/\* ==========================[ 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