Lab 5: Stopwatch

Design sources:

stopwatch.v

```
module stopwatch(
    input start,
    input stop,
    input store,
    input reset,
    input clk,
    output [3:0] anode,
    output [6:0] segment,
    output led_start,
    output led_stop,
    output led_store,
    output led_reset
    );

    reg newRecordFlag;
    reg [15:0] r0, r1;
    reg [19:0] timing = 5'h00000;
    wire [15:0] bcd_out, w0;

    stopwatch_control_unit scu(
        .start(start),
        .stop(stop),
        .store(store),
        .reset(reset),
        .newRecordFlag(newRecordFlag),
        .clk(timing[19]),
        .displayInputSel(displayInputSel),
        .registerEN(registerEN),
        .registerInputSel(registerInputSel),
        .counterEN(counterEN),
        .counterReset(counterReset)
    );

    four_digit_binary_coded_decimal_counter bcd(
        .clk(timing[19]),
        .enable(counterEN),
        .reset(counterReset),
        .out(bcd_out)
    );

    display_control dc(
        .clk(clk),
        .in(r2),
        .segment(segment),
        .anode(anode)
    );

    sixteen_bit_parallel_load_register plr(
        .clk(timing[19]),
        .enable(registerEN),
        .reset(),
        .in(r0),
        .out(w0)
    );

    always @(posedge clk)
    begin
        // 16-bit mux
        if (registerInputSel == 1) r0 = 16'h9999;
        else if (registerInputSel == 0) r0 = bcd_out;

        // 16-bit mux
```

stopwatch_control_unit.v

```
module stopwatch_control_unit(
   input start,
   input stop,
   input store,
   input reset,
   input newRecordFlag,
   input clk,
   output displayInputSel,
   output registerEN,
   output registerInputSel,
   output counterEN,
   output counterReset
   );

   reg [2:0] s;
   wire [2:0] n;

   stopwatch_state_transition_control sstc(
      .start(start),
      .stop(stop),
      .store(store),
      .reset(reset),
      .newRecordFlag(newRecordFlag),
      .s(s),
      .n(n)
    );

    stopwatch_output_control soc(
      .s(s),
      .displayInputSel(displayInputSel),
      .counterEN(counterEN),
      .registerEN(registerEN),
      .counterReset(counterReset),
      .registerInputSel(registerInputSel)
    );

   always @(posedge clk) s = n;

endmodule
```

stopwatch_state_transition_control.v

```
module stopwatch_state_transition_control(
   input start,
   input stop,
   input store,
   input reset,
   input newRecordFlag,
   input [2:0] s,
   output reg [2:0] n
   );

   always @(*)
   begin
      n[0] = (~start & ~reset & ~newRecordFlag & s[0]) |
           (~start & ~reset & ~s[1] & s[0]) |
           (~store & ~reset & s[1] & s[0]) |
           (~reset & s[2] & s[0]) |
           (reset & s[2] & s[1]) |
           (~start & store & ~reset & ~s[0]) |
           (stop & ~reset & ~newRecordFlag) |
           (stop & ~reset & ~s[1]) |
           (stop & ~reset & ~s[0]) |
           (start & ~reset & ~s[1] & ~s[0]) |
           (start & ~stop & ~reset & s[1] & s[0]) |
           (start & ~reset & s[2]);
      n[1] = (~start & ~store & s[1]) |
           reset |
           (stop & ~store) |
           (start & ~stop & ~s[2] & ~s[1] & s[0]) |
           (start & ~stop & ~s[2] & s[1] & ~s[0]);
      n[2] = (~start & ~stop & s[2]) |
           reset |
           (~start & store) |
           (stop & store);
   end
endmodule
```

stopwatch_output_control.v

```
module stopwatch_output_control(
   input wire [2:0] s,
   output wire displayInputSel,
   output wire counterEN,
   output wire registerEN,
   output wire counterReset,
   output wire registerInputSel
   );

   reg [4:0] out;

   always @(*)
   begin
      case(s)
         3'b000: out = 5'b10010;
         3'b001: out = 5'b11010;
         3'b010: out = 5'b11000;
         3'b011: out = 5'b10000;
         3'b100: out = 5'b10101;
         3'b101: out = 5'b00001;
         3'b110: out = 5'b10010;
         3'b111: out = 5'b10110;
      endcase
   end

   assign displayInputSel = out[4];
   assign counterEN = out[3];
   assign registerEN = out[2];
   assign counterReset = out[1];
   assign registerInputSel = out[0];

endmodule
```

four_digit_binary_coded_decimal_counter.v

```
module four_digit_binary_coded_decimal_counter(
   input clk,
   input enable,
   input reset,
   output [15:0] out
   );

   wire CO_0, CO_1, CO_2, CO_3;

   // four_bit_binary_coded_decimal_counter(clk,                  enable, reset, carryOut, out);
   four_bit_binary_coded_decimal_counter d0(clk,                  enable, reset,     CO_0, out[3:0]);
   four_bit_binary_coded_decimal_counter d1(clk,             CO_0 & enable, reset,     CO_1, out[7:4]);
   four_bit_binary_coded_decimal_counter d2(clk,       CO_0 & CO_1 & enable, reset,     CO_2, out[11:8]);
   four_bit_binary_coded_decimal_counter d3(clk, CO_0 & CO_1 & CO_2 & enable, reset,     CO_3, out[15:12]);

endmodule
```

four_bit_binary_coded_decimal_counter.v

```
module four_bit_binary_coded_decimal_counter(
   input clk,
   input enable,
   input reset,
   output reg carryOut,
   output reg [3:0] out
   );

   always @(posedge clk)
   begin

      if (reset) out = 0;

      else
      begin
         if (enable)
         begin
            case (out)
               4'b0000: out = 4'b0001;
               4'b0001: out = 4'b0010;
               4'b0010: out = 4'b0011;
               4'b0011: out = 4'b0100;
               4'b0100: out = 4'b0101;
               4'b0101: out = 4'b0110;
               4'b0110: out = 4'b0111;
               4'b0111: out = 4'b1000;
               4'b1000: out = 4'b1001;
               4'b1001: out = 4'b0000;
            endcase

            if (out == 4'b1001) carryOut = 1;
            else carryOut = 0;
         end
      end
   end

endmodule
```

display_control.v

```
module display_control(
   input clk,
   input [15:0] in,
   output [6:0] segment,
   output reg [3:0] anode
   );

   // Selecting between which digit/anode to refresh
   reg [1:0] digit_select;
   reg [19:0] refresh = 20'h00000;
   reg [3:0] gsv_in;
   reg [6:0] gsv_segment;

   get_segment_value gsv(gsv_in, segment);

   always @(posedge clk)
   begin

      refresh <= refresh + 1;
      digit_select <= refresh[19:18];

      case (digit_select)
         2'b00:
         begin
            anode = 4'b1110;
            gsv_in = in[3:0];
         end
         2'b01:
         begin
            anode = 4'b1101;
            gsv_in = in[7:4];
         end
         2'b10:
         begin
            anode = 4'b1011;
            gsv_in = in[11:8];
         end
         2'b11:
         begin
            anode = 4'b0111;
            gsv_in = in[15:12];
         end
      endcase
   end
endmodule
```

get_segment_value.v

```
module get_segment_value(
   input [3:0] switches,
   output reg [6:0] segment
   );

   // Define named constants for anode select values
   parameter ANODE_SELECT_DIGIT_0 = 4'b1110;
   parameter ANODE_SELECT_DIGIT_1 = 4'b1101;
   parameter ANODE_SELECT_DIGIT_2 = 4'b1011;
   parameter ANODE_SELECT_DIGIT_3 = 4'b0111;

   // Define named constants for switch values
   parameter SWITCH_VALUE_ZERO  = 4'b0000;
   parameter SWITCH_VALUE_ONE   = 4'b0001;
   parameter SWITCH_VALUE_TWO   = 4'b0010;
   parameter SWITCH_VALUE_THREE = 4'b0011;
   parameter SWITCH_VALUE_FOUR  = 4'b0100;
   parameter SWITCH_VALUE_FIVE  = 4'b0101;
   parameter SWITCH_VALUE_SIX   = 4'b0110;
   parameter SWITCH_VALUE_SEVEN = 4'b0111;
   parameter SWITCH_VALUE_EIGHT = 4'b1000;
   parameter SWITCH_VALUE_NINE  = 4'b1001;

   // Define named constants for segment values
   parameter SEGMENT_VALUE_ZERO  = 7'b1000000;
   parameter SEGMENT_VALUE_ONE   = 7'b1111001;
   parameter SEGMENT_VALUE_TWO   = 7'b0100100;
   parameter SEGMENT_VALUE_THREE = 7'b0110000;
   parameter SEGMENT_VALUE_FOUR  = 7'b0011001;
   parameter SEGMENT_VALUE_FIVE  = 7'b0010010;
   parameter SEGMENT_VALUE_SIX   = 7'b0000010;
   parameter SEGMENT_VALUE_SEVEN = 7'b1111000;
   parameter SEGMENT_VALUE_EIGHT = 7'b0000000;
   parameter SEGMENT_VALUE_NINE  = 7'b0010000;

   always @(*)
   begin
      case (switches[3:0])
         SWITCH_VALUE_ZERO:  segment = SEGMENT_VALUE_ZERO;
         SWITCH_VALUE_ONE:   segment = SEGMENT_VALUE_ONE;
         SWITCH_VALUE_TWO:   segment = SEGMENT_VALUE_TWO;
         SWITCH_VALUE_THREE: segment = SEGMENT_VALUE_THREE;
         SWITCH_VALUE_FOUR:  segment = SEGMENT_VALUE_FOUR;
         SWITCH_VALUE_FIVE:  segment = SEGMENT_VALUE_FIVE;
         SWITCH_VALUE_SIX:   segment = SEGMENT_VALUE_SIX;
         SWITCH_VALUE_SEVEN: segment = SEGMENT_VALUE_SEVEN;
         SWITCH_VALUE_EIGHT: segment = SEGMENT_VALUE_EIGHT;
         SWITCH_VALUE_NINE:  segment = SEGMENT_VALUE_NINE;
         default:            segment = SEGMENT_VALUE_ZERO;
      endcase
   end
endmodule
```

sixteen_bit_parallel_load_register.v

```
module sixteen_bit_parallel_load_register(
   input clk,
   input enable,
   input reset,
   input [15:0] in,
   output reg [15:0] out
   );

   always @(posedge clk & enable)
   begin
      if (reset) out = 0;
      else out = in;
   end
endmodule
```

Simulation Sources:

testbench_stopwatch.v

```
module testbench_stopwatch;

    reg start, stop, store, reset, clk;
    wire [3:0] anode;
    wire [6:0] segment;
    wire led_start, led_stop, led_store, led_reset;

    stopwatch dut(
        .start(start),
        .stop(stop),
        .store(store),
        .reset(reset),
        .clk(clk),
        .anode(anode),
        .segment(segment),
        .led_start(led_start),
        .led_stop(led_stop),
        .led_store(led_store),
        .led_reset(led_reset)
    );

    always #10 clk = ~clk;

    initial begin
        clk = 0;
        start = 0;
        stop = 0;
        store = 0;
        reset = 0;

        // Start stopwatch
        #100 start = 1;
        #100000 start = 0;

        // Stop stopwatch
        #1000000000 stop = 1;
        #100000000 stop = 0;

        // Store time
        #1000000 store = 1;
        #1000000 store = 0;

        // Reset stopwatch
        #1000000 reset = 1;
        #1000000 reset = 0;

        // Start stopwatch
        #1000000 start = 1;
        #1000000000 start = 0;

        // Stop stopwatch
        #200000000 stop = 1;
        #100000 stop = 0;

        // Store time
        #100000 store = 1;
        #100000 store = 0;

        // Reset stopwatch
        #100000 reset = 1;
        #100000 reset = 0;
```

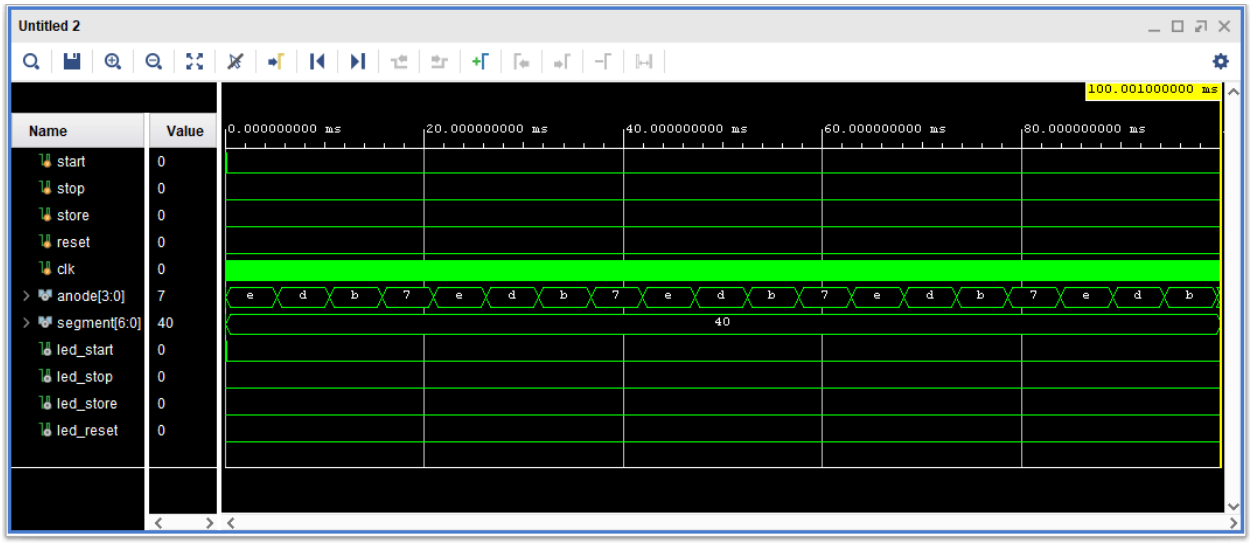
testbench_stopwatch_control_unit.v

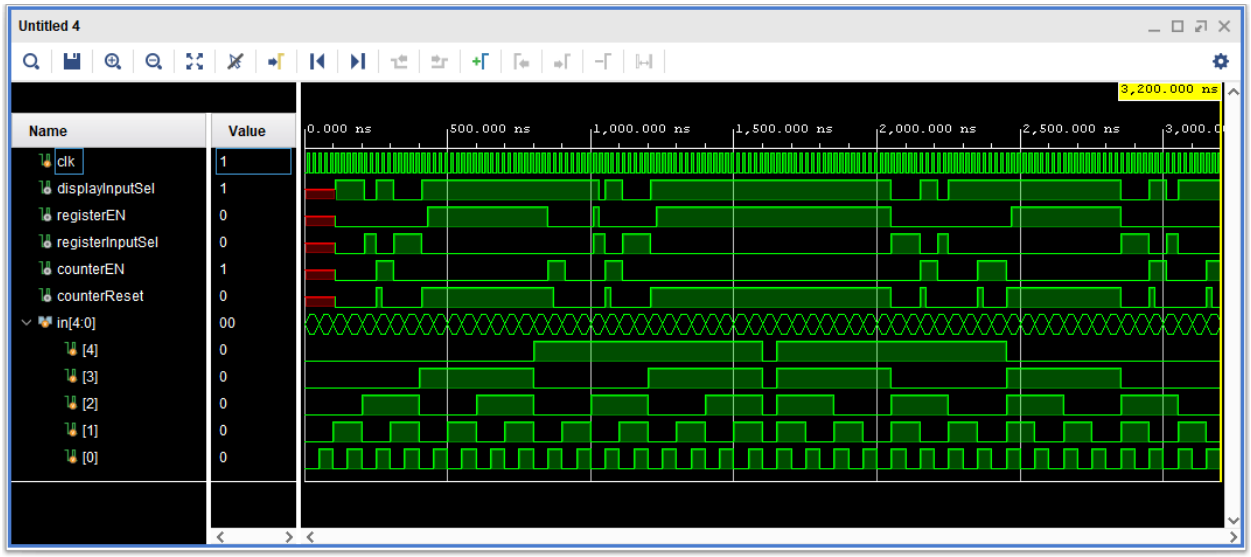
```
module testbench_stopwatch_control_unit;

    reg clk = 0;

    wire displayInputSel;
    wire registerEN;
    wire registerInputSel;
    wire counterEN;
    wire counterReset;

    stopwatch_control_unit scu(
        .start(in[0]),
        .stop(in[1]),
        .store(in[2]),
        .reset(in[3]),
        .newRecordFlag(in[4]),
        .clk(clk),
        .displayInputSel(displayInputSel),
        .registerEN(registerEN),
        .registerInputSel(registerInputSel),
        .counterEN(counterEN),
        .counterReset(counterReset)
    );

    reg [4:0] in = 5'b00000;

    initial
    begin
        #50 in = in + 1;
        while (in != 0)
        begin
            #50 in = in + 1;
        end
        #50 in = in - 1;
        while (in != 0)
        begin
            #50 in = in - 1;
        end
        $finish;
    end

    always @(*)
    begin
        #10 clk <= ~clk;
    end

endmodule
```

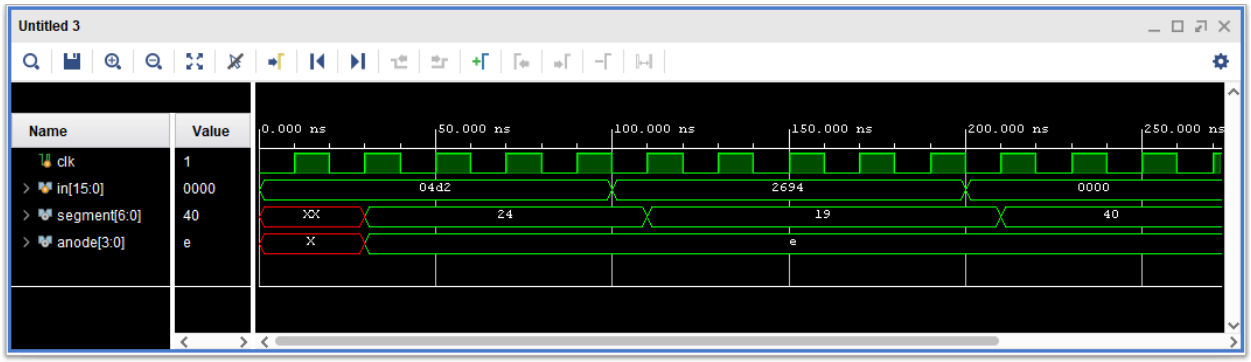
testbench_display_control.v

```
module testbench_display_control;

    // Declare signals for testbench
    reg clk = 0;
    reg [15:0] in;
    wire [6:0] segment;
    wire [3:0] anode;

    // Instantiate the module to be tested
    display_control dut(clk, in, segment, anode);

    always #10 clk = ~clk;

    // Provide input values to the module for testing
    initial
    begin
//        clk = 0;
        // Test case 1: Input value 1234
        in = 16'd1234;
        #100;
        // Expected output:
        // Segment = 7'b0110000
        // Anode = 4'b1110 (displaying digit 4)

        // Test case 2: Input value 9876
        in = 16'd9876;
        #100;
        // Expected output:
        // Segment = 7'b0011111
        // Anode = 4'b0111 (displaying digit 1)

        // Test case 3: Input value 0
        in = 16'd0;
        #100;
        // Expected output:
        // Segment = 7'b1111110
        // Anode = 4'b1110 (displaying digit 0)

        // Add more test cases here as needed

        $finish; // End simulation
    end

endmodule
```

Simulation Results:







RTL Schematic: