter has the features that it can count up or down or hold the count. The counter could be modeled by choosing a state consisting of the content of the register holding the count, but instead we choose to associate the state with the activity of the machine, which consists of idling, incrementing, or decrementing. The latter approach allows us to model the counter independently of its word length. We let the mode of counting be determined by a 2-bit input word, *up_dwn*, with options to count up, count down, or hold the count, and included an active low asynchronous reset of the counter. Two versions of the ASM chart for the machine are shown in Figure 5-25—one, Figure 5-25(a), focuses attention on the state transitions of the machine; the other, Figure 5-25(b), shows the conditional output/operation boxes annotated with register operations using the Verilog operator < = to effect concurrent (nonblocking) register assignments. Once the machine's operation is verified in (a), the concurrent register operations that are linked to the state transitions can be included, as shown by the conditional output boxes in Figure 5-25(b).

The activity of the counter has three states: idling (*S_idle*), incrementing (*S_incr*), and decrementing (*S_decr*). The asynchronous active-low reset signal, *reset_* drives the state to *S_idle* and its action is not confined to the active edges of the clock. The signal *reset_* is shown only at *S_idle*, to indicate that *S_idle* is reached from any state when *reset_* is asserted. The machine enters *S_idle* asynchronously from any state under the action of *reset_* and enters synchronously from *S_decr* and *S_incr* if *up_dwn* is 0 or 3. Otherwise, the count is either incremented or decremented.

Note that the ASM charts in Figure 5-25 are independent of the word length of the counter and that they capture the functionality of the machine. They can be adapted to a variety of applications.

A controller and datapath implementation of a sequential machine based on Figure 5-25 would require a 4-bit register to hold *count* and a separate 2-bit register to hold the state. A closer look at the machine suggests even further simplification. The counter can be viewed as having a single (equivalent) state, *S_running*, and there is no need for a state register, only a datapath register for *count*. Two ASMD charts are shown in Figure 5-26, one (a) for a machine with asynchronous reset, and the other (b) for a machine with synchronous reset. The action of *reset_* is to drive the state to *S_running* and flush the register holding *count*. *reset_* is shown as a synchronous entry into *S_running* in Figure 5-26(a). A decision diamond for *reset_* is shown in Figure 5-26(b)

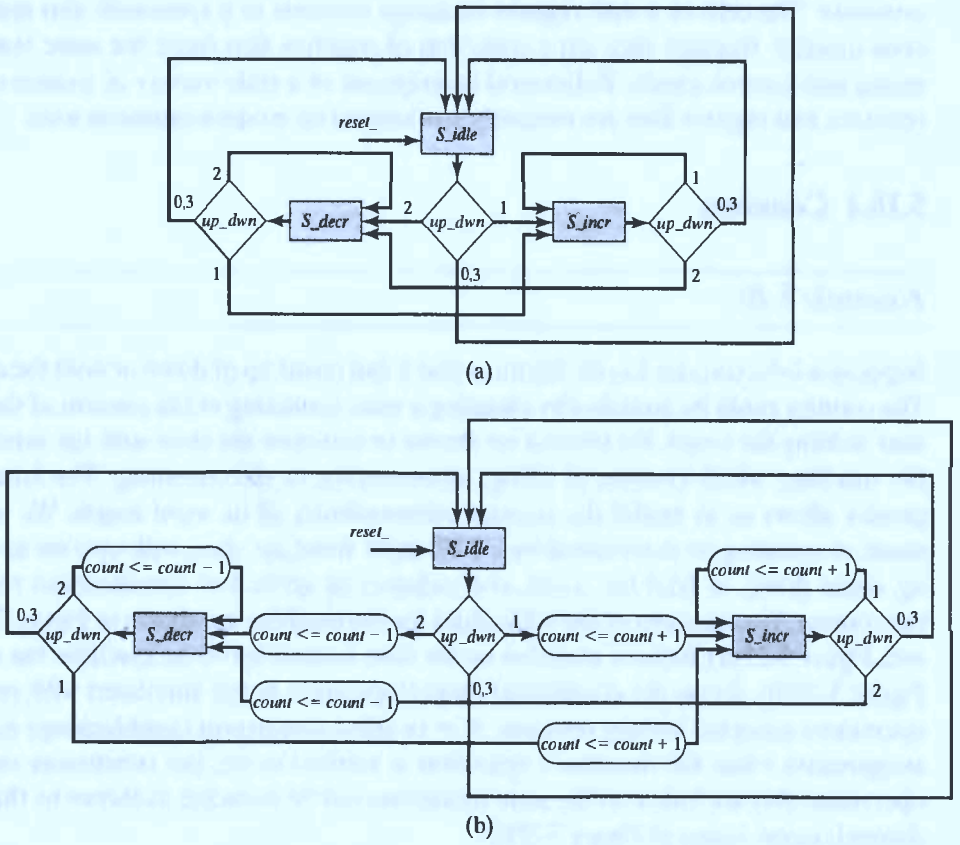


FIGURE 5-25 ASM charts for a behavioral model of an up–down counter having synchronous reset: (a) without conditional output boxes, and (b) with conditional output boxes for register operations outputs generated by the state machine.

on the path leaving *S_running* to remind us that the machine ignores *up_dwn* if *reset_* is asserted.

The Verilog model of the counter can be derived from the ASMD chart by noting that at every clock edge the machine either increments the count, decrements the count, or leaves the count unchanged. The cyclic behavior shown in *Up_Down_Implicit1* describes the decision tree for the state changes and the operations on the datapath register. It suppresses the details of the control signals that will control the hardware datapath.

```
module Up_Down_Implicit1 (count, up_dwn, clock, reset_);
  output  [2: 0]  count;
  input   [1: 0]  up_dwn;
  input           clock, reset_;

  reg     [2: 0]  count;
```

```
always @ (negedge clock or negedge reset_)
  if (reset_ == 0)                          count <= 3'b0; else
    if (up_dwn == 2'b00 || up_dwn == 2'b11) count <= count; else
      if (up_dwn == 2'b01)                  count <= count + 1; else
        if (up_dwn == 2'b10)                count <= count -1;
endmodule
```

End of Example 5.40

Example 5.41

A ring counter asserts a single bit that circulates through the counter in a synchronous manner. The movement of data in an 8-bit ring counter is illustrated in Figure 5-27. Given an external synchronizing signal, *clock*, the behavior described by *ring_counter* ensures the synchronous movement of the asserted bit through the register and the automatic restarting of the count at *count[0]* after *count[7]* is asserted at the end of a cycle. Note that the activity of the machine is the same in every clock cycle, and that *ring_counter* is an implicit state machine. The synthesized circuit is shown in Figure 5-28. The D-type flip-flops in the implementation are active on the rising edge of the clock, have gated data (i.e., the datapath of the flip-flop is driven by the output of a multiplexer whose control signal selects between the output of the flip-flop and the external datapath), and have an asynchronous active-low reset.

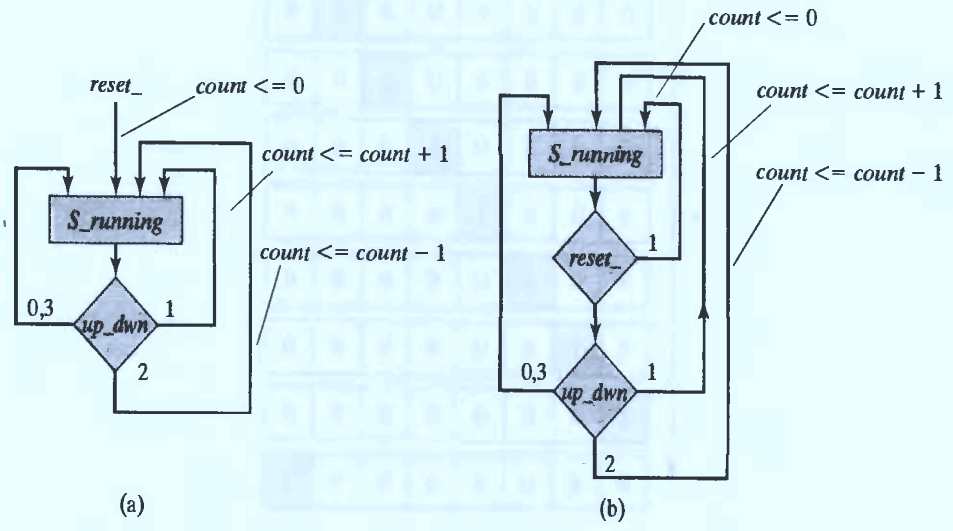


FIGURE 5-26 A simplified ASMD chart for a 4-bit binary counter: (a) With asynchronous active-low reset, and (b) synchronous active-low reset.