

```

module Count_Ones_STR_STR (count, Ready, data, Start, clock, reset_b);
// Mux – decoder implementation of control logic
// controller is structural
// datapath is structural
parameter R1_size = 8, R2_size = 4;
output [R2_size -1: 0] count;
output Ready;
input [R1_size -1: 0] data;
input Start, clock, reset_b;
wire Load_regs, Shift_left, Incr_R2, Zero, E;

Controller_STR M0 (Ready, Load_regs, Shift_left, Incr_R2, Start, E, Zero, clock,
reset_b);

Datapath_STR M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock);
endmodule

```

```

module Controller_STR (Ready, Load_regs, Shift_left, Incr_R2, Start, E, Zero,
clock, reset_b);
output Ready;
output Load_regs, Shift_left, Incr_R2;
input Start;
input E, Zero;
input clock, reset_b;
supply0 GND;
supply1 PWR;

parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11; // Binary code
wire Load_regs, Shift_left, Incr_R2;
wire G0, G0_b, D_in0, D_in1, G1, G1_b;
wire Zero_b = ~Zero;
wire E_b = ~E;
wire [1: 0] select = {G1, G0};
wire [0: 3] Decoder_out;
assign Ready = ~Decoder_out[0];
assign Incr_R2 = ~Decoder_out[1];
assign Shift_left = ~Decoder_out[2];
and (Load_regs, Ready, Start);
mux_4x1_beh Mux_1 (D_in1, GND, Zero_b, PWR, E_b, select);
mux_4x1_beh Mux_0 (D_in0, Start, GND, PWR, E, select);
D_flip_flop_AR_b M1 (G1, G1_b, D_in1, clock, reset_b);
D_flip_flop_AR_b M0 (G0, G0_b, D_in0, clock, reset_b);
decoder_2x4_df M2 (Decoder_out, G1, G0, GND);
endmodule

```

```

module Datapath_STR (count, E, Zero, data, Load_regs, Shift_left, Incr_R2,
clock);
parameter R1_size = 8, R2_size = 4;
output [R2_size -1: 0] count;

```

```

output          E, Zero;
input           [R1_size -1: 0]  data;
input           Load_regs, Shift_left, Incr_R2, clock;
wire           [R1_size -1: 0]   R1;
wire           Zero;
supply0        Gnd;
supply1        Pwr;
assign Zero = (R1 == 0);      // implicit combinational logic
Shift_Reg       M1      (R1, data, Gnd, Shift_left, Load_regs, clock,
Pwr);
Counter         M2      (count, Load_regs, Incr_R2, clock, Pwr);
D_flip_flop_AR  M3      (E, w1, clock, Pwr);
and            (w1, R1[R1_size - 1], Shift_left);
endmodule

```

```

module Shift_Reg (R1, data, SI_0, Shift_left, Load_regs, clock, reset_b);
parameter      R1_size = 8;
output         [R1_size -1: 0]    R1;
input          [R1_size -1: 0]    data;
input          SI_0, Shift_left, Load_regs;
input          clock, reset_b;
reg           [R1_size -1: 0]     R1;
always @ (posedge clock, negedge reset_b)
  if (reset_b == 0) R1 <= 0;
  else begin
    if (Load_regs) R1 <= data; else
      if (Shift_left) R1 <= {R1[R1_size -2: 0], SI_0}; end
endmodule

```

```

module Counter (R2, Load_regs, Incr_R2, clock, reset_b);
parameter      R2_size = 4;
output         [R2_size -1: 0]    R2;
input          Load_regs, Incr_R2;
input          clock, reset_b;
reg           [R2_size -1: 0]     R2;
always @ (posedge clock, negedge reset_b)
  if (reset_b == 0) R2 <= 0;
  else if (Load_regs) R2 <= {R2_size {1'b1}};    // Fill with 1
  else if (Incr_R2 == 1) R2 <= R2 + 1;
endmodule

```

```

module D_flip_flop_AR (Q, D, CLK, RST_b);
output         Q;
input          D, CLK, RST_b;
reg           Q;
always @ (posedge CLK, negedge RST_b)
  if (RST_b == 0) Q <= 1'b0;
  else Q <= D;
endmodule

```

```

module D_flip_flop_AR_b (Q, Q_b, D, CLK, RST_b);
    output      Q, Q_b;
    input  D, CLK, RST_b;
    reg    Q;
    assign      Q_b = ~Q;
    always @ (posedge CLK, negedge RST_b)
        if (RST_b == 0) Q <= 1'b0;
        else Q <= D;
endmodule

// Behavioral description of four-to-one line multiplexer
// Verilog 2005 port syntax
module mux_4x1_beh
(output reg    m_out,
 input  in_0, in_1, in_2, in_3,
 input [1: 0]  select
);
    always @ (in_0, in_1, in_2, in_3, select)      // Verilog 2005 syntax
        case (select)
            2'b00:    m_out = in_0;
            2'b01:    m_out = in_1;
            2'b10:    m_out = in_2;
            2'b11:    m_out = in_3;
        endcase
endmodule

// Dataflow description of two-to-four-line decoder
// See Fig. 4.19. Note: The figure uses symbol E, but the
// Verilog model uses enable to indicate functionality clearly.
module decoder_2x4_df (D, A, B, enable);
    output      [0: 3]  D;
    input  A, B;
    input  enable;
    assign      D[0] = !(A && !B && !enable),
                  D[1] = !(A && B && !enable),
                  D[2] = !(A && !B && !enable),
                  D[3] = !(A && B && !enable);
endmodule

module t_Count_Ones;
    parameter R1_size = 8, R2_size = 4;
    wire      [R2_size -1: 0]      R2;
    wire      [R2_size -1: 0]      count;
    wire      Ready;
    reg       [R1_size -1: 0]      data;
    reg       Start, clock, reset_b;
    wire      [1: 0]              state;    // Use only for debug
    assign state = {M0.M0.G1, M0.M0.G0};
    Count_Ones_STR_STR M0 (count, Ready, data, Start, clock, reset_b);
    initial #650 $finish;
    initial begin clock = 0; #5 forever #5 clock = ~clock; end
    initial fork

```

```
#1 reset_b = 1;
#3 reset_b = 0;
#4 reset_b = 1;
#27 reset_b = 0;
#29 reset_b = 1;
#355 reset_b = 0;
#365 reset_b = 1;
#4 data = 8'Hff;
#145 data = 8'haa;
# 25 Start = 1;
# 35 Start = 0;
#55 Start = 1;
#65 Start = 0;
#395 Start = 1;
#405 Start = 0;
join
endmodule
```
