**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. **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. **Draw the related block/logic diagram.**



* 1. **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:**

