```
/* =========================[ countdown.sv ]================================== */

/* This module is used for shutter counter
 *
 * Set the counter timer and enable signal to set the start time
 * each clock will tick down once
 * when the timer hits zero, the timer will stop
 * and the output signal "done" will turn HIGH
 *
 * Input signal RESET will reset the counter to its initial state
 */
module countdown(
        input logic clk,
        input logic reset,

        input logic [31:0] wait_cycles,
        input logic write_enable,
        input logic countdown_enable,

        output logic done
);

        // internal flip flop to hold the countdown number
        logic [31:0] count;

        // sequential logic
        always_ff @(posedge clk) begin
                if (reset) begin
                        count <= 0;
                end else if (write_enable) begin
                        count <= wait_cycles - 1;
                end else if (countdown_enable) begin

                        // Decrement count as long as it's not 0
                        if (count != 0) begin
                                count <= count - 1;
                        end
                end else begin
                        count <= count;
                end
        end

        // Output logic
        assign done = (count == 0);
endmodule

/* =========================[ decoders.sv ]========================== */

/* The higher the setting (fstop), the smaller the light */
/* Therefore we need to subtract from maximum setting */
module aperture_decoder(
        input logic [2:0] input_setting,
        output logic [7:0] output_multiplier
);
        assign output_multiplier = 1 << (7 - input_setting);
endmodule

/* For shutter speed, the higher the setting */
/* the faster the shutter speed */
/* Therefore we need to subtract from the maximum setting */
module shutter_decoder(
        input logic [2:0] input_setting,
        output logic [31:0] shutter_wait_time
);
        parameter BASE_WAIT_CYCLE = 2;
        assign shutter_wait_time = BASE_WAIT_CYCLE << (7 - input_setting);
endmodule

/* =========================[ dff.sv ]========================== */

module dff(d, q, en, clk, rst);
        input logic d;
        input logic en, clk, rst;
        output logic q;

        always_ff @(posedge clk, posedge rst) begin
```

```
                q <= rst ? 0 : (en ? d : q);
        end
endmodule

module dffs(d, q, en, clk, rst);
        parameter WIDTH = 32;

        input logic [WIDTH-1:0] d;
        input logic en, clk, rst;
        output logic [WIDTH-1:0] q;

        always_ff @(posedge clk, posedge rst) begin
                q <= rst ? 0 : (en ? d : q);
        end
endmodule

/* ==========================[ dff.sv ]========================== */
/*
 * Here is the main FSM module
 */

module fsm(
        input logic clk,
        input logic reset,

        input logic power_btn,

        input logic mode_inc,
        input logic mode_dec,
        input logic fstop_inc,
        input logic fstop_dec,
        input logic shutter_inc,
        input logic shutter_dec,

        input logic shutter_btn,

        input logic [15:0] sensor_data,

        output logic [15:0] output_data,
        output logic output_data_valid
);

        /* Power button FF async */
        logic power_on;
        always_ff @(posedge power_btn, posedge reset) begin
                if (reset) begin
                        power_on <= 0;
                end else begin
                        power_on <= ~power_on;
                end
        end

        /* Countdown module required for waiting on shutter (based on shutter speed) */
        /* Let SCD stand for shutter-countdown */
        logic scd_reset, scd_wr_en, scd_cd_en;
        logic [31:0] scd_cycles;
        logic scd_done;

        countdown COUNTDOWN_MODULE(
                .clk(clk), .reset(scd_reset), .wait_cycles(scd_cycles),
                .write_enable(scd_wr_en), .countdown_enable(scd_cd_en),
                .done(scd_done)
        );

        /* DFF to contain memory for aperture and shutter settings */
        /* There are 8 aperture and shutter settings, so only 3 bits is required */
        logic [2:0] next_aperture_setting, aperture_setting;
        logic aperture_setting_en;
        dffs #(3) APERTURE_SETTING_FF(
                .d(next_aperture_setting), .q(aperture_setting),
                .en(aperture_setting_en), .clk(clk), .rst(reset)
        );

        logic [2:0] next_shutter_setting, shutter_setting;
        logic shutter_setting_en;
        dffs #(3) SHUTTER_SETTING_FF(
```

```
		.d(next_shutter_setting), .q(shutter_setting),
		.en(shutter_setting_en), .clk(clk), .rst(reset)
	);

	/* Decoder to take aperture/shutter settings and turn them */
	/* into usable values */
	logic [7:0] aperture_multiplier;
	aperture_decoder FSTOP_DECODER(
		.input_setting(aperture_setting),
		.output_multiplier(aperture_multiplier)
	);
	shutter_decoder SHUTTER_DECODER(
		.input_setting(shutter_setting),
		.shutter_wait_time(scd_cycles)
	);

	/* State definitions */
	/* Since we're not using more than 32 states, 5 bit width should be more than enough */
	enum logic[4:0] {
		ST_IDLE,		/* [1]  The idle / reset state */

		ST_APERTURE_PRIORITY,	/* [2]  The default home state for aperture priority mode */
		ST_SHUTTER_PRIORITY,	/* [3]  The default home state for shutter priority mode */
		ST_MANUAL,		/* [4]  The default home state for manual exposure mode */

		ST_INC_FSTOP,		/* [5]  State for increasing f-stop number */
		ST_DEC_FSTOP,		/* [6]  State for decreasing f-stop number */
		ST_INC_SHUTTER,		/* [7]  State for increasing shutter speed */
		ST_DEC_SHUTTER,		/* [8]  State for decreasing shutter speed */

		ST_CALC_SHUTTER,	/* [9]  Intermediate state for aperture priority to caluclate shutter speed needed
*/
		ST_CALC_APERTURE,	/* [10] Intermediate state for shutter speed priority to caluclate aperture needed
*/

		ST_WAIT_SHUTTER,	/* [11] State for waiting for the shutter to open and close */
		ST_DONE			/* [12] state for outputting shutter */
	} current_state, next_state, prev_mode_state;

	/* Save the camera operating mode state */
	logic prev_mode_en;
	always_ff @(posedge clk, posedge reset) begin
		if (reset) begin
			prev_mode_state <= ST_APERTURE_PRIORITY;
		end else if (prev_mode_en) begin
			prev_mode_state <= current_state;
		end else begin
			prev_mode_state <= prev_mode_state;
		end
	end

	/* Next state combinational logic */
	always_comb begin

		// Default (cover all combinational cases)
		next_state = current_state;

		case (current_state)

			ST_IDLE: next_state <= power_on == 1 ? prev_mode_state : ST_IDLE;

			ST_APERTURE_PRIORITY: begin
				if (~power_on)		next_state <= ST_IDLE;
				else if (mode_inc)	next_state <= ST_SHUTTER_PRIORITY;
				else if (mode_dec)	next_state <= ST_MANUAL;
				else if (fstop_inc & (aperture_setting != 3'b111))
								next_state <= ST_INC_FSTOP;
				else if (fstop_dec & (aperture_setting != 3'b000))
								next_state <= ST_DEC_FSTOP;
				else if (shutter_btn)	next_state <= ST_CALC_SHUTTER; /* We need to calculate what
shutter we need because the user sets the aperture */
			end

			ST_SHUTTER_PRIORITY: begin
				if (~power_on)		next_state <= ST_IDLE;
				else if (mode_inc)	next_state <= ST_MANUAL;
```

```
                        else if (mode_dec)      next_state <= ST_APERTURE_PRIORITY;
                        else if (shutter_inc & (shutter_setting != 3'b111))
                                                                next_state <= ST_INC_SHUTTER;
                        else if (shutter_dec & (shutter_setting != 3'b000))
                                                                next_state <= ST_DEC_SHUTTER;
                        else if (shutter_btn)   next_state <= ST_CALC_APERTURE;
                end

                ST_MANUAL: begin
                        if (~power_on)          next_state <= ST_IDLE;
                        else if (mode_inc)      next_state <= ST_APERTURE_PRIORITY;
                        else if (mode_dec)      next_state <= ST_SHUTTER_PRIORITY;
                        else if ((fstop_inc) & (aperture_setting != 3'b111))
                                                                next_state <= ST_INC_FSTOP;
                        else if ((fstop_dec) & (aperture_setting != 3'b000))
                                                                next_state <= ST_DEC_FSTOP;
                        else if (shutter_inc & (shutter_setting != 3'b111))
                                                                next_state <= ST_INC_SHUTTER;
                        else if (shutter_dec & (shutter_setting != 3'b000))
                                                                next_state <= ST_DEC_SHUTTER;
                        else if (shutter_btn)   next_state <= ST_WAIT_SHUTTER;
                end

                ST_INC_FSTOP: next_state <= prev_mode_state;
                ST_DEC_FSTOP: next_state <= prev_mode_state;
                ST_INC_SHUTTER: next_state <= prev_mode_state;
                ST_DEC_SHUTTER: next_state <= prev_mode_state;

                ST_CALC_APERTURE: next_state <= ST_WAIT_SHUTTER;
                ST_CALC_SHUTTER: next_state <= ST_WAIT_SHUTTER;

                ST_WAIT_SHUTTER: next_state <= scd_done ? ST_DONE : ST_WAIT_SHUTTER;
                ST_DONE: next_state <= prev_mode_state;
        endcase
end

/* State sequential logic */
always_ff @(posedge clk, negedge reset) begin
        if (reset) begin
                current_state <= ST_IDLE;
        end else begin
                current_state <= next_state;
        end
end

/* FSM states to module connection logic */
always_comb begin

        // Reset all output to 0 (to cover all combinations)
        // All assignments are non-blocking on purpose
        scd_reset <= 0;
        prev_mode_en <= 0;
        scd_wr_en <= 0;
        scd_cd_en <= 0;
        next_aperture_setting <= 0;
        aperture_setting_en <= 0;
        next_shutter_setting <= 0;
        shutter_setting_en <= 0;

        // Turn on specific signals to overwrite default settings
        if (current_state == ST_IDLE) begin
                scd_reset <= 1;
        end

        if (current_state == ST_APERTURE_PRIORITY) begin
                prev_mode_en <= 1;
                // scd_wr_en <= 1;

                shutter_setting_en <= 1;
                next_shutter_setting <= 3'b111 - aperture_setting;
        end

        if (current_state == ST_SHUTTER_PRIORITY) begin
                prev_mode_en <= 1;
                scd_wr_en <= 1;
```

```
                        aperture_setting_en <= 1;
                        next_aperture_setting <= 3'b111 - shutter_setting;
                end

                if (current_state == ST_MANUAL) begin
                        prev_mode_en <= 1;
                        scd_wr_en <= 1;
                end

                if (current_state == ST_INC_FSTOP) begin
                        aperture_setting_en <= 1;
                        next_aperture_setting <= aperture_setting + 1;
                end

                if (current_state == ST_DEC_FSTOP) begin
                        aperture_setting_en <= 1;
                        next_aperture_setting <= aperture_setting - 1;
                end

                if (current_state == ST_INC_SHUTTER) begin
                        shutter_setting_en <= 1;
                        next_shutter_setting <= shutter_setting + 1;
                end

                if (current_state == ST_DEC_SHUTTER) begin
                        shutter_setting_en <= 1;
                        next_shutter_setting <= shutter_setting - 1;
                end

                if (current_state == ST_CALC_APERTURE) begin
                        /* nothing */
                end

                if (current_state == ST_CALC_SHUTTER) begin
                        scd_wr_en <= 1;
                end

                if (current_state == ST_WAIT_SHUTTER) begin
                        scd_cd_en <= 1;
                end
        end

        /* Output combination logic */
        always_comb begin
                // Reset all output to 0 to cover all comb cases
                output_data = 0;
                output_data_valid = 0;

                // State based output
                if (current_state == ST_DONE) begin
                        output_data = sensor_data * aperture_multiplier * scd_cycles;
                        output_data_valid = 1;
                end
        end

endmodule

/* =========================[ testbench_fsm.sv ]========================== */

module test_fsm();
        logic clk;
        logic reset;
        logic power_btn;
        logic mode_inc;
        logic mode_dec;
        logic fstop_inc;
        logic fstop_dec;
        logic shutter_inc;
        logic shutter_dec;
        logic shutter_btn;
        logic [15:0] sensor_data;

        logic [15:0] fsm_output;
        logic fsm_output_valid;

        // For simulation debugging
```

```
logic [799:0] current_test;

fsm DUT(
        .output_data(fsm_output),
        .output_data_valid(fsm_output_valid),
        .*
);

// Clock generator
always #2 clk = ~clk;

initial begin
        // Initial values
        clk = 1;
        reset = 0;
        power_btn = 0;
        mode_inc = 0;
        mode_dec = 0;
        fstop_inc = 0;
        fstop_dec = 0;
        shutter_inc = 0;
        shutter_dec = 0;
        shutter_btn = 0;
        sensor_data = 0;

        // Offset signals by half clock to avoid confusion
        #3;

        /* TEST: reset */
        current_test = "reset";
        reset = 1;
        #4;
        reset = 0;
        #20;
        assert (DUT.current_state == DUT.ST_IDLE);
        assert (DUT.prev_mode_state == DUT.ST_APERTURE_PRIORITY);
        // $stop;

        /* TEST: power button */
        /* we should see the state go from idle to aperture priority */
        current_test = "power button 1";
        power_btn = 1;
        #4;
        power_btn = 0;
        #20;
        assert (DUT.current_state == DUT.ST_APERTURE_PRIORITY);
        // $stop;

        /* TEST: power button 2 */
        /* Pressing power button again gets us back to idle */
        current_test = "power button 2";
        power_btn = 1;
        #4;
        power_btn = 0;
        #20;
        assert (DUT.current_state == DUT.ST_IDLE);
        // $stop;

        /* TEST: IDLE state should ignore inputs */
        current_test = "idle ignore input";
        mode_inc = 1;
        #4;
        assert (DUT.current_state == DUT.ST_IDLE);
        mode_inc = 0;
        mode_dec = 1;
        #4;
        assert (DUT.current_state == DUT.ST_IDLE);
        mode_dec = 0;
        fstop_inc = 1;
        #4;
        assert (DUT.current_state == DUT.ST_IDLE);
        fstop_inc = 0;
        fstop_dec = 1;
        #4;
        assert (DUT.current_state == DUT.ST_IDLE);
        fstop_dec = 0;
```

```
shutter_inc = 1;
#4;
assert (DUT.current_state == DUT.ST_IDLE);
shutter_inc = 0;
shutter_dec = 1;
#4;
assert (DUT.current_state == DUT.ST_IDLE);
shutter_dec = 0;
shutter_btn = 1;
#4;
assert (DUT.current_state == DUT.ST_IDLE);
shutter_btn = 0;
#20;
assert (DUT.current_state == DUT.ST_IDLE);
// $stop;

/* TEST: mode increment */
current_test = "mode increment";
power_btn = 1;
#4;
assert (DUT.current_state == DUT.ST_APERTURE_PRIORITY);
power_btn = 0;
#4;
mode_inc = 1;
#20;
mode_inc = 0;
#20;
assert (DUT.current_state == DUT.ST_MANUAL);
// $stop;

/* TEST: mode decrement */
current_test = "mode decrement";
mode_dec = 1;
#20;
mode_dec = 0;
#20;
assert (DUT.current_state == DUT.ST_APERTURE_PRIORITY);
// $stop;

/* TEST: Aperture increment test */
current_test = "aperture inc test";
fstop_inc = 1;
#80;
fstop_inc = 0;
#20;
assert (DUT.aperture_setting == 3'b111);
// $stop;

/* TEST: Aperture decrement test */
current_test = "aperture dec test";
fstop_dec = 1;
#80;
fstop_dec = 0;
#20;
assert (DUT.aperture_setting == 3'b000);
// $stop;

/* TEST: Shutter increment in aperture mode test (nothing should happen) */
current_test = "ignore shutter inc test";
shutter_inc = 1;
#20;
shutter_inc = 0;
#20;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
// $stop;

/* TEST: Shutter decrement in aperture mode test (nothing should happen) */
current_test = "ignore shutter dec test";
shutter_dec = 1;
#20;
shutter_dec = 0;
#20;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
// $stop;
```

```
/* TEST: switching to shutter priority */
current_test = "switching to shutter priority";
mode_inc = 1;
#4;
mode_inc = 0;
#4;
assert (DUT.current_state == DUT.ST_SHUTTER_PRIORITY);

/* TEST: Shutter decrement test */
current_test = "shutter dec test";
shutter_dec = 1;
#80;
shutter_dec = 0;
#20;
assert (DUT.shutter_setting == 3'b000);
// $stop;

/* TEST: Shutter increment test */
current_test = "shutter inc test";
shutter_inc = 1;
#80;
shutter_inc = 0;
#20;
assert (DUT.shutter_setting == 3'b111);
// $stop;

/* TEST: Aperture increment in shutter mode test (nothing should happen) */
current_test = "ignore aperture inc test";
fstop_inc = 1;
#20;
fstop_inc = 0;
#20;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
// $stop;

/* TEST: Aperture decrement in shutter mode test (nothing should happen) */
current_test = "ignore aperture dec test";
fstop_dec = 1;
#20;
fstop_dec = 0;
#20;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
// $stop;

/* TEST: Switch to Manual mode */
current_test = "switch to manual mode test";
mode_inc = 1;
#4;
mode_inc = 0;
#4;
assert (DUT.current_state == DUT.ST_MANUAL);

/* TEST: set every setting to 000 */
current_test = "set aperture & shutter to 0";
shutter_dec = 1;
#80;
shutter_dec = 0;
#4;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b000);

/* TEST: manul mode test */
current_test = "manual mode test";
fstop_inc = 1;
#4;
fstop_inc = 0;
#4;
shutter_inc = 1;
#4;
shutter_inc = 0;
#4;
assert (DUT.aperture_setting == 3'b001);
assert (DUT.shutter_setting == 3'b001);
```

```
fstop_inc = 1;
#4;
fstop_inc = 0;
#4;
shutter_inc = 1;
#4;
shutter_inc = 0;
#4;
assert (DUT.aperture_setting == 3'b010);
assert (DUT.shutter_setting == 3'b010);
fstop_dec = 1;
#40;
fstop_dec = 0;
#4;
shutter_inc = 1;
#40;
shutter_inc = 0;
#4;
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
#20;
// $stop;

/* TEST: taking a manual picture 1: highest setting (highest fstop, fastest shutter, darkest picture) */
current_test = "manual photo 1";
fstop_inc = 1;
#64;
fstop_inc = 0;
assert (DUT.current_state == DUT.ST_MANUAL);
assert (DUT.aperture_setting == 3'b111);
assert (DUT.shutter_setting == 3'b111);
shutter_btn = 1;
sensor_data = 42;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid); /* Continue waiting until we see output signal posedge */
#3;
assert (fsm_output == 84);
#8;
// $stop;

/* TEST: manual photo 2: lowest fstop fastest shutter */
/* input data is 100, output should be 2,560 */
current_test = "manual photo 2";
fstop_dec = 1;
#64;
fstop_dec = 0;
assert (DUT.current_state == DUT.ST_MANUAL);
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
shutter_btn = 1;
sensor_data = 100;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 25600);
#8;
// $stop;

/* TEST: manual photo 2: lowest fstop slowest shutter */
/* input data is 1, Output should be */
current_test = "manual photo 3";
shutter_dec = 1;
#64;
shutter_dec = 0;
assert (DUT.current_state == DUT.ST_MANUAL);
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b000);
shutter_btn = 1;
sensor_data = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 32768); /* 256 * 128 * 1 = 32768 */
```

```
#8;
// $stop;

/* TEST: burst mode */
/* Setting shutter speed to the fastest */
current_test = "manual 10-burst";
shutter_inc = 1;
#64;
shutter_inc = 0;
assert (DUT.current_state == DUT.ST_MANUAL);
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
shutter_btn = 1;
for (int i = 1; i <= 10; i = i + 1) begin
        #4;
        sensor_data = i;
        @(posedge fsm_output_valid);
        #3;
        assert (fsm_output == (i * 2 * 128));
end
shutter_btn = 0;
#8;
// $stop;

/* TEST: power off and on */
current_test = "power off and on";
power_btn = 1;
#8;
power_btn = 0;
#8;
power_btn = 1;
#8;
power_btn = 0;
#8;
assert (DUT.current_state != DUT.ST_IDLE);
#8;
// $stop;

/* TEST: Aperture priority picture 1: min-f-stop */
current_test = "aperture priority 1: smallest fstop";
mode_inc = 1;
#4;
mode_inc = 0;
assert (DUT.aperture_setting == 3'b000);
shutter_btn = 1;
sensor_data = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 256); /* 128 (ap) * 2 (shutter cycle) * 1 (sensor) = 256 */
#8;
// $stop;

/* TEST: Aperture priority picture 2: small-f-stop */
current_test = "aperture priority 2: small fstop";
fstop_inc = 1;
#4;
fstop_inc = 0;
#4;
assert (DUT.aperture_setting == 3'b001);
shutter_btn = 1;
sensor_data = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 256); /* 64 (ap) * 4 (shutter cycle) * 1 (sensor) = 256 */
#8;
// $stop;

/* TEST: Aperture priority picture 3: medium-f-stop */
current_test = "aperture priority 3: medium fstop";
fstop_inc = 1;
#12;
fstop_inc = 0;
```

```
#4;
assert (DUT.aperture_setting == 3'b011);
shutter_btn = 1;
sensor_data = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 256); /* 16 (ap) * 16 (shutter cycle) * 1 (sensor) = 256 */
#8;
// $stop;

/* TEST: Aperture priority picture 3: larget-f-stop */
current_test = "aperture priority 4: largest fstop";
fstop_inc = 1;
#40;
fstop_inc = 0;
#4;
assert (DUT.aperture_setting == 3'b111);
shutter_btn = 1;
sensor_data = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 256); /* 1 (ap) * 256 (shutter cycle) * 1 (sensor) = 256 */
#8;
// $stop;

/* TEST: dial aperture back to largest f-stop */
current_test = "reset largest aperture";
fstop_dec = 1;
#80;
fstop_dec = 0;
assert (DUT.aperture_setting == 3'b000);

/* TEST: burst mode aperture priority */
current_test = "aperture priority 10-burst";
assert (DUT.current_state == DUT.ST_APERTURE_PRIORITY);
assert (DUT.aperture_setting == 3'b000);
assert (DUT.shutter_setting == 3'b111);
shutter_btn = 1;
for (int i = 1; i <= 10; i = i + 1) begin
        #4;
        sensor_data = i;
        @(posedge fsm_output_valid);
        #3;
        assert (fsm_output == (i * 2 * 128)); /* 2 cycles * 128 aperture mult */
end
shutter_btn = 0;
#8;

/* TEST: switch mode to shutter priority */
current_test = "switch to shutter priority";
mode_inc = 1;
#4;
mode_inc = 0;
#4;
assert (DUT.current_state == DUT.ST_SHUTTER_PRIORITY);

/* TEST: fastest shutter speed */
current_test = "shutter priority 1: fastest";
sensor_data = 1;
assert (DUT.shutter_setting == 3'b111);
shutter_btn = 1;
#4;
shutter_btn = 0;
@(posedge fsm_output_valid);
#3;
assert (fsm_output == 256);
#8;

/* TEST: fast shutter speed */
current_test = "shutter priority 2: fast";
sensor_data = 1;
shutter_dec = 1;
```

```
                #8;
                shutter_dec = 0;
                assert (DUT.shutter_setting == 3'b110);
                shutter_btn = 1;
                #4;
                shutter_btn = 0;
                @(posedge fsm_output_valid);
                #3;
                assert (fsm_output == 256);
                #8;

                /* TEST: fast shutter speed burst */
                current_test = "shutter priority 3: fast burst";
                assert (DUT.shutter_setting == 3'b110);
                assert (DUT.aperture_setting == 3'b001);
                shutter_btn = 1;
                for (int i = 1; i <= 10; i = i + 1) begin
                        #4;
                        sensor_data = i;
                        @(posedge fsm_output_valid);
                        #3;
                        assert (fsm_output == (i * 4 * 64)); /* 4 cycles * 64 aperture mult */
                end
                shutter_btn = 0;
                #8;

                /* TEST: slowest shutter speed */
                current_test = "shutter priority 4: slowest";
                sensor_data = 1;
                shutter_dec = 1;
                #60;
                shutter_dec = 0;
                assert (DUT.shutter_setting == 3'b000);
                assert (DUT.aperture_setting == 3'b111);
                shutter_btn = 1;
                #4;
                shutter_btn = 0;
                @(posedge fsm_output_valid);
                #3;
                assert (fsm_output == 256);
                #8;

                /* TEST: Power off */
                current_test = "power off";
                power_btn = 1;
                #4;
                power_btn = 0;
                #8;
                assert (DUT.current_state == DUT.ST_IDLE);

                /* TEST: reset */
                current_test = "reset";
                reset = 1;
                #8;
                reset = 0;
                #20;

                $stop;
        end
endmodule
```