**Controller Design and Integration**

Vincent Martin

TUID: 913012274

ECE 2613

Lab #9 and #10 (11/8/2012)

**Introduction:**

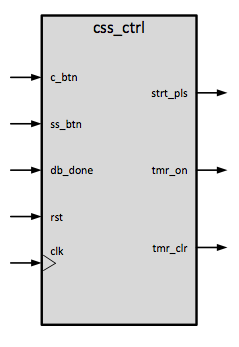
The objective of this lab is to create three new modules consisting of a clear/start/stop controller, a debouncer controller and finally a stopwatch controller which will include instantiations of both the clear/star/stop and debouncer controller.

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.

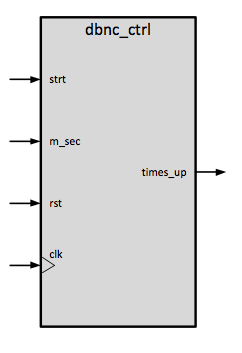
This lab will be the ultimate test of our previous work and will give us the best demonstration of code ruse and instantiation that we have seen to date.

Applying the Theory to Block Diagrams:

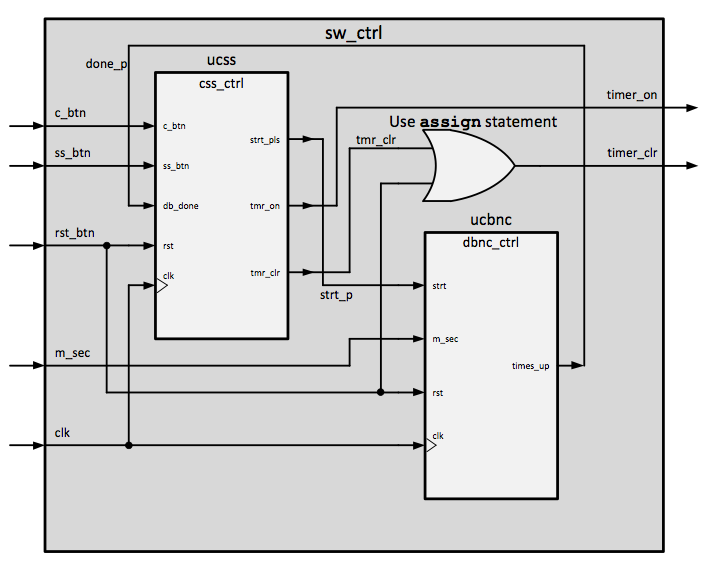
To best understand the modules that will be involved in our design it is good to look at block diagrams of our controllers. Includes below are block diagrams for our css\_ctrl, dbnc\_ctrl and sw\_ctrl modules.



**clear/start/stop controller css\_ctrl.v**



**debouncer\_controller dbnc\_ctrl.v**



**stopwatch controller sw\_ctrl.v**

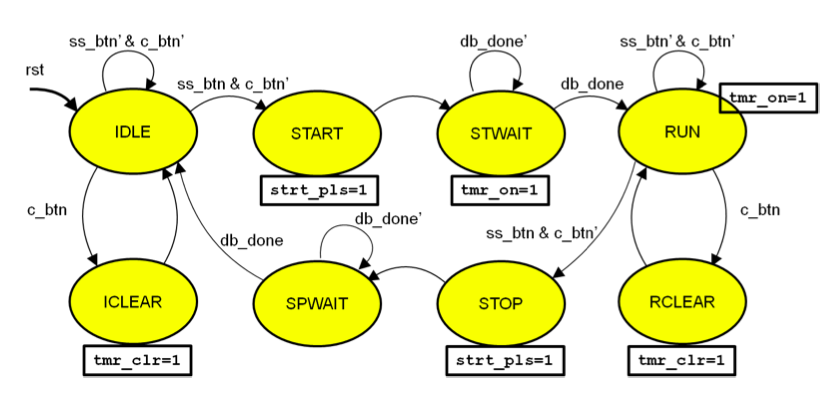
Additionally we will want to implement testing modules. This scheme will utilize a number of .txt files containing expected outcomes, which will allow us to test all of our expected results for our design.

**Procedures:**

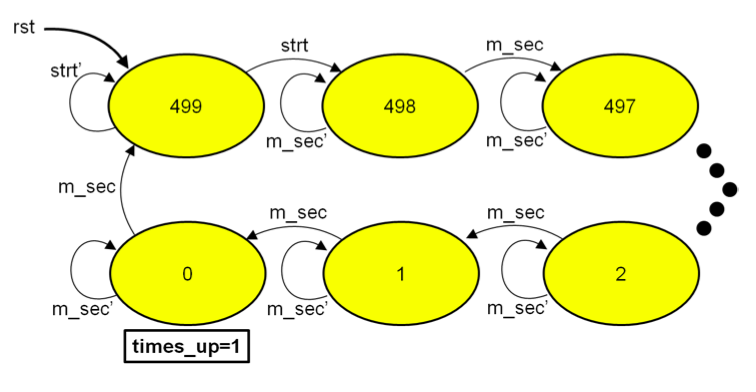
Import the source code from previous labs

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
   1. sw\_core.v
   2. dsp\_drvr.v
   3. svn\_seg\_decoder.v
   4. counter\_0to9.v
   5. counter\_0to5.v

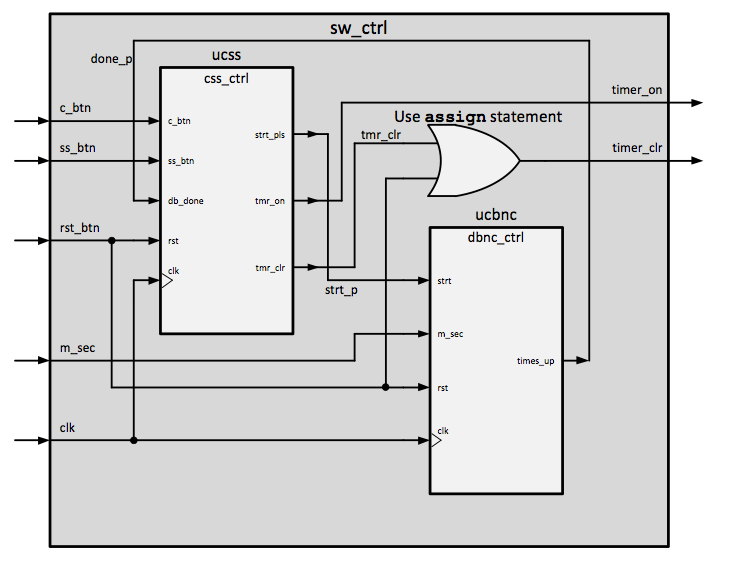
Create the css\_ctrl module

1. Create a css\_ctrl.v file using the new source wizard and give it the following input/output settings
   1. Input c\_btn
   2. Input ss\_btn
   3. Input rst
   4. Input clk
   5. Output reg strt\_pls
   6. Output reg tmr\_on
2. Create the logic that will cause css\_ctrl to behave as seen below in the state diagram
   1. 
3. Simulate the created module with tb\_css\_ctrl and correct any mismatches found.

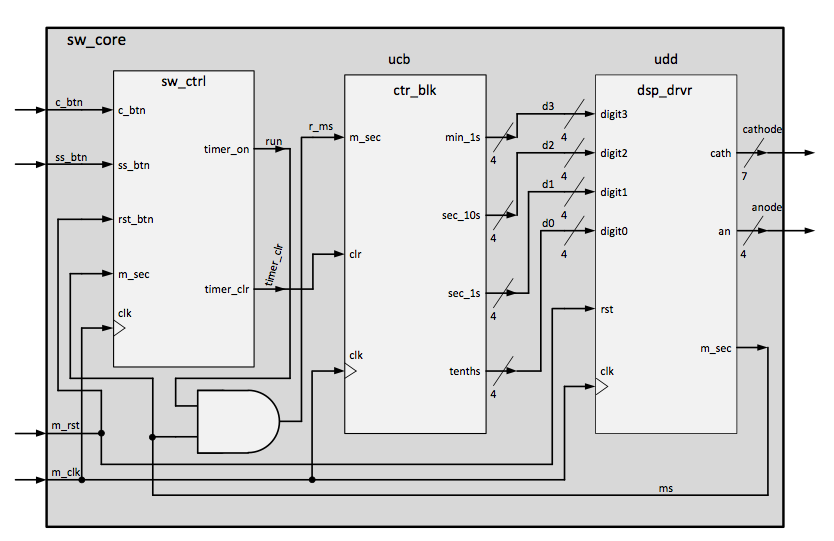
Create the dbnc\_ctrl module

1. Create a dbnc\_ctrl.v file using the new source wizard to give it the following input/output settings
   1. Input strt
   2. Input m\_sec
   3. Input rst
   4. Input clk
   5. Output reg times\_up
2. Create the following internal registers
   1. Reg [8:0] millisecond\_count
   2. Reg [8:0] next\_millisecond\_count
3. Create the logic that will cause dbnc\_ctrl to behave as seen below in the state diagram.
   1. 
4. Simulate the created module with tb\_dbnc\_ctrl and correct any found mismatches.

Create the sw\_ctrl module

1. Create a sw\_ctrl.v file using the new source wizard to give it the following input/output settings
   1. Input c\_btn
   2. Input ss\_btn
   3. Input rst\_btn
   4. Input m\_sec
   5. Input clk
   6. Output timer\_on
   7. Output timer\_clr
2. Create the following wires
   1. Done\_p
   2. Tmr\_clr
   3. Strt\_p
3. Instantiate the css\_ctrl and dbnc\_ctrl modules as seen below
   1. 
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.
   1. 
   2. Instantiate the stopwatch as usc as seen in the above diagram.
   3. Add the following inputs to sw\_core.
      1. ss\_btn
      2. c\_btn
   4. 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
   1. Implementation
      1. Xc3s500-e4g320
         1. Lab9\_10\_io\_wrapper
            1. Implement design
            2. 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