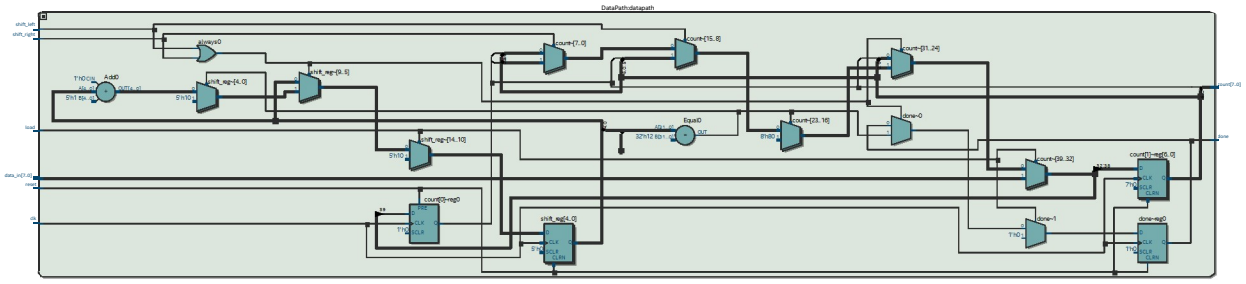
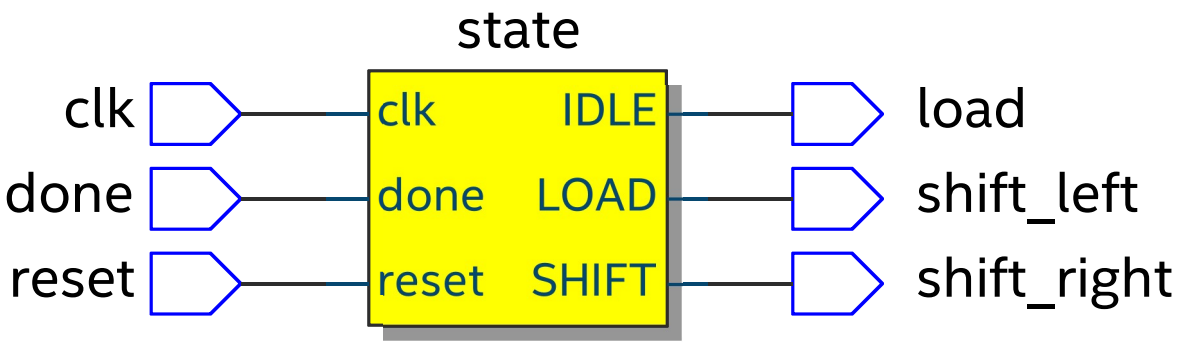
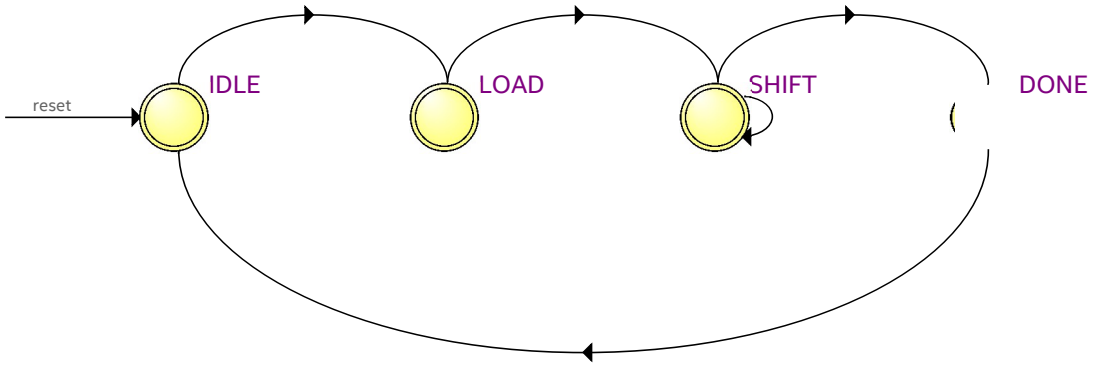
```
1   /*-------------------------------------------------------------------------------
2   Name Lamin Jammeh
3   CLass: EE417 Summer 2024
4   Lesson 07 HW Question 2
5   Group: Ron Kalin/ Lamin Jammeh
6   Project Description: Datapath module takes comman from controoler module and counts
7   use a one hot count by shifting the value 1 left and right in an 8 bit input
8   -------------------------------------------------------------------------------*/
9
10  /*------------up/down count-------------------------------*/
11  module DataPath #(parameter WIDTH = 8, parameter CYCLES = 18) (
12      input clk,
13      input reset,
14      input shift_left,
15      input shift_right,
16      input load,
17      input [WIDTH-1:0] data_in,
18      output reg [WIDTH-1:0] count,
19      output reg done
20  );
21
22  reg [4:0] shift_reg; // 5-bit counter to count up to 18 CYCLES
23
24  always @ (posedge clk or posedge reset)
25  begin
26      //condition for reset
27      if (reset)
28      begin
29          count <= 8'b00000001; // output count at reset is the first state of the counter
30          shift_reg <= 0;
31          done <= 0;
32      end
33      //condition to start loading the output
34      else if (load)
35      begin
36          count <= data_in; // count is still 1 when load comes on
37          shift_reg <= 1;
38          done <= 0;
39      end
40      //condition to shift 1 left and right in count
41      else if (shift_left || shift_right)
42      begin
43          if (shift_reg == CYCLES)
44          begin
45              shift_reg <= 1;
46              count <= 8'b00000001;
47              //done <= 1;
48          end
49          else begin
50              shift_reg <= shift_reg + 1;            //all start conditions are active,
    increment shift_reg
51              //define a condition for counting up
52              if (shift_left)
53              begin
54                 count <= {count[WIDTH-2:0], count[WIDTH-1]}; // Shift left by joining
    count[6:0] to count[7]
55              end
56              else if (shift_right)
57              begin
58                 count <= {count[0], count[WIDTH-1:1]}; // Shift right by joining count[0] to
    count[7:1]
59              end
60          end
61          done <= (shift_reg == CYCLES);
62      end
63  end
64
65  endmodule
66
```



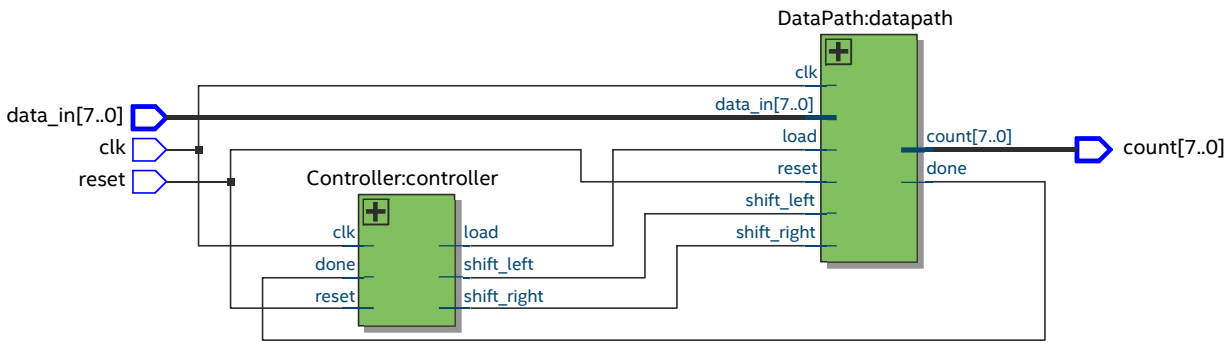
```
/*-------------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 07 HW Question 2
Group: Ron Kalin/ Lamin Jammeh
Project Description: COntroller module sends control command to the Datapath to shift
the value 1 left or right in an input data
-------------------------------------------------------------------------------*/

/*----------------------------up/down counter controller--------------*/
module Controller (
    input clk,
    input reset,
    input done,
    output reg shift_left,
    output reg shift_right,
    output reg load
);

reg [1:0] state, next_state;

parameter    IDLE  = 2'b00;
parameter    LOAD  = 2'b01;
parameter    SHIFT = 2'b10;
parameter    DONE  = 2'b11;

// State transition logic
always @ (posedge clk or posedge reset) begin
    if (reset)
        state <= IDLE;
    else
        state <= next_state;
end

// Next state logic
always @ (*) begin
    case (state)
        IDLE: next_state = LOAD;
        LOAD: next_state = SHIFT;
        SHIFT: next_state = done ? DONE : SHIFT; //keep shifting if Done is false
        DONE: next_state = IDLE;
        default: next_state = IDLE;
    endcase
end

// Output logic
always @ (*) begin
    shift_left  = 0;   // Default direction
    shift_right = 0;
           load = 0;
    case (state)
        IDLE: begin
               load = 1;
             end
        LOAD: begin
               shift_left = 1;
              end
        SHIFT: begin
                 if (shift_left)
                 begin
                   shift_left = 1;
                   shift_right = 0;
                 end
            else begin
                   shift_left = 0;
                   shift_right = 1;
                 end
        end
        DONE: begin
```

```
70                // No control signals
71            end
72        endcase
73    end
74
75    endmodule
```

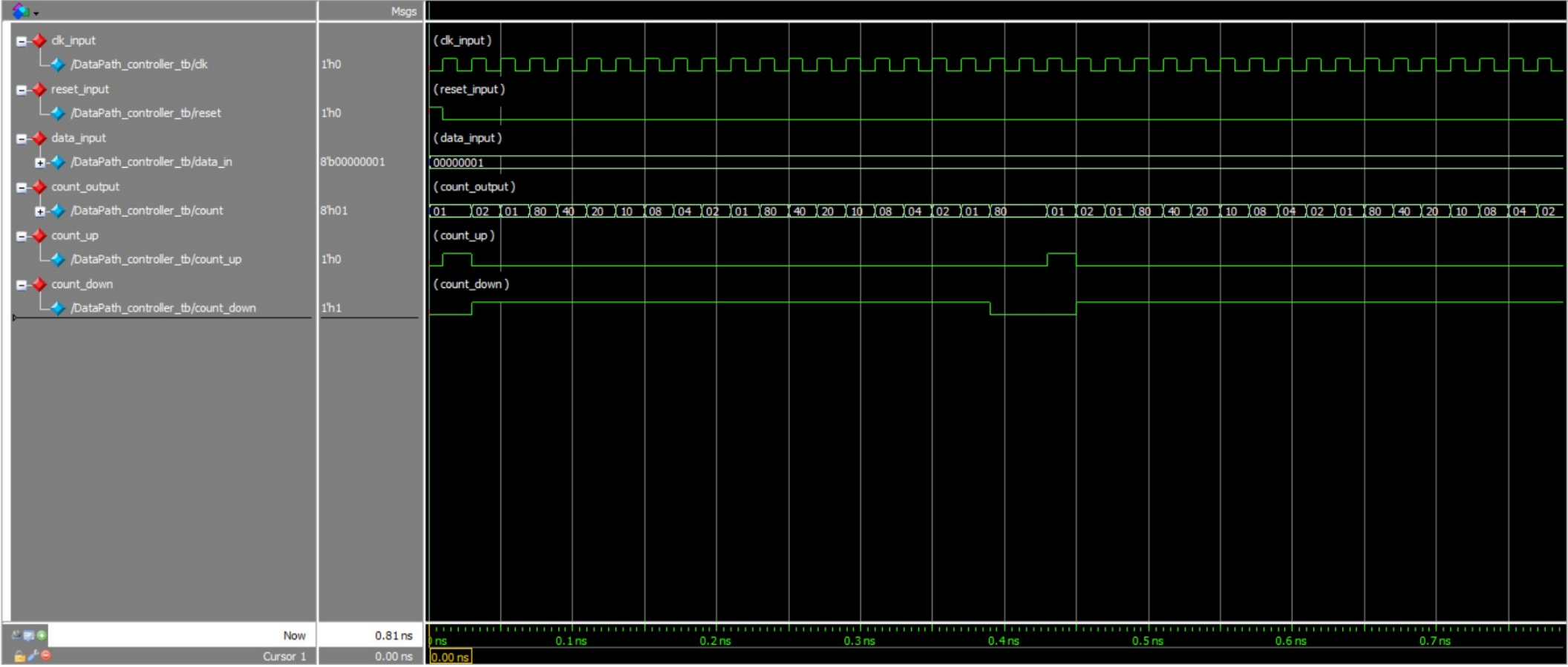

state
clk
done
reset
clk
done
reset
IDLE
LOAD
SHIFT
load
shift_left
shift_right


reset
IDLE
LOAD
SHIFT
DONE

```
1  /*-------------------------------------------------------------------------------
2  Name Lamin Jammeh
3  CLass: EE417 Summer 2024
4  Lesson 07 HW Question 2
5  Group: Ron Kalin/ Lamin Jammeh
6  Project Description: This module combines submodule DataPath and Controller to perform
7  the shift functions. It is the top module for the design
8  -------------------------------------------------------------------------------*/
9
10 /*-----------------------up/down counter-----------------------*/
11 module DataPath_controller #(parameter WIDTH = 8, parameter CYCLES = 18) (
12     input clk,
13     input reset,
14     input  [WIDTH-1:0] data_in,
15     output [WIDTH-1:0] count
16 );
17
18 wire shift_left, shift_right, load, done;
19
20 DataPath #(WIDTH, CYCLES) datapath (
21     .clk(clk),
22     .reset(reset),
23     .shift_left(shift_left),
24     .shift_right(shift_right),
25     .load(load),
26     .data_in(data_in),
27     .count(count),
28     .done(done)
29 );
30
31 Controller controller (
32     .clk(clk),
33     .reset(reset),
34     .done(done),
35     .shift_left(shift_left),
36     .shift_right(shift_right),
37     .load(load)
38 );
39
40 endmodule
41
```


DataPath:datapath
data_in[7..0]
clk
reset
Controller:controller
clk
done
reset
load
shift_left
shift_right
clk
data_in[7..0]
load
reset
shift_left
shift_right
count[7..0]
done
count[7..0]

```
1  /*-------------------------------------------------------------------------------
2  Name Lamin Jammeh
3  CLass: EE417 Summer 2024
4  Lesson 07 HW Question 2
5  Group: Ron Kalin/ Lamin Jammeh
6  Project Description: This is testbench shows the outputs and how the value 1 is shifting
7  -------------------------------------------------------------------------------*/
8
9  /*-----------------------TestBench for up/down count---------------*/
10 module DataPath_controller_tb ();
11
12 //define registers and wires
13 reg clk;
14 reg reset;
15 reg  [7:0] data_in;
16 wire [7:0] count;
17 wire shift_left;
18 wire shift_right;
19
20 //define the units under test UUT
21 DataPath_controller #(8, 18) UUT (
22                                    .clk(clk),
23                                    .reset(reset),
24                                    .data_in(data_in),
25                                    .count(count)
26                                    );
27
28 //monitor internal probe
29 assign count_up   = UUT.controller.shift_left;
30 assign count_down = UUT.controller.shift_right;
31 //clock cycle
32 initial
33    begin
34       clk = 0;
35       forever #10 clk = ~clk;
36    end
37
38 //count output is base on posedge of clk and reset
39 //initialie reset and run enough clk cycle to get all desired counts
40 initial
41    begin
42       reset = 1;
43       data_in = 8'b00000001;  //load initial value
44       #10 reset = 0;
45       #800 $stop;          //run clk for 100 time units change if necessary
46    end
47
48 //display results for each change in count
49 initial
50    begin
51       $monitor ("Time: %0t |  Count_up: %b | Count_down: %b |  count: %b", $time, count_up,
   count_down, count);
52    end
53 endmodule
```


Msgs
clk_input
/DataPath_controller_tb/clk
1'h0
reset_input
/DataPath_controller_tb/reset
1'h0
data_input
/DataPath_controller_tb/data_in
8'b00000001
count_output
/DataPath_controller_tb/count
8'h01
count_up
/DataPath_controller_tb/count_up
1'h0
count_down
/DataPath_controller_tb/count_down
1'h1
(clk_input)
(reset_input)
(data_input)
00000001
(count_output)
01 02 01 80 40 20 10 08 04 02 01 80 40 20 10 08 04 02 01 80 01 02 01 80 40 20 10 08 04 02 01 80 40 20 10 08 04 02
(count_up)
(count_down)
Now
0.81 ns
Cursor 1
0.00 ns
0.1 ns
0.2 ns
0.3 ns
0.4 ns
0.5 ns
0.6 ns
0.7 ns

```
# Time: 0 |     Count_up: 0 | Count_down: 0 | count: 00000001
# Time: 10 |    Count_up: 1 | Count_down: 0 | count: 00000001
# Time: 30 |    Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 50 |    Count_up: 0 | Count_down: 1 | count: 00000001
# Time: 70 |    Count_up: 0 | Count_down: 1 | count: 10000000
# Time: 90 |    Count_up: 0 | Count_down: 1 | count: 01000000
# Time: 110 |   Count_up: 0 | Count_down: 1 | count: 00100000
# Time: 130 |   Count_up: 0 | Count_down: 1 | count: 00010000
# Time: 150 |   Count_up: 0 | Count_down: 1 | count: 00001000
# Time: 170 |   Count_up: 0 | Count_down: 1 | count: 00000100
# Time: 190 |   Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 210 |   Count_up: 0 | Count_down: 1 | count: 00000001
# Time: 230 |   Count_up: 0 | Count_down: 1 | count: 10000000
# Time: 250 |   Count_up: 0 | Count_down: 1 | count: 01000000
# Time: 270 |   Count_up: 0 | Count_down: 1 | count: 00100000
# Time: 290 |   Count_up: 0 | Count_down: 1 | count: 00010000
# Time: 310 |   Count_up: 0 | Count_down: 1 | count: 00001000
# Time: 330 |   Count_up: 0 | Count_down: 1 | count: 00000100
# Time: 350 |   Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 370 |   Count_up: 0 | Count_down: 1 | count: 00000001
# Time: 390 |   Count_up: 0 | Count_down: 0 | count: 10000000
# Time: 430 |   Count_up: 1 | Count_down: 0 | count: 00000001
# Time: 450 |   Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 470 |   Count_up: 0 | Count_down: 1 | count: 00000001
# Time: 490 |   Count_up: 0 | Count_down: 1 | count: 10000000
# Time: 510 |   Count_up: 0 | Count_down: 1 | count: 01000000
# Time: 530 |   Count_up: 0 | Count_down: 1 | count: 00100000
# Time: 550 |   Count_up: 0 | Count_down: 1 | count: 00010000
# Time: 570 |   Count_up: 0 | Count_down: 1 | count: 00001000
# Time: 590 |   Count_up: 0 | Count_down: 1 | count: 00000100
# Time: 610 |   Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 630 |   Count_up: 0 | Count_down: 1 | count: 00000001
# Time: 650 |   Count_up: 0 | Count_down: 1 | count: 10000000
# Time: 670 |   Count_up: 0 | Count_down: 1 | count: 01000000
# Time: 690 |   Count_up: 0 | Count_down: 1 | count: 00100000
# Time: 710 |   Count_up: 0 | Count_down: 1 | count: 00010000
# Time: 730 |   Count_up: 0 | Count_down: 1 | count: 00001000
# Time: 750 |   Count_up: 0 | Count_down: 1 | count: 00000100
# Time: 770 |   Count_up: 0 | Count_down: 1 | count: 00000010
# Time: 790 |   Count_up: 0 | Count_down: 1 | count: 00000001
```