Controller Design and Integration

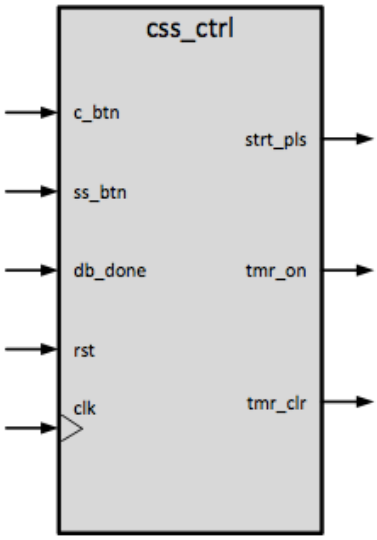
Vincent Martin
TUID: 913012274
ECE 2613
Lab #9 and #10 (11/8/2012)

Introduction:
The objective of this lab is to create three new modules consisting of a clear/start/stop controller, a debouncer controller and finally a stopwatch controller which will include instantiations of both the clear/star/stop and debouncer controller.

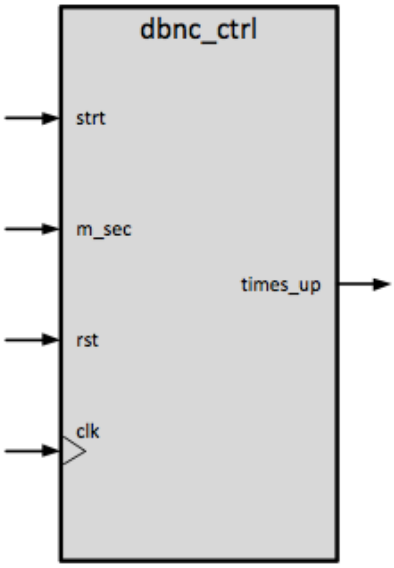
These created modules will be put together with previous modules created in our older labs to create a working stopwatch The previous modules includes in this project will be sw_core, sw_ctrl, ctr_blk, dsp_drvr, svn_seg_decoder, counter_0to9, counter_0to5 and finally the divideby100 module.

This lab will be the ultimate test of our previous work and will give us the best demonstration of code ruse and instantiation that we have seen to date.

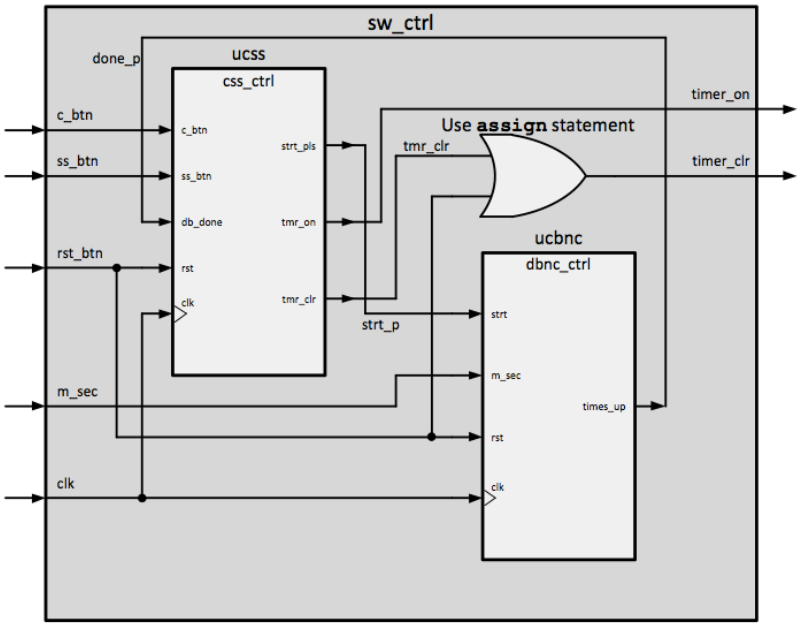
Applying the Theory to Block Diagrams:
To best understand the modules that will be involved in our design it is good to look at block diagrams of our controllers. Includes below are block diagrams for our css_ctrl, dbnc_ctrl and sw_ctrl modules.



clear/start/stop controller css_ctrl.v



debouncer_controller dbnc_ctrl.v



stopwatch controller sw_ctrl.v

Additionally we will want to implement testing modules. This scheme will utilize a number of .txt files containing expected outcomes, which will allow us to test all of our expected results for our design.

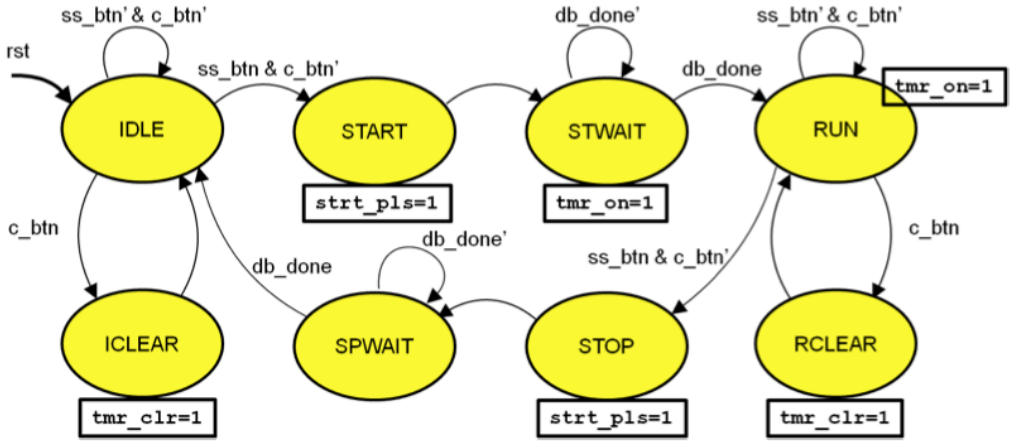
Procedures:

Import the source code from previous labs

1. Connect to the design server using no machine
2. Execute XISE
3. Open the lab9_10 project in ~/Xilinx/lab9_10
4. Add copies of the following files taken from your previous lab files
    a. sw_core.v
    b. dsp_drvr.v
    c. svn_seg_decoder.v
    d. counter_0to9.v
    e. counter_0to5.v

Create the css_ctrl module

1. Create a css_ctrl.v file using the new source wizard and give it the following input/output settings
    a. Input c_btn
    b. Input ss_btn
    c. Input rst
    d. Input clk
    e. Output reg strt_pls
    f. Output reg tmr_on
2. Create the logic that will cause css_ctrl to behave as seen below in the state diagram

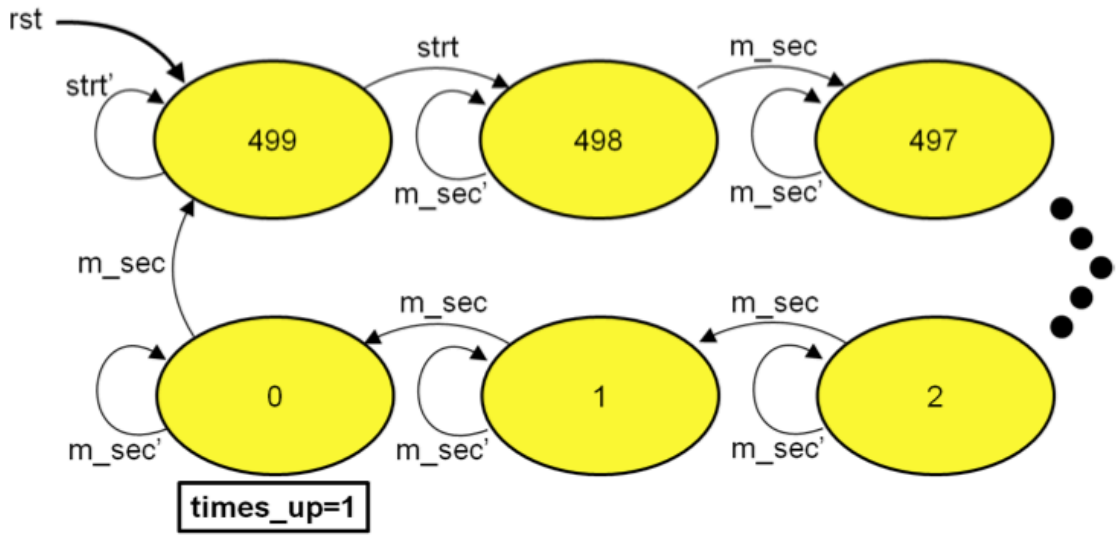


    a.
3. Simulate the created module with tb_css_ctrl and correct any mismatches found.

Create the dbnc_ctrl module

1. Create a dbnc_ctrl.v file using the new source wizard to give it the following input/output settings
    a. Input strt

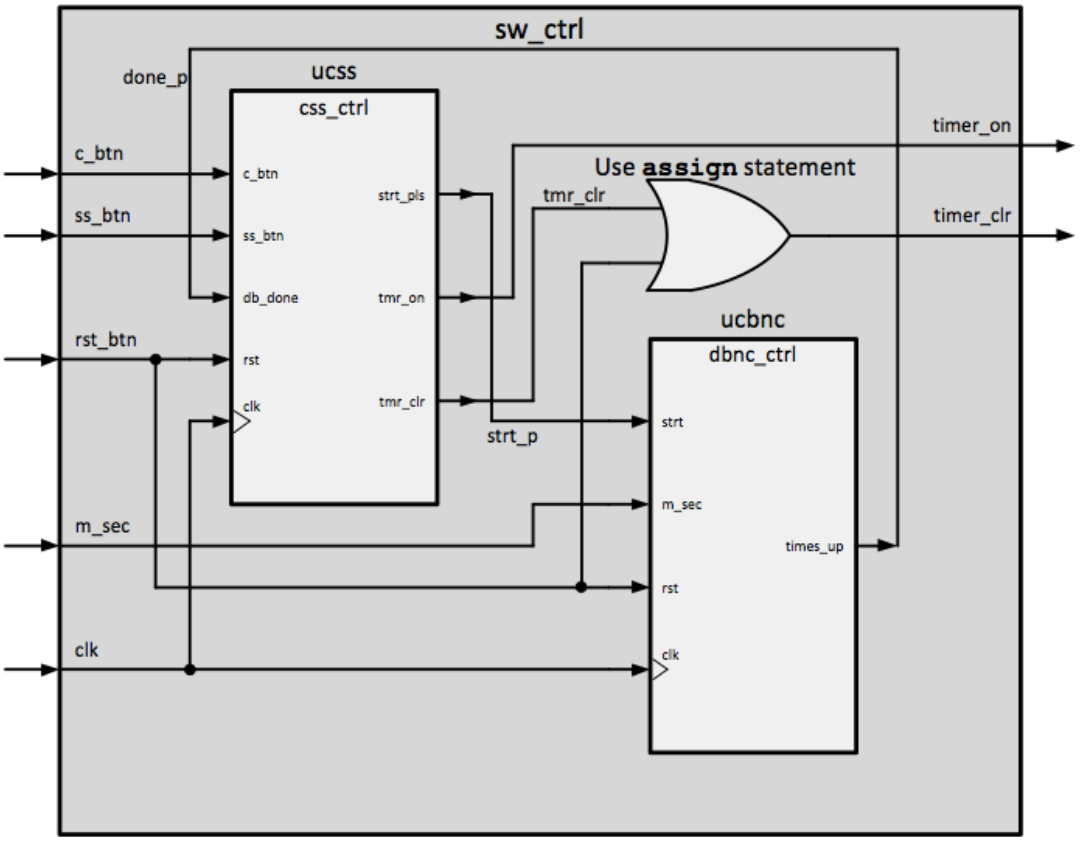
    b. Input m_sec
    c. Input rst
    d. Input clk
    e. Output reg times_up
2. Create the following internal registers
    a. Reg [8:0] millisecond_count
    b. Reg [8:0] next_millisecond_count
3. Create the logic that will cause dbnc_ctrl to behave as seen below in the state diagram.



    a.
4. Simulate the created module with tb_dbnc_ctrl and correct any found mismatches.

<u>Create the sw_ctrl module</u>

1. Create a sw_ctrl.v file using the new source wizard to give it the following input/output settings
    a. Input c_btn
    b. Input ss_btn
    c. Input rst_btn
    d. Input m_sec
    e. Input clk
    f. Output timer_on
    g. Output timer_clr
2. Create the following wires
    a. Done_p
    b. Tmr_clr
    c. Strt_p
3. Instantiate the css_ctrl and dbnc_ctrl modules as seen below

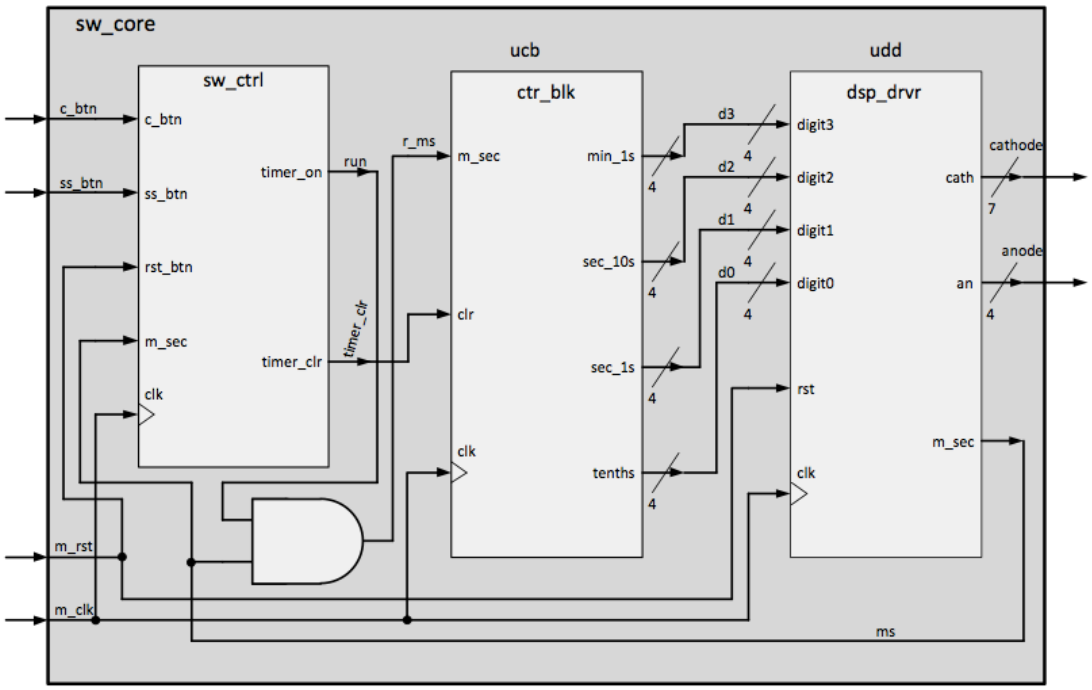


a.

4. Create an assign statement for the timer_clr output as seen above in the diagram.
5. Simulate the created module with tb_sw_ctrl and correct any mismatches found.

Instantiate all of the above into the sw_core module

1. Modify the the imported sw_core.v module so that it matches the image below.



a.
b. Instantiate the stopwatch as usc as seen in the above diagram.
c. Add the following inputs to sw_core.
    i. ss_btn
    ii. c_btn
d. Remove the input run and replace it with a wire run.

2. Simulate the created module with tb_sw_core and correct any problems found.

Compile to a .bit file

1. Compile to a .bit file by navigating to
    a. Implementation
        i. Xc3s500-e4g320
            1. Lab9_10_io_wrapper
                a. Implement design
                b. Generate programming file

Transfer .bit file to board

1. Use your favorite network transfer program to move the .bit file from the development server to your local workstation.
2. Plug the board into the USB port.
3. Launch the Digilent Adept application.
4. Click config tab.
5. Click on browse by the PROM icon.
6. Click Program
7. Reset the board to test.

Results:
Below is the iSim output from the tb_sw_core process. This is our most important outcome. It appears to function as expected. All other tests completed correctly and did not show any mismatches.

Simulator is doing circuit initialization process.
Finished circuit initialization process.
Start button pressed.
digit0 changed to: 1001111 - time: 152000000 ns
digit0 changed to: 0100100 - time: 250000040 ns
digit0 changed to: 0000110 - time: 250002040 ns
digit0 changed to: 0001011 - time: 250004040 ns
digit0 changed to: 0010010 - time: 250006040 ns
digit0 changed to: 0010000 - time: 250008040 ns
digit0 changed to: 1000111 - time: 250010040 ns
digit0 changed to: 0000000 - time: 250012040 ns
digit0 changed to: 0000010 - time: 250014040 ns
digit0 changed to: 1000000 - time: 250016040 ns
digit1 changed to: 1001111 - time: 250016060 ns
digit1 changed to: 0100100 - time: 250036060 ns
digit1 changed to: 0000110 - time: 250056060 ns
digit1 changed to: 0001011 - time: 250076060 ns
digit1 changed to: 0010010 - time: 250096060 ns
digit1 changed to: 0010000 - time: 250116060 ns
digit1 changed to: 1000111 - time: 250136060 ns
digit1 changed to: 0000000 - time: 250156060 ns
digit1 changed to: 0000010 - time: 250176060 ns
digit2 changed - to 1001111 - time = 250196000 ns
digit2 changed - to 0100100 - time = 250396000 ns
digit2 changed - to 0000110 - time = 250596000 ns
digit2 changed - to 0001011 - time = 250796000 ns
digit2 changed - to 0010010 - time = 250996000 ns
digit2 changed - to 1000000 - time = 251196000 ns
digit3 changed - to 1001111 - time = 251196020 ns
digit3 changed - to 0100100 - time = 252396020 ns
digit3 changed - to 0000110 - time = 253596020 ns
digit3 changed - to 0001011 - time = 254796020 ns
digit3 changed - to 0010010 - time = 255996020 ns
digit3 changed - to 0010000 - time = 257196020 ns
digit3 changed - to 1000111 - time = 258396020 ns
digit3 changed - to 0000000 - time = 259596020 ns
digit3 changed - to 0000010 - time = 260796020 ns
Clear button pressed.
Stop button pressed.

digit3 changed - to 1000000 - time = 262000000 ns
Digits should equal zero pattern: 1000000
digit0: 1000000
digit1: 1000000
digit2: 1000000
digit3: 1000000
Simulation complete!!!
Stopped at time : 265995120 ns : File "/home/students/tuc56100/xilinx/lab9_10/tb_sw_core.v" Line 151

Results on the board
Once the .bit file was transferred to the board everything functioned as expected.

Discussion:
This lab was the most difficult on for me. I feel that I understood the concepts perfectly fine, however, there was a lot to keep track of in the form of instantiations and logic.

I made a few mistakes in this lab. The first mistake that I made was quite simple but cost me hours of time. In the sw_ctrl I made the mistake where I set variables in the wrong order.

The final mistake I made was that I forgot to add all of the wires to the sw_core module. This caused my board to not reset.

This lab was a good experience for me because I now have a real board that does something real. This is pretty rewarding.

Source:
Sw_core.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:14:01 10/12/2012
// Design Name:
// Module Name:    sw_core
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sw_core(
    //input run,
    input m_rst,
    input m_clk,
    input ss_btn,
    input c_btn,
    output [6:0] cathode,
    output [3:0] anode,
    output m_sec
    );


//Wires and such
wire [3:0] d0, d1, d2, d3;
wire ms, run, timer_clr;
assign r_ms = run & ms;




//Instantiate the dsp_driver module as udd with kin of crazy format
```

```
//let us see that the spaces do not matter in this language.
dsp_drvr udd(                                   .digit0(d0),
                                                .digit1(d1),
                                                .digit2(d2),
                                                .digit3(d3),
                                                .rst(m_rst),
                                                .clk(m_clk),
                                                .cath(cathode),
                                                .an(anode),
                                                .m_sec(ms));


ctr_blk ucb (                                   .m_sec(r_ms),
                                                .clr(timer_clr),
                                                .clk(m_clk),
                                                .min_1s(d3),
                                                .sec_10s(d2),
                                                .sec_1s(d1),
                                                .tenths(d0));



sw_ctrl usc(                                    .c_btn(c_btn),
                                                .ss_btn(ss_btn),
                                                .rst_btn(m_rst),
                                                .m_sec(ms),
                                                .clk(m_clk),
                                                .timer_on(run),
                                                .timer_clr(timer_clr));


endmodule
```

css_ctrl.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////// // Company:
// Engineer:
//
// Create Date:    14:55:23 11/02/2012
// Design Name:
// Module Name:    css_ctrl
// Project Name:
```

```
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
// //////////////////////////////////////////////////////////////////////////////////
module css_ctrl(
    input c_btn,
    input ss_btn,
    input db_done,
    input rst,
    input clk,
    output reg strt_pls,
    output reg tmr_on,
    output reg tmr_clr
    );


//Paramters for the state machine states
parameter IDLE = 0, START = 1, STWAIT = 2, RUN = 3, RCLEAR = 4, SPWAIT
= 5, STOP = 6, ICLEAR = 7;

//Store state and next_state
reg [2:0] state, next_state;

//Sequential  Logic
always @(posedge clk)
begin
        state <= next_state;
end  //End of sequential logic

//Combinational Logic
always @(state, c_btn, ss_btn, db_done, rst)

begin

        //I did this backwards!
        //state = next_state;
        next_state = state;
```

```
strt_pls = 0;
tmr_on = 0;
tmr_clr = 0;

case (state)

        IDLE:
                begin
                        if((ss_btn == 1) && (c_btn == 0)) next_state = START;
                        //if((ss_btn == 0) && (c_btn == 0)) next_state = IDLE;
                        if(c_btn == 1) next_state = ICLEAR;
                end

        START:
                begin
                        strt_pls = 1;
                        next_state = STWAIT;
                end

        STWAIT:
                begin
                        tmr_on = 1;
                        if(db_done == 1) next_state = RUN;
                        //if(db_done == 0) next_state = STWAIT;
                end

        RUN:
                begin
                        tmr_on = 1;
                        //I shouldnt need the below code because it
                        //holds by default...
                        //if((ss_btn == 0) && (c_btn == 0)) next_state = RUN;
                        if(c_btn == 1) next_state = RCLEAR;
                        if((ss_btn == 1) && (c_btn == 0)) next_state = STOP;
                end

        ICLEAR:
                begin
                        tmr_clr = 1;
                        next_state = IDLE;
                end
```

```
                SPWAIT:
                        begin
                                //if(db_done == 0) next_state = SPWAIT;
                                if(db_done == 1) next_state = IDLE;
                        end

                STOP:
                        begin
                                strt_pls = 1;
                                next_state = SPWAIT;
                        end

                RCLEAR:
                        begin
                                tmr_clr = 1;
                                next_state = RUN;
                        end


        endcase

        //high priority stuff
        if (rst == 1) next_state = IDLE;

end //End of combinational logic for our state machine

endmodule
```

dbnc_ctrl.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:14:09 11/07/2012
// Design Name:
// Module Name:    dbnc_ctrl
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module dbnc_ctrl(
    input strt,
    input m_sec,
    input rst,
    input clk,
    output reg times_up
    );


reg [8:0] millisecond_count, next_millisecond_count;


//Begin of sequential logic
always @(posedge clk)
begin
        millisecond_count <= next_millisecond_count;
end


//Combinational logic
always @(strt, m_sec, rst, millisecond_count)
begin

        //Default settings
        times_up = 0;
        next_millisecond_count = millisecond_count;


        //If we are not at 0 or 499 then let us count down.
        if ((m_sec == 1) && (millisecond_count != 499) && (millisecond_count !=
0))
                begin
                        next_millisecond_count = millisecond_count - 1;
                end



        //If we are not at 0 or 499 and m_sec is set to 0, let us just hold
```

```
	//the conditon.
	if ((m_sec == 0) && (millisecond_count != 0) && (millisecond_count !=
499))
		begin
			next_millisecond_count = millisecond_count;
		end


	//Handle the times up!
	if ((m_sec == 1) && (millisecond_count == 0))
		begin
			times_up = 1;
			next_millisecond_count = 499;
		end

	//Handle the starting of the counter.
	if ((strt == 1) && (millisecond_count == 499))
		begin
			next_millisecond_count = 498;
			times_up = 0;
		end

	//Finally lets handle RST
	if (rst == 1)
		begin
			next_millisecond_count = 499;
			times_up = 0;
		end


end // End of combinational logic block


endmodule
```