Assignment 4 – Problem 3 – Up and Down counter in BCD or 2421 Code

Create a counter code2421_downCounter (blue block in the above diagram) to count down from (1111) to (0000) and then loop back to (1111) using the 2421 code as given in the table above. The counter should have a synchronized reset, meaning that if the reset goes high at any point, it will be only applied at the next positive clock edge. When the reset is active the output remains at (1111). If the reset is inactive and the enable input is at logic high, the counter counts down at every positive clock edge (Following the 2421 down counting: 1111, 1110, 1101, 1100, 1011, 0100, 0011, 0010, 0001, 0000). If not enabled, it will freeze at the last count reached. The design should output the values in the 2421-code. The module should be designed using a Moore FSM implemented using a case structure.

Design a top module BCD_or_2521_up_down_counter. The counter should have an output of 4 bits displaying digits in BCD or 2421code based on a ***selectCode*** input, a synchronous reset option to reset the counter, an enable to count or freeze the count, a ***mode_updown*** input to select between up counting and down counting, and a clock input to trigger the count at the positive edge. The reset and enable inputs are active high. The count should only output values from 0 to 9 and then loop back.

The top module should instantiate the code2421_downCounter module, and other submodules shown in the block diagram.

This top design must instantiate code2421_2_BCD converter that you have created in the previous lesson 3 (reusability of code).

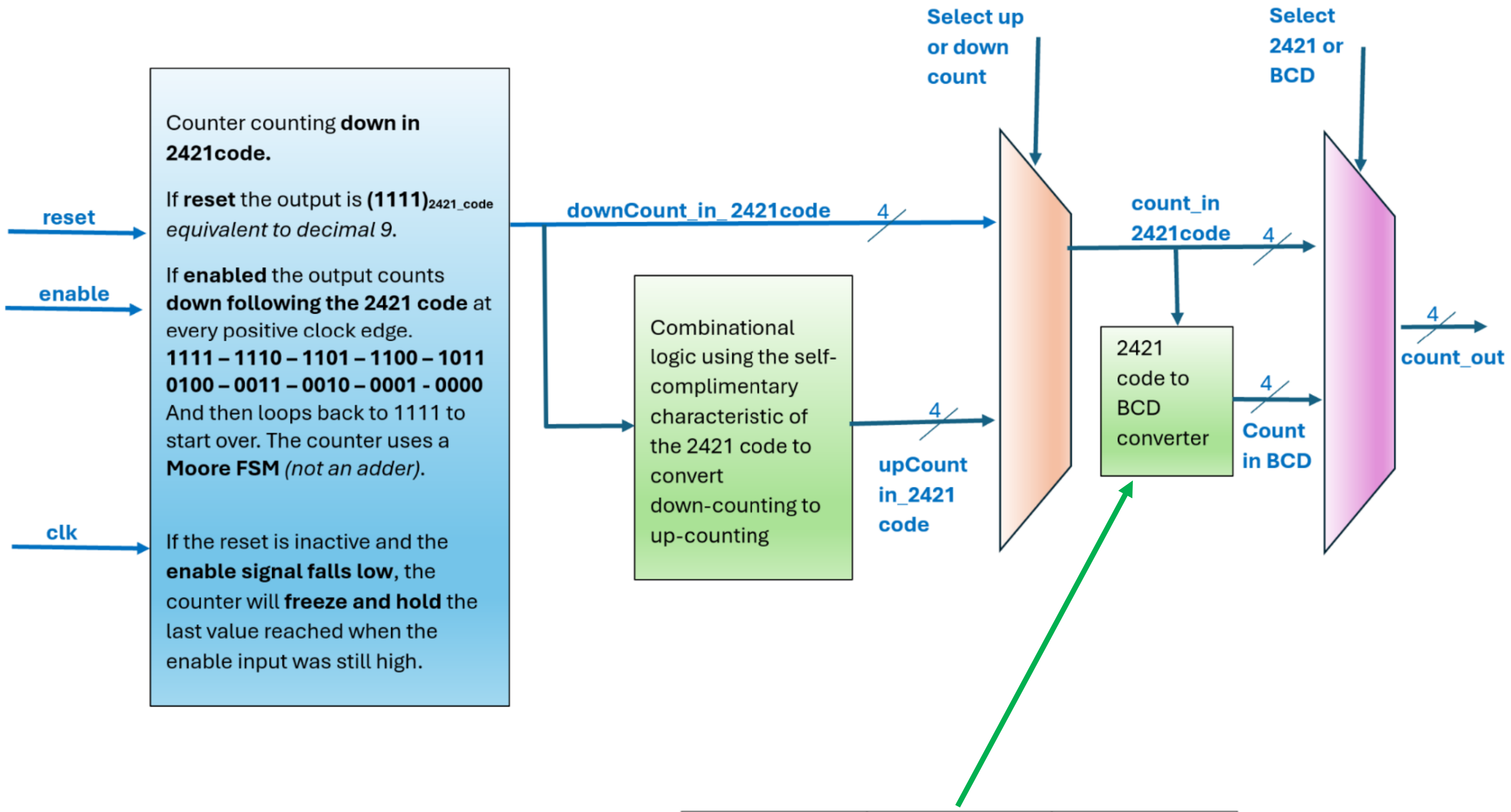
The combinational logic that will convert from up-counting to down-counting will use the self-complimentary characteristics of the 2421 code:

For down counting: 1111 – 1110 – 1110 – 1100 – 0111 – 1000 - 0011 – 0001 – 0001- 0000
(FSM Moore)

For up counting: 0000 – 0001 – 0001 – 0011 – 1000 – 0111 - 1100 – 1110 –1110 - 1111
(self-complimentary characteristic)

Include two 2-to-1 multiplexers to select between up and down counting, and to select between 2421code and BCD count.

Test the counter using a testbench. Feel free to use internal probes to verify your design functionality.


Counter counting **down in 2421code.**

If **reset** the output is **(1111)**$_{2421\_code}$ *equivalent to decimal 9.*

If **enabled** the output counts **down following the 2421 code** at every positive clock edge.
1111 – 1110 – 1101 – 1100 – 1011
0100 – 0011 – 0010 – 0001 - 0000
And then loops back to 1111 to start over. The counter uses a **Moore FSM** *(not an adder)*.

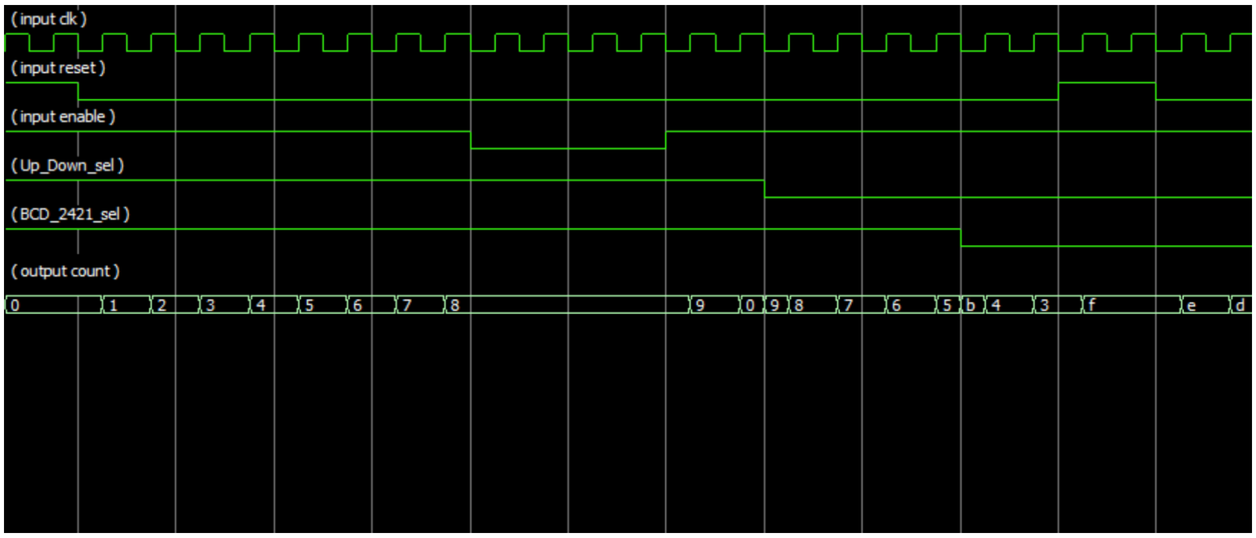
If the reset is inactive and the **enable signal falls low,** the counter will **freeze and hold** the last value reached when the enable input was still high.

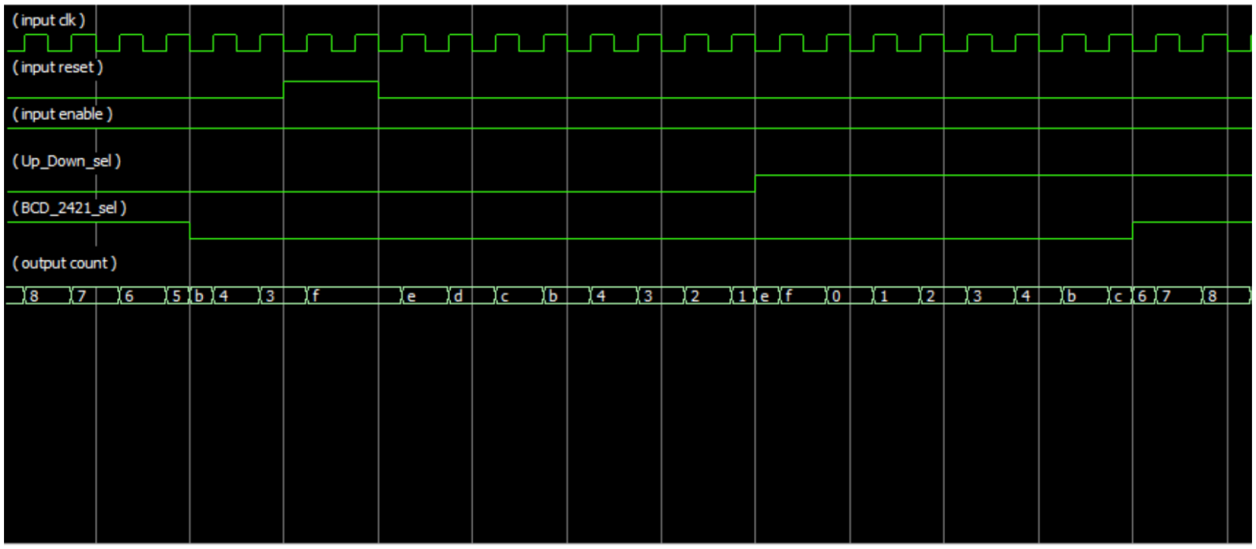
Combinational logic using the self-complimentary characteristic of the 2421 code to convert down-counting to up-counting

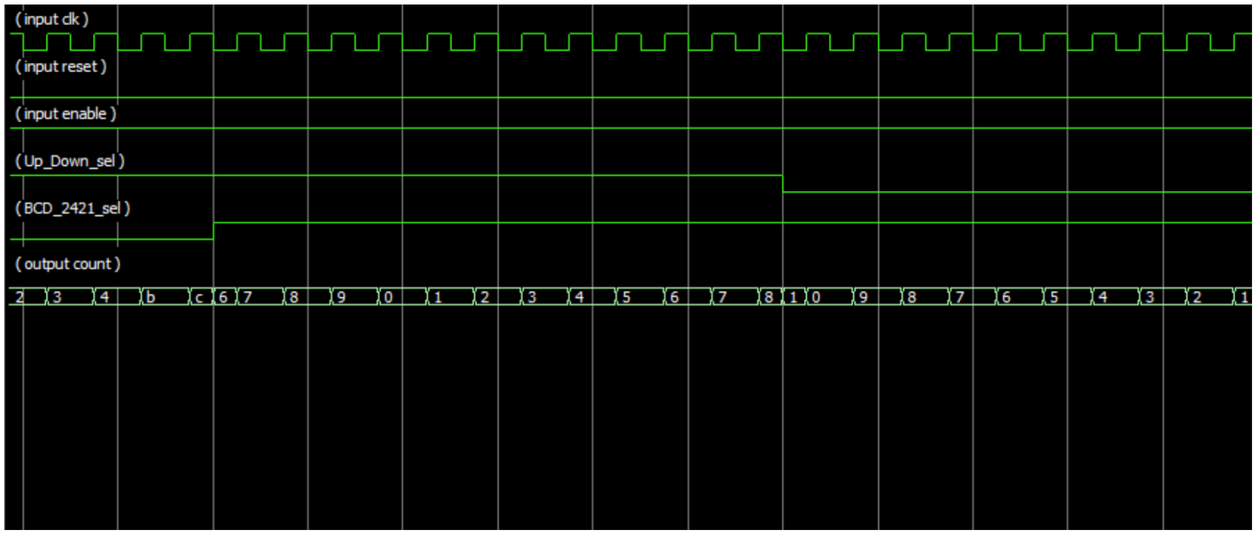
Decimal value	BCD – 8421 weighted code	Weighted 2421 code
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110
9	1001	1111

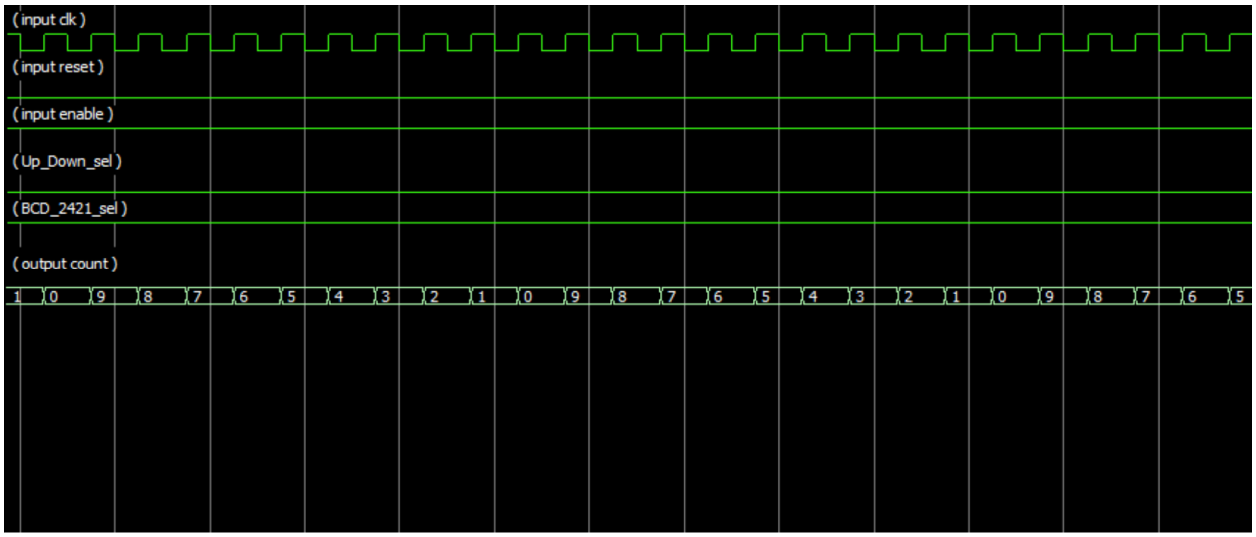
```
module DownCounter_2421 (clk, reset, enable, count);

input              clk, reset, enable;
output reg [3:0]   count;
reg [3:0]  state, next_state;

// Moore Machine
parameter S9 = 4'b1111;     //f = 9
parameter S8 = 4'b1110;     //e = 8
parameter S7 = 4'b1101;     //d = 7
parameter S6 = 4'b1100;     //c = 6
parameter S5 = 4'b1011;     //b = 5
parameter S4 = 4'b0100;     //4 = 4
parameter S3 = 4'b0011;     //3 = 3
parameter S2 = 4'b0010;     //2 = 2
parameter S1 = 4'b0001;     //1 = 1
parameter S0 = 4'b0000;     //0 = 0

always @ (posedge clk)
        if (reset)    state <= S9;           // reset case
   else if (enable)   state <= next_state;   // down counting
   else               state <= state;        // hold value     (optional statement)


 always @ *
    case (state)
       S9 : begin count = 4'b1111;   next_state = S8; end
       S8 : begin count = 4'b1110;   next_state = S7; end
       S7 : begin count = 4'b1101;   next_state = S6; end
       S6 : begin count = 4'b1100;   next_state = S5; end
       S5 : begin count = 4'b1011;   next_state = S4; end
       S4 : begin count = 4'b0100;   next_state = S3; end
       S3 : begin count = 4'b0011;   next_state = S2; end
       S2 : begin count = 4'b0010;   next_state = S1; end
       S1 : begin count = 4'b0001;   next_state = S0; end
       S0 : begin count = 4'b0000;   next_state = S9; end
    endcase

 endmodule
//-----------------------------------------------------------------------------------

 module Code2421_BCD_converter (BCD, Code2421);
 input       [3:0]   Code2421;
 output reg [3:0]    BCD;

 always @ *
 case (Code2421)
   4'b1111 : BCD = 4'b1001;
   4'b1110 : BCD = 4'b1000;
   4'b1101 : BCD = 4'b0111;
   4'b1100 : BCD = 4'b0110;
   4'b1011 : BCD = 4'b0101;
   4'b0100 : BCD = 4'b0100;
   4'b0011 : BCD = 4'b0011;
   4'b0010 : BCD = 4'b0010;
   4'b0001 : BCD = 4'b0001;
   4'b0000 : BCD = 4'b0000;
   endcase
 endmodule
//------------------------------------------------------------------------------------------
```

```
module UpDownCounter_BCD_2421 (clk, reset, enable, Up_Down_sel, BCD_2421_sel, count);

input          clk, reset, enable;
input          Up_Down_sel, BCD_2421_sel;
output [3:0]   count;

wire [3:0]  downCount;
wire [3:0]  upCount;
wire [3:0]  BCD;
wire [3:0]  Code2421;

Code2421_BCD_converter   converter   (BCD, Code2421);
DownCounter_2421         counter     (clk, reset, enable, downCount);

assign upCount  = ~downCount;
assign Code2421 = Up_Down_sel  ? upCount : downCount;
assign count    = BCD_2421_sel ? BCD     : Code2421;

endmodule


module UpDownCounter_BCD_2421_tb ();

reg         clk, reset, enable;
reg         Up_Down_sel, BCD_2421_sel;
wire [3:0]  count;

UpDownCounter_BCD_2421  UUT  (clk, reset, enable, Up_Down_sel, BCD_2421_sel, count);

initial
begin  clk = 1'b0;
   forever begin
   #5  clk = ~clk; end
   end

initial
begin  reset = 1'b1;
  #20  reset = 1'b0;
  #200 reset = 1'b1;
  #20  reset = 1'b0; end

initial
begin  enable = 1'b1;
  #100 enable = 1'b0;
  #40  enable = 1'b1;
  end

initial
begin   Up_Down_sel  = 1'b1;
  #160  Up_Down_sel  = 1'b0;
  #160  Up_Down_sel  = 1'b1;
  #200  Up_Down_sel  = 1'b0; end


initial
begin   BCD_2421_sel = 1'b1;
  #200  BCD_2421_sel = 1'b0;
  #200  BCD_2421_sel = 1'b1; end

endmodule
```


(input clk)
(input reset)
(input enable)
(Up_Down_sel)
(BCD_2421_sel)
(output count)
0 1 2 3 4 5 6 7 8 9 0 9 8 7 6 5 b 4 3 f e d


(input clk)
(input reset)
(input enable)
(Up_Down_sel)
(BCD_2421_sel)
(output count)
8 7 6 5 b 4 3 f e d c b 4 3 2 1 e f 0 1 2 3 4 b c 6 7 8


(input clk)
(input reset)
(input enable)
(Up_Down_sel)
(BCD_2421_sel)
(output count)
2 3 4 b c 6 7 8 9 0 1 2 3 4 5 6 7 8 1 0 9 8 7 6 5 4 3 2 1


(input clk)
(input reset)
(input enable)
(Up_Down_sel)
(BCD_2421_sel)
(output count)
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5