EGR 400 A
Advanced Digital
System Design Using FPGAs

Lab 3: Enhanced Stopwatch Design

Prepared for: Dr. Foist

Christopher Parisi

College of Engineering
California Baptist University

10/19/12

Introduction

The objective for this lab was to design, synthesize, and verify a stopwatch that can count both forwards and backwards. The counter tells the time past in the form of M.SS.D using the seven-segment display attachment to the Xilinx board. The timer can be reset by pressing a button on the board. The forward and backward counting functions are also utilized by a push button. The purpose of the lab was to teach students how to program a stop watch and how to implement the logic of a modulo counter for accurate time measurement . This lab also teaches students more about the ISE IDE, Spartan 3E-Starter board, and Xilinx Design Flow.

Procedure

Dr. Chu's Stopwatch

Before starting the lab 3 assignment, I began by downloading Dr. Chu's stopwatch code. I downloaded the code onto the Spartan 3E-Starter board and observed the design. I found that the West button started the timer, and the South button reset the timer. One thing that was slightly off about this design was a denouncing issue. The button occasionally needed to be pressed multiple times to actually start the timer. Another design problem was the "go" button had to be held to keep the stopwatch running.

Dr. Foist's Stopwatch

After observing Dr. Chu's code, I downloaded Dr. Foist's stopwatch files and downloaded the design to the Spartan-3E Starter board. I observed the functionality of this stopwatch and found that the "go" button no longer needed to be held to continuously run the timer. This design also contained a debouncing glitch which cause the button to not work at times.

After testing the design, I was instructed to create a testbench for the State Machine utilized by Dr. Foist's design. This testbench tested the design for each state and displayed this information on the waveform. I had to learn how to view the "state_reg" variable in my waveform even though it was not part of the top level design. After working in ISE for some time, I learned how to do this and dragged the signal into my waveform. The Verilog code for my test fixture can be found in Appendix A along with the simulation results in Figure 2.

Lab Assignment

The lab assignment was to improve these designs by adding two modifications. Both previous designs had stopwatches that counted to 9.99.9. This does not count minutes, but continuously increase numbers as the seconds go on. Part of the lab assignment was fixing this issue so the stopwatch kept tract of minutes as well. The second modification involved adding another signal to control the direction the stopwatch counted.

Step 1

"Add an additional signal, up, to control the direction of counting. The stopwatch counts up when the up signal is asserted and counts down otherwise."

Working of Dr. Foist's design, I added a new signal named "up" to his design and assigned it to another push button. When this signal goes high (button is pushed and held), the timer revers directions and starts counting down to 0.00.0. I downloaded this modified design to the board and tested it. When the button was pressed, the circuit counted backwards as required. This still counted only seconds and not minutes.

Step 2

"Add a minute digit to the display. The LED display format should be like M.SS.D, where D represents 0.1 second and its range is between 0 and 9, SS represents seconds and its range is between 00 and 59, and M represents minutes and its range is between 0 and 9."

After I tested the multi-direction stopwatch, I then began editing the logic to incorporate minutes into the design. This was changed by making the max value of seconds to be 59. When the program hits 59 seconds, the seconds reset to 00 and the minute digit is incremented by 1. I downloaded this final design to the Spartan-3E Starter board and verified its functionality. The timer went to X.59.99 and then incremented the minute and reset the rest to 0. This also worked for when the circuit counted down. The top-level Verilog module code can be found in Appendix B, the stopwatch module Verilog code can be found in Appendix C, and the UCF can be found in Appendix D. A summary of the Device Utilization can be found on Figure 1 of the next page.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	59	9,312	1%	
Number of 4 input LUTs	78	9,312	1%	
Number of occupied Slices	63	4,656	1%	
Number of Slices containing only related logic	63	63	100%	
Number of Slices containing unrelated logic	0	63	0%	
Total Number of 4 input LUTs	117	9,312	1%	
Number used as logic	78			
Number used as a route-thru	39			
Number of bonded IOBs	22	232	9%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.12			

Figure 1: Enhanced Stopwatch Device Utilization Summary

Conclusion

In summary, this lab has taught me the functionality and implementation of a multidirectional stopwatch. This lab has also given me more experience "thinking hardware" as I write Verilog code. Everything during this lab worked correctly and there were no major hindrances. I learned how to read and modify Verilog code written by another programmer as well as thinking about the logic of a modulo counter needed to implement time tracking functions. I also learned how to bring in signals from lower levels of the code into top level design testbench waveforms.

Appendix A

State Machine Verilog Test Fixture

```
`timescale 1ns / 1ps

module state_mc_swatch_tb;

	localparam T=20;

	// Inputs
	reg clk;
	reg reset;
	reg go;

	// Outputs
	wire cnt_ena;
	wire cntr_clr;

	// Instantiate the Unit Under Test (UUT)
	state_mc_swatch uut (
		.clk(clk),
		.reset(reset),
		.go(go),
		.cnt_ena(cnt_ena),
		.cntr_clr(cntr_clr)
	);

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

	initial begin
		// Initialize Inputs
		reset = 0;
		go = 0;

		// Wait 100 ns for global reset to finish
		#100;
```

```
        // Test Vector 1
        reset = 0;
        go = 1;

        #200;

        // Test Vector 2
        reset = 1;
        go = 1;

        #100;

        // Test Vector 3
        reset = 0;
        go = 1;

        #200;

    end

endmodule
```

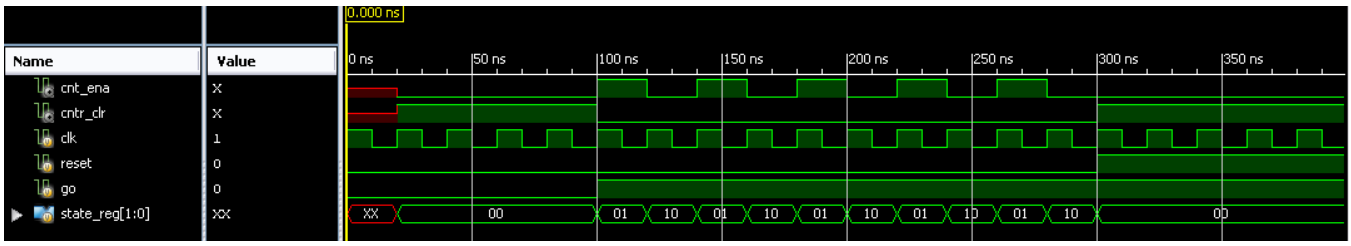

Figure 2: State Machine Verilog Test Fixture Simulation Results

Appendix B

Enhanced Stopwatch Top-Level Verilog Module

```
module s3e_stopwatch_sm_test
  (
   input wire clk,
   input wire [1:0] btn,
            input wire up,
   output wire an_lo,
   output wire an_hi,
   output wire [6:0] sseg_lo,
   output wire [6:0] sseg_hi,
            output wire [1:0] led
  );

  // signal declaration
  wire [3:0] d3, d2, d1, d0;
            wire cnt_ena, cntr_clr;

  disp_hex_mux_s3e disp_unit_lo
    (.clk(clk), .reset(1'b0), .hex0(d1), .hex1(d0),
     .an(an_lo), .sseg(sseg_lo));

  disp_hex_mux_s3e disp_unit_hi
    (.clk(clk), .reset(1'b0), .hex0(d3), .hex1(d2),
     .an(an_hi), .sseg(sseg_hi));


  // instantiate stopwatch
  stop_watch_if counter_unit
  // (.clk(clk), .go(btn[1]), .up(BTN[East], .clr(btn[0]),
    (.clk(clk), .go(cnt_ena), .up(up), .clr(cntr_clr),

     .d3(d3), .d2(d2), .d1(d1), .d0(d0),
                        .ms_reg_b22(led[1]), .ms_reg_b21(led[0]) // added by Rod

                        );

            // instantiate state machine to handle "go" input
  state_mc_swatch sm1(
                        .clk(clk),
                        .reset(btn[0]),
    .go(btn[1]),
                        .cnt_ena(cnt_ena),
                        .cntr_clr(cntr_clr)
  );


endmodule
```

Appendix C

stop_watch_if module Verilog Code

```
// Listing 4.18
module stop_watch_if
  (
   input wire clk,
   input wire go, clr, up,
   output wire [3:0] d3, d2, d1, d0,
              // added by Rod
   output wire ms_reg_b22, ms_reg_b21

  );

  // declaration
  localparam  DVSR = 5000000;
  reg [22:0] ms_reg;
  wire [22:0] ms_next;
  reg [3:0] d3_reg, d2_reg, d1_reg, d0_reg;
  reg [3:0] d3_next, d2_next, d1_next, d0_next;
  wire ms_tick;

  // body
  // register
  always @(posedge clk)
  begin
    ms_reg <= ms_next;
                        d3_reg <= d3_next;
    d2_reg <= d2_next;
    d1_reg <= d1_next;
    d0_reg <= d0_next;
  end

  // next-state logic
  // 0.1 sec tick generator: mod-5000000
  assign ms_next = (clr || (ms_reg==DVSR && go)) ? 4'b0 :(go) ? ms_reg + 1 : ms_reg;
  assign ms_tick = (ms_reg==DVSR) ? 1'b1 : 1'b0;
  // 3-digit bcd counter
  always @*
  begin
    // default: keep the previous value
    d0_next = d0_reg;
    d1_next = d1_reg;
    d2_next = d2_reg;
                        d3_next = d3_reg;
    if (clr)
      begin
        d0_next = 4'b0; //decimal
        d1_next = 4'b0; //seconds1
        d2_next = 4'b0; //seconds2
                                              d3_next = 4'b0; //minutes
      end
    else if (ms_tick)
            if (!up) //count up

                        if (d0_reg != 9)
                                d0_next = d0_reg + 1;
                        else            // reach XXX.9
```

```
                begin
                        d0_next = 4'b0;
                        if (d1_reg != 9)
                                d1_next = d1_reg + 1;
                        else      // reach XX9.9
                                begin
                                        d1_next = 4'b0;
                                        if (d2_reg != 5)
                                                d2_next = d2_reg + 1;
                                        else // reach X59.9
                                                begin
                                                        d2_next = 4'b0;
                                                        if (d3_reg != 9)
                                                                d3_next = d3_reg + 1;
                                                                else // reach 959.9
                                                                        d3_next = 4'b0;
                                                        end
                                        end
                        end
        else //count down
                if (d0_reg != 0)
                        d0_next = d0_reg - 1;
                else      // reach 0XX.X
                        begin
                                d0_next = 4'b1001;
                                if (d1_reg != 0)
                                        d1_next = d1_reg - 1;
                                else      // reach 00X.X
                                        begin
                                                d1_next = 4'b1001;
                                                if (d2_reg != 0)
                                                        d2_next = d2_reg - 1;
                                                else // reach 000.X
                                                        begin
                                                                d2_next = 4'b0101;
                                                                if (d3_reg != 0)
                                                                        d3_next = d3_reg -1;
                                                                else // reach 000.0
                                                                        d3_next = 4'b1001;
                                                        end
                                        end
                        end
  end

  // output logic
  assign d0 = d0_reg;
  assign d1 = d1_reg;
  assign d2 = d2_reg;
            assign d3 = d3_reg;
            // added by Rod
  assign ms_reg_b22 = ms_reg[22];
  assign ms_reg_b21 = ms_reg[21];


endmodule
```

Appendix D

Enhanced Stop Watch UCF (User Constraints File)

```
#=========================================================
#    Pin assignment for Xilinx
#    Spartan-3E Starter board
#=========================================================

#=========================================================
# clock and reset
#=========================================================
# Period constraint for 50MHz operation, from Ken C.
#
NET "clk" PERIOD = 20.0ns HIGH 50%;
#
# soldered 50MHz Clock.
#
NET "clk" LOC = "C9" | IOSTANDARD = LVTTL;


#=========================================================
# buttons & switches
#=========================================================
# 4 pushbuttons
#NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
#NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn<0>" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ; # "BTN_SOUTH"
NET "btn<1>" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ; # "BTN_WEST"
NET "up" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ; # "BTN_EAST"

# 4 slide switches
#NET "sw<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
#NET "sw<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
#NET "sw<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
#NET "sw<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;

# ------------------ Leds for debug ------------------
#NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<1>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ; # led1,0 for debug

#NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<0>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

#NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
#=========================================================
# 4-digit time-multiplexed 7-segment LED display
#=========================================================
NET "sseg_lo<6>" LOC = "A4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;  # segment a
```

```
NET "sseg_lo<5>" LOC = "C5" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;  # segment b
NET "sseg_lo<4>" LOC = "B6" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;  # segment c
NET "sseg_lo<3>" LOC = "F7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;  # segment d
NET "sseg_lo<2>" LOC = "C7" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ; # segment e
NET "sseg_lo<1>" LOC = "E8" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ; # segment f
NET "sseg_lo<0>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ; # segment g
NET "an_lo"          LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# 2-digits enable

NET "sseg_hi<6>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment a
NET "sseg_hi<5>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment b
NET "sseg_hi<4>" LOC = "B13" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment c
NET "sseg_hi<3>" LOC = "B14" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment d
NET "sseg_hi<2>" LOC = "D14" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment e
NET "sseg_hi<1>" LOC = "B16" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# segment f
NET "sseg_hi<0>" LOC = "C4"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ; # segment g
NET "an_hi"          LOC = "A11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 6 ;# 2-digits enable
```