Figure 7.6. Gated SR latch.

Figure 7.8. Gated D latch.

Figure 7.10. Master-slave D flip-flop.

Figure 7.12. Comparison of level-sensitive and edge-triggered.

Figure 7.14. Positive-edge-triggered D flip-flop with Clear and Preset.

Figure 7.16. T flip-flop.
**Figure 7.24.** A four-bit counter with D flip-flops.

**Figure 7.25.** A counter with parallel-load capability.

**Figure 7.29.** Ring counter.

**Figure 7.33.** Implementation of a circuit in a CPLD.

**Figure 7.69.** A digital system like the one in Figure 7.60.

**Figure 7.70.** Using multiplexers to implement a bus.
module swapmux (Data, Resetn, w, Clock, RinExt, BusWires);
input [7:0] Data;
input Resetn, w, Clock;
input [1:3] RinExt;
output [7:0] BusWires;
reg [7:0] BusWires;
wire [1:3] Rin, Q;
wire [7:0] R1, R2, R3;

shift control (Resetn, w, Clock, Q);
defparam control.m = 3;
reg reg_1 (BusWires, Rin[1], Clock, R1);
reg reg_2 (BusWires, Rin[2], Clock, R2);
reg reg_3 (BusWires, Rin[3], Clock, R3);
always @(Q or Data or R1 or R2 or R3) begin
if (Q == 3'b000) BusWires = Data;
else if (Q == 3'b100) BusWires = R2;
else if (Q == 3'b010) BusWires = R1;
else BusWires = R3;
end
endmodule

Figure 7.71. A simplified version of the specification in Figure 7.70.

module proc (Data, Reset, w, Clock, F, Rx, Ry, Done, BusWires);
input [7:0] Data;
input Reset, w, Clock;
input [1:3] F, Rx, Ry;
output [7:0] BusWires;
reg [7:0] BusWires;
wire [1:3] Rin, Rout;
wire [7:0] Sum, Gin, Gout, Frin, Frout, Rout;
wire [1:0] Count, C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;

always @ (Data or R0 or R1 or R2 or R3)
begin
if (Data == 3'b000) BusWires = Data;
else if (Data == 3'b100) BusWires = R2;
else if (Data == 3'b010) BusWires = R1;
else BusWires = R3;
end
endmodule

Figure 7.77. Code for the processor (Part a).

module proc2 (Data, Reset, w, Clock, F, Rx, Ry, Done, BusWires);
input [7:0] Data;
input Reset, w, Clock;
input [1:3] F, Rx, Ry;
output [7:0] BusWires;
reg [7:0] BusWires;
wire [1:3] Rin, Rout;
wire [7:0] Sum, Gin, Gout, Frin, Frout, Rout;
wire [1:0] Count, C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;

always @ (Data or R0 or R1 or R2 or R3)
begin
if (Data == 3'b000) BusWires = Data;
else if (Data == 3'b100) BusWires = R2;
else if (Data == 3'b010) BusWires = R1;
else BusWires = R3;
end
endmodule

Figure 7.77. Code for the processor (Part b).

module proc3 (Data, Reset, w, Clock, F, Rx, Ry, Done, BusWires);
input [7:0] Data;
input Reset, w, Clock;
input [1:3] F, Rx, Ry;
output [7:0] BusWires;
reg [7:0] BusWires;
wire [1:3] Rin, Rout;
wire [7:0] Sum, Gin, Gout, Frin, Frout, Rout;
wire [1:0] Count, C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;
wire [1:0] F, Rx, Ry;
wire [7:0] R0, R1, R2, R3, A, G;
wire [1:0] C2, Q, T;
wire [7:0] BusWires;

always @ (Data or R0 or R1 or R2 or R3)
begin
if (Data == 3'b000) BusWires = Data;
else if (Data == 3'b100) BusWires = R2;
else if (Data == 3'b010) BusWires = R1;
else BusWires = R3;
end
endmodule

Figure 7.77. Code for the processor (Part c).

// RegCntl
always ((b or T or Xreg or Y)
begin
Reg[1] = (I[0] | I[1]) & T[1] & Xreg[0];
Reg[1] = (I[0] | I[1]) & T[1] & Xreg[1];
end

// busExt
assign busExt[0] = busExt[0];
assign busExt[1] = busExt[1];
est generate
module
  BusWires = R0;
eles generate
  BusWires = R1;
elest generate
  BusWires = R2;
ende
est generate
module
  BusWires = R3;
eles generate
  BusWires = G;
endmodule

the BusWires = Data;
end
cendmodule

Figure 7.78. Alternative code for the processor (Part c).
Figure 7.80. A reaction-timer circuit.

Figure 7.81. Code for the two-digit BCD counter in Figure 7.28.