

```

module proc (Data, Reset, w, Clock, F, Rx, Ry, Done, BusWires);
  input [7:0] Data;
  input Reset, w, Clock;
  input [1:0] F, Rx, Ry;
  output wire [7:0] BusWires;
  output Done;
  reg [0:3] Rin, Rout;
  reg [7:0] Sum;
  wire Clear, AddSub, Extern, Ain, Gin, Gout, FRin;
  wire [1:0] Count;
  wire [0:3] T, I, Xreg, Y;
  wire [7:0] R0, R1, R2, R3, A, G;
  wire [1:6] Func, FuncReg;
  integer k;

  upcount counter (Clear, Clock, Count);
  dec2to4 decT (Count, 1'b1, T);

  assign Clear = Reset | Done | (~w & T[0]);
  assign Func = {F, Rx, Ry};
  assign FRin = w & T[0];

  regn functionreg (Func, FRin, Clock, FuncReg);
  defparam functionreg.n = 6;
  dec2to4 decI (FuncReg[1:2], I, 1'b1);
  dec2to4 decX (FuncReg[3:4], Xreg, 1'b1);
  dec2to4 decY (FuncReg[5:6], Y, 1'b1);

  assign Extern = I[0] & T[1];
  assign Done = ((I[0] | I[1]) & T[1]) | ((I[2] | I[3]) & T[3]);
  assign Ain = (I[2] | I[3]) & T[1];
  assign Gin = (I[2] | I[3]) & T[2];
  assign Gout = (I[2] | I[3]) & T[3];
  assign AddSub = I[3];

  . . . continued in Part b.

```

Figure 7.77 Code for the processor (Part a).

At the end of Figure 7.78, the bus is described with an **if-else** statement which represents multiplexers that place the appropriate data onto *BusWires*, depending on the values of R_{out} , G_{out} , and *Extern*.

The circuits synthesized from the code in Figures 7.77 and 7.78 are functionally equivalent. The style of code in Figure 7.78 has the advantage that it does not require the manual

```

// RegCntl
always @(I, T, Xreg, Y)
  for (k = 0; k < 4; k = k+1)
    begin
      Rin[k] = ((I[0] | I[1]) & T[1] & Xreg[k]) |
        ((I[2] | I[3]) & T[3] & Xreg[k]);
      Rout[k] = (I[1] & T[1] & Y[k]) | ((I[2] | I[3]) &
        ((T[1] & Xreg[k]) | (T[2] & Y[k])));
    end

trin tri_ext (Data, Extern, BusWires);
regn reg_0 (BusWires, Rin[0], Clock, R0);
regn reg_1 (BusWires, Rin[1], Clock, R1);
regn reg_2 (BusWires, Rin[2], Clock, R2);
regn reg_3 (BusWires, Rin[3], Clock, R3);

trin tri_0 (R0, Rout[0], BusWires);
trin tri_1 (R1, Rout[1], BusWires);
trin tri_2 (R2, Rout[2], BusWires);
trin tri_3 (R3, Rout[3], BusWires);
regn reg_A (BusWires, Ain, Clock, A);

// alu
always @(AddSub, A, BusWires)
  if (!AddSub)
    Sum = A + BusWires;
  else
    Sum = A - BusWires;

regn reg_G (Sum, Gin, Clock, G);
trin tri_G (G, Gout, BusWires);

endmodule

```

Figure 7.77 Code for the processor (Part b).

effort of analyzing Table 7.3 to generate the logic expressions for the control signals in Figure 7.77. By using the style of code in Figure 7.78, these expressions are produced automatically by the Verilog compiler as a result of analyzing the **case** statements. The style of code in Figure 7.78 is less prone to careless errors. Also, using this style of code it would be straightforward to provide additional capabilities in the processor, such as adding other operations.

```

module proc (Data, Reset, w, Clock, F, Rx, Ry, Done, BusWires);
  input [7:0] Data;
  input Reset, w, Clock;
  input [1:0] F, Rx, Ry;
  output reg [7:0] BusWires;
  output reg Done;
  reg [7:0] Sum;
  reg [0:3] Rin, Rout;
  reg Extern, Ain, Gin, Gout, AddSub;
  wire [1:0] Count, I;
  wire [0:3] Xreg, Y;
  wire [7:0] R0, R1, R2, R3, A, G;
  wire [1:6] Func, FuncReg, Sel;

  wire Clear = Reset | Done | (~w & ~Count[1] & ~Count[0]);
  upcount counter (Clear, Clock, Count);
  assign Func = {F, Rx, Ry};
  wire FRin = w & ~Count[1] & ~Count[0];
  regn functionreg (Func, FRin, Clock, FuncReg);
  defparam functionreg.n = 6;
  assign I = FuncReg[1:2];
  dec2to4 decX (FuncReg[3:4], Xreg, 1'b1);
  dec2to4 decY (FuncReg[5:6], Y, 1'b1);

  always @(Count, I, Xreg, Y)
  begin
    Extern = 1'b0; Done = 1'b0; Ain = 1'b0; Gin = 1'b0;
    Gout = 1'b0; AddSub = 1'b0; Rin = 4'b0; Rout = 4'b0;
    case (Count)
      2'b00: ; //no signals asserted in time step T0
      2'b01: //define signals in time step T1
        case (I)
          2'b00: begin //Load
            Extern = 1'b1; Rin = Xreg; Done = 1'b1;
          end
          2'b01: begin //Move
            Rout = Y; Rin = Xreg; Done = 1'b1;
          end
          default: begin //Add, Sub
            Rout = Xreg; Ain = 1'b1;
          end
        endcase
    end
  end
  . . . continued in Part b.

```

Figure 7.78 Alternative code for the processor (Part a).