THE UNIVERSITY OF DANANG

UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Advanced Science and Technology

REPORT PROJECT

DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

TOPIC: TIMER

Instructor	: Huynh Viet Thang
Class	: 21ECE
Group	: 3
Group members	:Luong Nhu Quynh
	Vo Van Buu
	Tran Hoang Minh
	Nguyen Thi Tam

Da Nang, December 26 th 2024

Contents

Requirement

Design a timer circuit for the following purposes:

- Countdown: The timer should count down from an initial time set by the user to zero, then stop and provide an indication.
- Display: Display minutes and seconds elapsed on 4 seven-segment LEDs (2 LEDs for minutes, 2 LEDs for seconds).
- Alarm: When the countdown reaches zero, trigger an alarm by blinking an LED (dot or seven-segment).
- User input: Allow the user to set the initial countdown time (in minutes) using switches.
- Pause/resume: Enable the user to pause and resume the countdown.
- Reset: Allow the user to reset the timer, returning the display to the initial set time.
- Customization: Students can customize the design as desired to achieve the best results.

I. Block diagram

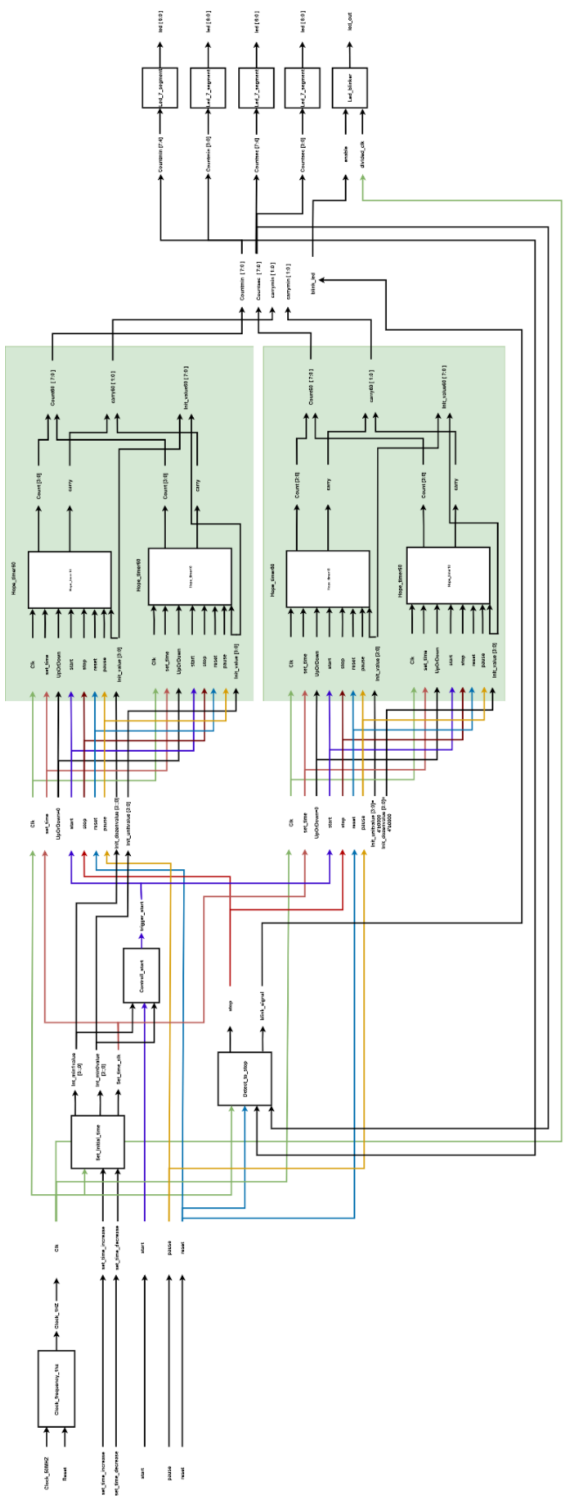


Fig 1. Block diagram of the timer

II. State diagram

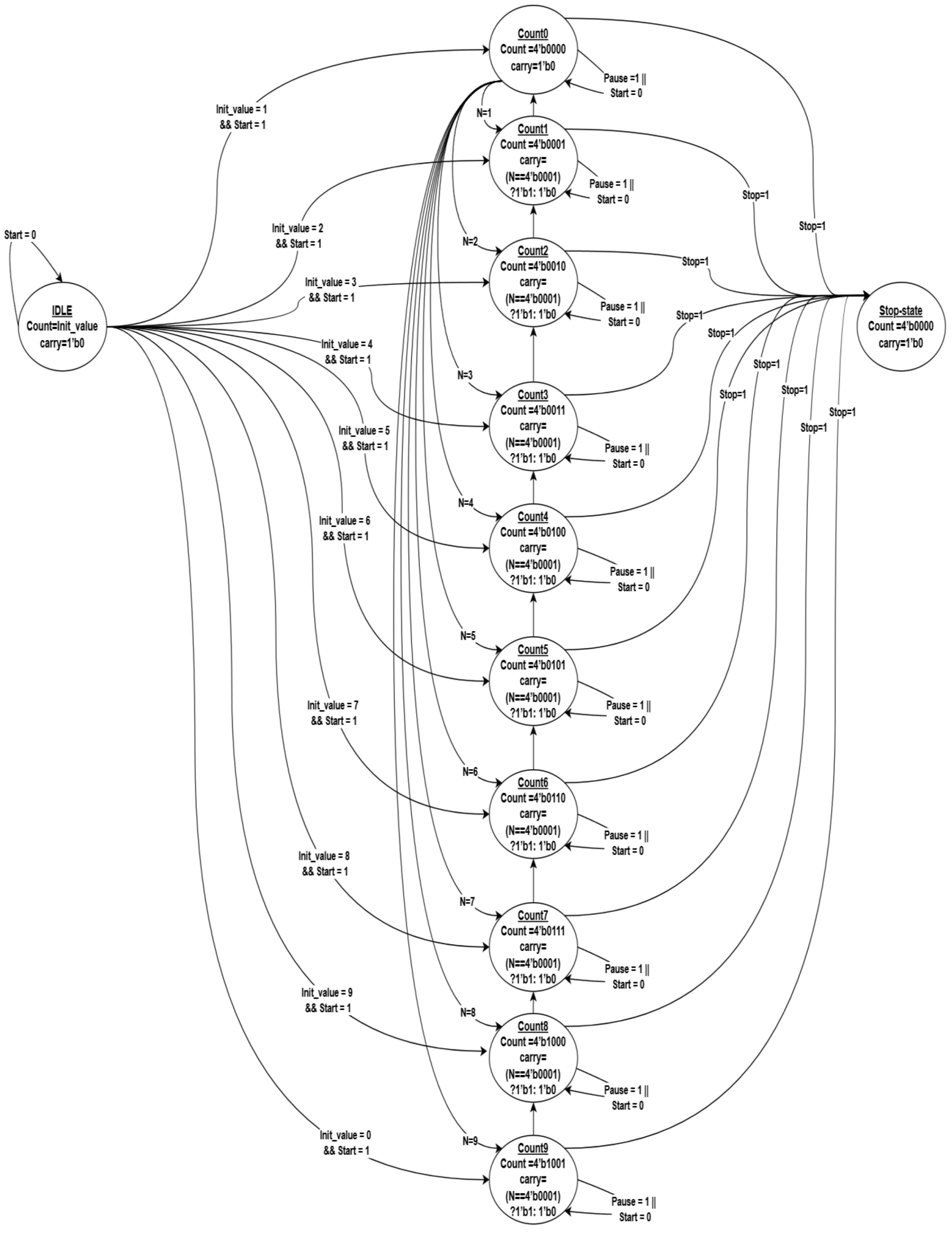


Fig 2. State diagram of the counter

III. Code Verilog

```
module FSM_timer_final

(

	input wire sys_clk,

	input wire set_time_increase,

	input wire set_time_decrease,

	input wire start,

	input wire pause,

	input wire reset,

	output wire [6:0] min1_led,

	output wire [6:0] min0_led,

	output wire [6:0] sec1_led,

	output wire [6:0] sec0_led,

	output wire blink_led

);

wire clk_1hz;

wire [7:0] Init_minvalue ;

wire [7:0] Init_secvalue ;

wire [7:0] Countmin ;

wire [7:0] Countsec ;
wire [1:0] carrymin ;

wire [1:0] carrysec ;

wire enable_led ;



// Create the clock 1 Hz for realtime timer

frequency_1hz Clk_block (.clk_in(sys_clk),.reset(reset),.clk_out(clk_1hz));

// Create the timer

FSM_timer_minsec FPGA_timer (

        .Clk(clk_1hz),

        .set_time_increase(set_time_increase),

        .set_time_decrease(set_time_decrease),

        .start(start),

        .pause(pause),

        .reset(reset),

        .Init_minvalue(Init_minvalue),

        .Init_secvalue(Init_secvalue),

        .Countmin(Countmin),

        .Countsec(Countsec),

        .carrymin(carrymin),

        .carrysec(carrysec),
        .blink_led(enable_led)

);

// Decode Output Count for display with 7 segment led

led_7_segment leg1_min (

        .bcd(Countmin[7:4]),

        .led(min1_led)

);



led_7_segment leg0_min (

        .bcd(Countmin[3:0]),

        .led(min0_led)

);

led_7_segment leg1_sec (

        .bcd(Countsec[7:4]),

        .led(sec1_led)

);

led_7_segment leg0_sec (

        .bcd(Countsec[3:0]),

        .led(sec0_led)

);

// Blink led when the timer is finished...
```

```
led_blinker toggle_led (

		.divided_clk(clk_1hz),

		.enable(enable_led),

		.led_out(blink_led)

);

endmodule
```

– Code Verilog Testbench

```
`timescale 1ns / 1ns

module FSM_timer_final_tb ;

localparam T = 20;  //( T=20ns )

reg sys_clk;

reg set_time_increase;

reg set_time_decrease;

reg start;

reg pause;

reg reset;

wire [6:0] min1_led;

wire [6:0] min0_led;

wire [6:0] sec1_led;

wire [6:0] sec0_led;

wire blink_led;
```

```
FSM_timer_final dut (
            .sys_clk(sys_clk),
            .set_time_increase(set_time_increase),
            .set_time_decrease(set_time_decrease),
            .start(start),
            .pause(pause),
            .reset(reset),
            .min1_led(min1_led),
            .min0_led(min0_led),
            .sec1_led(sec1_led),
            .sec0_led(sec0_led),
            .blink_led(blink_led)
);
   // Clock generation with T=20 ns
    always begin
       sys_clk = 1'b1;
        #(T / 2);
       sys_clk = 1'b0;
        #(T / 2);
    end
```

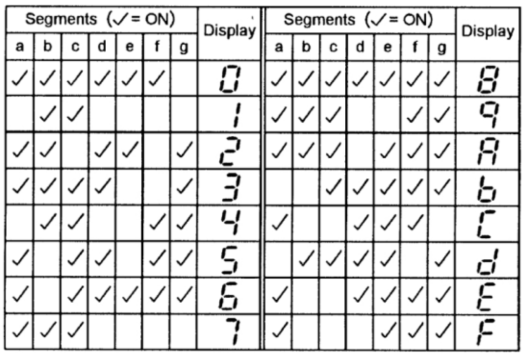
```
  // Reset logic

  initial begin

        reset = 1'b0;

        pause = 1'b0;

        start = 1'b0;

        set_time_increase = 1'b1;

        set_time_decrease =1'b1;

// 1s correspond with 50000000*T

        #(50000000*T);

  end

 initial

 begin

 #(25000000*T);

// Increase initial time

set_time_increase = 1'b0;

 #(50000000*T);

 set_time_increase = 1'b1;

 #(50000000*T);

set_time_increase = 1'b0;

 #(50000000*T);

 set_time_increase = 1'b1;
 #(50000000*T);

// decrease initial time

set_time_decrease = 1'b0;

 #(50000000*T);

 set_time_decrease = 1'b1;

 #(50000000*T);

set_time_decrease = 1'b0;

 #(50000000*T);

 set_time_decrease = 1'b1;

 #(50000000*T);

// Test decreases initial time when initial time = 0;

set_time_decrease = 1'b0;

 #(50000000*T);

 set_time_decrease = 1'b1;

 #(50000000*T);

set_time_decrease = 1'b0;

 #(50000000*T);

 set_time_decrease = 1'b1;

 #(100000000*T);

// Increase initial time to 1 minute

set_time_increase = 1'b0;
```

```
 #(50000000*T);

 set_time_increase = 1'b1;

 #(50000000*T);

//Running counter

start = 1'b1;

 #(250000000*T);

// Test set initial time when the timer is starting

set_time_increase = 1'b0;

 #(50000000*T);

 set_time_increase = 1'b1;

 #(50000000*T);

set_time_decrease = 1'b0;

 #(50000000*T);

set_time_decrease = 1'b1;

 #(50000000*T);

// Test Pause function

pause=1'b1;

 #(100000000*T);

// Continousing counting

pause=1'b0;

 #(64'd3000000000*T);
```

```
// Test increasing initial time when timer stopped

set_time_increase = 1'b0;

 #(50000000*T);

 set_time_increase = 1'b1;

 #(50000000*T);

set_time_increase = 1'b0;

 #(50000000*T);

 set_time_increase = 1'b1;

 #(100000000*T);

// Stop timer and turn off blink leb by resetting



start=1'b0;

 #(50000000*T);

reset=1'b1;

 #(50000000*T);

reset=1'b0;

#(50000000*T);

// Start Timer again

start=1'b1;

#(100000000*T);

// Test timer start =0 when timer is  counting
```

```
start=1'b0;

#(100000000*T);

// Start timer again

start=1'b1;

#(200000000*T);

// Test reset to the initial time function

reset=1'b1;

 #(100000000*T);

reset=1'b0;

// Finish timer

#(50000000*T);

  $finish;   // End simulation

  end

endmodule
```

IV. Simulation result (ModelSIM)

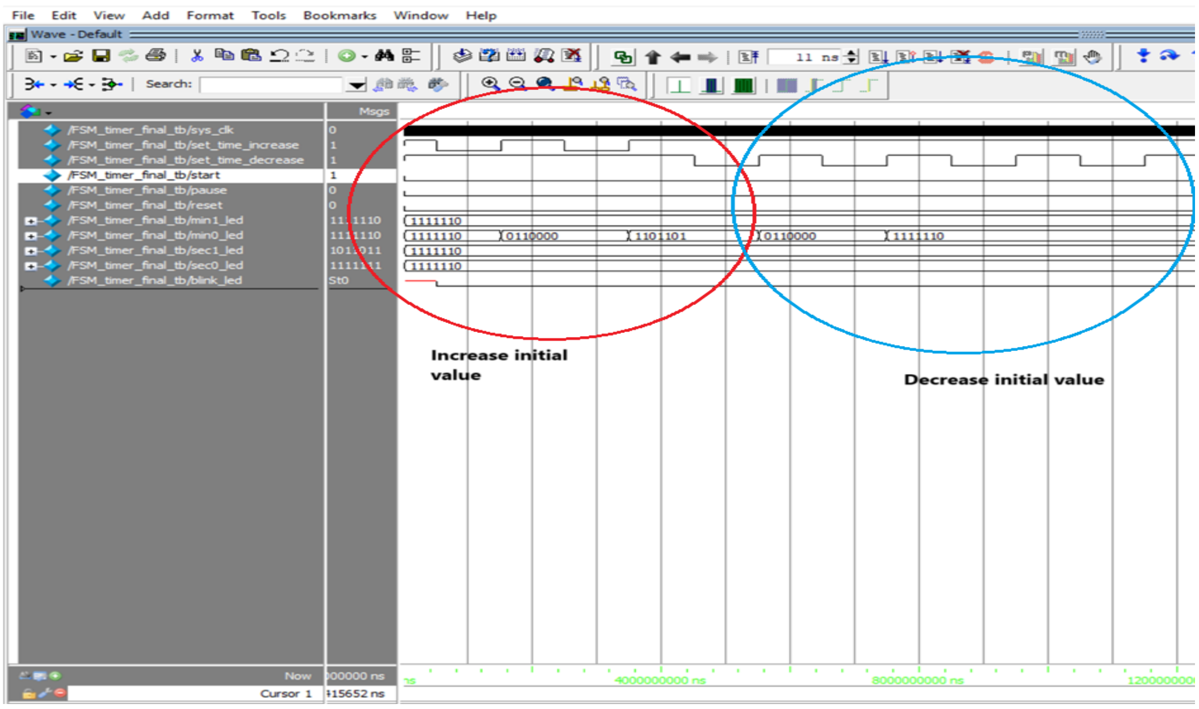


Segments (✓= ON)							Display	Segments (✓= ON)							Display
a	b	c	d	e	f	g		a	b	c	d	e	f	g	
✓	✓	✓	✓	✓	✓		0	✓	✓	✓	✓	✓	✓	✓	8
	✓	✓					1	✓	✓	✓			✓	✓	9
✓	✓		✓	✓		✓	2	✓	✓	✓		✓	✓	✓	A
✓	✓	✓	✓			✓	3			✓	✓	✓	✓	✓	b
	✓	✓			✓	✓	4	✓			✓	✓	✓		C
✓		✓	✓		✓	✓	5		✓	✓	✓	✓		✓	d
✓		✓	✓	✓	✓	✓	6	✓			✓	✓	✓	✓	E
✓	✓	✓					7	✓				✓	✓	✓	F

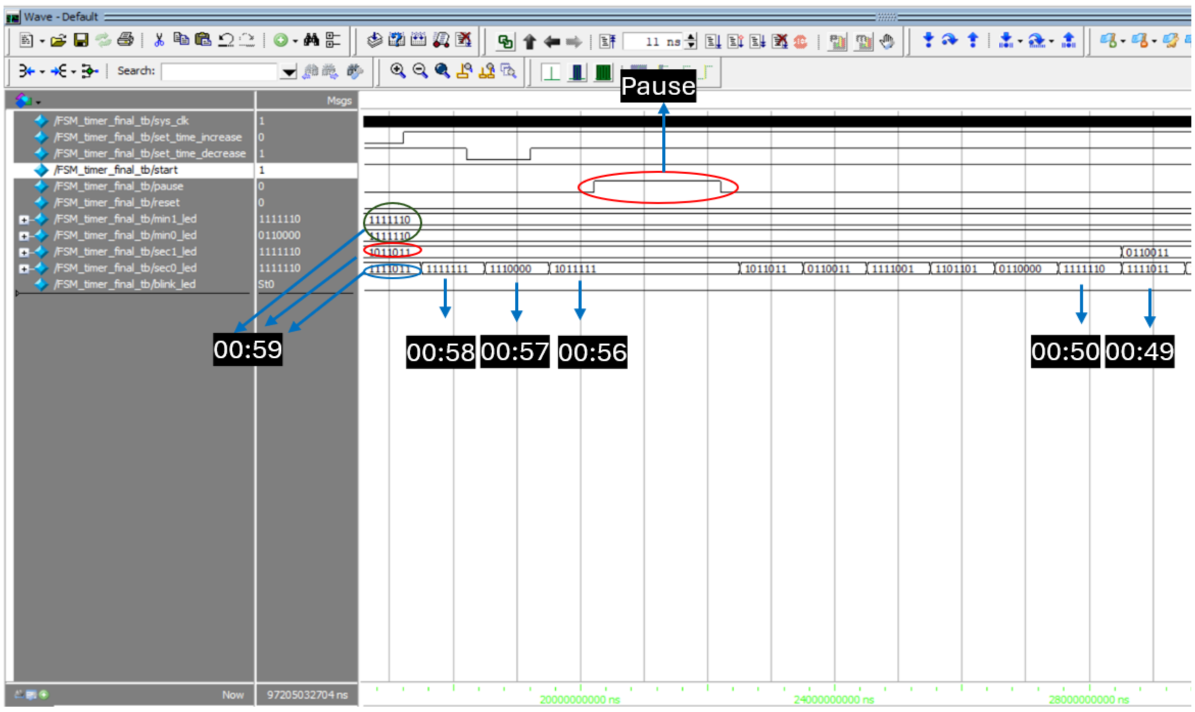
Table 1: The output of 7-segment commoning anode led corresponding with numbers

Number	Decoded number
0	1111110
1	0110000
2	1101101
3	1111001
4	0110011
5	1011011
6	1011111
7	1110000
8	1111111
9	111101

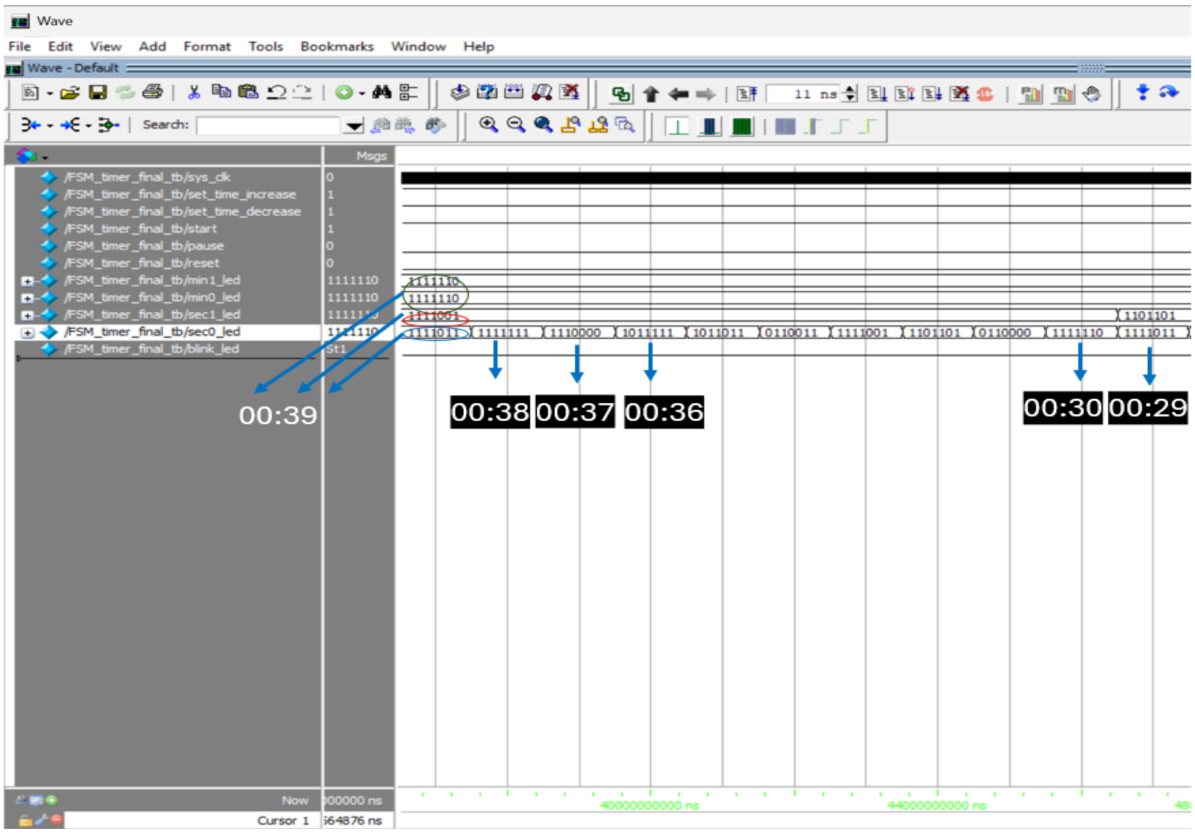
When set_ increase_time is low at the positive edge, the initial time is increased by 1. Reversely, set_decrease_time is low at the positive edge of clock, the initial time is decreased by 1, and if initial time is 0, we can not continuously decrease.


```
/FSM_timer_final_tb/sys_clk                0
/FSM_timer_final_tb/set_time_increase      1
/FSM_timer_final_tb/set_time_decrease      1
/FSM_timer_final_tb/start                  1
/FSM_timer_final_tb/pause                  0
/FSM_timer_final_tb/reset                  0
/FSM_timer_final_tb/min1_led               1111110
/FSM_timer_final_tb/min0_led               1111110
/FSM_timer_final_tb/sec1_led               1011011
/FSM_timer_final_tb/sec0_led               1111111
/FSM_timer_final_tb/blink_led              St0
```

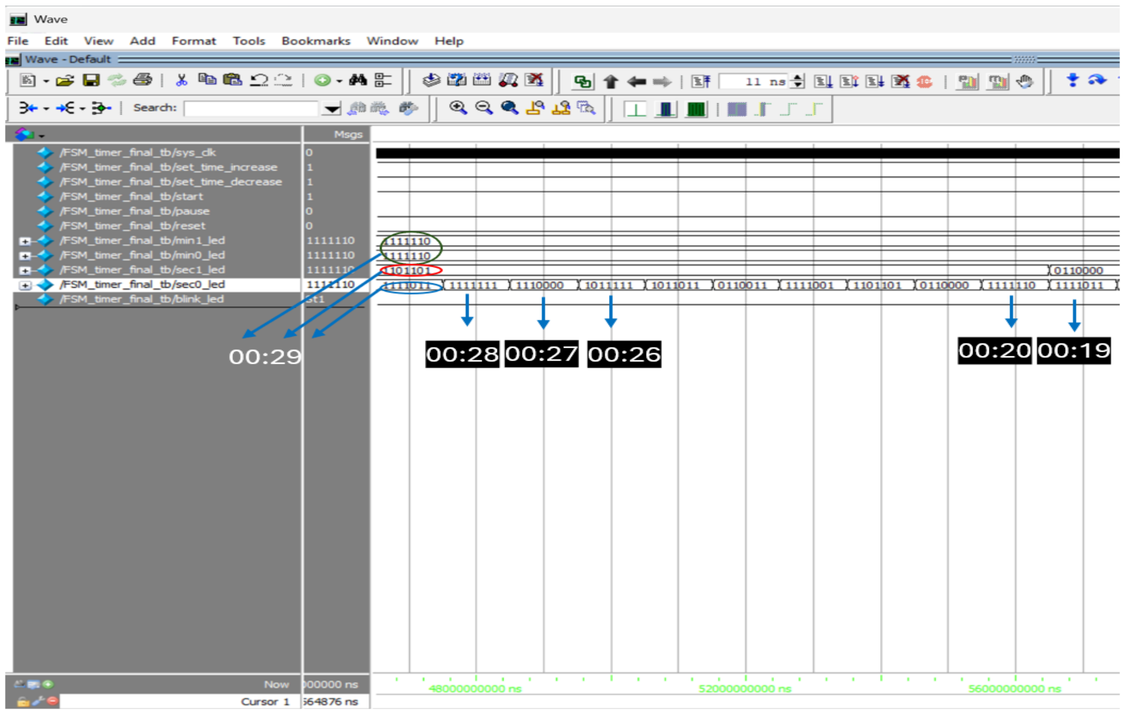
Start to count, from 00:59 to 00:50


Wave - Default
Msgs
/FSM_timer_final_tb/sys_clk
/FSM_timer_final_tb/set_time_increase
/FSM_timer_final_tb/set_time_decrease
/FSM_timer_final_tb/start
/FSM_timer_final_tb/pause
/FSM_timer_final_tb/reset
/FSM_timer_final_tb/min1_led
/FSM_timer_final_tb/min0_led
/FSM_timer_final_tb/sec1_led
/FSM_timer_final_tb/sec0_led
/FSM_timer_final_tb/blink_led
Pause
00:59
00:58
00:57
00:56
00:50
00:49
Now
97205032704 ns
20000000000 ns
24000000000 ns
28000000000 ns

Count from 00:39 to 00:30


Wave
File
Edit
View
Add
Format
Tools
Bookmarks
Window
Help
Wave - Default
Msgs
/FSM_timer_final_tb/sys_clk
/FSM_timer_final_tb/set_time_increase
/FSM_timer_final_tb/set_time_decrease
/FSM_timer_final_tb/start
/FSM_timer_final_tb/pause
/FSM_timer_final_tb/reset
/FSM_timer_final_tb/min1_led
/FSM_timer_final_tb/min0_led
/FSM_timer_final_tb/sec1_led
/FSM_timer_final_tb/sec0_led
/FSM_timer_final_tb/blink_led
00:39
00:38
00:37
00:36
00:30
00:29
Now
Cursor 1
40000000000 ns
44000000000 ns

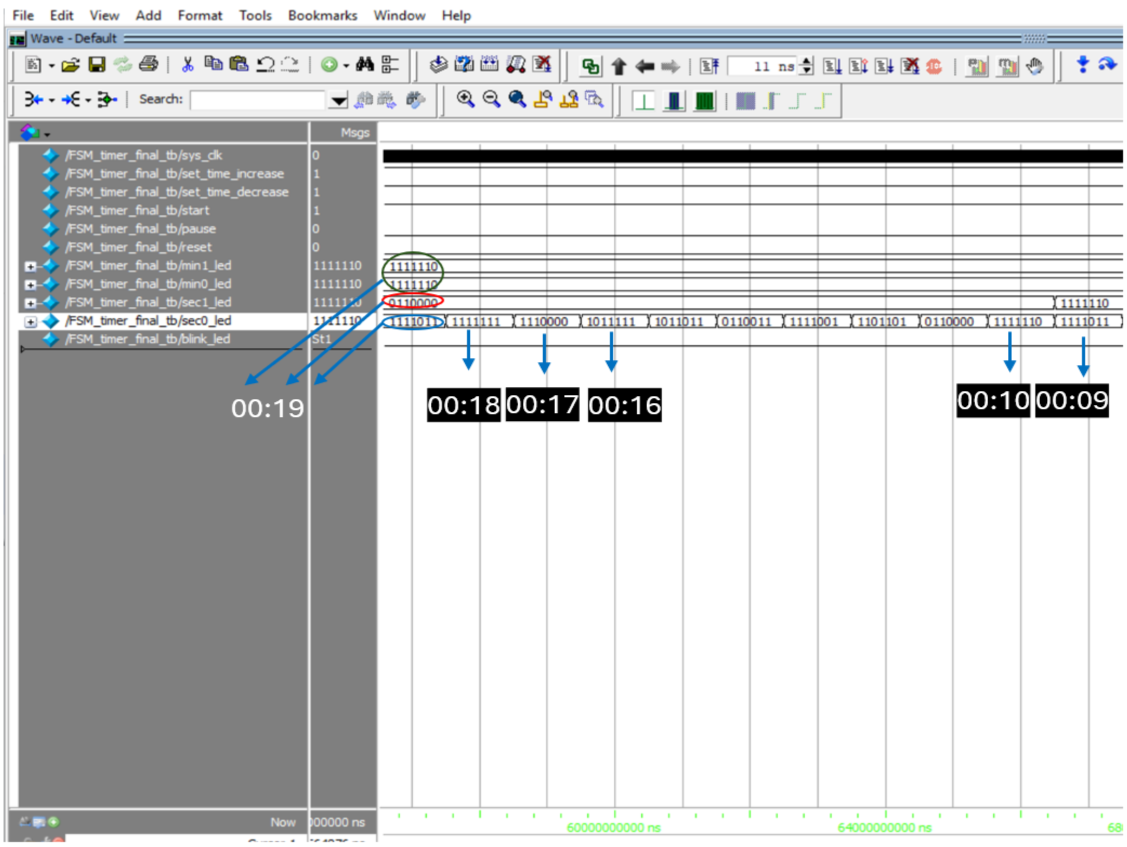
Count from 00:29 to 00:20



	Msgs
/FSM_timer_final_tb/sys_clk	0
/FSM_timer_final_tb/set_time_increase	1
/FSM_timer_final_tb/set_time_decrease	1
/FSM_timer_final_tb/start	1
/FSM_timer_final_tb/pause	0
/FSM_timer_final_tb/reset	0
/FSM_timer_final_tb/min1_led	1111110
/FSM_timer_final_tb/min0_led	1111110
/FSM_timer_final_tb/sec1_led	1111110
/FSM_timer_final_tb/sec0_led	1111110
/FSM_timer_final_tb/blink_led	St1

```
min1_led  1111110
min0_led  1111110
sec1_led  0101101                                                         0110000
sec0_led  1111011 1111111 1110000 1011111 1011011 0110011 1111001 1101101 0110000 1111110 1111011
```

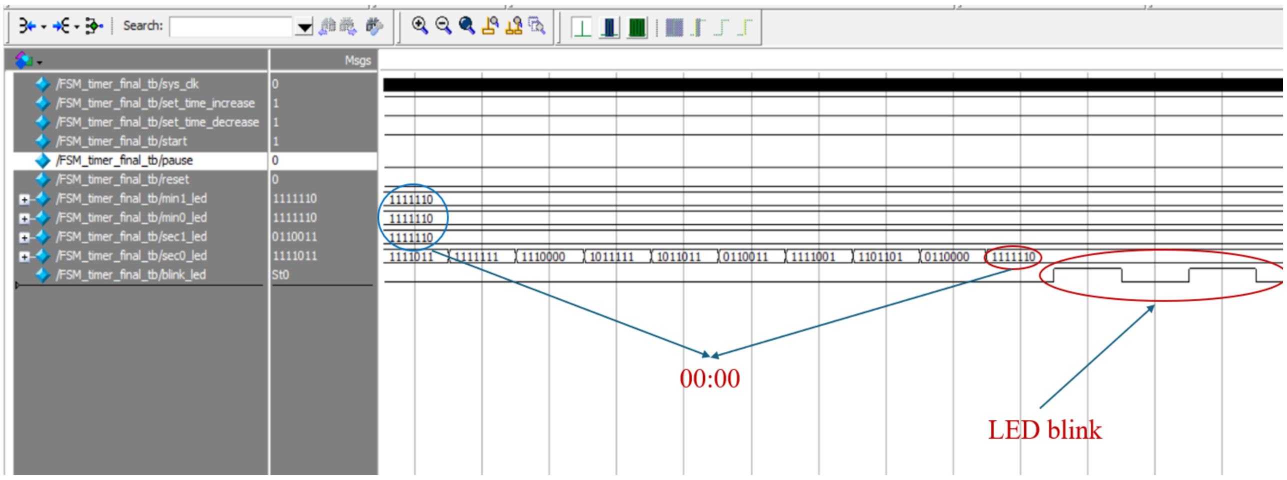
Count from 00:19 to 00:10



	Msgs
/FSM_timer_final_tb/sys_clk	0
/FSM_timer_final_tb/set_time_increase	1
/FSM_timer_final_tb/set_time_decrease	1
/FSM_timer_final_tb/start	1
/FSM_timer_final_tb/pause	0
/FSM_timer_final_tb/reset	0
/FSM_timer_final_tb/min1_led	1111110
/FSM_timer_final_tb/min0_led	1111110
/FSM_timer_final_tb/sec1_led	1111110
/FSM_timer_final_tb/sec0_led	1111110
/FSM_timer_final_tb/blink_led	St1

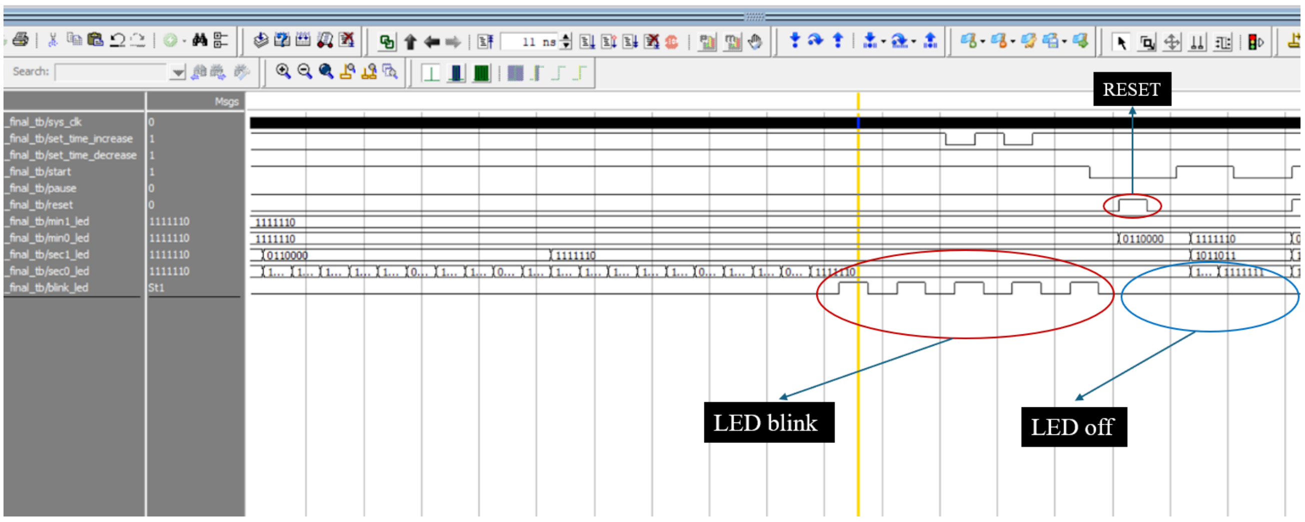
```
min1_led  1111110
min0_led  1111110
sec1_led  0110000                                                         1111110
sec0_led  1111011 1111111 1110000 1011111 1011011 0110011 1111001 1101101 0110000 1111110 1111011
```

Count from 00:09 to 00:00


Search:
Msgs
/FSM_timer_final_tb/sys_clk 0
/FSM_timer_final_tb/set_time_increase 1
/FSM_timer_final_tb/set_time_decrease 1
/FSM_timer_final_tb/start 1
/FSM_timer_final_tb/pause 0
/FSM_timer_final_tb/reset 0
/FSM_timer_final_tb/min1_led 1111110
/FSM_timer_final_tb/min0_led 1111110
/FSM_timer_final_tb/sec1_led 0110011
/FSM_timer_final_tb/sec0_led 1111011
/FSM_timer_final_tb/blink_led St0
1111110
1111110
1111110
1111011 1111111 1110000 1011111 1011011 0110011 1111001 1101101 0110000 1111110
00:00
LED blink

The timer enters the Pause state when the system counts and the user turns off the Start switch.

When the system counts down to 0, the LED blinks. The system is in a Stop state (cannot increase or decrease the value) until the user resets the timer to the initial value.


Search:
Msgs
11 ns
_final_tb/sys_clk 0
_final_tb/set_time_increase 1
_final_tb/set_time_decrease 1
_final_tb/start 1
_final_tb/pause 0
_final_tb/reset 0
_final_tb/min1_led 1111110
_final_tb/min0_led 1111110
_final_tb/sec1_led 1111110
_final_tb/sec0_led 1111110
_final_tb/blink_led St1
RESET
LED blink
LED off

V. Conclusion

1. Project application

Common Applications:

- Household:
  - Timer functions in microwaves, rice cookers, washing machines, and other household appliances.

  - Alarms or time reminders in portable devices.
- Industrial:
  - Timing processes in production lines, measuring intervals between stages.
  - Countdown to automatically turn devices on/off or schedule periodic maintenance.
- Transportation: Countdown timers in traffic lights to help drivers and pedestrians prepare for changes in signals.
- Education and Sports: Timing during exams or sports activities.
- Special Applications:
  - Security devices such as timed locks...
  - Controlling events like motor activation or controlled industrial explosions.

2. Advantages and disadvantages

Advantages:

- The timer is a combination of sub-modules, with a 10-count module, the maximum value of the counter can change from 0 to 9, so it can be applied to combine into different counting values such as 99, 12, and 24... for use in many other applications.
- .The timer can increase or decrease the initial time by holding down or pressing and releasing to help adjust the time easily, along with the logic setting during operation that does not allow the timer to be started when the initial time has not been set, the initial value cannot be increased or decreased while the timer is running, helping the timer not to be interrupted during operation, and when finished, it can only be reactivated when the timer is started and reset is turned off, otherwise, it will not perform any other operations, always helping to signal the user until they perform the shutdown operation.

Disadvantages:

The timer is not optimal yet. Using multiple states for counting is a waste of resources and can cause conflicts in output values. Using two separate switches for start and pause, although it is possible to use only one which is also fine.

3. Future Work and Development Potential of the Projec

Future work:

- Enhancing the optimization of the timer: Simplify counting states to save resources and minimize output value conflicts.
- Integrating smart sensors: Combine the timer with sensors for temperature, light, or motion to expand applications in smart homes and industries.
- Developing a user interface: Create an intuitive display or a mobile app for users to easily manage time settings.
- Increasing flexibility: Add more flexible timer functions, such as repeat timers or the ability to set multiple time intervals.
- Integrating IoT (Internet of Things): Connect the timer to other IoT devices to synchronize operations within an automation system.

Development Potential of the Project:

- Applications in smart homes: The timer can manage household devices like lights, air conditioners, heaters, fans, etc., helping to save energy and increase convenience.
- Development in the industrial sector: Applications in automated production lines, especially in industries requiring high precision, such as electronics, food, and beverage manufacturing.
- Applications in security: Use in security devices like automatic locks, smart doors, and fire alarm systems.
- Applications in education and sports: Develop devices to monitor time during competitions or sports training, improving accuracy and fairness.
- Integration with green energy: Develop timers powered by solar energy or other renewable energy sources to align with environmental protection trends.

VI. Reference

1. University of Washington. (2009). *Finite State Machines (FSM) [Lecture slides]*. Retrieved from https://courses.cs.washington.edu/courses/cse370/09wi/LectureSlides/18-FSM.pdf.
2. YouTube. (n.d.). *[CpE 100 Module 23: FSM Counters]*. YouTube. Retrieved December 26, 2024, from https://www.youtube.com/watch?v=FmiyDP3O6ag.
3. YouTube. (n.d.). *[[VLSIE001] Bài 11 - Mô tả máy trạng thái (FSM) bằng Verilog]*. YouTube. Retrieved December 26, 2024, from https://www.youtube.com/watch?v=I6TqiRhBank.

VII. Appendix

1. Module: FSM_timer_minsec

Code:

```
module FSM_timer_minsec (
    input wire Clk,
    input wire set_time_increase,
    input wire set_time_decrease,
    input wire start,
    input wire pause,
    input wire reset,
    output wire [7:0] Init_minvalue,
    output wire [7:0] Init_secvalue,
    output wire [7:0] Countmin,
    output wire [7:0] Countsec,
    output wire [1:0] carrymin,
    output wire [1:0] carrysec,
    output wire blink_led
);
    // Internal registers
    reg [3:0] Int_min1value, Int_min0value;
reg [3:0] Int_sec1value, Int_sec0value;
reg blink_signal;
```

```
    reg Set_clk;
    reg stop;
    wire trigger_start;
    wire UpOrDown = 1'b0; // Always count down
    // Initialization
    initial begin
        Int_min1value = 4'b0000;
        Int_min0value = 4'b0000;
        Int_sec1value = 4'b0000;
        Int_sec0value = 4'b0000;
        stop = 1'b0;
        Set_clk = 1'b1;
        blink_signal = 1'b0;
    end
    assign blink_led = blink_signal;
assign trigger_start = (start && (|Int_min1value || |Int_min0value)) ? 1 : 0;
      always @(posedge Clk) begin
if (!start && !stop) begin
        if (set_time_increase==1'b0 ) begin
            if (Int_min0value < 4'b1001) begin
                Int_min0value <= Int_min0value + 1'b1;
            end else if (Int_min1value < 4'b0101) begin
                Int_min0value <= 4'b0000;
                Int_min1value <= Int_min1value + 1'b1;
            end else begin
                Int_min0value <= 4'b0000;
                Int_min1value <= 4'b0000;
                   end
                   Set_clk <= 1'b1;
               end else if (set_time_decrease==1'b0 ) begin
```

```
                if (Int_min0value > 4'b0000) begin
                    Int_min0value <= Int_min0value - 1'b;
                 end else if (Int_min1value > 4'b0000) begin
                    Int_min1value <= Int_min1value - 1'b1;
                    Int_min0value <= 4'b1001;
                end else begin
                    Int_min1value <= 4'b0000;
                    Int_min0value <= 4'b0000;
                end
                Set_clk <= 1'b1;
            end else begin
                Set_clk <= 1'b0;
            end
      end
     end
      // Countdown logic and blink signal
    always @(posedge Clk or posedge reset) begin
        if (reset) begin
            blink_signal <= 1'b0;
            stop <= 1'b0;
        end else if (trigger_start) begin
            if (Countmin==8'b00000000 && Countsec==8'b00000001 ) begin
                // Stop the timer when the countdown reaches zero
                blink_signal <= 1'b1;
                stop <= 1'b1;
            end
        end
    end
        // Second timer module
        timer60 seccount (
            .Clk(Clk),
            .set_time(Set_clk),
        .UpOrDown(UpOrDown),
        .start(trigger_start),
        .stop(stop),
        .reset(reset),
        .pause(pause),
        .Init_unitvalue(Int_sec0value),
        .Init_dozenvalue(Int_sec1value),
        .Init_value60(Init_secvalue),
        .Count60(Countsec),
        .carry60(carrysec)
    );

    // Minute timer module
    timer60 mincount (
        .Clk(carrysec[1]),
        .set_time(Set_clk),
        .UpOrDown(UpOrDown),
        .start(trigger_start),
        .stop(stop),
        .reset(reset),
        .pause(pause),
        .Init_unitvalue(Int_min0value),
        .Init_dozenvalue(Int_min1value),
        .Init_value60(Init_minvalue),
        .Count60(Countmin),
        .carry60(carrymin)
    );

endmodule
```

Testbench:

```
module FSM_timer_minsec_tb;
localparam T = 20; // Clock period
// Inputs
reg clk;
reg set_time_increase;
reg set_time_decrease;
reg start;
reg reset;
reg pause;
// Outputs
wire [7:0] Init_minvalue;
wire [7:0] Init_secvalue;
wire [7:0] Countmin;
wire [7:0] Countsec;
wire [1:0] carrymin;
wire [1:0] carrysec;
wire blink_led;

// Instantiate the Unit Under Test (UUT)
FSM_timer_minsec uut (
.Clk(clk),
.set_time_increase(set_time_increase),
.set_time_decrease(set_time_decrease),
.start(start),
.pause(pause),
.reset(reset),
.Init_minvalue(Init_minvalue),
. Init_secvalue( Init_secvalue),
.Countmin(Countmin),
.Countsec(Countsec),
.carrymin(carrymin),
. carrysec(carrysec),
.blink_led(blink_led)
);

// Clock generation
always begin
clk = 1'b1;
#(T / 2);
clk = 1'b0;
#(T / 2);
end

// Reset logic
initial begin
reset = 1'b0;
pause = 1'b0;
start = 1'b0;
set_time_increase = 1'b1;
set_time_decrease =1'b1;
#(T);
end
initial
begin

#(T/2);

// Increase initial time

set_time_increase = 1'b0;
#(T);
set_time_increase = 1'b1;
#(T);
set_time_increase = 1'b0;
#(T);
```

```
set_time_increase = 1'b1;
#(T);

// decrease initial time

set_time_decrease = 1'b0;
#(T);
set_time_decrease = 1'b1;
#(T);

set_time_decrease = 1'b0;
#(T);
set_time_decrease = 1'b1;
#(T);

// Test decreases initial time when initial time = 0;

set_time_decrease = 1'b0;
#(T);
set_time_decrease = 1'b1;
#(T);

set_time_decrease = 1'b0;
#(T);
set_time_decrease = 1'b1;
#(2*T);

// Increase initial time to 1 minute

set_time_increase = 1'b0;
#(T);
set_time_increase = 1'b1;
#(T);
```

```
//Running counter
start = 1'b1;
#(5*T);

// Test set initial time when the timer is starting


set_time_increase = 1'b0;
#(T);
set_time_increase = 1'b1;
#(T);


set_time_decrease = 1'b0;
#(T);
set_time_decrease = 1'b1;


#(T);
// Test Pause function
pause=1'b1;
#(2*T);
// Continousing counting
pause=1'b0;
#(55*T);
// Test increasing initial time when timer stopped
set_time_increase = 1'b0;
#(T);
set_time_increase = 1'b1;
#(T);
set_time_increase = 1'b0;
#(T);
set_time_increase = 1'b1;
#(2*T);
// Stop timer and turn off blink leb by resetting
```

```
start=1'b0;
#(T);
reset=1'b1;
#(T);
reset=1'b0;
#(T);

// Start Timer again
start=1'b1;
#(2*T);

// Test timer start =0 when timer is counting
start=1'b0;
#(2*T);

// Start timer again
start=1'b1;
#(4*T);

// Test reset to the initial time function
reset=1'b1;
#(2*T);
reset=1'b0;
// Finish timer
#(T);
$finish; // End simulation
end
endmodule
```

2. Module: timer60 :

Code:

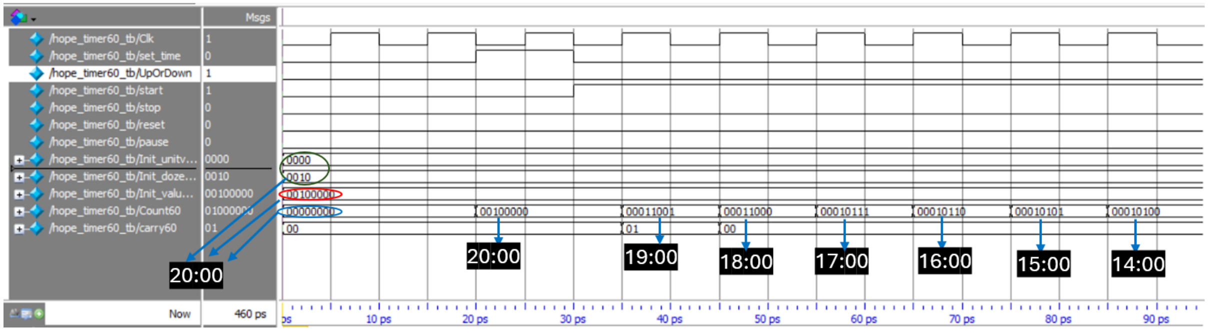
```
    module timer60 (
    input wire Clk,
    input wire set_time,
    input wire UpOrDown,
    input wire start,
    input wire stop,
    input wire reset,
    input wire pause,
    input wire [3:0] Init_unitvalue,
    input wire [3:0] Init_dozenvalue,
    output wire [7:0] Init_value60,
    output wire [7:0] Count60,
    output wire [1:0] carry60
);
    // Wires for lower and upper counters
    wire [3:0] count_lower, count_upper;
    wire carry_lower, carry_upper;
    // Assign combined outputs
    assign Init_value60 = {Init_dozenvalue, Init_unitvalue};
    assign Count60 = {count_upper, count_lower};
    assign carry60 = {carry_upper, carry_lower};
    // Instantiate lower timer (Units)
    timer10 #(.N(4'b1001)) Unit (
        .Clk(Clk),
        .set_time(set_time),

        .start(start),

        .stop(stop),

        .reset(reset),

        .pause(pause),

        .UpOrDown(UpOrDown),

        .Init_value(Init_unitvalue),

        .Count(count_lower),

        .carry(carry_lower)

    );

    // Instantiate upper timer (Dozens) with carry_lower as clock

    timer10 #(.N(4'b0101)) Dozen ( // Dozen counter wraps at 5 (0–59 counting)

        .Clk(carry_lower), // Use carry_lower as clock

        .set_time(set_time),

        .start(start),

        .stop(stop),

        .reset(reset),

        .pause(pause),

        .UpOrDown(UpOrDown),

        .Init_value(Init_dozenvalue),

        .Count(count_upper),

        .carry(carry_upper)

    );

endmodule
```

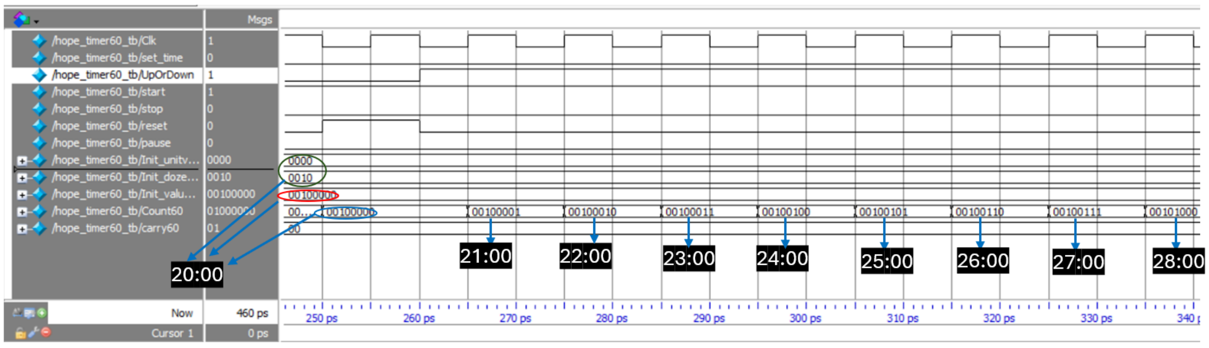
Testbench:

```
module timer60_tb;
// Inputs
   reg Clk;
   reg set_time;
   reg UpOrDown;
   reg start;
   reg stop;
   reg reset;
   reg pause;
   reg [3:0] Init_unitvalue;
   reg [3:0] Init_dozenvalue;
   // Outputs
   wire [7:0] Init_value60;
   wire [7:0] Count60;
   wire [1:0] carry60;
   // Instantiate the Unit Under Test (UUT)
   timer60 uut (
      .Clk(Clk),
      .set_time(set_time),
      .UpOrDown(UpOrDown),
      .start(start),
      .stop(stop),
      .reset(reset),
      .pause(pause),
      .Init_unitvalue(Init_unitvalue),
        .Init_dozenvalue(Init_dozenvalue),
        .Init_value60(Init_value60),
        .Count60(Count60),
        .carry60(carry60)
);
    // Clock generation
    initial begin
        Clk = 0;
        forever #5 Clk = ~Clk; // 10ns clock period
    end
    // Testbench logic
    initial begin
        // Initialize inputs
        set_time = 0;
        UpOrDown = 0; // Start with down-counting
        start = 0;
        stop = 0;
        reset = 0;
        pause = 0;
        Init_unitvalue = 4'b0101;  // Initial value = 5 (units)
        Init_dozenvalue = 4'b0010; // Initial value = 2 (dozens)
        // Wait for global reset
        #20;
        // Set initial time
        set_time = 1;
        #10;
        set_time = 0;
    // Start counting down
    start = 1;
    #100;
    // Pause the timer
    pause = 1;
    #20;
    pause = 0;
    #50;
    // Stop the timer
    stop = 1;
    #20;
    stop = 0;
    #30;
    // Reset the timer
    reset = 1;
    #10;
    reset = 0;
    // Change to up-counting
    UpOrDown = 1;
    start = 1;
    #200;
    // Finish simulation
    $stop;
end
// Monitor values
initial begin
```

```
        $monitor("Time=%0t | Init_value60=%b | Count60=%b | carry60=%b | start=%b |
pause=%b | stop=%b | reset=%b",

                 $time, Init_value60, Count60, carry60, start, pause, stop, reset);

    end

endmodule
```

Results:





3. timer10

Code:

```
  module timer10

  #(parameter N = 4'b1001) // Max count value

  (

      input wire set_time,

      input wire Clk,

      input wire start,
    input wire stop,
    input wire reset,
input wire pause,
input wire UpOrDown,  // High for UP counter, Low for DOWN counter
input wire [3:0] Init_value,
output reg [3:0] Count,
output reg carry
        );
        // State Definitions
        localparam [3:0]
           Count0 = 4'b0000,
           Count1 = 4'b0001,
           Count2 = 4'b0010,
           Count3 = 4'b0011,
           Count4 = 4'b0100,
           Count5 = 4'b0101,
           Count6 = 4'b0110,
           Count7 = 4'b0111,
           Count8 = 4'b1000,
           Count9 = 4'b1001,
           IDE    = 4'b1010,
           Stop_state = 4'b1100;
        reg [3:0] state_reg, state_next;
        // Synchronous State Update
        always @(posedge Clk or posedge reset or posedge set_time) begin
           if (reset || set_time)
               state_reg <= IDE;
        else
            state_reg <= state_next;
    end
    // Combinational Next-State Logic
    always @* begin
        state_next = state_reg;
        case (state_reg)
            IDE: begin
                if (start) begin
                    if (UpOrDown)
                        state_next = (Init_value == N) ? Count0 : Init_value + 1'b1;
                    else
                        state_next = (Init_value == 4'b0000) ? N : Init_value - 1'b1;
                end
            end
            Count0: begin
                if (stop)
                    state_next = Stop_state;
                else if (!start || pause)
                    state_next = Count0;
                else
                    state_next = UpOrDown ? Count1 : N;
            end
            Count1: begin
                if (stop)
                    state_next = Stop_state;
                else if (!start || pause)
        state_next = Count1;
    else
        state_next = UpOrDown ? ((Count == N) ? Count0 : Count2) : Count0;
end
Count2: begin
    if (stop)
        state_next = Stop_state;
    else if (!start || pause)
        state_next = Count2;
    else
        state_next = UpOrDown ? ((Count == N) ? Count0 : Count3) : Count1;
end
Count3: begin
    if (stop)
        state_next = Stop_state;
    else if (!start || pause)
        state_next = Count3;
    else
        state_next = UpOrDown ? ((Count == N) ? Count0 : Count4) : Count2;
end
Count4: begin
    if (stop)
        state_next = Stop_state;
    else if (!start || pause)
        state_next = Count4;
    else
        state_next = UpOrDown ? ((Count == N) ? Count0 : Count5) : Count3;
```

```
end
Count5: begin
   if (stop)
      state_next = Stop_state;
   else if (!start || pause)
      state_next = Count5;
   else
      state_next = UpOrDown ? ((Count == N) ? Count0 : Count6) : Count4;
end
Count6: begin
   if (stop)
      state_next = Stop_state;
   else if (!start || pause)
      state_next = Count6;
   else
      state_next = UpOrDown ? ((Count == N) ? Count0 : Count7) : Count5;
end
Count7: begin
   if (stop)
      state_next = Stop_state;
   else if (!start || pause)
      state_next = Count7;
   else
      state_next = UpOrDown ? ((Count == N) ? Count0 : Count8) : Count6;
end
Count8: begin
   if (stop)
```

```
                    state_next = Stop_state;
                else if (!start || pause)
                    state_next = Count8;
                else
                    state_next = UpOrDown ? ((Count == N) ? Count0 : Count9) : Count7;
            end
            Count9: begin
                if (stop)
                    state_next = Stop_state;
                else if (!start || pause)
                    state_next = Count9;
                else
                    state_next = UpOrDown ? Count0 : Count8;
            end
            Stop_state: ; // No operation
            default: state_next = IDE;
        endcase
    end

    // Output Logic
    always @* begin
        carry = 1'b0;
        case (state_reg)
            IDE: Count = Init_value;
            Count0: begin
                Count = 4'b0000;
                carry = UpOrDown;
```

```
          end
      Count1: begin
            Count = 4'b0001;
            carry = (!UpOrDown && (Count == N));
          end
          Count2: begin
            Count = 4'b0010;
            carry = (!UpOrDown && (Count == N));
          end
          Count3: begin
            Count = 4'b0011;
            carry = (!UpOrDown && (Count == N));
          end
          Count4: begin
            Count = 4'b0100;
            carry = (!UpOrDown && (Count == N));
          end
          Count5: begin
            Count = 4'b0101;
            carry = (!UpOrDown && (Count == N));
          end
          Count6: begin
            Count = 4'b0110;
            carry = (!UpOrDown && (Count == N));
          end
          Count7: begin
            Count = 4'b0111;
            carry = (!UpOrDown && (Count == N));
        end
        Count8: begin
            Count = 4'b1000;
            carry = (!UpOrDown && (Count == N));
        end
        Count9: begin
            Count = 4'b1001;
            carry = (!UpOrDown);
        end
        Stop_state: begin
            Count = 4'b0000;
            carry = 1'b0;
        end
        default: begin
            Count = 4'b0000;
            carry = 1'b0;
        end
    endcase
end
endmodule
```

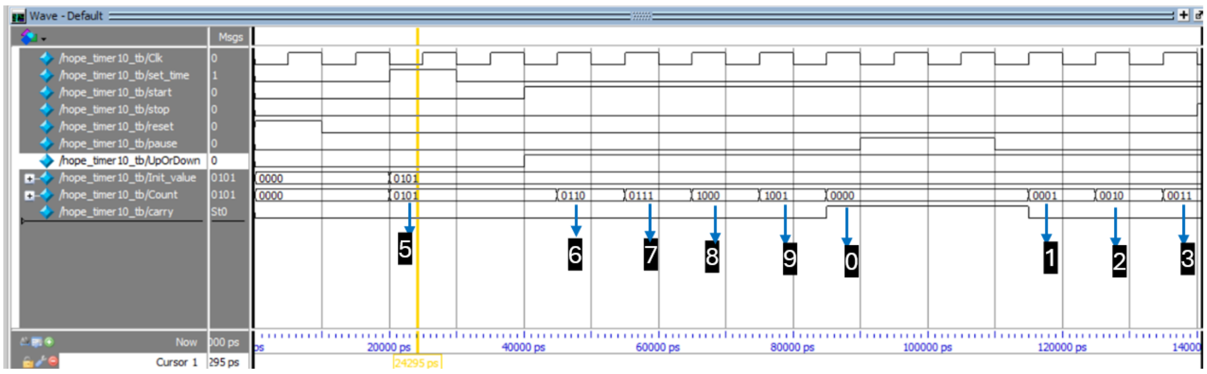
Testbench:

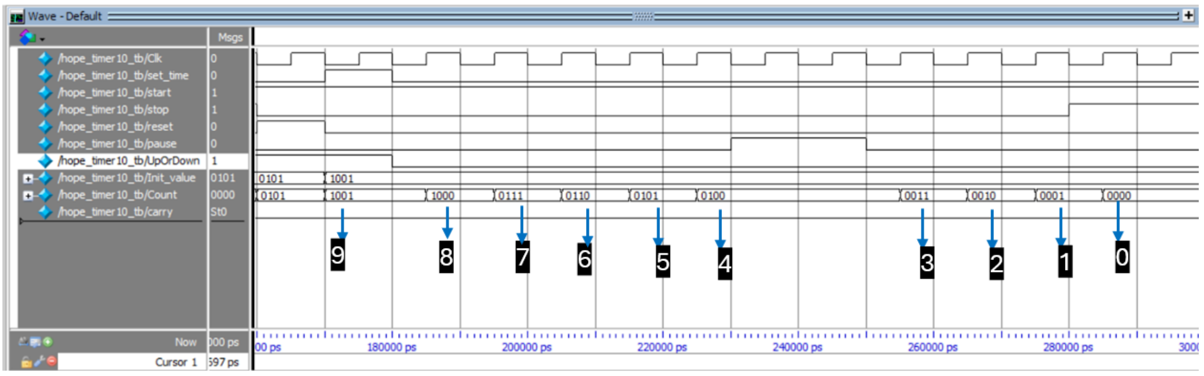
```
`timescale 1ns / 1ps

module timer10_tb;

   // Inputs

   reg set_time;

   reg Clk;

   reg start;

   reg stop;

   reg reset;

   reg pause;

   reg UpOrDown;

   reg [3:0] Init_value;

   // Outputs

   wire [3:0] Count;

   wire carry;

          // Instantiate the Unit Under Test (UUT)

          timer10 uut (

             .set_time(set_time),

             .Clk(Clk),

             .start(start),

             .stop(stop),

             .reset(reset),

             .pause(pause),

             .UpOrDown(UpOrDown),

             .Init_value(Init_value),

             .Count(Count),
    .carry(carry)
);
// Clock generation
always #5 Clk = ~Clk;
// Test sequence
initial begin
    // Initialize Inputs
    set_time = 0;
    Clk = 0;
    start = 0;
    stop = 0;
    reset = 0;
    pause = 0;
    UpOrDown = 0;
    Init_value = 4'b0000;
    // Reset the timer
    reset = 1;
    #10;
    reset = 0;
    #10;
    // Set initial value
    set_time = 1;
    Init_value = 4'b0101;
    #10;
    set_time = 0;
    #10;
    // Start counting up
```

```
start = 1;
UpOrDown = 1;
#50;
// Pause
pause = 1;
#20;
pause = 0;
#30;
// Stop counting
stop = 1;
#20;
stop = 0;
// Reset again
reset = 1;
#10;
reset = 0;
// Set initial value for countdown
set_time = 1;
Init_value = 4'b1001;
#10;
set_time = 0;
// Start counting down
start = 1;
UpOrDown = 0;
#50;
// Pause
pause = 1;
```

```
            #20;

            pause = 0;

            #30;

            // Stop counting

            stop = 1;

            #20;

            stop = 0;

            // End simulation

            $stop;

        end

    endmodule
```

Results:





4. Module frequency_1hz

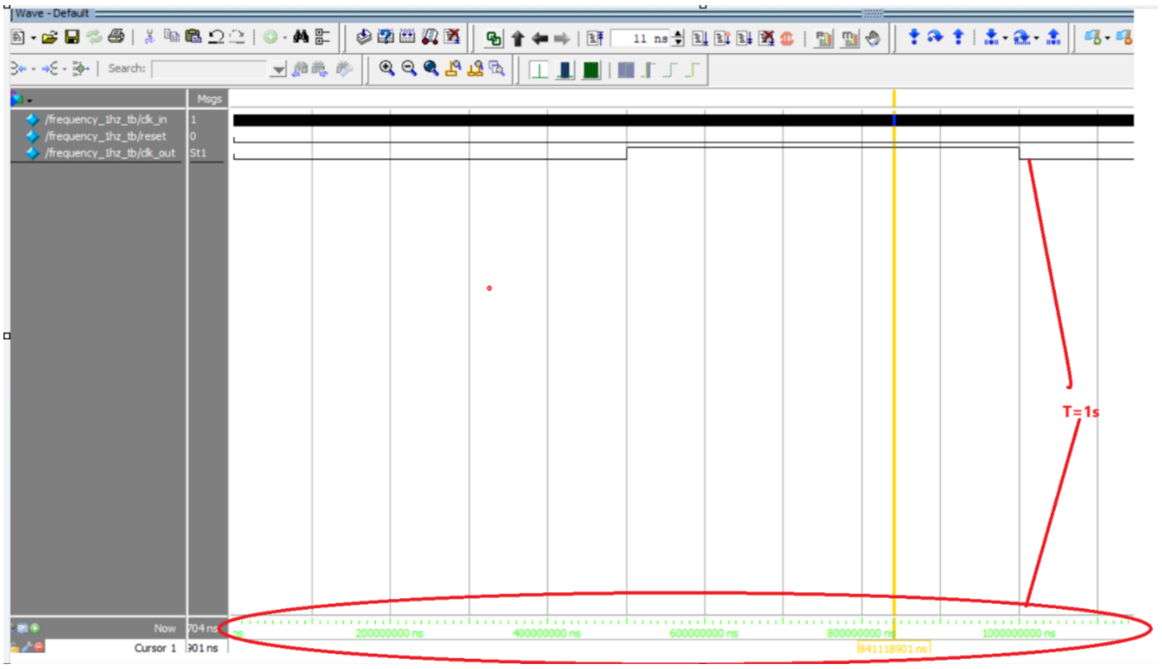
Code:

```
module frequency_1hz (
   input wire clk_in,       // Input clock (50 MHz)
   input wire reset,        // Reset signal
   output reg clk_out       // Output clock (0.5s HIGH, 0.5s LOW) );
   reg [25:0] counter; // 26-bit counter
initial
begin
clk_out =1'b0;
counter =26'd0;
end
   always @(posedge clk_in or posedge reset) begin
      if (reset) begin
         counter <= 26'd0;  // Reset the counter
         clk_out <= 1'b0;   // Reset the output clock
      end else begin
               if (counter == 26'd24999999) begin
                  counter <= 26'd0;         // Reset the counter when it reaches 25 million
                  clk_out <= ~clk_out;      // Toggle the output clock
               end else begin
                  counter <= counter + 1'b1;   // Increment the counter
               end
            end
         end
      endmodule
```

Testbench:

```
`timescale 1ns/1ns  // Define time unit and precision

module frequency_1hz_tb ;

   // Testbench signals

   reg clk_in;        // Input clock (simulating 50 MHz)

   reg reset;         // Reset signal

   wire clk_out;      // Output clock

   // Clock period for 50 MHz = 20 ns (1 / 50 MHz)

   localparam T= 20;

   // Instantiate the DUT (Device Under Test)

   frequency_1hz uut (

       .clk_in(clk_in),

       .reset(reset),

       .clk_out(clk_out)

);

             // Generate a 50 MHz clock

          // Clock generation

            always begin

              clk_in = 1'b1;

              #(T / 2);

              clk_in = 1'b0;

              #(T / 2);

            end

           // Reset logic

            initial begin

              reset = 1'b0;
            end

            // Stimulus block with structured 'd' changes over time

            initial

           begin

             #(300000000*T);

            $finish;   // End simulation

            end

          endmodule
```

Results:



```
/frequency_1hz_tb/clk_in    1
/frequency_1hz_tb/reset     0
/frequency_1hz_tb/clk_out   St1
Now      704 ns
Cursor 1 901 ns
```

5. Module led_7_segment

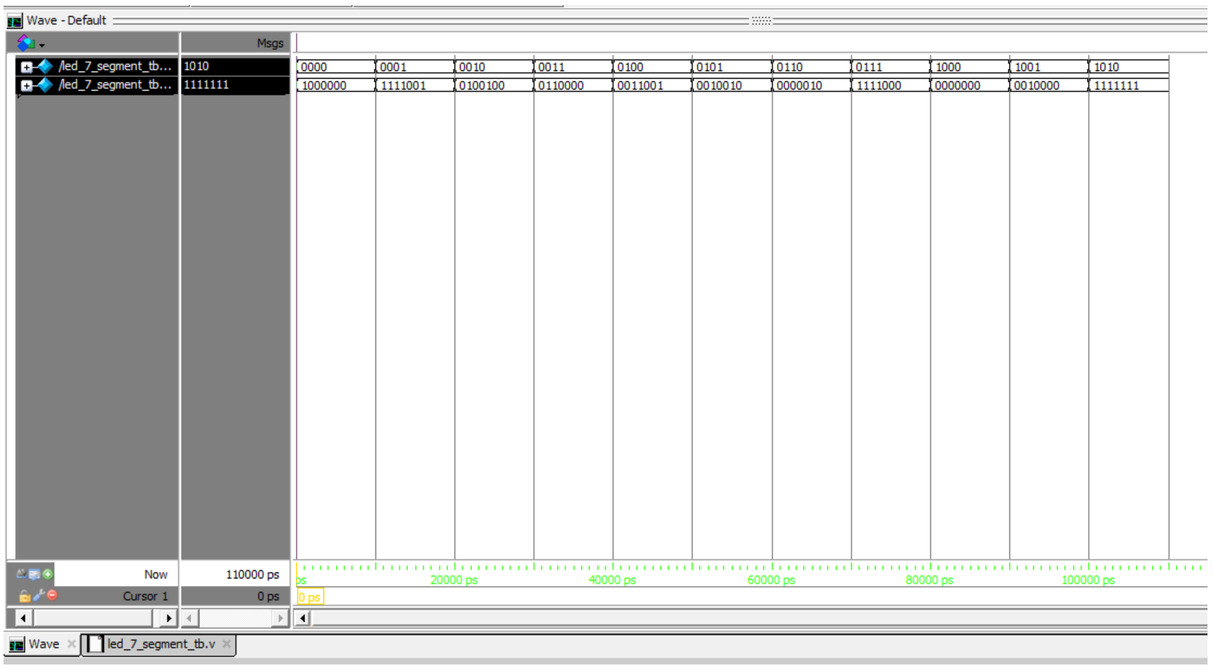
Code:

```
module led_7_segment(
    input [3:0] bcd,    // Input 4-bit BCD
    output reg [6:0] led // Output 7-bit LED segments
  );
    always @(*) begin
        case (bcd) // common anode
            4'b0000: led = 7'b1111110; // 0
            4'b0001: led = 7'b0110000; // 1
            4'b0010: led = 7'b1101101; // 2
            4'b0011: led = 7'b1111001; // 3
             4'b0100: led = 7'b0110011; // 4
             4'b0101: led = 7'b1011011; // 5
             4'b0110: led = 7'b1011111; // 6
             4'b0111: led = 7'b1110000; // 7
             4'b1000: led = 7'b1111111; // 8
             4'b1001: led = 7'b1111011; // 9
             default: led = 7'b0000000; // Tắt LED nếu không hợp lệ
              endcase
             end
        endmodule
```

Testbench:

```
led_7_segment_tb.v
`timescale 1ns / 1ps


module led_7_segment_tb;


    reg [3:0] bcd;
    wire [6:0] led;


    led_7_segment dut (
        .bcd(bcd),
        .led(led)
    );
    initial begin
            bcd = 4'b0000;
            #10 bcd = 4'b0001;
            #10 bcd = 4'b0010;
            #10 bcd = 4'b0011;
            #10 bcd = 4'b0100;
            #10 bcd = 4'b0101;
            #10 bcd = 4'b0110;
            #10 bcd = 4'b0111;
            #10 bcd = 4'b1000;
            #10 bcd = 4'b1001;
            #10 bcd = 4'b1010; // Test default case
            #10 $finish;
        end


        always @(*) begin
            $display("Time: %0t, bcd: %b, led: %b", $time, bcd, led);
        end
    endmodule
```

Results:



```
Wave - Default
                         Msgs
/led_7_segment_tb...  1010     0000    0001    0010    0011    0100    0101    0110    0111    1000    1001    1010
/led_7_segment_tb...  1111111  1000000 1111001 0100100 0110000 0011001 0010010 0000010 1111000 0000000 0010000 1111111

Now       110000 ps   ps    20000 ps    40000 ps    60000 ps    80000 ps    100000 ps
Cursor 1  0 ps        0 ps
```

6. Module led_blinker

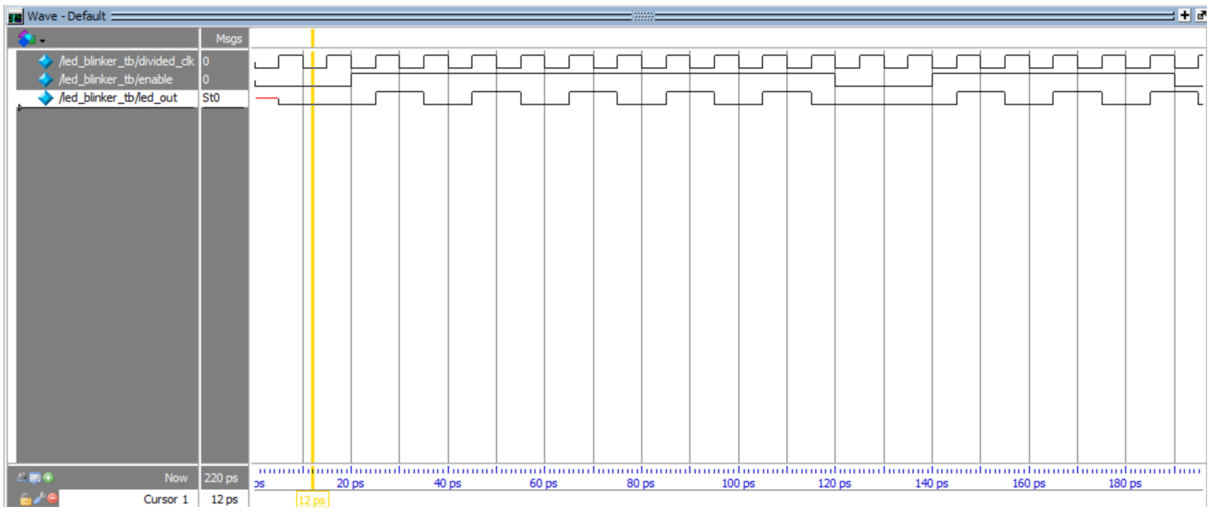
Code:

```
module led_blinker (
    input wire divided_clk,  // Input clock (already divided, e.g., 1 Hz)
    input wire enable,       // Enable signal (1 = blink, 0 = off)
    output reg led_out       // Output LED signal
);
    always @(posedge divided_clk ) begin
        if (enable) begin
            led_out <= ~led_out;  // Toggle LED when enabled
        end else begin
            led_out <= 1'b0;  // Turn off LED when not enabled
        end
    end
endmodule
```

Testbench:

```
// Testbench for led_blinker module
module led_blinker_tb;
    // Testbench signals
    reg divided_clk;
         reg enable;
     wire led_out;
         // Instantiate the led_blinker module
         led_blinker uut (
             .divided_clk(divided_clk),

             .enable(enable),

             .led_out(led_out)

         );

         // Clock generation (simulating a 1 Hz clock)

         initial begin

             divided_clk = 0;

             forever #5 divided_clk = ~divided_clk; // 10 time units period

         end


         // Testbench process

         initial begin

             // Monitor signals for debugging

             $monitor($time, " divided_clk=%b, enable=%b, led_out=%b", divided_clk,
     enable, led_out);


             // Initialize inputs

             enable = 0;


             // Test Case 1: LED off when enable = 0

             #20; // Wait for 2 clock cycles


             // Test Case 2: Enable blinking

             enable = 1;

             #100; // Wait for several clock cycles to observe LED toggling


             // Test Case 3: Disable blinking
             enable = 0;
             #20; // Wait for 2 clock cycles

             // Test Case 4: Re-enable blinking
             enable = 1;
             #50; // Observe LED toggling again

             // Test Case 5: Reset enable and observe LED off
             enable = 0;
             #30;

             // Finish simulation
             $stop;
         end
endmodule
```

Results:



Wave - Default
Msgs
/led_blinker_tb/divided_clk 0
/led_blinker_tb/enable 0
/led_blinker_tb/led_out St0
Now 220 ps
Cursor 1 12 ps
12 ps
ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps