complemented output of the rightmost flip-flop to the input of the leftmost flip-flop. The register shifts its contents once to the right with every clock pulse, and at the same time, the complemented value of the E flip-flop is transferred into the A flip-flop. Starting from a cleared state, the switch-tail ring counter goes through a sequence of eight states, as listed in Fig. 6.18(b). In general, a k-bit switch-tail ring counter will go through a sequence of $2k$ states. Starting from all 0's, each shift operation inserts 1's from the left until the register is filled with all 1's. In the next sequences, 0's are inserted from the left until the register is again filled with all 0's.

A Johnson counter is a k-bit switch-tail ring counter with $2k$ decoding gates to provide outputs for $2k$ timing signals. The decoding gates are not shown in Fig. 6.18, but are specified in the last column of the table. The eight AND gates listed in the table, when connected to the circuit, will complete the construction of the Johnson counter. Since each gate is enabled during one particular state sequence, the outputs of the gates generate eight timing signals in succession.

The decoding of a k-bit switch-tail ring counter to obtain $2k$ timing signals follows a regular pattern. The all-0's state is decoded by taking the complement of the two extreme flip-flop outputs. The all-1's state is decoded by taking the normal outputs of the two extreme flip-flops. All other states are decoded from an adjacent 1, 0 or 0, 1 pattern in the sequence. For example, sequence 7 has an adjacent 0, 1 pattern in flip-flops B and C. The decoded output is then obtained by taking the complement of B and the normal output of C, or $B'C$.

One disadvantage of the circuit in Fig. 6.18(a) is that if it finds itself in an unused state, it will persist in moving from one invalid state to another and never find its way to a valid state. The difficulty can be corrected by modifying the circuit to avoid this undesirable condition. One correcting procedure is to disconnect the output from flip-flop B that goes to the D input of flip-flop C and instead enable the input of flip-flop C by the function

$$D_C = (A + C)B$$

where D_C is the flip-flop input equation for the D input of flip-flop C.

Johnson counters can be constructed for any number of timing sequences. The number of flip-flops needed is one-half the number of timing signals. The number of decoding gates is equal to the number of timing signals, and only two-input gates are needed.

6.6 HDL FOR REGISTERS AND COUNTERS

Registers and counters can be described in Verilog at either the behavioral or the structural level. Behavioral modeling describes only the operations of the register, as prescribed by a function table, without a preconceived structure. A structural-level description shows the circuit in terms of a collection of components such as gates, flip-flops, and multiplexers. The various components are instantiated to form a hierarchical description of the design similar to a representation of a multilevel logic diagram. The examples in this section will illustrate both types of descriptions. Both are useful. When a machine is complex, a hierarchical description creates a physical partition of the machine into simpler and more easily described units.

Shift Register

The universal shift register presented in Section 6.2 is a bidirectional shift register with a parallel load. The four clocked operations that are performed with the register are specified in Table 6.6. The register also can be cleared asynchronously. Our chosen name for a behavioral description of the four-bit universal shift register shown in Fig. 6.7(a), the name *Shift_Register_4_beh*, signifies the behavioral model of the internal detail of the top-level block diagram symbol and distinguishes that model from a structural one. The behavioral model is presented in HDL Example 6.1, and the structural model is given in HDL Example 6.2. The top-level block diagram symbol in Fig. 6.7(a) indicates that the four-bit universal shift register has two selection inputs (*s1*, *s0*), two serial inputs (*shift_left*, *shift_right*), for controlling the shift register, two serial datapath inputs (MSB_in and LSB_in), a four-bit parallel input (*I_par*), and a four-bit parallel output (*A_par*). The elements of vector *I_par[3: 0]* correspond to the bits $I_3, \ldots, I_0$ in Fig. 6.7, and similarly for *A_par[3: 0]*. The **always** block describes the five operations that can be performed with the register. The *Clear* input clears the register asynchronously with an active-low signal. *Clear* must be high for the register to respond to the positive edge of the clock. The four clocked operations of the register are determined from the values of the two select inputs in the **case** statement. (*s1* and *s0* are concatenated into a two-bit vector and are used as the expression argument of the **case** statement.) The shifting operation is specified by the concatenation of the serial input and three bits of the register. For example, the statement

```
A_par <= {MSB_in, A_par [3: 1]}
```

specifies a concatenation of the serial data input for a right shift operation (*MSB_in*) with bits *A_par[3: 1] of the output data bus*. A reference to a contiguous range of bits within a vector is referred to as a *part select*. The four-bit result of the concatenation is transferred to register *A_par [3: 0]* when the clock pulse triggers the operation. This transfer produces a shift-right operation and updates the register with new information. The shift operation overwrites the contents of *A_par[0]* with the contents of *A_par[1]*. Note that only the functionality of the circuit has been described, irrespective of any particular hardware. A synthesis tool would create a netlist of ASIC cells to implement the shift register in the structure of Fig. 6.7(b).

HDL Example 6.1 (Universal Shift Register-Behavioral Model)

```
// Behavioral description of a 4-bit universal shift register
// Fig. 6.7 and Table 6.3
module Shift_Register_4_beh (                    // V2001, 2005
 output reg      [3: 0]    A_par,              // Register output
 input           [3: 0]    I_par,              // Parallel input
 input           s1, s0,                       // Select inputs
                 MSB_in, LSB_in,               // Serial inputs
                 CLK, Clear_b                  // Clock and Clear
);
```

```
always @ (posedge CLK, negedge Clear_b)    // V2001, 2005
  if (Clear_b == 0) A_par <= 4'b0000;
  else
   case ({s1, s0})
     2'b00: A_par <= A_par;                // No change
     2'b01: A_par <= {MSB_in, A_par[3: 1]};  // Shift right
     2'b10: A_par <= {A_par[2: 0], LSB_in};  // Shift left
     2'b11: A_par <= I_par;                // Parallel load of input
   endcase
endmodule
```

Variables of type **reg** retain their value until they are assigned a new value by an assignment statement. Consider the following alternative **case** statement for the shift register model:

```
case ({s1, s0})
  // 2'b00: A_par <= A_par;                // No change
  2'b01: A_par <= {MSB_in, A_par [3: 1]};  // Shift right
  2'b10: A_par <= {A_par [2: 0], LSB_in};  // Shift left
  2'b11: A_par <= I_par;                // Parallel load of input
endcase
```

Without the case item 2′b00, the **case** statement would not find a match between $\{s1, s0\}$ and the case items, so register *A_par* would be left unchanged.

A structural model of the universal shift register can be described by referring to the logic diagram of Fig. 6.7(b). The diagram shows that the register has four multiplexers and four *D* flip-flops. A mux and flip-flop together are modeled as a stage of the shift register. The stage is a structural model, too, with an instantiation and interconnection of a module for a mux and another for a *D* flip-flop. For simplicity, the lowest-level modules of the structure are behavioral models of the multiplexer and flip-flop. Attention must be paid to the details of connecting the stages correctly. The structural description of the register is shown in HDL Example 6.2. The top-level module declares the inputs and outputs and then instantiates four copies of a stage of the register. The four instantiations specify the interconnections between the four stages and provide the detailed construction of the register as specified in the logic diagram. The behavioral description of the flip-flop uses a single edge-sensitive cyclic behavior (an **always** block). The assignment statements use the nonblocking assignment operator (<=) the model of the mux employs a single level-sensitive behavior, and the assignments use the blocking assignment operator (=).

HDL Example 6.2 (Universal Shift Register-Structural Model)

```
// Structural description of a 4-bit universal shift register (see Fig. 6.7)
module Shift_Register_4_str (                // V2001, 2005
 output [3: 0] A_par,                        // Parallel output
 input [3: 0]  I_par,                        // Parallel input
```

```
  input       s1, s0,                          // Mode select
  input       MSB_in, LSB_in, CLK, Clear_b     // Serial inputs, clock, clear
);

// bus for mode control
  assign  [1:0]   select = {s1, s0};

// Instantiate the four stages
  stage ST0 (A_par[0], A_par[1], LSB_in, I_par[0], A_par[0], select, CLK, Clear_b);
  stage ST1 (A_par[1], A_par[2], A_par[0], I_par[1], A_par[1], select, CLK, Clear_b);
  stage ST2 (A_par[2], A_par[3], A_par[1], I_par[2], A_par[2], select, CLK, Clear_b);
  stage ST3 (A_par[3], MSB_in, A_par[2], I_par[3], A_par[3], select, CLK, Clear_b);
endmodule

// One stage of shift register
module stage (i0, i1, i2, i3, Q, select, CLK, Clr_b);
  input       i0,          // circulation bit selection
              i1,          // data from left neighbor or serial input for shift-right
              i2,          // data from right neighbor or serial input for shift-left
              i3;          // data from parallel input
  output      Q;
  input [1: 0]  select;    // stage mode control bus
  input       CLK, Clr_b;  // Clock, Clear for flip-flops
  wire        mux_out;

// instantiate mux and flip-flop
  Mux_4_x_1   M0      (mux_out, i0, i1, i2, i3, select);
  D_flip_flop M1      (Q, mux_out, CLK, Clr_b);
endmodule

// 4x1 multiplexer              // behavioral model
module Mux_4_x_1 (mux_out, i0, i1, i2, i3, select);
  output      mux_out;
  input       i0, i1, i2, i3;
  input [1: 0]  select;
  reg         mux_out;
  always @ (select, i0, i1, i2, i3)
    case (select)
      2'b00:    mux_out = i0;
      2'b01:    mux_out = i1;
      2'b10:    mux_out = i2;
      2'b11:    mux_out = i3;
    endcase
endmodule
```

```
// Behavioral model of D flip-flop
module D_flip_flop (Q, D, CLK, Clr_b);
  output      Q;
  input       D, CLK, Clr;
  reg         Q;

  always @ (posedge CLK, negedge Clr_b)
    if (!Clr_b) Q <= 1'b0; else Q <= D;
endmodule
```

The above examples presented two descriptions of a universal shift register to illustrate the different styles for modeling a digital circuit. A simulation should verify that the models have the same functionality. In practice, a designer develops only the behavioral model, which is then synthesized. The function of the synthesized circuit can be compared with the behavioral description from which it was compiled. Eliminating the need for the designer to develop a structural model produces a huge improvement in the efficiency of the design process.

Synchronous Counter

HDL Example 6.3 presents *Binary_Counter_4_Par_Load,* a behavioral model of the synchronous counter with a parallel load from Fig. 6.14. *Count*, *Load*, *CLK*, and *Clear_b* are inputs that determine the operation of the counter according to the function specified in Table 6.6. The counter has four data inputs, four data outputs, and a carry output. The internal data lines (*I3*, *I2*, *I1*, *I0*) are bundled as *Data_in[3: 0]* in the behavioral model. Likewise, the register that holds the bits of the count (*A3*, *A2*, *A1*, *A0*) is *A_count[3: 0].* It is good practice to have identifiers in the HDL model of a circuit correspond exactly to those in the documentation of the model. That is not always feasible, however, if the circuit-level identifiers are those found in a handbook, for they are often short and cryptic and do not exploit the text that is available with an HDL. The top-level block diagram symbol in Fig. 6.14(a) serves as an interface between the names used in a circuit diagram and the expressive names that can be used in the HDL model. The carry output *C_out* is generated by a combinational circuit and is specified with an **assign** statement. *C_out* = 1when the count reaches 15 and the counter is in the count state. Thus, *C_out* = 1if *Count* = 1, *Load* = 0, and *A* = 1111; otherwise *C_out* = 0. The **always** block specifies the operation to be performed in the register, depending on the values of *Clear_b, Load,* and *Count*. A 0 (active-low signal) at *Clear_b* resets *A* to 0. Otherwise, if *Clear_b* = 1, one out of three operations is triggered by the positive edge of the clock. The **if**, **else if**, and **else** statements establish a precedence among the control signals *Clear*, *Load*, and *Count* corresponding to the specification in Table 6.6. *Clear_b* overrides *Load* and *Count*; *Load* overrides *Count*. A synthesis tool will produce the circuit of Fig. 6.14(b) from the behavioral model.

HDL Example 6.3 (Synchronous Counter)

```
// Four-bit binary counter with parallel load (V2001, 2005)
// See Figure 6.14 and Table 6.6
module Binary_Counter_4_Par_Load (
  output reg [3: 0]                 A_count,     // Data output
  output                            C_out,       // Output carry
  input [3: 0]                      Data_in,     // Data input
  input                             Count,       // Active high to count
                                    Load,        // Active high to load
                                    CLK,         // Positive-edge sensitive
                                    Clear_b      // Active low
);
assign C_out = Count && (~Load) && (A_count == 4'b1111);
always @ (posedge CLK, negedge Clear_b)
   if (~Clear_b)                    A_count <= 4'b0000;
   else if (Load)                   A_count <= Data_in;
   else if (Count)                  A_count <= A_count + 1'b1;
   else                             A_count <= A_count;  // redundant statement
endmodule
```

Ripple Counter

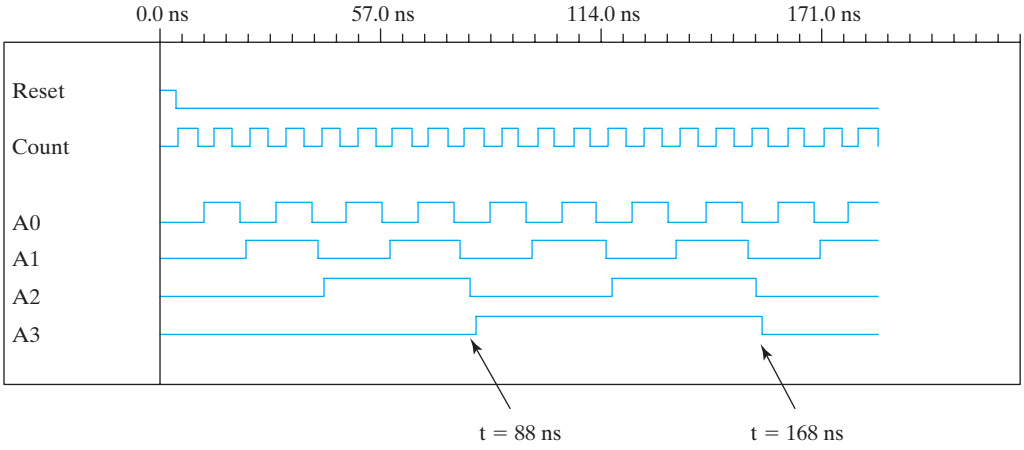
The structural description of a ripple counter is shown in HDL Example 6.4. The first module instantiates four internally complementing flip-flops defined in the second module as *Comp_D_flip_flop (Q, CLK, Reset)*. The clock (input *CLK*) of the first flip-flop is connected to the external control signal *Count*. (*Count* replaces *CLK* in the port list of instance *F0*.) The clock input of the second flip-flop is connected to the output of the first. (*A0* replaces *CLK* in instance *F1*.) Similarly, the clock of each of the other flip-flops is connected to the output of the previous flip-flop. In this way, the flip-flops are chained together to create a ripple counter as shown in Fig. 6.8(b).

The second module describes a complementing flip-flop with delay. The circuit of a complementing flip-flop is constructed by connecting the complement output to the *D* input. A reset input is included with the flip-flop in order to be able to initialize the counter; otherwise the simulator would assign the unknown value (*x*) to the output of the flip-flop and produce useless results. The flip-flop is assigned a delay of two time units from the time that the clock is applied to the time that the flip-flop complements its output. The delay is specified by the statement $Q <= \#2 \sim Q$. Notice that the delay operator is placed to the right of the nonblocking assignment operator. This form of delay, called *intra-assignment delay*, has the effect of postponing the assignment of the complemented value of Q to Q. The effect of modeling the delay will be apparent in the simulation results. This style of modeling might be useful in simulation, but it is to be avoided when the model is to be synthesized. The results of synthesis depend on the ASIC cell library that is accessed by the tool, not on any propagation delays that might appear within the model that is to be synthesized.

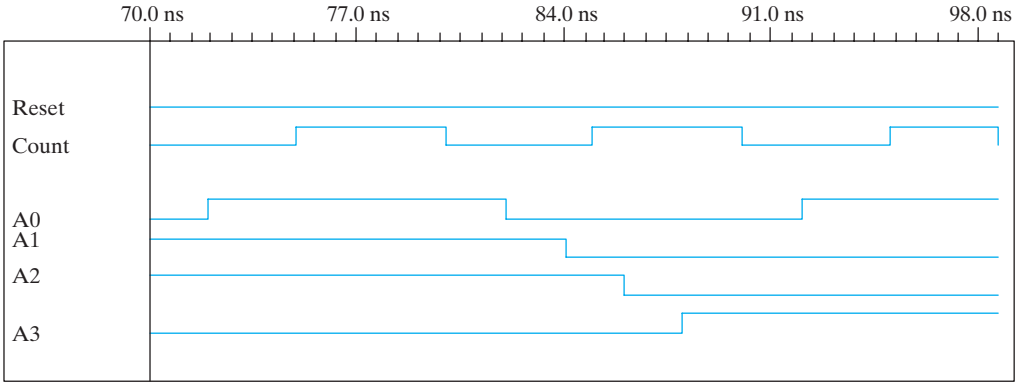
HDL Example 6.4 (Ripple Counter)

```
// Ripple counter (See Fig. 6.8(b))
'timescale 1ns / 100 ps
module Ripple_Counter_4bit (A3, A2, A1, A0, Count, Reset);
  output A3, A2, A1, A0;
  input Count, Reset;
// Instantiate complementing flip-flop
  Comp_D_flip_flop F0 (A0, Count, Reset);
  Comp_D_flip_flop F1 (A1, A0, Reset);
  Comp_D_flip_flop F2 (A2, A1, Reset);
  Comp_D_flip_flop F3 (A3, A2, Reset);
endmodule
// Complementing flip-flop with delay
// Input to D flip-flop = Q'
module Comp_D_flip_flop (Q, CLK, Reset);
  output   Q;
  input    CLK, Reset;
  reg      Q;
  always @ (negedge CLK, posedge Reset)
  if (Reset) Q <= 1'b0;
  else Q <= #2 ~Q;              // intra-assignment delay
endmodule
// Stimulus for testing ripple counter
module t_Ripple_Counter_4bit;
  reg      Count;
  reg      Reset;
  wire     A0, A1, A2, A3;
// Instantiate ripple counter
  Ripple_Counter_4bit M0 (A3, A2, A1, A0, Count, Reset);
   always
  #5 Count = ~Count;
   initial
    begin
  Count = 1'b0;
  Reset = 1'b1;
  #4 Reset = 1'b0;
    end

   initial #170 $finish;

endmodule
```

The test bench module in HDL Example 6.4 provides a stimulus for simulating and verifying the functionality of the ripple counter. The **always** statement generates a free-running clock with a cycle of 10 time units. The flip-flops trigger on the negative edge of the clock, which occurs at $t = 10, 20, 30$, and every 10 time units thereafter. The waveforms obtained from this simulation are shown in Fig. 6.19. The control signal *Count* goes negative every 10 ns. *A0* is complemented with each negative edge of *Count*, but is delayed by 2 ns. Each flip-flop is complemented when its previous flip-flop goes from 1 to 0. After $t = 80$ ns, all four flip-flops complement because the counter goes from 0111 to 1000. Each output is delayed by 2 ns, and because of that, *A*3 goes from 0 to 1 at $t = 88$ ns and from 1 to 0 at 168 ns. Notice how the propagation delays accumulate to the last bit of the counter, resulting in very slow counter action. This limits the practical utility of the counter.



(a) From 0 to 180 ns



(b) From 70 to 98 ns

FIGURE 6.19
Simulation output of HDL Example 6.4