prelab(10)
Design Specification (2)
block diagram of the design or Logic Diagram (3)
verify the design with simulation results + Function explanation (5)
(只有 FSM 即可)

邏輯設計實驗 Lab05 預報

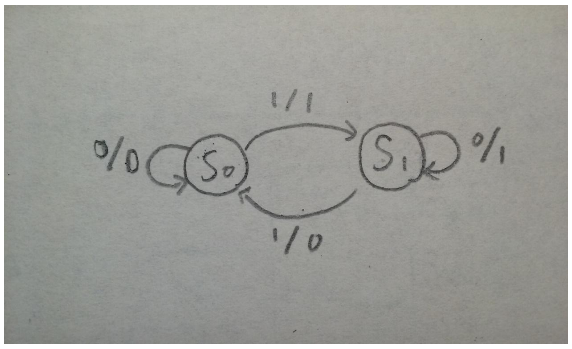
104060012 邱怡庭

Construct a 30-second down counter with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.

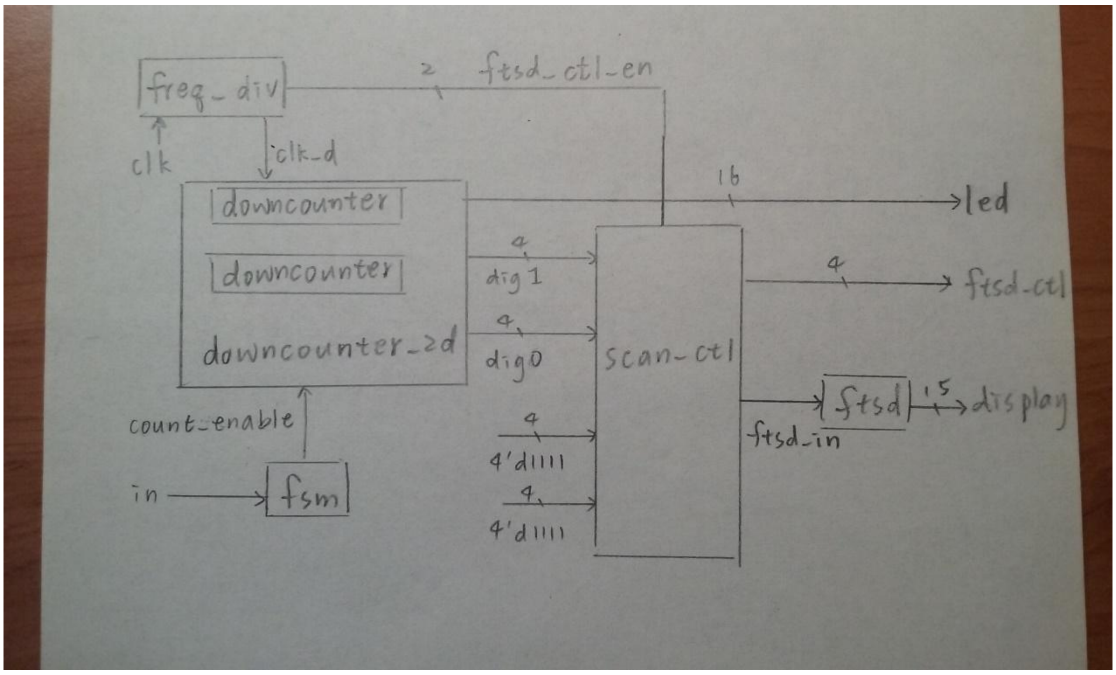
1.1 Write the spec (inputs, outputs, and function table) of the design.

output [14:0] display; // 14 segment display control
output [3:0] ftsd_ctl; // scan control for ftsd
output [15:0] led; //led display control
input clk; // clock
input rst_n; // low active reset
input in; // input control for FSM
wire [1:0] ftsd_ctl_en; // divided output for ftsd ctl
wire clk_d; // divided clock
wire count_enable; // if count is enabled
wire [3:0] dig0,dig1; // second counter output

rst_n	in	clk	count_enable	function
0	x	x	x	Return to 30
1	0	↑	x	Present state
1	1	↑	0	Next state(pause)
1	1	↑	1	Next state(count)



1.2 Draw the related block/logic diagram.



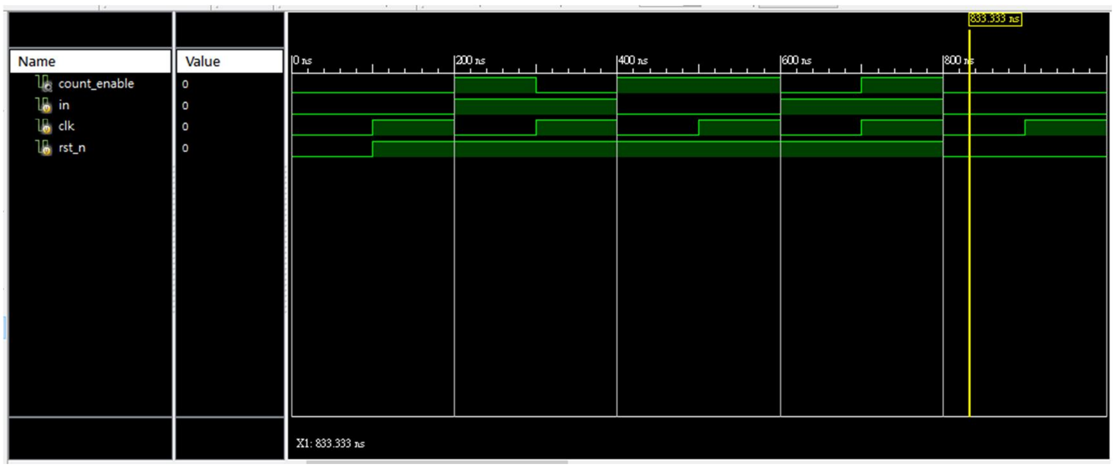
1.3 Use a FSM to implement the function of pause/start function. Use one LED to represent current state.

```
`define STAT_DEF 1'b0
`define STAT_COUNT 1'b1
`define STAT_PAUSE 1'b0
`define ENABLED 1
`define DISABLED 0
module fsm(
   count_enable, // if counter is enabled
    in, //input control
    clk, // global clock signal
    rst_n // low active reset
    );

    // outputs
    output count_enable; // if counter is enabled

    // inputs
    input clk; // global clock signal
    input rst_n; // low active reset
    input in; //input control

    reg count_enable; // if counter is enabled
    reg state; // state of FSM
    reg next_state; // next state of FSM

    // FSM state decision
    always @*
            case (state)
            `STAT_DEF:
                    if (in)
                            begin
                            next_state = `STAT_COUNT;
                            count_enable = `ENABLED;
                            end
                    else
                            begin
                            next_state = `STAT_DEF;
                            count_enable = `DISABLED;
                            end
            `STAT_COUNT:
                    if (in)
                            begin
                            next_state = `STAT_PAUSE;
                            count_enable = `DISABLED;
                            end
                    else
                            begin
                            next_state = `STAT_COUNT;
                            count_enable = `ENABLED;
                            end
            `STAT_PAUSE:
                    if (in)
                            begin
                            next_state = `STAT_COUNT;
                            count_enable = `ENABLED;
                            end
                    else
```

```
                        begin
                        next_state = `STAT_PAUSE;
                        count_enable = `DISABLED;
                        end
                default:
                        begin
                        next_state = `STAT_DEF;
                        count_enable = `DISABLED;
                        end
                endcase

        // FSM state transition
        always @(posedge clk or negedge rst_n)
                if (~rst_n)
                        state <= `STAT_DEF;
                else
                        state <= next_state;



         endmodule
```

1.4 Use Verilog to implement 1.3 and verify the design with simulation results.

The Final Result:



833.333 ns
Name
Value
count_enable 0
in 0
clk 0
rst_n 0
0 ns
200 ns
400 ns
600 ns
800 ns
X1: 833.333 ns