Counter Integration

Vincent Martin
TUID: 913012274
ECE 2613
Lab #: 8 (10/22/2012)

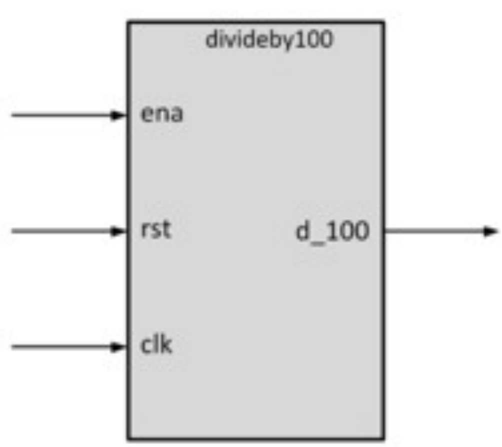
Introduction:
The objective of this lab is to use our counters from lab 6 along with our display driver from lab 7 in order to create a counter block module that in this particular implementation, will be referred to as ctr_blk. Combining all of this functionality together will have the end result of being able to design a very rudimentary stopwatch/timer with limited functionality on to our Xilinx board.

This design will be accomplished by copying the code files from our previous labs and also instantiating new modules that we create from scratch, in particular a clock divider divide by 100 module that will set an output bit high for every count of 100.

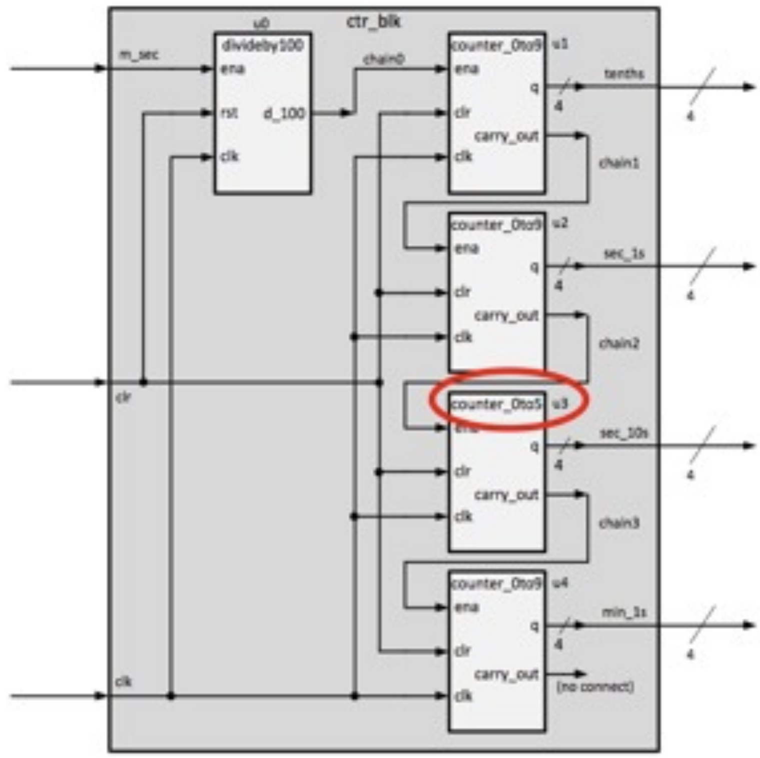
Instantiation Applied to Our Design
The top level of our design will consist of the sw_core which inside will include an instantiation of the dsp_drvr which we will label as udd. This module was designed in our previous lab. For more understanding of this please refer to past labs.

Additionally, there will be another instance called ucb which will be of the ctr_blk. Inside of the ucb instance we will find yet more instantiations, this time called u0, u1, u2, u3 and finally u4. These modules will all be of the counter_0to9 type execpt for u3, which will be a counter_0to5 variety and u0 which will be an instantiation of divideby100.

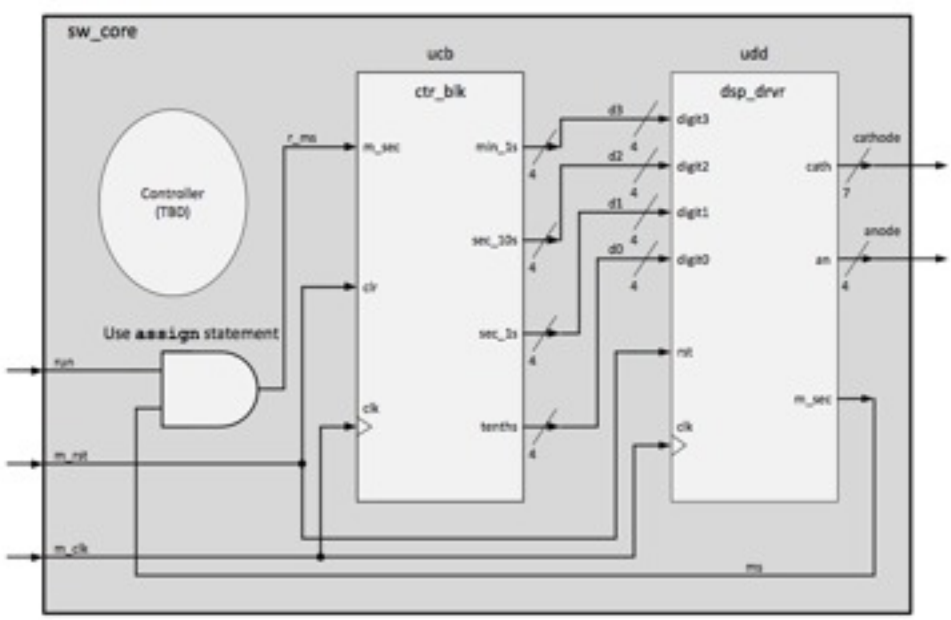
`Applying the Theory to Block Diagrams:
To best understand the modules that will be involved in our design it is good to look at block diagrams. Included below are block diagrams for our divideby100 module, ctr_block and sw_core.



divideby100 block diagram



ctr_blk block diagram



sw_core block diagram

Additionally we will want to implement a testing module. This scheme will utilize a .txt file, which will allow us to test all of our expected outcomes 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 lab8 project in ~/xilinx/lab8
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. 2612_lab7.ucf
   e. counter_0to9.v
   f. counter_0to5.v

Create the divideby100 module

1. Create a divideby100.v file using the new source wizard to give it the following input/output settings
   a. input ena
   b. input rst
   c. input clk
   d. output reg d_100

2. Create the logic that will cause d_100 be high every 100 count

Test the divideby100 module

1. Run iSim to check that there are no mis match errors
2. If there are errors correct the module

Create the ctr_blk module

1. Create a ctr_blk.v file using the new source wizard to give it the following input/output settings
   a. input m_sec
   b. input clr
   c. input clk
   d. output [3:0] tenths
   e. output [3:0] sec_1s
   f. output [3:0] sec_10s
   g. output [3:0] min_1s
2. Define wires for chain 0, 1, 2 and 3.
3. Instantiate the following modules with the proper wire connections (See attached source for more details)

   a. divideby100 u0
   b. counter_0to9 u1
   c. counter_0to9 u2
   d. counter_0to5 u3
   e. counter_0to9 u4

Prepare sw_core

1. Instantiate the ctr_blk as ucb with all the proper connections (See attached source for more details)
2. Instantiate the dsp_drvr as udd with all the proper connections (See attached source for more details)
3. Create Wires with 3 bits called
   a. d0
   b. d1
   c. d2
   d. d3
4. Create a wire called ms
5. Create an assign statement of the following
   a. "assign r_ms = run & ms"

Simulate the design

1. Navigate to the simulation mode
2. Launch iSim for the tb_sw_core
3. Run al simulation
4. Verify that the output works as expected (See results)

Compile to .bit file

1. Compile to a .bit file by navigating to
   a. Implementation
      i. Xc3s500-e4g320
         1. Lab7_top_io_wrapper (didn't rename)
            a. Implement design
            b. Generate programming file

Transfer .bit file to board

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

Results:

Below is iSim output from an input of the design simulation of tb_sw_core which appears to have worked properly and as expected.

<u>Output from simulation of tb_sw_core:</u>

ISim O.76xd (signature 0x8ddf5b5d)
This is a Full version of ISim.
WARNING: For instance udd/U1/, width 1 of formal port display_on is not equal to width 32 of actual constant.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim> run all
digit0 changed to: 1001111 - time: 100000020 ns
digit0 changed to: 0100100 - time: 200000020 ns
digit0 changed to: 0000110 - time: 200002020 ns
digit0 changed to: 0001011 - time: 200004020 ns
digit0 changed to: 0010010 - time: 200006020 ns
digit0 changed to: 0010000 - time: 200008020 ns
digit0 changed to: 1000111 - time: 200010020 ns
digit0 changed to: 0000000 - time: 200012020 ns
digit0 changed to: 0000010 - time: 200014020 ns
digit0 changed to: 1000000 - time: 200016020 ns
digit1 changed to: 1001111 - time: 200016040 ns
digit1 changed to: 0100100 - time: 200036040 ns
digit1 changed to: 0000110 - time: 200056040 ns
digit1 changed to: 0001011 - time: 200076040 ns
digit1 changed to: 0010010 - time: 200096040 ns
digit1 changed to: 0010000 - time: 200116040 ns
digit1 changed to: 1000111 - time: 200136040 ns
digit1 changed to: 0000000 - time: 200156040 ns
digit1 changed to: 0000010 - time: 200176040 ns
digit1 changed to: 1000000 - time: 200196040 ns
digit2 changed - to 1001111 - time = 200196060 ns
digit2 changed - to 0100100 - time = 200396060 ns
digit2 changed - to 0000110 - time = 200596060 ns
digit2 changed - to 0001011 - time = 200796060 ns
digit2 changed - to 0010010 - time = 200996060 ns
digit2 changed - to 1000000 - time = 201196060 ns
digit3 changed - to 1001111 - time = 201196080 ns
digit3 changed - to 0100100 - time = 202396080 ns
digit3 changed - to 0000110 - time = 203596080 ns
digit3 changed - to 0001011 - time = 204796080 ns
digit3 changed - to 0010010 - time = 205996080 ns
digit3 changed - to 0010000 - time = 207196080 ns
digit3 changed - to 1000111 - time = 208396080 ns
digit3 changed - to 0000000 - time = 209596080 ns

digit3 changed - to 0000010 - time = 210796080 ns
Simulation complete!!!

<u>Results on board</u>
Once the .bit file was transferred to the board I was able to see that I could reset and hold the counter. This works as expected.

Discussion:
This lab was exciting for me because it showed me in a more realistic way how all of our theory can be useful. Now we are left with a project that is more complete. I can now actually show someone that I made a partial stop watch which is pretty cool.

It was also interesting to note how the testing was completed in this lab. If we were to go ahead and test every permutation of the clock in the iSim simulator it would take a very long time. Instead we did a small hack so that the testing did not take as long. This is good that it saves us quite a bit of time.

Finally, as always, this lab allowed me to make quite a few mistakes and learn from them. The first mistake that cost me was that I accidentally made the wires 1 bit instead of 4 bit busses. This caused me all sorts of problems. Additionally I did not wire up all the modules correctly the first time. This also cost me some time. Surely this teaches me to be more careful in the future when I'm doing this sort of thing, especially with regards to design block diagrams.

Source Code:
ctr_blk.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    08:01:17 10/23/2012
// Design Name:
// Module Name:    ctr_blk
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ctr_blk(
    input m_sec,
    input clr,
    input clk,
    output [3:0] tenths,
    output [3:0] sec_1s,
    output [3:0] sec_10s,
    output [3:0] min_1s
    );

//Wires!
wire chain0, chain1, chain2, chain3;

//Instantiations

divideby100 u0 (.ena(m_sec), .rst(clr), .clk(clk), .d_100(chain0));

counter_0to9 u1 (.ena(chain0), .clr(clr), .clk(clk), .carry_out(chain1), .q(tenths));
counter_0to9 u2 (.ena(chain1), .clr(clr), .clk(clk), .carry_out(chain2), .q(sec_1s));
counter_0to5 u3 (.ena(chain2), .clr(clr), .clk(clk), .carry_out(chain3), .q(sec_10s));
counter_0to9 u4 (.ena(chain3), .clr(clr), .clk(clk), .q(min_1s));

endmodule
```

counter_0to5.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:19:11 10/08/2012
// Design Name:
// Module Name:    counter_0to5
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module counter_0to5(
    input ena,
    input clr,
    input clk,
    output reg [3:0] q,
    output reg carry_out);

 //Declare reg for Next_q
 reg [3:0] Next_q;


always @(posedge clk)
			begin
			q <= Next_q;
			end


		always @(q or ena or clr)
			begin

				Next_q = 4'b0000;
				carry_out = 4'b0;


				// ** HOLD **
				//Logic when ena and clr are set to 0 to hold current values
				if( (ena == 1'b0) && (clr == 1'b0))
					begin
						carry_out = 1'b0;
						Next_q[0] = q[0];
```

```
                        Next_q[1] = q[1];
                        Next_q[2] = q[2];
                        Next_q[3] = q[3];
                end



        // ** MAX **
        //Logic to handle when ena =1 and clr = 1 to get ouput
        //of 5

        if ( (ena ==1'b1) && (clr == 1'b1) )
                begin
                        carry_out = 1'b1;
                        Next_q = 4'b0101;
                end



        // ** NEXT COUNT **
        //Logic when ena = 1 to calculate our next bit

        if ((ena == 1'b1) && (clr == 1'b0))
                begin

                        Next_q = q + 1;

                        if ( q == 5) carry_out = 1;
                        if ( q >= 5) Next_q = 0;

                end // end of if handling the next bit



        // ** CLEAR **
        //Logic when clr = 1 and ena = 0 to clear
        if ((clr == 1'b1) && (ena == 1'b0))
                begin
                        Next_q = 4'b0000; // set to 0000.
                        carry_out = 1'b0;                       // set the carry
bit to 0.
                end // end if to set Next_1 and carry_out to 0.


    end  // End my entire always block

endmodule
```

counter_0to9.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:28:37 10/08/2012
// Design Name:
// Module Name:    counter_0to9
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module counter_0to9(
    input ena,
    input clr,
    input clk,
    output reg[3:0] q,
    output reg carry_out
);

	 //Declare Next_q reg type
	 reg[3:0] Next_q ;


	//Let us do some logic on our input switches


	always @(posedge clk)
		begin
		q <= Next_q;
		end


	always @(q or ena or clr)
		begin

			Next_q = 4'b0000;
			carry_out = 4'b0;


			// ** HOLD **
			//Logic when ena and clr are set to 0 to hold current values
```

```
		if( (ena == 1'b0) && (clr == 1'b0))
			begin
				carry_out = 1'b0;
				Next_q[0] = q[0];
				Next_q[1] = q[1];
				Next_q[2] = q[2];
				Next_q[3] = q[3];
			end


		// ** MAX **
		//Logic to handle when ena =1 and clr = 1 to get ouput
		//of 9

		if ( (ena ==1'b1) && (clr == 1'b1) )
			begin
				carry_out = 1'b1;
				Next_q = 4'b1001;
			end


		// ** NEXT COUNT **
		//Logic when ena = 1 to calculate our next bit

		if ((ena == 1'b1) && (clr == 1'b0))
			begin

				Next_q = q + 1;

				if ( q == 9) carry_out = 1;
				if ( q >= 9) Next_q = 0;

			end // end of if handling the next bit


		// ** CLEAR **
		//Logic when clr = 1 and ena = 0 to clear
		if ((clr == 1'b1) && (ena == 1'b0))
			begin
				Next_q = 4'b0000; // set to 0000.
				carry_out = 1'b0;
		end // end if to set Next_1 and carry_out to 0.

end  // End my entire always block
endmodule
```

dividebу100.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:50:10 10/22/2012
// Design Name:
// Module Name:    divideby100
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module divideby100(
    input ena,
    input rst,
    input clk,
    output reg d_100
    );

        // Register to store count and next count so that we can
        // count from 1 to 100 and then set d_100 to 1 every cycle.
        reg [0:7] count, next_count;

        // Sequential block
        always @(posedge clk) begin
                count <= next_count;
        end

        // Combinational block for count
        always @(ena, rst, count) begin


                // Default logic
                next_count = count;
                d_100 = 0;


                //Handle adding 1
                if ( ena == 1) begin
                        next_count = count + 1;

                                if ( count == 99) begin
```

```
                    next_count = 0;
                    d_100 = 1;
                end
        end

        //Priority
        if (rst == 1) begin
            next_count = 0;
        end



    end // end of combinational block


endmodule
```

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,
    output [6:0] cathode,
    output [3:0] anode,
    output m_sec
    );


//Wires and such
wire [3:0] d0, d1, d2, d3;
wire ms;
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(m_rst),
                                 .clk(m_clk),
                                 .min_1s(d3),
                                 .sec_10s(d2),
                                 .sec_1s(d1),
                                 .tenths(d0));


endmodule
```

dsp_drvr.v

```
//
// lab7 : version 10/09/2012
//
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
module dsp_drvr(
    input clk,
    input [3:0] digit0,
    input [3:0] digit1,
    input [3:0] digit2,
    input [3:0] digit3,
    input rst,
    output reg [3:0] an,
    output [6:0] cath,
    output reg m_sec
            );




        // register for dividing by 50000
        reg [15:0] count, next_count;
        // register for anode control
        reg [1:0] anreg, next_anreg;
        // for multiplexed input into the decoder
        reg [3:0] mux_digit;
        // output from the decoder
        wire [6:0] seg_out;

        // sequential block
        always @(posedge clk) begin
                count <= next_count;
                anreg <= next_anreg;
        end     // end of always block

        // combinational logic for count
        always @(count, rst) begin
                // take care of defaults
                next_count = count - 1;
                m_sec = 0;

                // normal logic
                if (count == 0) begin
                        next_count = 49999;
                        m_sec = 1;
                end
                // priority logic
```

```
		if (rst == 1) next_count = 49999;
	end	// end of count combinational logic

	// anode control counter logic
	always @(anreg, rst, m_sec) begin
		// defaults
		next_anreg = anreg;	// hold count
		// regular logic
		if (m_sec == 1) next_anreg = anreg + 1;
		// priority logic
		if (rst == 1) next_anreg = 0;
	end	// end of anode combinational logic

	// now do two muxes: digit mux and an signals
	always @(anreg,digit0,digit1,digit2,digit3) begin
		an = 4'b1111;	// default
		case (anreg)
			0: begin
				mux_digit = digit0;
				an[3] = 0;
			end
			1: begin
				mux_digit = digit1;
				an[2] = 0;
			end
			2: begin
				mux_digit = digit2;
				an[1] = 0;
			end
			3: begin
				mux_digit = digit3;
				an[0] = 0;
			end
		endcase
	end	// end of combinational mux

	// finally instantiate the decoder here
	svn_seg_decoder U1(.display_on(1), .bcd_in(mux_digit), .seg_out(seg_out));
	assign cath = ~seg_out; // invert

endmodule
```

svn_seg_decoder.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:16:21 09/11/2012
// Design Name:             seven segment decoder module
// Module Name:    svn_seg_decoder
// Project Name:   lab 03 seven segment decoder
// Target Devices:  xilinx board
// Tool versions:
// Description: Take in 4 bits as a descriptor of the number you want to show and also
// 1bit as an on/off switch and then output the appropriate signal to drive
// the seven segment display.
//
// Dependencies:
//
// Revision: 2 (now using case statement)
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module svn_seg_decoder(
    input [3:0] bcd_in,
    input display_on,
    output reg [6:0] seg_out
    );



// My always logic to replace old code
always @(display_on, bcd_in[3] or bcd_in[2] or bcd_in[1] or bcd_in[0]) begin

        case({display_on,bcd_in[3],bcd_in[2],bcd_in[1],bcd_in[0]})

        // Takes into account all combinations with the display turned off

                //Number 0000 Decimal Val: 0 Display: on *
                5'b10000: seg_out = 7'b0111111;

                //Number 0001 Decimal Val: 1 Display: on *
                5'b10001: seg_out = 7'b0110000;

                //Number 0010 Decimal Val: 2 Display: on *
                5'b10010: seg_out = 7'b1011011;

                //Number 0011 Decimal Val: 3 Display: on
                5'b10011: seg_out = 7'b1111001;

                //Number 0100 Decimal Val: 4 Display: on
```

```
        5'b10100: seg_out = 7'b1110100;

        //Number 0101 Decimal Val: 5 Display: on
        5'b10101: seg_out = 7'b1101101;

        //Number 0110 Decimal Val: 6 Display: on
        5'b10110: seg_out = 7'b1101111;

        //Number 0111 Decimal Val: 7 Display: on
        5'b10111: seg_out = 7'b0111000;

        //Number 1000 Decimal Val: 8 Display: on
        5'b11000: seg_out = 7'b1111111;

        //Number 1001 Decimal Val: 9 Display: on
        5'b11001: seg_out = 7'b1111101;

        default: seg_out = 7'b0000000;

    endcase

end

endmodule
```