Figure 7.1. Control of an alarm system.

Figure 7.2. A simple memory element.

Figure 7.3. A controlled memory element.

Figure 7.4. A memory element with NOR gates.

Figure 7.5. A latch built with NOR gates.

Figure 7.7. Gated SR latch with NAND gates.
Figure 7.9. Setup and hold times.

Figure 7.11. A positive-edge-triggered D flip-flop.

Figure 7.13. Master-slave D flip-flop with Clear and Preset.

Figure 7.15. Synchronous reset for a D flip-flop.

Figure 7.17. JK flip-flop.

Figure 7.18. A simple shift register.
Figure 7.26. A modulo-6 counter with synchronous reset.

Figure 7.27. A modulo-6 counter with asynchronous reset.

Figure 7.28. A two-digit BCD counter.

Figure 7.30. Johnson counter.

Figure 7.31. Three types of storage elements in a schematic.

Figure 7.32. Gated D latch generated by CAD tools.
module D_latch (D, Clk, Q);
  input D, Clk;
  output Q;
  reg Q;

  always @(D or Clk)
  if (Clk)
    Q = D;
endmodule

module flipflop (D, Clock, Q);
  input D, Clock;
  output Q;
  reg Q;

  always @(posedge Clock)
    Q = D;
endmodule

module example7_3 (D, Clock, Q1, Q2);
  input D, Clock;
  output Q1, Q2;
  reg Q1, Q2;

  always @(posedge Clock)
    begin
      Q1 = D;
      Q2 = Q1;
    end
endmodule

module example7_4 (D, Clock, Q1, Q2);
  input D, Clock;
  output Q1, Q2;
  reg Q1, Q2;

  always @(posedge Clock)
    begin
      Q1 <= D;
      Q2 <= Q1;
    end
endmodule

Figure 7.34. Timing simulation of storage elements.

Figure 7.35. Code for a gated D latch.

Figure 7.36. Code for a D flip-flop.

Figure 7.37. Incorrect code for two cascaded flip-flops.

Figure 7.38. Circuit for Example 7.3.

Figure 7.39. Code for two cascaded flip-flops.
Figure 7.40. Circuit defined in Figure 7.39.

Figure 7.41. Code for Example 7.5.

module example7_5 (x1, x2, x3, Clock, f, g);
  input x1, x2, x3, Clock;
  output f, g;
  reg f, g;

  always @(posedge Clock)
  begin
    f = x1 & x2;
    g = f | x3;
  end
endmodule

Figure 7.42. Circuit for Example 7.5.

Figure 7.43. Code for Example 7.6.

module example7_6 (x1, x2, x3, Clock, f, g);
  input x1, x2, x3, Clock;
  output f, g;
  reg f, g;

  always @(posedge Clock)
  begin
    f <= x1 & x2;
    g <= f | x3;
  end
endmodule

Figure 7.44. Circuit for Example 7.6.

Figure 7.45. D flip-flop with asynchronous reset.

module flipflop (D, Clock, Resetn, Q);
  input D, Clock, Resetn;
  output Q;
  reg Q;

  always @(negedge Resetn or posedge Clock)
  if (!Resetn)
    Q <= 0;
  else
    Q <= D;
endmodule
module flipflop (D, Clock, Resetn, Q);
  input D, Clock, Resetn;
  output Q;
  reg Q;

  always @(posedge Clock)
    if (!Resetn)
      Q <= 0;
    else
      Q <= D;

endmodule

Figure 7.46. D flip-flop with synchronous reset.

module shift (Clock, Reset, w, Load, R, Q);
  input Clock, Reset, w, Load;
  input [3:0] R;
  output [3:0] Q;
  lpm_shiftreg shift_right (.data(R), .aclr(Reset), .clock(Clock), .load(Load), .shiftin(w), .q(Q));
  defparam shift_right.lpm_width = 4;
  defparam shift_right.lpm_direction = "RIGHT";
endmodule

Figure 7.48. An adder with registered feedback.

module regn (D, Clock, Resetn, Q);
  parameter n = 16;
  input [n-1:0] D;
  input Clock, Resetn;
  output [n-1:0] Q;
  reg [n-1:0] Q;

  always @(negedge Resetn or posedge Clock)
    if (!Resetn)
      Q <= 0;
    else
      Q <= D;
endmodule

Figure 7.51. Code for an n-bit register with asynchronous clear.
module muxdff (D0, D1, Sel, Clock, Q);
    input D0, D1, Sel, Clock;
    output Q;
    reg Q;

    always @(posedge Clock)
        if (!Sel)
            Q <= D0;
        else
            Q <= D1;

endmodule

module shift4 (R, L, w, Clock, Q);
    input [3:0] R;
    input L, w, Clock;
    output [3:0] Q;
    wire [3:0] Q;

    muxdff Stage3 (w, R[3], L, Clock, Q[3]);
    muxdff Stage2 (Q[3], R[2], L, Clock, Q[2]);
    muxdff Stage1 (Q[2], R[1], L, Clock, Q[1]);
    muxdff Stage0 (Q[1], R[0], L, Clock, Q[0]);

endmodule

module shiftn (R, L, w, Clock, Q);
    parameter n = 16;
    input [n-1:0] R;
    input L, w, Clock;
    output [n-1:0] Q;
    reg [n-1:0] Q;
    integer k;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
            begin
                for (k = 0; k < n-1; k = k+1)
                    Q[k] <= Q[k+1];
                Q[n-1] <= w;
            end

endmodule

module upcount (Resetn, Clock, E, Q);
    input Resetn, Clock, E;
    output [3:0] Q;
    reg [3:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (E)
            Q <= Q + 1;

endmodule

module upcount (R, Resetn, Clock, E, L, Q);
    input [3:0] R;
    input Resetn, Clock, E, L;
    output [3:0] Q;
    reg [3:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1;

endmodule

module upcount (R, Resetn, Clock, E, L, Q);
    input [3:0] R;
    input Resetn, Clock, E, L;
    output [3:0] Q;
    reg [3:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1;

endmodule
module downcount (R, Clock, E, L, Q);
  parameter n = 8;
  input [n-1:0] R;
  input Clock, L, E;
  output [n-1:0] Q;
  reg [n-1:0] Q;
  always @(posedge Clock)
    if (L)
      Q <= R;
    else if (E)
      Q <= Q - 1;
endmodule

module updowncount (R, Clock, L, E, up_down, Q);
  parameter n = 8;
  input [n-1:0] R;
  input Clock, L, E, up_down;
  output [n-1:0] Q;
  reg [n-1:0] Q;
  integer direction;
  always @(posedge Clock)
    begin
      if (up_down)
        direction = 1;
      else
        direction = 0;
      if (L)
        Q <= R;
      else if (E)
        Q <= Q + direction;
    end
endmodule

Figure 7.58. A down-counter with a parallel load.

Figure 7.59. Code for an up/down counter.

Figure 7.60. A digital system with \( k \) registers.

Figure 7.61. Details for connecting registers to a bus.

Figure 7.62. A shift-register control circuit.

Figure 7.63. A modified control circuit.
module regn (R, Rin, Clock, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Rin, Clock;
    output [n-1:0] Q;
    reg [n-1:0] Q;

    always @(posedge Clock)
        if (Rin)
            Q <= R;

endmodule

Figure 7.66. Code for an n-bit register of the type in Figure 7.61.

module trin (Y, E, F);
    parameter n = 8;
    input [n-1:0] Y;
    input E;
    output [n-1:0] F;
    wire [n-1:0] F;

    assign F = E ? Y : 'bz;

endmodule

Figure 7.67. Code for an n-bit tri-state module.

module shiftr (Resetn, w, Clock, Q);
    parameter m = 4;
    input Resetn, w, Clock;
    output [1:m] Q;
    reg [1:m] Q;
    integer k;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else
            begin
                for (k = m; k > 1; k = k-1)
                    Q[k] <= Q[k-1];
                Q[1] <= w;
            end

endmodule

Figure 7.68. Code for the shift register in Figure 7.62.

Figure 7.72. Timing simulation.
Table 7.2. Operations performed in the processor.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Rx, Lay</td>
<td>Rx ← Lay</td>
</tr>
<tr>
<td>Move Rx, Ry</td>
<td>Rx ← Ry</td>
</tr>
<tr>
<td>Add Rx, Rz</td>
<td>Rx ← Rx + Rz</td>
</tr>
<tr>
<td>Sub Rx, Rz</td>
<td>Rx ← Rx − Rz</td>
</tr>
</tbody>
</table>

Figure 7.3. A digital system that implements a simple processor.

Table 7.3. Control signals asserted in each operation/time step.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Rx, Lay</td>
<td>Q0, Q1, Q2, Q3</td>
</tr>
<tr>
<td>Move Rx, Ry</td>
<td>Q0, Q1, Q2, Q3</td>
</tr>
<tr>
<td>Add Rx, Rz</td>
<td>Q0, Q1, Q2, Q3</td>
</tr>
<tr>
<td>Sub Rx, Rz</td>
<td>Q0, Q1, Q2, Q3</td>
</tr>
</tbody>
</table>

Figure 7.4. A part of the control circuit for the processor.

Figure 7.5. The function register and decoders.

```verilog
testmodule upcount (Clear, Clock, Q);

input Clear, Clock;
output [1:0] Q;
reg [1:0] Q;

always @(posedge Clock)
  if (Clear)
    Q <= 0;
  else
    Q <= Q + 1;

endmodule
```

Figure 7.6. A two-bit up-counter with synchronous reset.
module reaction (c9, Reset, w, Pushn, LEDn, Digit1, Digit0);
input c9, Reset, w, Pushn;
output LEDn;
output [1:7] Digit1, Digit0;
wire LEDn;
wire [1:7] Digit1, Digit0;
reg LED;
wire [3:0] BCD1, BCD0;
always @ (posedge c9)
begin
  if (Pushn == 0)
    LED <= 0;
  else if (w)
    LED <= 1;
end
assign LEDn = ~LED;
BCDcount counter (c9, Reset, LED, BCD1, BCD0);
seg7 seg1 (BCD1, Digit1);
seg7 seg0 (BCD0, Digit0);
endmodule
module lfsr (R, L, Clock, Q);
  input [0:2] R;
  input L, Clock;
  output [0:2] Q;
  reg [0:2] Q;
  
  always @(posedge Clock)
    if (L)
      Q <= R;
    else
      Q <= {Q[2], Q[0] ^ Q[2], Q[1]};

endmodule
module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output [0:2] Q;
    reg [0:2] Q;
    always @(posedge Clock)
        if (L)
            Q <= R;
        else
            Q <= {Q[2], Q[0], Q[1] ^ Q[2]};
endmodule

Figure P7.10. Code for a linear-feedback shift register.

module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output [0:2] Q;
    reg [0:2] Q;
    always @(posedge Clock)
        if (L)
            Q <= R;
        else
            begin
                Q[0] = Q[2];
                Q[1] = Q[0] ^ Q[2];
                Q[2] = Q[1];
            end
endmodule

Figure P7.11. Code for problem 7.37.

module lfsr (R, L, Clock, Q);
    input [0:2] R;
    input L, Clock;
    output [0:2] Q;
    reg [0:2] Q;
    always @ (posedge Clock)
        if (L)
            Q <= R;
        else
            begin
                Q[0] = Q[2];
                Q[1] = Q[0] ^ Q[2];
                Q[2] = Q[1];
            end
endmodule

Figure P7.12. Code for problem 7.38.