# Lab9

Monday, October 25, 2021 9:37 PM

Prelab9_step1.sv

```
`default_nettype none
// Empty top module

module top (
  // I/O ports
  input  logic hz100, reset,
  input  logic [20:0] pb,
  output logic [7:0] left, right,
         ss7, ss6, ss5, ss4, ss3, ss2, ss1, ss0,
  output logic red, green, blue,

  // UART ports
  output logic [7:0] txdata,
  input  logic [7:0] rxdata,
  output logic txclk, rxclk,
  input  logic txready, rxready
);

  // Your code goes here...
  count8du c8_1(.CLK(hz100), .RST(reset), .Q(right));

endmodule

// Add more modules down here...
module count8du(input logic CLK, input logic RST, output logic [7:0] Q);

  logic [7:0] next_Q;
  always_ff @ (posedge CLK, posedge RST) begin //state variables

    if(RST == 1'b1)
      Q <= 8'b0;

    else
      Q <= next_Q;
  end

  always_comb begin //next-state equations
    if (Q == 8'd99)
      next_Q = 8'b0; //d or b doesn't matter
    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end
  end

endmodule
```

Lab9_step1.sv

```
`default_nettype none
// Empty top module

module top (
  // I/O ports
  input  logic hz100, reset,
  input  logic [20:0] pb,
  output logic [7:0] left, right,
         ss7, ss6, ss5, ss4, ss3, ss2, ss1, ss0,
  output logic red, green, blue,

  // UART ports
  output logic [7:0] txdata,
  input  logic [7:0] rxdata,
  output logic txclk, rxclk,
  input  logic txready, rxready
);

  // Your code goes here...
  count8du c8_1(.CLK(hz100), .RST(reset), .DIR(pb[17]), .Q(right));

endmodule

// Add more modules down here...
module count8du(input logic CLK, RST, DIR, output logic [7:0] Q);

  logic [7:0] next_Q;
  always_ff @ (posedge CLK, posedge RST) begin //state variables

    if(RST == 1'b1)
      Q <= 8'b0;

    else
      Q <= next_Q;
  end

  always_comb begin //next-state equations
    if(DIR == 1'b0) begin
      next_Q[0] = ~Q[0];
      next_Q[1] = Q[1] ^ ~Q[0];
      next_Q[2] = Q[2] ^ (~Q[1] & ~Q[0]);
      next_Q[3] = Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[4] = Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[5] = Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[6] = Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[7] = Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] &
~Q[0]);
    end

    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end

    if (Q == 8'd99)
      next_Q = 8'b0; //d or b doesn't matter
    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end
  end

endmodule
```

Lab9_step2.sv

```
`default_nettype none
// Empty top module

module top (
  // I/O ports
  input  logic hz100, reset,
  input  logic [20:0] pb,
  output logic [7:0] left, right,
         ss7, ss6, ss5, ss4, ss3, ss2, ss1, ss0,
  output logic red, green, blue,

  // UART ports
  output logic [7:0] txdata,
  input  logic [7:0] rxdata,
  output logic txclk, rxclk,
  input  logic txready, rxready
);

  // Your code goes here...
  count8du c8_
1(.CLK(hz100), .RST(reset), .DIR(pb[17]), .MAX(8'd49), .Q(right));

endmodule

// Add more modules down here...
module count8du(input logic CLK, RST, DIR, input logic [7:0]MAX, output logic
[7:0] Q);

  logic [7:0] next_Q;
  always_ff @ (posedge CLK, posedge RST) begin //state variables

    if(RST == 1'b1)
      Q <= 8'b0;

    else
      Q <= next_Q;
  end

  always_comb begin //next-state equations
    if(DIR == 1'b0) begin
      next_Q[0] = ~Q[0];
      next_Q[1] = Q[1] ^ ~Q[0];
      next_Q[2] = Q[2] ^ (~Q[1] & ~Q[0]);
      next_Q[3] = Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[4] = Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[5] = Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[6] = Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[7] = Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] &
~Q[0]);
    end

    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end

    if (Q == MAX)
      next_Q = 8'b0; //d or b doesn't matter
    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end
  end

endmodule
```

Lab9_step3.sv

```
`default_nettype none
// Empty top module

module top (
  // I/O ports
  input  logic hz100, reset,
  input  logic [20:0] pb,
  output logic [7:0] left, right,
         ss7, ss6, ss5, ss4, ss3, ss2, ss1, ss0,
  output logic red, green, blue,

  // UART ports
  output logic [7:0] txdata,
  input  logic [7:0] rxdata,
  output logic txclk, rxclk,
  input  logic txready, rxready
);

  // Your code goes here...
  count8du c8_
1(.CLK(hz100), .RST(reset), .DIR(pb[17]), .E(pb[18]), .MAX(8'd49), .Q(right))
;

endmodule

// Add more modules down here...
module count8du(input logic CLK, RST, DIR, E, input logic [7:0]MAX, output
logic [7:0] Q);

  logic [7:0] next_Q;
  always_ff @ (posedge CLK, posedge RST) begin //state variables

    if(RST == 1'b1)
      Q <= 8'b0;

    else if (E == 1'b1)
      Q <= next_Q;
  end

  always_comb begin //next-state equations
    if(DIR == 1'b0) begin
      next_Q[0] = ~Q[0];
      next_Q[1] = Q[1] ^ ~Q[0];
      next_Q[2] = Q[2] ^ (~Q[1] & ~Q[0]);
      next_Q[3] = Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[4] = Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[5] = Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[6] = Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0]);
      next_Q[7] = Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] &
~Q[0]);
    end

    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end

    if (Q == MAX)
      next_Q = 8'b0; //d or b doesn't matter
    else begin
    next_Q[0] = ~Q[0];
    next_Q[1] = Q[1] ^ Q[0];
    next_Q[2] = Q[2] ^ (&Q[1:0]);
    next_Q[3] = Q[3] ^ (&Q[2:0]);
    next_Q[4] = Q[4] ^ (&Q[3:0]);
    next_Q[5] = Q[5] ^ (&Q[4:0]);
    next_Q[6] = Q[6] ^ (&Q[5:0]);
    next_Q[7] = Q[7] ^ (&Q[6:0]);
    end
  end

endmodule
```