module smartphone\_automation(

input wire clk, // System clock

input wire reset, // System reset

input wire [2:0] command, // Command from smartphone/tablet (3-bit for simplicity)

output reg light\_control, // Output control for lights

output reg heating\_control, // Output control for heating

output reg security\_control // Output control for security

);

// Command Definitions

parameter CMD\_LIGHT\_ON = 3'b001; // Command to turn on the light

parameter CMD\_LIGHT\_OFF = 3'b010; // Command to turn off the light

parameter CMD\_HEAT\_ON = 3'b011; // Command to turn on heating

parameter CMD\_HEAT\_OFF = 3'b100; // Command to turn off heating

parameter CMD\_SECURITY\_ON = 3'b101; // Command to arm security system

parameter CMD\_SECURITY\_OFF = 3'b110; // Command to disarm security system

// State Definitions

typedef enum logic [1:0] {

IDLE = 2'b00,

LIGHT = 2'b01,

HEATING = 2'b10,

SECURITY = 2'b11

} state\_t;

state\_t current\_state, next\_state;

// Sequential Logic for State Transitions

always @(posedge clk or posedge reset) begin

if (reset) begin

current\_state <= IDLE;

end else begin

current\_state <= next\_state;

end

end

// Combinational Logic for Next State

always @(\*) begin

case (current\_state)

IDLE: begin

case (command)

CMD\_LIGHT\_ON, CMD\_LIGHT\_OFF: next\_state = LIGHT;

CMD\_HEAT\_ON, CMD\_HEAT\_OFF: next\_state = HEATING;

CMD\_SECURITY\_ON, CMD\_SECURITY\_OFF: next\_state = SECURITY;

default: next\_state = IDLE;

endcase

end

LIGHT: begin

if (command == CMD\_LIGHT\_ON || command == CMD\_LIGHT\_OFF) begin

next\_state = IDLE;

end else begin

next\_state = LIGHT;

end

end

HEATING: begin

if (command == CMD\_HEAT\_ON || command == CMD\_HEAT\_OFF) begin

next\_state = IDLE;

end else begin

next\_state = HEATING;

end

end

SECURITY: begin

if (command == CMD\_SECURITY\_ON || command == CMD\_SECURITY\_OFF) begin

next\_state = IDLE;

end else begin

next\_state = SECURITY;

end

end

default: next\_state = IDLE;

endcase

end

// Output Logic for Controlling Devices

always @(posedge clk or posedge reset) begin

if (reset) begin

light\_control <= 1'b0;

heating\_control <= 1'b0;

security\_control <= 1'b0;

end else begin

case (next\_state)

LIGHT: begin

if (command == CMD\_LIGHT\_ON) begin

light\_control <= 1'b1;

end else if (command == CMD\_LIGHT\_OFF) begin

light\_control <= 1'b0;

end

end

HEATING: begin

if (command == CMD\_HEAT\_ON) begin

heating\_control <= 1'b1;

end else if (command == CMD\_HEAT\_OFF) begin

heating\_control <= 1'b0;

end

end

SECURITY: begin

if (command == CMD\_SECURITY\_ON) begin

security\_control <= 1'b1;

end else if (command == CMD\_SECURITY\_OFF) begin

security\_control <= 1'b0;

end

end

default: begin

light\_control <= light\_control;

heating\_control <= heating\_control;

security\_control <= security\_control;

end

endcase

end

end

endmodule

// Test Bench

module tb\_smartphone\_automation;

reg clk;

reg reset;

reg [2:0] command;

wire light\_control;

wire heating\_control;

wire security\_control;

// Instantiate the module

smartphone\_automation uut (

.clk(clk),

.reset(reset),

.command(command),

.light\_control(light\_control),

.heating\_control(heating\_control),

.security\_control(security\_control)

);

// Clock generation

initial clk = 0;

always #5 clk = ~clk; // 10 ns clock period

// Test sequence

initial begin

// Initialize inputs

reset = 1;

command = 3'b000;

#20;

// Release reset

reset = 0;

#20;

// Test light control on

command = 3'b001;

#20;

// Test light control off

command = 3'b010;

#20;

// Test heating control on

command = 3'b011;

#20;

// Test heating control off

command = 3'b100;

#20;

// Test security control on

command = 3'b101;

#20;

// Test security control off

command = 3'b110;

#20;

// End simulation

$stop;

end

endmodule