10.12.2019

CS 223-2

LAB-5

PRELIMINARY REPORT

ZÜBEYİR BODUR

21702382

Trainer No: 22

Auxilary SystemVerilog Modules

```
//The register module that will be used in master & slave process.
module register( input logic load, clk, rst, [15:0]d,
                        output logic [15:0] q
    );
    always_ff@(posedge clk, posedge rst) begin
        if (rst) q <= 0;
        else if (load) q <= d;
    end
endmodule

// The timer that will be used in master & slave process
// Here multiplier M is the multiplier of base time
module timer( input logic clk, load, enable, [31:0]M,
                        output logic overflow
    );
    logic [31:0]downcount;
```

```
    always_ff @(posedge clk) begin
        // If downcounting has reached 0 or the value is manually loaded
        // reset the timer
        if (load | overflow) downcount <= M - 1;

        // If the timer is enabled, start downcounting
        else if (enable) downcount <= downcount - 1;
    end
    always_ff @(posedge clk) begin
        // Output an overflow flag when downcounting has ended
        if (downcount == 0) overflow = 1;
        else overflow = 0;
    end
endmodule

module counter(input logic clk, load, enable, [31:0]numOfCounts,
                    output logic overflow, [31:0]count
    );
    always_ff @(posedge clk) begin
        // If downcounting has reached numOfCounts or the value is manually
loaded
        // reset the counter
        if (load | overflow) count <= 0;

        // If the counter is enabled, start counting
        else if (enable) count <= count + 1;
    end
    always_ff @(posedge clk) begin
        // Output an overflow flag when counting has ended
```

```
        if (count == numOfCounts - 1) overflow = 1;
        else overflow = 0;
    end
endmodule
```

SystemVerilog codes for Master and Slave Processes

```
module master_process( input logic clk, load, enable,
                                output logic masterLed, overflowFlag
    );
    logic tc_20ms;
    logic[31:0]dummyCount;
    logic[31:0]M = 2000000;
    logic[31:0]numOfCount = 50;

    timer T20ms(clk, load, enable, M, tc_20ms);
    counter Cmaster(tc_20ms, load, enable, numOfCount, overflowFlag,
dummyCount);

    always_ff@(posedge tc_20ms, negedge enable) begin
        if (!enable) masterLed <= 0;
        else if (overflowFlag) masterLed <= ~masterLed;
        else masterLed <= masterLed;
    end
endmodule

module slave_process( input logic clk, load, enable,
                                output logic slaveLed, overflowFlag
    );
    logic tc_20ms;
    logic[31:0]dummyCount;
```

```
    logic[31:0]M = 2000000;
    logic[31:0]numOfCount = 25;

    timer T20ms(clk, load, enable, M, tc_20ms);
    counter Cslave(tc_20ms, load, enable, numOfCount, overflowFlag,
dummyCount);

    always_ff@(posedge tc_20ms, negedge enable) begin
        if (!enable) slaveLed <= 0;
        else if (overflowFlag) slaveLed <= ~slaveLed;
        else slaveLed <= slaveLed;
    end
endmodule
```

SystemVerilog code for master_slave Module

```
module master_slave( input logic clk, master_switch, slave_switch, enable_m,
enable_s, rst,
                     output logic master_mode_led, slave_mode_led,
master_led, slave_led,
                     s0, s1, s2, s3, s4, s5, s6, dp, [3:0]an
    );
    logic overflowFlag_master;
    logic overflowFlag_slave;

    assign master_mode_led = master_switch | ~master_switch & ~slave_switch &
                             enable_m;

    assign slave_mode_led = ~master_switch & slave_switch | ~master_switch &
                            ~slave_switch & ~enable_m & enable_s;
```

```
    master_process master(clk, rst, master_mode_led, master_led,
                                    overflowFlag_master);

    slave_process slave(clk, rst, slave_mode_led, slave_led, overflowFlag_slave);


    logic [15:0]count_master, count_slave;

    register reg_master(master_mode_led, overflowFlag_master, rst |
                                ~master_mode_led, count_master + 1, count_master);

    register reg_slave(slave_mode_led, overflowFlag_slave, rst | ~slave_mode_led,
                                count_slave + 1, count_slave);

    logic [15:0]count;
    always_comb begin
        if (master_mode_led & ~slave_mode_led) begin
            count <= count_master;
        end
        else if (~master_mode_led & slave_mode_led) begin
            count <= count_slave;
        end
        else count <= 0;
    end

    SevSeg_4digit display(clk, count[3:0], count[7:4], count[11:8], count[15:12], s0,
                                s1, s2, s3, s4, s5, s6, dp, an);
endmodule
```

Testbench for master_slave Module

```
// Testbench for the master_slave module

module master_slaveTest();

	logic clk, master_switch, slave_switch, enable_m, enable_s, rst,
                                    master_mode_led, slave_mode_led, master_led,
                                     slave_led, s0, s1, s2, s3, s4, s5, s6, dp;

	logic [3:0]an;

	master_slave modeUUT( clk, master_switch, slave_switch, enable_m, enable_s,
                                     rst, master_mode_led, slave_mode_led, master_led,
                                     slave_led, s0, s1, s2, s3, s4, s5, s6, dp, an);

	initial begin

		clk = 0; master_switch = 0; slave_switch = 0; enable_m = 0; enable_s = 0;
rst = 0; #1;

		repeat(2) begin

			repeat(2) begin

				repeat(250) begin

					clk = ~clk; #1;

				end

				master_switch = ~master_switch; #1;

			end

			slave_switch = ~slave_switch; #1;

		end

	$stop;

	end

endmodule
```