

Figure 7.1. Control of an alarm system.

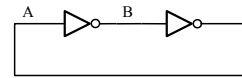


Figure 7.2. A simple memory element.

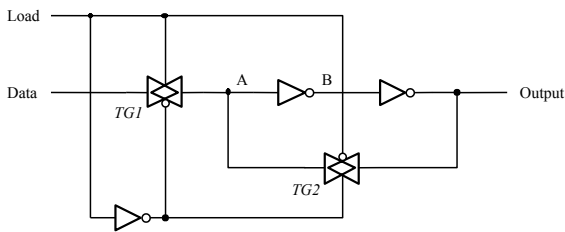


Figure 7.3. A controlled memory element.

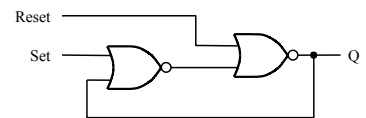


Figure 7.4. A memory element with NOR gates.

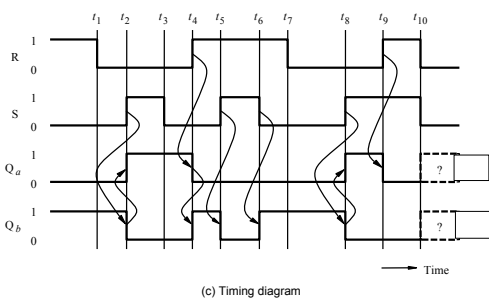
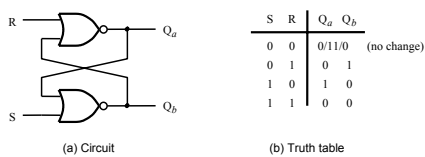


Figure 7.5. A latch built with NOR gates.

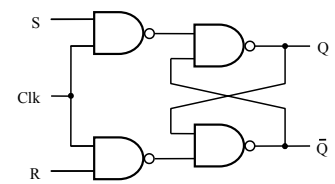


Figure 7.7. Gated SR latch with NAND gates.

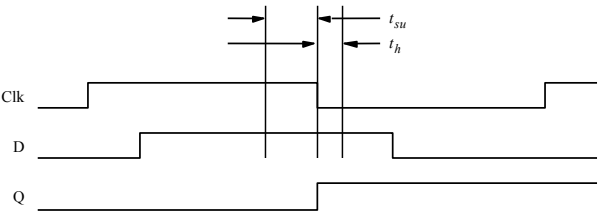


Figure 7.9. Setup and hold times.

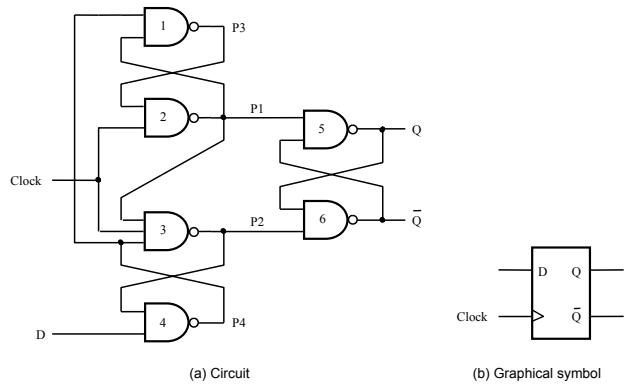


Figure 7.11. A positive-edge-triggered D flip-flop.

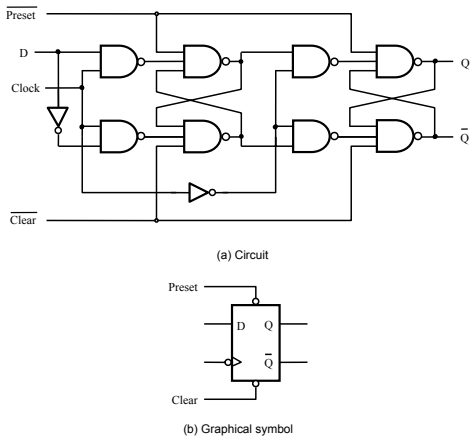


Figure 7.13. Master-slave D flip-flop with Clear and Preset.

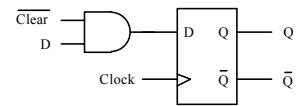
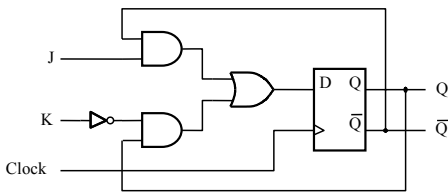


Figure 7.15. Synchronous reset for a D flip-flop.



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

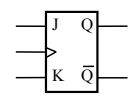
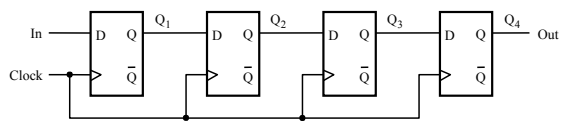


Figure 7.17. JK flip-flop.



(a) Circuit

	In	Q ₁	Q ₂	Q ₃	Q ₄ = Out
t ₀	1	0	0	0	0
t ₁	0	1	0	0	0
t ₂	1	0	1	0	0
t ₃	1	1	0	1	0
t ₄	1	1	1	0	1
t ₅	0	1	1	1	0
t ₆	0	0	1	1	1
t ₇	0	0	0	1	1

(b) A sample sequence

Figure 7.18. A simple shift register.

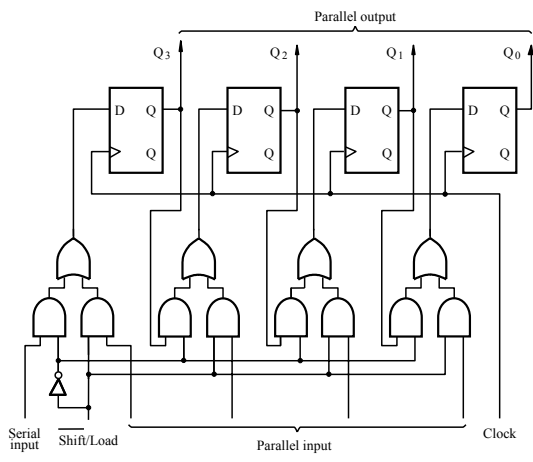
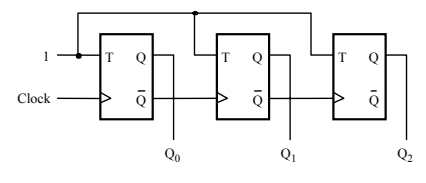
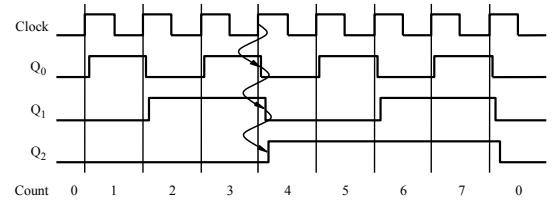


Figure 7.19. Parallel access shift register.

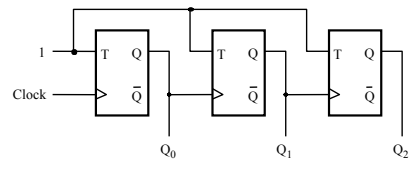


(a) Circuit

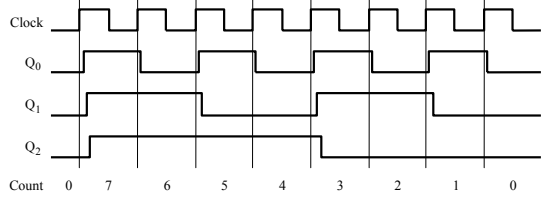


(b) Timing diagram

Figure 7.20. A three-bit up-counter.



(a) Circuit



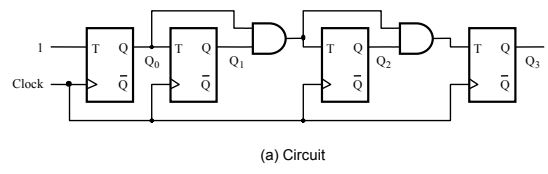
(b) Timing diagram

Figure 7.21. A three-bit down-counter.

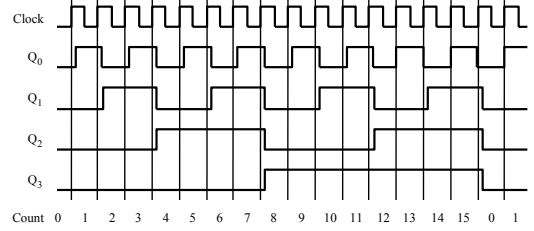
Clock cycle	Q ₂	Q ₁	Q ₀
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

Q₁ changes
Q₂ changes

Table 7.1. Derivation of the synchronous up-counter.



(a) Circuit



(b) Timing diagram

Figure 7.22. A four-bit synchronous up-counter.

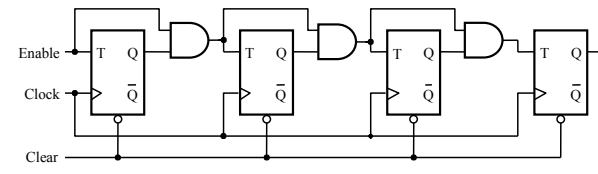
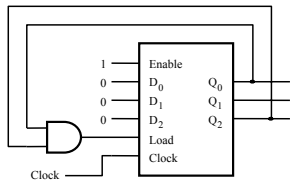
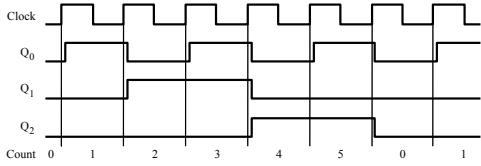


Figure 7.23. Inclusion of Enable and Clear capability.

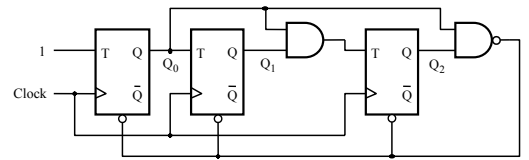


(a) Circuit

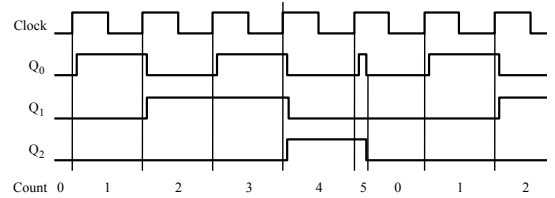


(b) Timing diagram

Figure 7.26. A modulo-6 counter with synchronous reset.



(a) Circuit



(b) Timing diagram

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

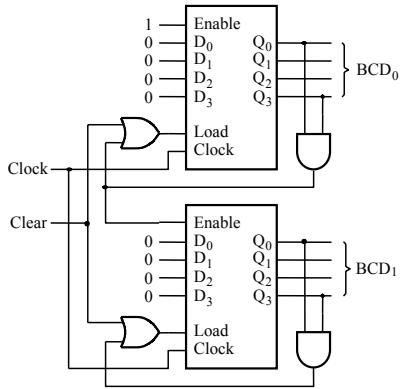


Figure 7.28. A two-digit BCD counter.

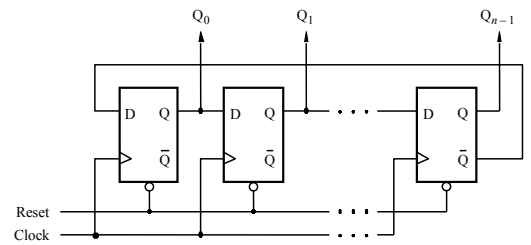


Figure 7.30. Johnson counter.

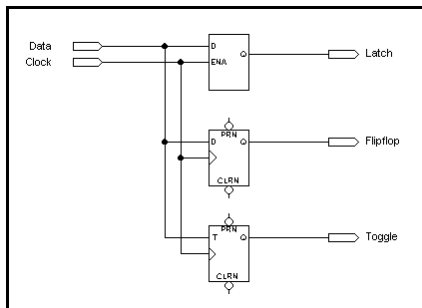


Figure 7.31. Three types of storage elements in a schematic.

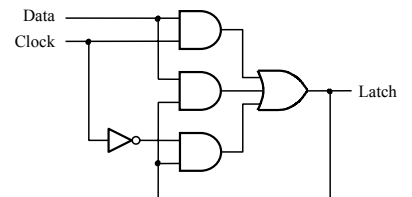


Figure 7.32. Gated D latch generated by CAD tools.

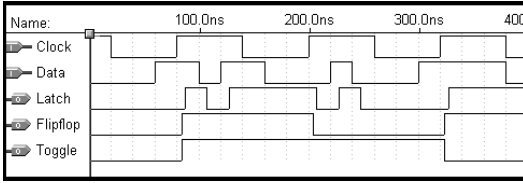


Figure 7.34. Timing simulation of storage elements.

```

module D_latch (D, Clk, Q);
  input D, Clk;
  output Q;
  reg Q;

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

endmodule

```

Figure 7.35. Code for a gated D latch.

```

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

  always @(posedge Clock)
    Q = D;

endmodule

```

Figure 7.36. Code for a D flip-flop.

```

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

```

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

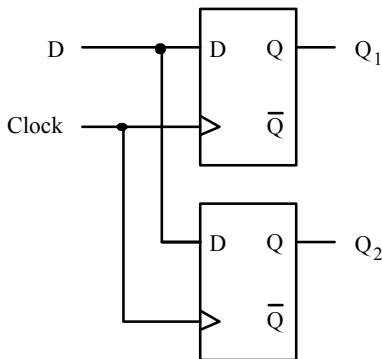


Figure 7.38. Circuit for Example 7.3.

```

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 7.39. Code for two cascaded flip-flops.

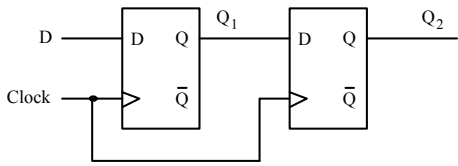


Figure 7.40. Circuit defined in Figure 7.39.

```

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 7.41. Code for Example 7.5.

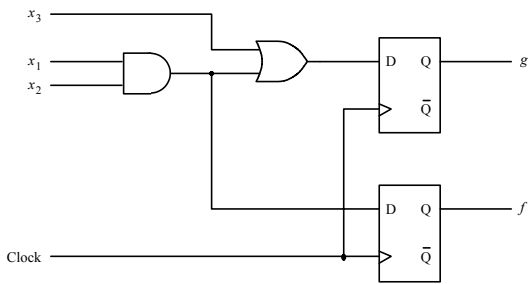


Figure 7.42. Circuit for Example 7.5.

```

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 7.43. Code for Example 7.6.

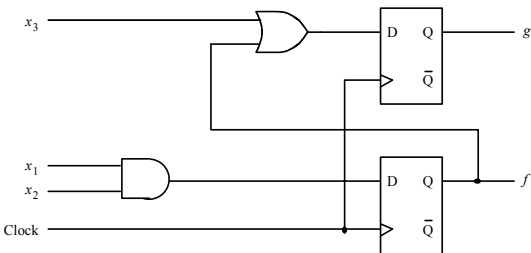


Figure 7.44. Circuit for Example 7.6.

```

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

```

Figure 7.45. D flip-flop with asynchronous reset.

```

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.

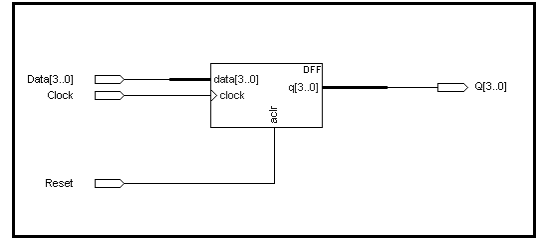


Figure 7.47. The *lpm_ff* parameterized flip-flop module.

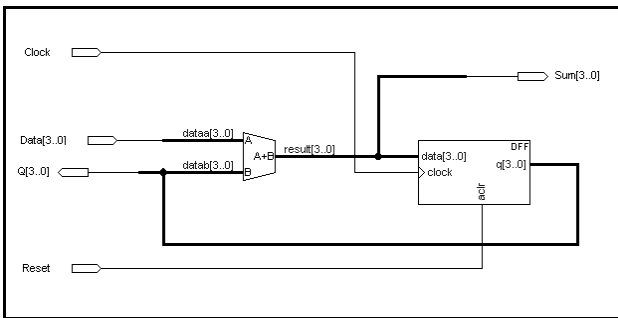


Figure 7.48. An adder with registered feedback.

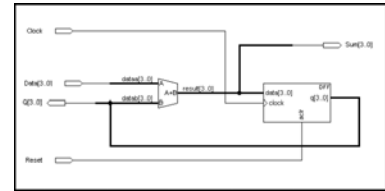


Figure 7.49. Timing simulation.

```

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 7.50. Instantiation of the *lpm_shiftreg* module.

```

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

```

Figure 7.52. Code for a D flip-flop with a 2-to-1 multiplexer on the D input.

```

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

```

Figure 7.53. Hierarchical code for a four-bit shift register.

```

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

  always @(posedge Clock)
    if (L)
      Q <= R;
    else
      begin
        Q[0] <= Q[1];
        Q[1] <= Q[2];
        Q[2] <= Q[3];
        Q[3] <= w;
      end

endmodule

```

Figure 7.54. Alternative code for a four-bit shift register.

```

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

```

Figure 7.55. An n -bit shift register.

```

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

```

Figure 7.56. Code for a four-bit up-counter.

```

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

```

Figure 7.57. A four-bit up-counter with parallel load.

```

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

```

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

```

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 = -1;
      if (L)
        Q <= R;
      else if (E)
        Q <= Q + direction;
    end

endmodule

```

Figure 7.59. Code for an up/down counter.

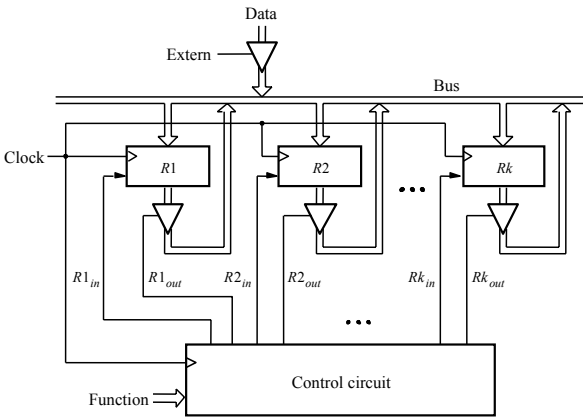


Figure 7.60. A digital system with k registers.

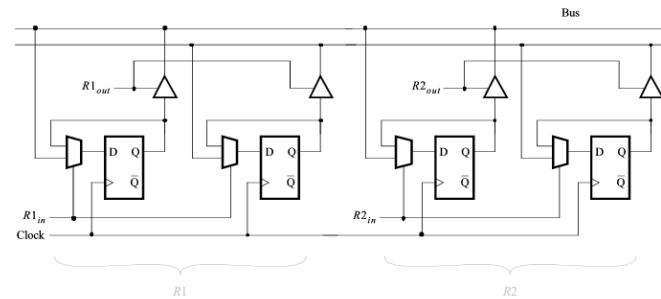


Figure 7.61. Details for connecting registers to a bus.

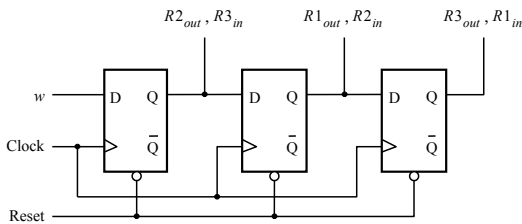


Figure 7.62. A shift-register control circuit.

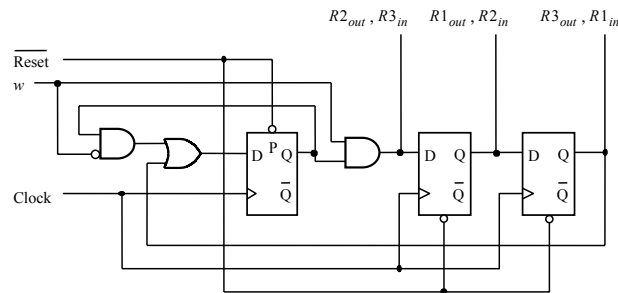


Figure 7.63. A modified control circuit.

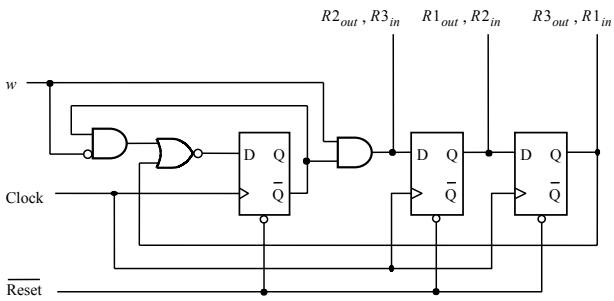


Figure 7.64. A control circuit that does not require flip-flop preset inputs.

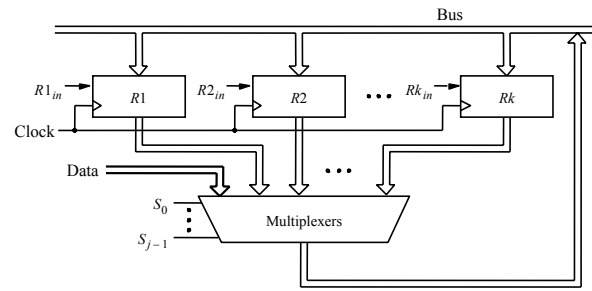


Figure 7.65. Using multiplexers to implement a bus.

```

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
    end

endmodule

```

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

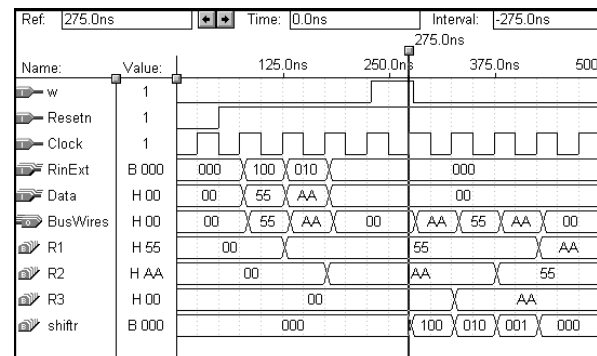


Figure 7.72. Timing simulation.

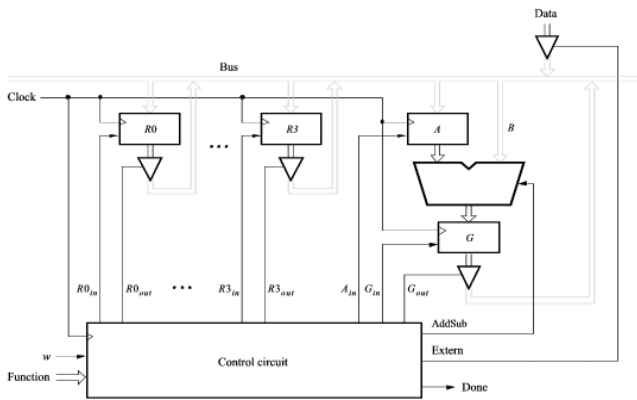


Figure 7.73. A digital system that implements a simple processor.

Operation	Function performed
Load $Rx, Data$	$Rx \leftarrow Data$
Move Rx, Ry	$Rx \leftarrow [Ry]$
Add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
Sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 7.2. Operations performed in the processor.

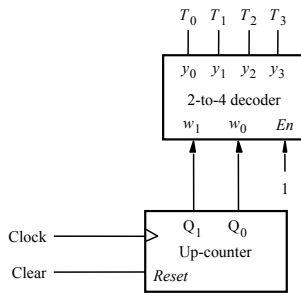


Figure 7.74. A part of the control circuit for the processor.

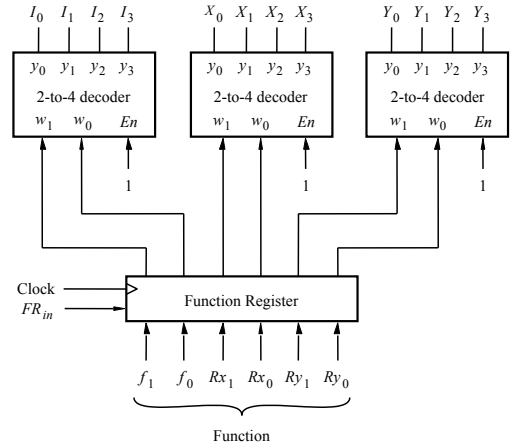


Figure 7.75. The function register and decoders.

	T_1	T_2	T_3
(Load): I_0	$Extern, R_{in} = X,$ $Done$		
(Move): I_1	$R_{in} = X, R_{out} = Y,$ $Done$		
(Add): I_2	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 0$	$G_{out}, R_{in} = X,$ $Done$
(Sub): I_3	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 1$	$G_{out}, R_{in} = X,$ $Done$

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

```

module 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 7.76. A two-bit up-counter with synchronous reset.

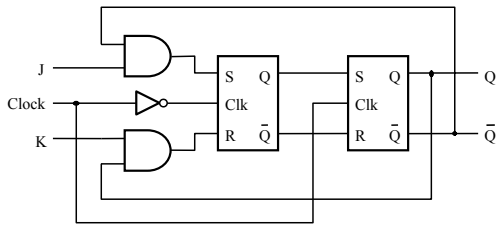


Figure P7.4. The circuit for problem 7.19.

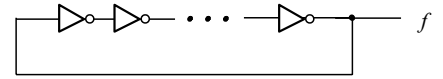


Figure P7.5. A ring oscillator.

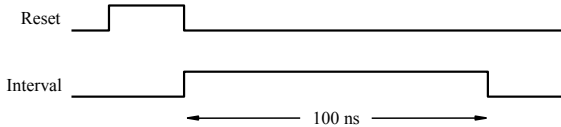


Figure P7.6. Timing of signals for problem 7.31.

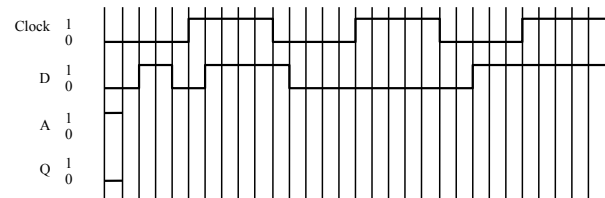
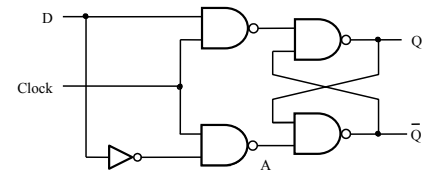


Figure P7.7. Circuit and timing diagram for problem 7.32.

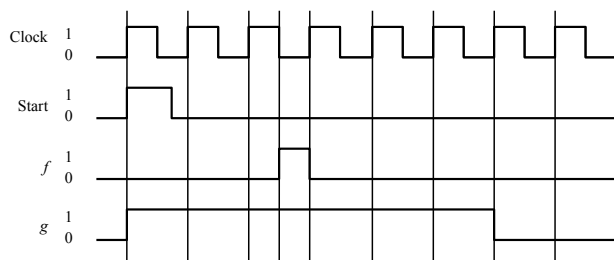


Figure P7.8. Timing diagram for problem 7.33.

```

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

```

Figure P7.9. 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
      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[1] ^ Q[2];
      end
endmodule

```

Figure P7.12. Code for problem 7.38.