Australian National University

ASSIGNMENT COVER SHEET

ANU College of Engineering, Computing and Cybernetics
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au
+61 2 6125 5254

Student 1 ID	U6474528		
Student 1 Name	Zefan Wu		
Student 2 ID	U7607296		
Student 2 Name	Max Georg Wierse		
Course Code	ENGN4213/6213		
Course Name	Digital Systems and Microprocessors		
Assignment Item	☒ FPGA Project Report	☐ Micro-controller Project Report	
Word Count	5333	Due Date	21/04/2023
Date Submitted	21/04/2023	Extension Granted	

Initials

For group assignments,
each student must initial.

Australian
National
University

DECLARATION OF CONTRIBUTION

We, the undersigned members of ENGN4213/6213 FPGA Project Group No:___15__, hereby declare that we have contributed approximately equally to this project and agree to receive the same marks for the project report assessment.

Signature 1 with name____ Zefan Wu ____ Date__16/04/2023__

Signature 2 with name _____ [illegible] _______ Date__17/04/2023__

If any member does not agree to this declaration of equal contribution, the group must email the course co-conveners (A/Prof. Nan Yang and A/Prof. Xiangyun Zhou) and tutors (Shaoheng Xu and Zhifeng Tang) before 14th April, 2023 stating your individual contributions and percentage of contribution. Such statements may be considered during the assessment of the FPGA project report.

Australian
National
University

Smart-Home Controller using a Basys3 FPGA Development Board

FPGA Project Group: [15]

[Zefan Wu], [u6474528]
[Max Georg Wierse], [u7607296]

ENGN[4213/6213] Digital Systems and Microprocessors
Semester 1, 2023

ANU College of Engineering, Computing and Cybernetics
The Australian National University

Table of Contents

Australian
National
University

1 INTRODUCTION

1.1 Background Information

The purpose of this project is to design and implement a smart home controller using an FPGA development board. The controller will manage two main functions: Secure Garage Door Control and Climate Control (Home Heating and Cooling). The project aims to demonstrate the ability to consider real-world problems, use creativity to generate specifications, and apply digital design and Verilog knowledge to implement a solution using available resources.

The scope of the project includes designing, implementing, and testing two separate modules for the smart home functions, then integrating them into a single, hierarchical design. The objectives of the project include:

1. Designing and implementing a secure garage door control module with features such as pin-code security, alarm system, and collision detection.
2. Designing and implementing a climate control module for home heating and cooling, considering current and desired temperatures.
3. Integrating the two modules into a single smart home controller capable of handling both functions simultaneously.

1.2 Report Outline

The report begins with an introduction that presents the project, its purpose, scope, and objectives. The resource requirements section follows, detailing the hardware, software, and other resources necessary for the project's implementation. Next, the design methodology section describes the design process for the two modules and the integrated controller. Design considerations are addressed in a separate section, focusing on constraints and important aspects of the project. The report concludes with results and discussion, where findings are presented, and potential improvements are explored. Additionally, references and appendices are provided, including Verilog code and other relevant information.

2 RESOURCE REQUIREMENTS

2.1 Hardware

- Basys3 FPGA Development Board (part number: xc7a35tcpg236-1)
- 16 switches and 5 push buttons on the Basys3 board to mimic the inputs of sensors or users.
- 16 LEDs and 4 Seven-Segment Displays (SSDs) to show the outputs.

2.2 Software

- Xilinx Vivado (Version 2022.1) - for synthesizing, implementing, and programming the FPGA board with the designed Verilog code.

3 DESIGN METHODOLOGY

3.1 Function I: Secure Garage Door Control

3.1.1 Assumptions

1. Default the garage door is closed.
2. The correct pin code is set 15 (group number).
3. The alarm reset pin code is set 13.
4. The door can be closed without the pin.
5. The alarm is immediately disabled when the correct alarm pin is set. (Note that this means, that there won't be an alarm if the alarm pin is used to open the door. The door stays closed)
6. Seven segments displays (SSDs) show the status of the door only when the door is fully opened or closed.
7. The collision sensor is only considered when the door is not fully closed.
8. When attempting to close the door with the collision sensor activated, the first LED will blink to show that it was tried to close the door (but immediately opens due to collision).

3.1.2 Inputs and Outputs

Please refer to the annotated image of the board (Appendix [II.1]) for comparison.

Inputs:

1. The up-push button (T18) serves as the control for opening and closing the garage door (GARAGE_CONTROL).
2. Six switches (from right to left, starting with the second switch V16, W16, W17, W15, V15 and W14) used to input the binary equivalent of the Project Group Number as the pin code for opening the garage door. These switches are also used to enter the alarm pin code to turn off the alarm.

3. The down-push button (U17) that resets the alarm when an incorrect pin is entered (ALARM_RESET).
4. Right-most switch (V17) that detects if the garage door collides with an obstacle while closing (COLLISION_SENSOR).

Outputs:

1. LD0 to LD3: Door status - 4 LEDs that indicate the status of the garage door:
    a. All 4 LEDs ON: Door is closed.
    b. LEDs turn OFF one by one at every 1 second: Door is opening.
    c. All 4 LEDs OFF: Door is completely open.
    d. LEDs turn ON one by one at every 1 second: Door is closing.
    e. All 4 LEDs blinking every 0.5 seconds: Alarm is triggered due to incorrect pin entry.
2. Two SSDs (right to left, third and fourth one): indicate the status of the garage door.
    a. "DO": Door Open.
    b. "DC": Door Close.
    c. "CL": Collision Triggered.
    d. "AL": Alarm Triggered.

3.1.3 FSM Design

For this problem, it is better to use a **Moore finite state** machine (FSM) instead of a Mealy FSM because it involves interfacing with external components, such as sensors and display units, which require a fixed timing relationship between the inputs and outputs. Moore FSMs provide a more predictable output timing since the outputs depend only on the current state and not on the inputs.

Here is a brief description of each state according to the state transition diagram [II.2]:

1. ALARM: Alarm state, triggered when an incorrect pin is entered.
2. CLOSED: Garage door is closed.
3. OPEN: Garage door is open.

4-6. OPENING_1, OPENING_2, and OPENING_3: Three states representing the different stages of the garage door opening process.

7-9. STOP_OPENING_1, STOP_OPENING_2, and STOP_OPENING_3: Three states representing the garage door being stopped during the opening process.

10-12. CLOSING_1, CLOSING_2, and CLOSING_3: Three states representing the different. stages of the garage door closing process.

13-14. STOP_CLOSING_1, STOP_CLOSING_2, and STOP_CLOSING_3: Three states representing the garage door being stopped during the closing process.

The garage door system has 14 states, which represent various stages in the process of opening and closing the garage door, as well as the alarm state. The code provided contains three sections: Next State Logic [I.2], SSD Value Logic [I.3.1], and LED Output Logic [I.3.2]. The Next State Logic block [I.2] handles the state transitions, the SSD Value Logic block [I.3.1] updates the values for the seven-segment display, and the Output Logic block [I.3.2] sets the LED values depending on the current state.

3.1.4 Module Hierarchy

garage_FSM_top (Top-level module)

The top-level module (Appendix [II.3]) connects various sub-modules and handles input/output (I/O) signals. The main sub-modules in this module are:

a. garage_FSM (FSM for garage door control)

   This is the core module for the garage door control. It implements the Finite State Machine (FSM) for controlling the garage door's various states and transitions between those states. The module processes the input signals, such as garage control button, alarm reset button, collision sensor, and switch inputs. Based on the input signals, the module drives the outputs, such as SSD values, LEDs, and door open signal.

b. debouncer (Input debouncing for push buttons)

   These sub-modules are used for debouncing the push-button / switch inputs (GARAGE_CONTROL, ALARM_RESET, SW [6:0], IDLE_RESET). Debouncing is necessary to prevent false triggering due to mechanical bouncing of the push buttons. The debouncer module filters out the noise in the input signal and provides a clean signal for the FSM.

c. spot (Edge detection for push buttons)

   This sub-module is used for edge detection of push buttons (GARAGE_CONTROL and ALARM_RESET). The spot module detects a rising edge in the debounced input signal, which indicates that the button has been pressed. The output of the spot module is then fed into the garage_FSM module.

Block Diagram Interconnections:

The garage_FSM_top module connects the sub-modules as follows:

- The input signals (GARAGE_CONTROL, ALARM_RESET, COLLISION_SENSOR, and SW) are connected to the corresponding inputs of the garage_FSM module.
- The debouncer modules are connected to the push button inputs (GARAGE_CONTROL, ALARM_RESET, and COLLISION_SENSOR) and the debounced outputs are connected to the spot modules.
- The spot modules are connected to the debounced signals (db_GARAGE_CONTROL, db_ALARM_RESET, and db_COLLISION_SENSOR) and their outputs are connected to the garage_FSM module as spot_GARAGE_CONTROL, spot_ALARM_RESET and spot_ COLLISION_SENSOR.
- The outputs of the garage_FSM module (ssdValueOne, ssdValueTwo, LED, and doorOpen)

are connected to the corresponding outputs of the garage_FSM_top module.

3.1.5 Implementation

The Current State Register and Transition Block (Appendix [I.1]):
This module updates at every clock cycle, loading the contents of the 'next_state' into the 'state' register. This ensures that the system's current state transitions to the appropriate next state based on the input conditions and previous state.

The Nextstate Logic (Appendix [I.2]):
This module follows the state transition diagram strictly. The next state is determined by the current state and input conditions (GARAGE_CONTROL, COLLISION_SENSOR, ALARM_RESET, correct_pin, alarm_pin, second_beat). This module is responsible for controlling the state transitions of the system.

The Output Logic (Appendix [I.3]):
This module is responsible for determining the output values based on the current state. It is divided into two parts:

1. **SSD (Appendix [I.3.1])**
   This part of the module defines the Seven Segment Display (SSD) output values based on the current state. It displays different values for different states, making it easier for users to understand the current state of the system.
2. **LED (Appendix [I.3.2])**
   This part of the module controls the LED output based on the current state. The LED output serves as a visual indicator of the current state, making it easier for users to identify the system's status.

Since the debouncer and spot modules were taken from the previous labs of this course these have been tested via simulations. None of the other parts have been tested via simulations.

Australian National University

3.2 Function II: Climate Control (Home Heating and Cooling)

3.2.1 Assumptions

1. The outside temperature serves as an input, and the initial room temperature is set equal to the outside temperature.
2. The temperature range is from 20 to 27 degrees, providing an 8-degree span.

3.2.2 Inputs and Outputs

Please refer to the annotated image of the board (Appendix [II.1]) for comparison.
Inputs:

1. A switch (seventh from the right: T3) to turn the climate control ON/OFF.
2. Three switches (first, second, and third from the right: R2, T1 and U1) for inputting the outside temperature.
3. Three switches (fourth, fifth, and sixth from the right: W2, R3 and T2) for inputting the desired temperature.
4. A switch (eighth from the right: V2) to turn the fan ON/OFF.

Note: Temperature input from three switches can be interpreted as an integer between 0 and 7 such that 0 indicates 20 degrees and 7 indicates 27 degrees.

Outputs:

1. Two SSDs to display the current room temperature (between 20 to 27).

3.2.3 FSM Design

Here is a brief description of each state according to the state transition diagram [II.2] for the climate control system:

1. OFF: The climate control system is not active.
2. IDLE: The system is ON but not actively heating or cooling.
3. COOLING: The system is actively cooling the room.
4. HEATING: The system is actively heating the room.

The climate control system has four states, which represent various stages of operation. The code provided contains several sections: Next State Logic [I.5], Output Logic [I.6] and State Transition Logic [I.4].

Next State Logic: Handles the state transitions based on the input signals and the current state.

Output Logic: Updates the current room temperature based on the state, heartbeat signals, and input conditions such as the fan_speed and doorOpen signals. Also, it includes SSD Value Logic, which assigns the values to be displayed on the Seven-Segment Display (ssdValueOne and ssdValueTwo).

State Transition Logic: Updates the current state based on the next state determined by the Next State Logic.

Current Temperature Logic: This component is responsible for updating the current temperature based on the system state and the desired temperature. Although it shares similarities with the next state logic, separating these two logics improves code readability and maintains a more straightforward structure.

3.2.4 Module Hierarchy

AC_FSM_top (Top-level module)

The top-level module connects various sub-modules and handles input/output (I/O) signals. The main sub-modules in this module are (Appendix [II.5)):

a. AC_FSM (FSM for climate control)

   This is the core module for the climate control system. It implements the Finite State Machine (FSM) for controlling the air conditioning system's various states and transitions between those states. The module processes the input signals, such as switch inputs, fan speed, and door open signal. Based on the input signals, the module drives the outputs, such as ssdValueOne and ssdValueTwo.

b. Debouncer (Input debouncing for switches)

   These sub-modules are used for debouncing the switch inputs (SW [0] to SW [7]). Debouncing is necessary to prevent false triggering due to mechanical bouncing of the switches. The debouncer module filters out the noise in the input signal and provides a clean signal for the FSM.

Block Diagram Interconnections:

The AC_FSM_top module connects the sub-modules as follows:

- The input signals (clk, reset, SW, and doorOpen) are connected to the corresponding inputs of the AC_FSM module.
- The debouncer modules are connected to the switch inputs (SW [0] to SW [7]) and the debounced outputs are connected to the AC_FSM module as db_SW[0] to db_SW[7].
- The outputs of the AC_FSM module (ssdValueOne and ssdValueTwo) are connected to the corresponding outputs of the AC_FSM_top module.

3.2.5 Implementation

The AC_FSM_top_minimum and AC_FSM_minimum modules work together to implement a climate control system that regulates the room temperature according to the desired temperature set by the user.

The top-level module, AC_FSM_top_minimum, handles input/output signals, debouncing of the switch inputs, and instantiates the AC_FSM_minimum module.

The AC_FSM_minimum module uses a Finite State Machine (FSM) with four states (OFF, IDLE, COOLING, and HEATING) to control the room temperature based on the desired temperature and the outside temperature. The room temperature is updated every 0.5, 1, 2, or 3 seconds, depending on the state of the air conditioning system, the fan speed, and whether the garage door is open. The desired and outside temperature values are updated every second and three seconds, respectively, through flip-flops. This ensures that the system only updates the temperature values when appropriate, to circumvent unwanted behavior.

The inner workings of the design allow it to behave as required by using heartbeat generators (clockDividerHB modules) to create timing signals that update the current, desired, and outside temperatures. The use of debouncers for switch inputs ensures that clean signals are fed into the FSM, preventing false triggering due to mechanical bouncing. Spot modules detect the edge of on/off switch signals, allowing the synchronization of the heartbeat generators with the on/off switch of the AC.

3.3 Integrated Design: Smart-home Controller

3.3.1 Module Hierarchy

The TOP_BOTH module (Appendix [II.6]) integrates the functionalities of the air conditioning (AC) system and the garage system into a single module. The design connects the input switches for both systems, combines the outputs for the seven-segment displays, and shares common input signals like the clock and reset.

Integrated Design of Two Functions:

1. Garage System (garage_FSM_top) – Section see 3.1: The garage_FSM_top module manages the garage door's opening and closing, as well as the security alarm. It takes input from switches for garage control and alarm reset. The garage_FSM_top module also outputs the values to be displayed on the third and fourth seven-segment displays (ssdValueThree and ssdValueFour) and controls the doorOpen signal, which is fed into the AC_FSM_top module.
2. Air Conditioning System (AC_FSM_top) – Section 3.2: The AC_FSM_top module is responsible for controlling the temperature of a room based on the desired temperature and outside temperature. It uses an FSM (Finite State Machine) to regulate the room temperature and update it accordingly. The module takes input from switches for on/off, fan speed, desired temperature, and outside temperature. It also receives a doorOpen signal from the garage system. The AC_FSM_top module outputs the values to be displayed on

the first and second seven-segment displays (ssdValueOne and ssdValueTwo).
The integrated design first separates the input switch signals for the AC and garage systems by assigning AC_SW [9:0] and garage_SW [6:0] to the respective portions of the SW [15:0] input. Then, the design instantiates both the AC_FSM_top and garage_FSM_top modules, connecting the respective input and output signals.

The seven-segment display values (ssdValueOne, ssdValueTwo, ssdValueThree, and ssdValueFour) from both the AC and garage systems are connected to a ssdTop module, which is responsible for driving the seven-segment displays. The ssdTop module takes the input values from both systems and generates the corresponding ssdAnode and ssdCathode outputs to control the displays.

By combining the AC and garage systems into the TOP_BOTH module, the design achieves a higher level of integration, allowing for more efficient resource utilization and simplified system management.

3.3.2 User Interface

According to the Basys3 board with markings of the inputs and outputs (Appendix [II.1]).

Inputs:

1. Three switches (1st, 2nd and 3rd from the right): for inputting the outside temperature.
2. Three switches (4th, 5th and 6th from the right): for inputting the desired temperature.
3. A switch (7th from the right) to turn the climate control ON/OFF.
4. A switch (8th from the right) to turn the fan ON/OFF.
5. The up-push button that serves as the control for opening and closing the garage door.
6. The down-push button that resets the alarm when an incorrect pin is entered.
7. The right-push button that resets the control function to an idle closed-door state.
8. A switch (1st from the left) that detects if the garage door collides with an obstacle while closing.
9. Six switches (2nd, 3rd, 4th, 5th, 6th and 7th from the left) used to input the binary equivalent of the Project Group Number as the pin code for opening the garage door, also alarm pin code will stop blinking the LEDs.

Outputs:

1. LEDs: LD0 to LD3 indicate the status of the garage door.
2. SSDs:
   a) Two SSDs (1st and 2nd) to display the current room temperature (between 20 to 27).
   b) Two SSDs (3rd and 4th) to indicate that the status of the garage door.

Only the global reset push button input (the left one) is exclusive to the top module. This input serves to reset both the air conditioning and garage door systems, returning them to their initial states.

4 DESIGN CONSIDERATIONS

The project uses the default clock frequency of 100 MHz since there was no reason to change that. All Button inputs have a debouncer connected to it and a "spot" module to synchronize the input with the rest of the system. The only exception to that is the "global reset" button since it is used to reset the debouncers as well and that would have introduced further complexities.

Both functions are implemented using an FSM since that is the preferred way that was taught in this course. One way of optimizing the behavior of an FSM is to represent the states via one-hot encoding or gray encoding to prevent glitches. For both FSMs this would have meant to add more bits to the state variable which would use more hardware resources and adds more complexity to the project. That is why both FSMs do not use any of those encodings.

Both functions were developed on their own and then combined via a top module. The problem with that was that both functions use two SSDs each. That means that each function had their own SSD driver. Since the SSDs of the Basys 3 board must be updated in a very specific way it is easier to do that with one central driver than with two separated ones. For that purpose, the top module of the project includes an SSD driver and the function modules only output the values that should be displayed.

5 RESULTS & DISCUSSION

Overall, our design performed very good. In the hardware presentation everything worked as expected. The biggest strength of our design lies in the fact that both functions use an FSM as a basis. This made it very easy to structure the code in a good and readable way. This also allowed us to derive a lot of information from the current state of the machine (e.g., what status to display for the garage, go up or down after stopping the garage, etc.).

Furthermore, we thought from the beginning about the combination of both functions. Except for the SSD drivers we just had to organize the switches and buttons which made the combination very simple. There are several smaller weaknesses though. The first one is that the garage door function uses a lot of states (14). This made the programming easier, but at the same time it uses a lot of the hardware resources. Additionally, to that both FSMs do not implement any specific encodings for the states (grey or one-hot) which is a further weakness.

Another weakness is that we did not perform any simulations on our modules but relied on intensive manual testing. This worked out for the scenarios of the hardware demo but does not guarantee correctness for all scenarios.

We also had some challenges in the development of the design. The first one was about the way to synchronize the input of the temperatures for the second function. We did clarify that with a Tutor in one of the Lab sessions. Another challenge was the way of handling the “stop” of the garage door opening/closing. In the beginning we tried to represent that with one state which was a bit tedious. After some brainstorming we introduced more states to make it easier.

One improvement to our design would be to add gray encoding to the FSMs and maybe rethink the states that were used for the “stopping” of the garage door. Furthermore, the synchronization of the temperature input for the second function could be designed in such a way that also the first update takes exactly x seconds (and not a bit more/less). Another improvement would be to use multiple testbenches to verify the correctness of the design.

One main learning outcome was that FSMs are great! For specific types of problems, a well-designed FSM really simplifies the implementation. We spent a good amount of time on thinking about the design of the FSMs. This invests in time really paid off in the implementation phase of the project by simplifying the implementation and reducing the behavioral bugs of the design.

References

List of sources cited in the report.

[1] J. Wakerly, *Digital Design: Principles and Practices*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.

[2] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Hoboken, NJ, USA: Wiley, 2007.

[3] Inc., D. (2015) Basys 3 artix-7 FPGA trainer board: Recommended for introductory users, Flickr. Yahoo! Available at: https://www.flickr.com/photos/127815101@N07/16340067591/ (Accessed: April 21, 2023).

Appendix I: Verilog Code

Secure Garage Door Control

I.1 The Current State Register and Transition Block:

```
    always @(posedge clk) begin
        if (reset) state <= CLOSED;
        else state <= next_state;
    end
```

I.2 The Next State Logic:

```
    always @(*) begin
        case (state)
            CLOSED:
                if (GARAGE_CONTROL) begin
                    if (correct_pin == SW) next_state = OPENING_1;
                    else next_state = ALARM;
                end
                else next_state = CLOSED;
            OPENING_1:
                if (GARAGE_CONTROL) next_state = STOP_OPENING_1;
                else if (second_beat) next_state = OPENING_2;
                else next_state = OPENING_1;
            STOP_OPENING_1:
                if (GARAGE_CONTROL) next_state = CLOSING_3;
                else next_state = STOP_OPENING_1;
            OPENING_2:
                if (GARAGE_CONTROL) next_state = STOP_OPENING_2;
                else if (second_beat) next_state = OPENING_3;
                else next_state = OPENING_2;
            STOP_OPENING_2:
                if (GARAGE_CONTROL) next_state = CLOSING_2;
                else next_state = STOP_OPENING_2;
            OPENING_3:
                if (GARAGE_CONTROL) next_state = STOP_OPENING_3;
                else if (second_beat) next_state = OPEN;
                else next_state = OPENING_3;
            STOP_OPENING_3:
                if (GARAGE_CONTROL) next_state = CLOSING_1;
                else next_state = STOP_OPENING_3;
            OPEN:
                if (GARAGE_CONTROL) next_state = CLOSING_1;
                else next_state = OPEN;
            CLOSING_1:
                if (GARAGE_CONTROL) next_state = STOP_CLOSING_1;
                else if (COLLISION_SENSOR) next_state = OPENING_3;
                else if (second_beat) next_state = CLOSING_2;
                else next_state = CLOSING_1;
            STOP_CLOSING_1:
                if (GARAGE_CONTROL) next_state = OPENING_3;
                else next_state = STOP_CLOSING_1;
            CLOSING_2:
                if (GARAGE_CONTROL) next_state = STOP_CLOSING_2;
                else if (COLLISION_SENSOR) next_state = OPENING_2;
                else if (second_beat) next_state = CLOSING_3;
                else next_state = CLOSING_2;
            STOP_CLOSING_2:
                if (GARAGE_CONTROL) next_state = OPENING_2;
                else next_state = STOP_CLOSING_2;
            CLOSING_3:
                if (GARAGE_CONTROL) next_state = STOP_CLOSING_3;
                else if (COLLISION_SENSOR) next_state = OPENING_1;
                else if (second_beat) next_state = CLOSED;
                else next_state = CLOSING_3;
            STOP_CLOSING_3:
                if (GARAGE_CONTROL) next_state = OPENING_1;
                else next_state = STOP_CLOSING_3;
            ALARM:
                if (ALARM_RESET) next_state = CLOSED;
                else if (alarm_pin == SW) next_state = CLOSED;
                else next_state = ALARM;
            default: next_state = CLOSED;
        endcase
    end
```

I.3.1 The Output Logic (SSD):

```
    always @(*) begin
        case (state)
            CLOSED: begin
                ssdValueOne = 4'd12;    // D
                ssdValueTwo = 4'd11;    // C
            end
            CLOSING_1:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            CLOSING_2:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            CLOSING_3:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_CLOSING_1:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_CLOSING_2:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_CLOSING_3:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            OPEN:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd12;    // D
                    ssdValueTwo = 4'd12;    // O
            end
            OPENING_1:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            OPENING_2:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            OPENING_3:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_OPENING_1:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
```

```
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_OPENING_2:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            STOP_OPENING_3:
                if (COLLISION_SENSOR) begin
                    ssdValueOne = 4'd11;    // C
                    ssdValueTwo = 4'd13;    // L
                end else begin
                    ssdValueOne = 4'd14;    // turn off otherwise
                    ssdValueTwo = 4'd14;    // turn off otherwise
                end
            ALARM: begin
                ssdValueOne = 4'd10;    // A
                ssdValueTwo = 4'd13;    // L
            end
            default:
                begin
                ssdValueOne = 4'd14;    // turn off otherwise
                ssdValueTwo = 4'd14;    // turn off otherwise
            end
        endcase
    end
```

I.3.2 The Output Logic (LED):

```
        always @(posedge clk) begin
            case (state)
                CLOSED: LED = 4'b1111;
                OPENING_1: LED = 4'b0111;
                STOP_OPENING_1: LED = 4'b0111;
                OPENING_2: LED = 4'b0011;
                STOP_OPENING_2: LED = 4'b0011;
                OPENING_3: LED = 4'b0001;
                STOP_OPENING_3: LED = 4'b0001;
                OPEN: LED = 4'b0000;
                CLOSING_1: LED = 4'b0001;
                STOP_CLOSING_1: LED = 4'b0001;
                CLOSING_2: LED = 4'b0011;
                STOP_CLOSING_2: LED = 4'b0011;
                CLOSING_3: LED = 4'b0111;
                STOP_CLOSING_3: LED = 4'b0111;
                ALARM: if (half_second_beat) LED = ~LED;
            endcase
        end
```

Climate Control (Home Heating and Cooling)

I.4 The Current State Register and Transition Block:

```
        always @(posedge clk) begin
            if (reset) state <= OFF;
            else state <= next_state;
        end
```

I.5 The Next State Logic:

```
        always @(*) begin
            case (state)
                OFF:
                    if (on_off_switch)
                        next_state = IDLE;
                    else
                        next_state = OFF;
                IDLE:
                    if (!on_off_switch)
                        next_state = OFF;
                    else if (current_temp > desired_temp)
                        next_state = COOLING;
                    else if (current_temp < desired_temp)
                        next_state = HEATING;
                    else
                        next_state = IDLE;
                COOLING:
                    if (!on_off_switch)
                        next_state = OFF;
                    else if (current_temp > desired_temp)
                        next_state = COOLING;
                    else
                        next_state = IDLE;
                HEATING:
                    if (!on_off_switch)
                        next_state = OFF;
                    else if (current_temp < desired_temp)
                        next_state = HEATING;
                    else
                        next_state = IDLE;
            endcase
        end
```

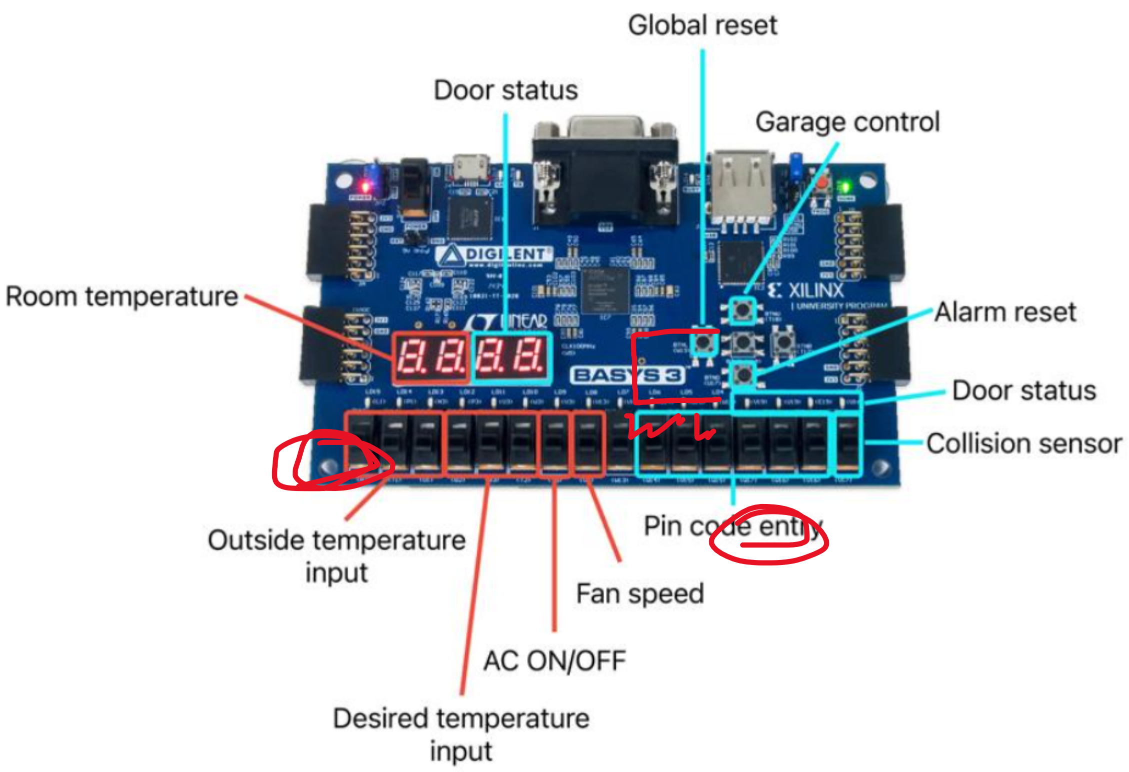
I.6 The Output Logic (SSD):

```
        always @(posedge clk) begin
            if (reset) current_temp <= outside_temp;        // Initial room
 temperature = outside
            else
                case (state)
                    OFF:
                        if (three_second_beat)
                            if (current_temp < outside_temp)
                                current_temp <= current_temp + ONE;
                            else if (current_temp > outside_temp)
                                current temp <= current temp - ONE;
                            else
                                current temp <= current temp;
                    COOLING:
                        if (fan_speed) begin // depending on the fan_speed
value decrease every second/half a second
                            if (doorOpen) begin
                                if (one_second_beat)
                                    current temp <= current temp - ONE;
                            end
                            else if (half_second_beat)
                                current temp <= current temp - ONE;
                        end
                        else if (doorOpen) begin
                                if (two second beat)
                                    current_temp <= current_temp - ONE;
                        end
                        else if (one second beat)
                            current_temp <= current_temp - ONE;
                    HEATING:
                        if (fan_speed) begin// depending on the fan_speed value
decrease every second/half a second
                            if (doorOpen) begin
                                if (one_second_beat)
                                    current temp <= current temp + ONE;
                            end
                            else if (half_second_beat)
                                current temp <= current temp + ONE;
                        end
                        else if (doorOpen) begin
                                if (two second beat)
                                    current_temp <= current_temp + ONE;
                        end
                        else if (one second beat)
                            current_temp <= current_temp + ONE;
                    default: current temp <= current temp;
                endcase
        end
```

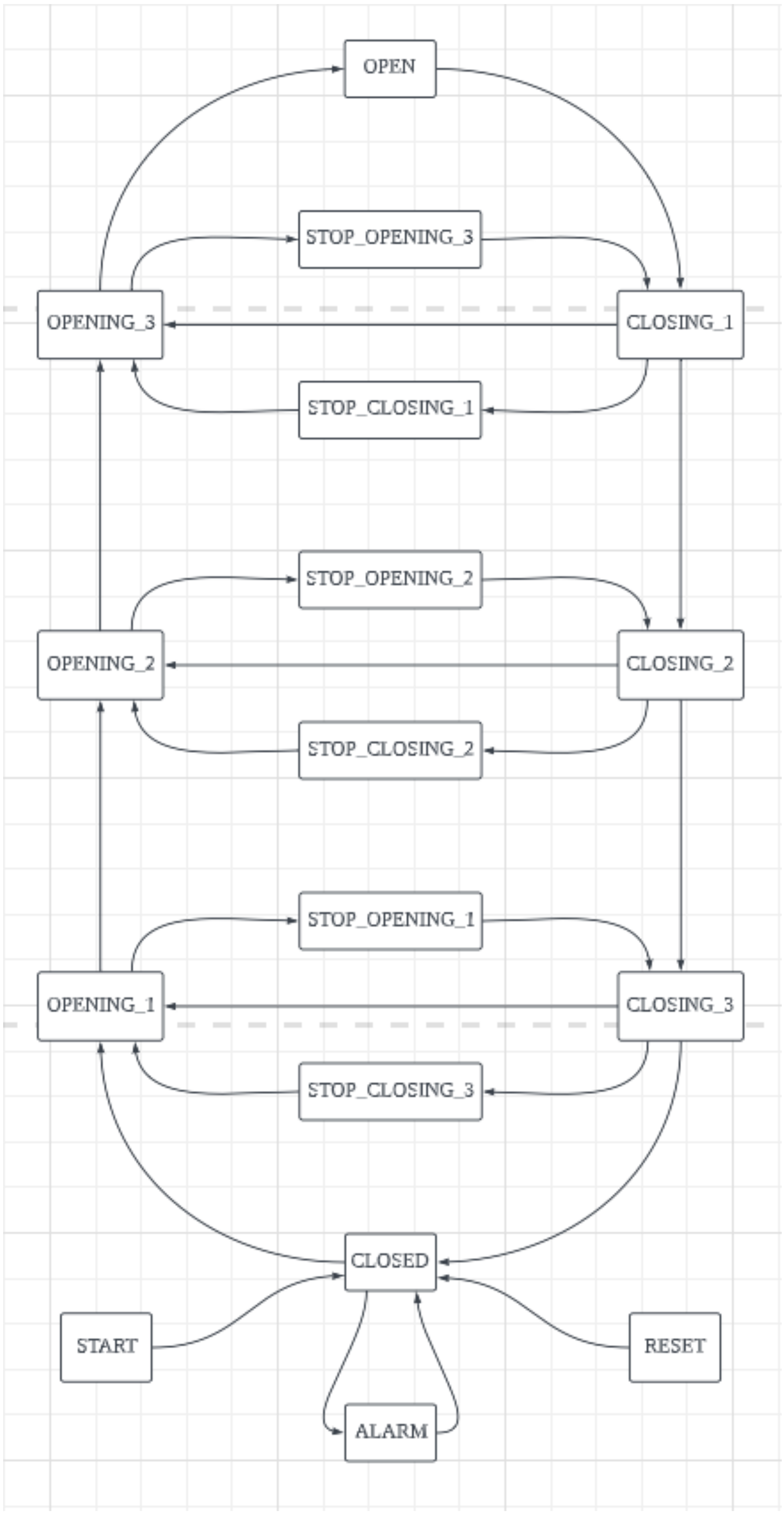
Australian
National
University

Appendix II: Others

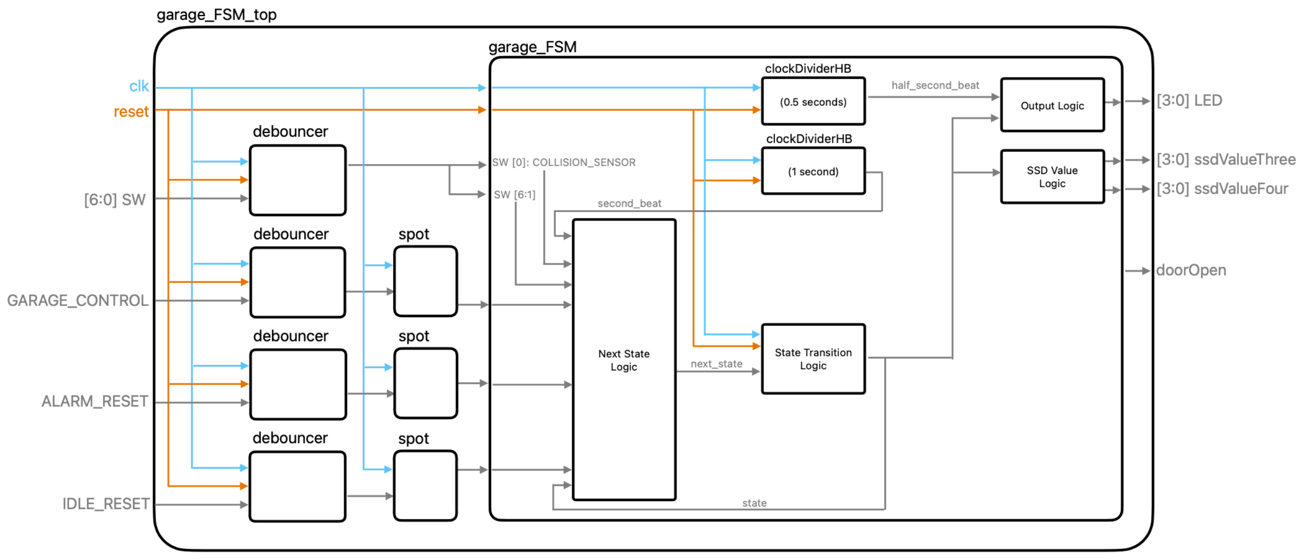
II.1 Input / output user interface:



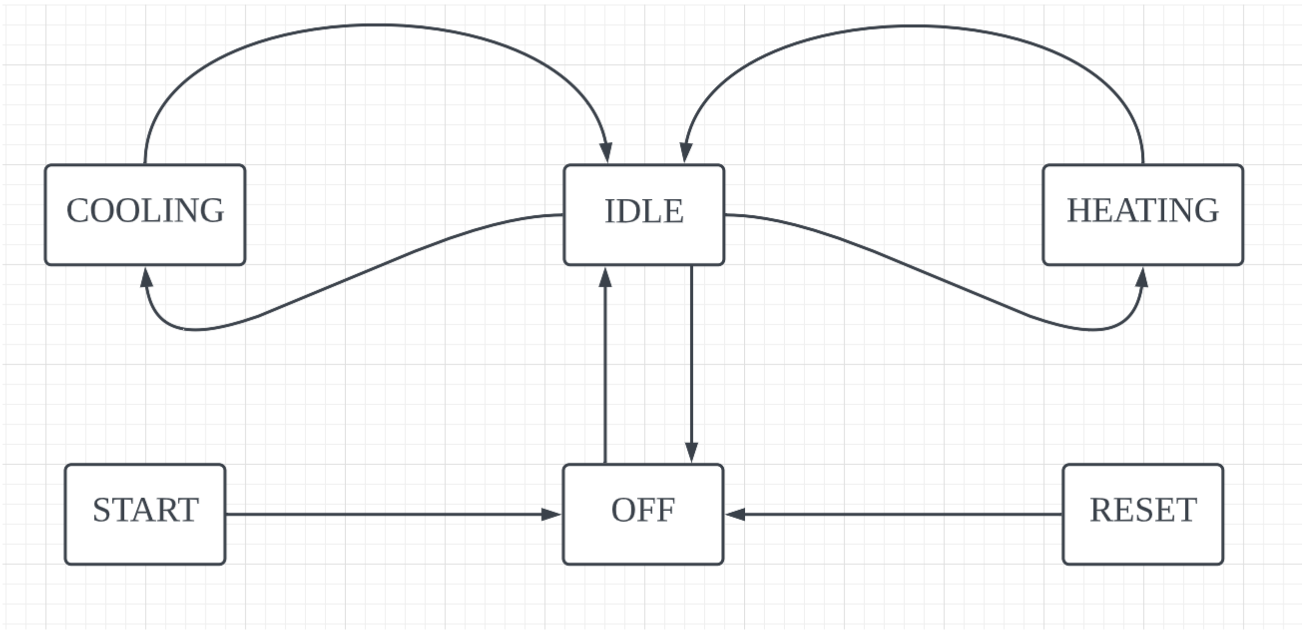
II.2 State Transition Diagram for function 1:



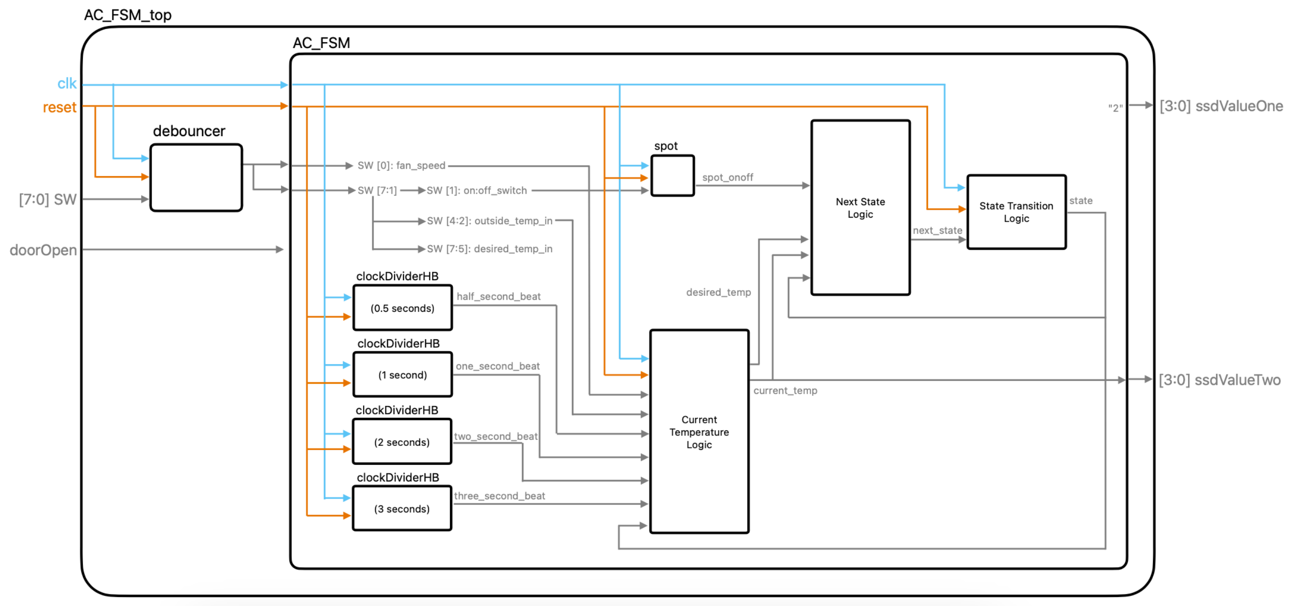
II.3 garage door module block diagram:



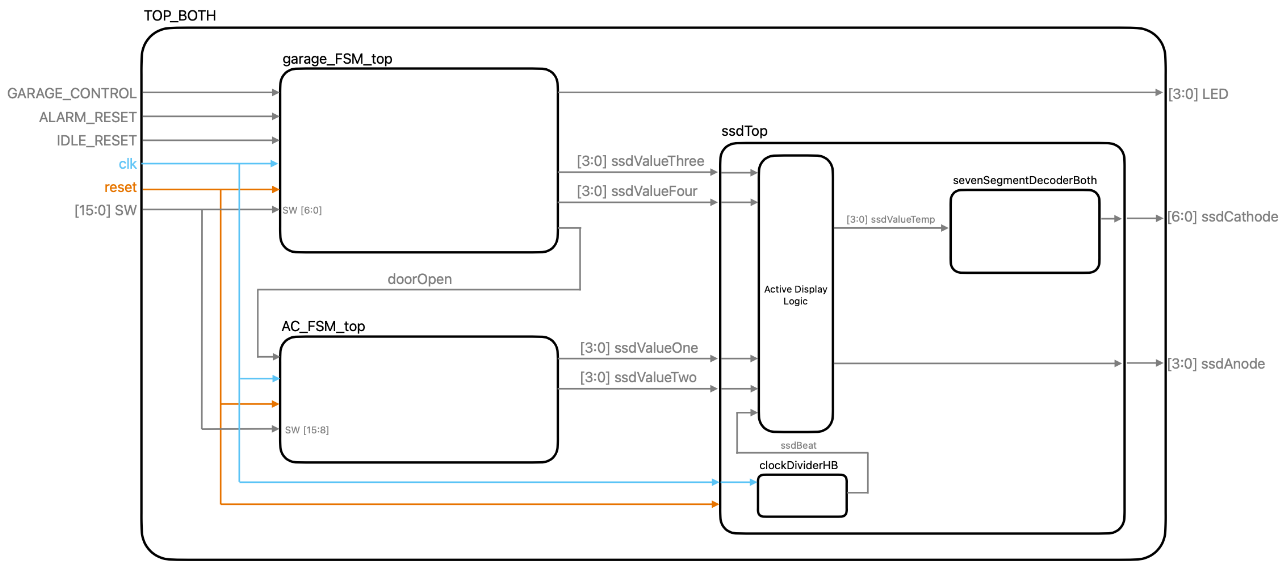
II.4 State Transition Diagram for function 2:



II.5 AC control system block diagram:



II.6 Integrated system block diagram:


TOP_BOTH
GARAGE_CONTROL
ALARM_RESET
IDLE_RESET
clk
reset
[15:0] SW
garage_FSM_top
SW [6:0]
[3:0] LED
ssdTop
[3:0] ssdValueThree
[3:0] ssdValueFour
doorOpen
AC_FSM_top
SW [15:8]
[3:0] ssdValueOne
[3:0] ssdValueTwo
Active Display Logic
[3:0] ssdValueTemp
sevenSegmentDecoderBoth
[6:0] ssdCathode
[3:0] ssdAnode
ssdBeat
clockDividerHB