Digital Logic Final Project Report

Student Name: Lianji Bai SID: 12110201 Student Name: Haoheng Liu SID: 12310627

Team Division of Labor *团队分工*

Work Contribution Description *各成员工作简述*

Lianji Bai

- **Project Architecture Design**: Entire project architecture design.
- **Coding**: State machine implementation (state transitions).
- **Testbench**: State machine testing (state transitions).
- **Project Report**: Entire project report drafting.

Haoheng Liu

- **Project Architecture Design**: Null.
- **Coding**: Transaction Handling Section, for example, output and configuration (parts outside the state machine).
- **Testbench**: Transaction Handling Section, for example, output and configuration (parts outside the state machine).
- **Project Report**: Null.

Contribution Percentage *贡献百分比*

Lianji Bai : Haoheng Liu = 1 : 1

Development Plan Schedule *开发计划日程安排*

Week 1: Planning and Initial Development

- **Day 1–2: Requirement Analysis**
  - Confirm project goals and specifications.
  - Identify functional requirements (state transitions, transaction handling).
  - Create a detailed module breakdown and assign responsibilities.
- **Day 3–4: System Architecture Design**
  - Design overall project architecture.
  - Create block diagrams, state diagrams, and data flow charts.
  - Finalize the design of the state machine and transaction handling logic.
- **Day 5–6: Code Implementation - Part 1**
  - Begin Verilog coding for the state machine (state transitions).
  - Test initial functionality of smaller modules individually.

Week 2: Integration, Testing, and Documentation

- **Day 7–8: Code Implementation - Part 2**
  - Complete coding for transaction handling (output and configuration).
  - Integrate all modules into the main system.
- **Day 9–10: Testbench Development**
  - Develop and verify testbenches for the state machine and transaction handling.
  - Test all edge cases and validate functionality against requirements.
- **Day 11–12: System Optimization**
  - Debug and optimize Verilog code.
  - Address any timing or resource utilization issues.
  - Ensure proper simulation waveforms and results.
- **Day 13: Documentation**
  - Compile project report including architecture, code structure, and test results.
  - Finalize diagrams, screenshots, and data tables for the report.
- **Day 14: Final Review and Submission Preparation**
  - Perform a final system test to ensure reliability.
  - Review and polish the project report.
  - Prepare materials for submission or presentation.

Implementation Status *实施情况*

Due to its complexity and error-prone nature, state machine development took longer than expected. However, the independent division of tasks between the state machine and transaction processing ensures that the project is completed on time.

System Function List *项目功能列表*

> 给出项目功能的简要概述，具体操作方式合并于System Usage Instructions *系统使用说明* 部分，完全参考教师给定项目说明文档。

Core Functions

1. **Power Control**
   - Power on and off via button.
   - Initialization upon power-on.
   - Long-press for power-off.
   - Simulated gesture control for power on/off (bonus).
2. **Mode Switching**
   - Switch between standby, extraction (Level 1, 2, 3), and self-cleaning modes.
   - Return to standby from active modes with specific conditions for each.
3. **Extraction Function**

- Operates at three levels (Level 1, Level 2, Level 3 - Hurricane mode).
- Accumulated working time tracking for extraction modes.
- Special conditions for Hurricane mode:
  - Limited to one use per power cycle.
  - Automatic switch to Level 2 after 60 seconds or forced standby countdown.

4. **Self-Cleaning Function**
   - Accessible only from standby mode.
   - 3-minute countdown for cleaning.
   - Automatic return to standby with reminders after completion.

Auxiliary Functions

1. **Lighting**
   - Toggle lighting in any mode after power-on.
2. **Time Display and Setting**
   - Real-time clock display.
   - Manual time setting in standby mode.
3. **Smart Reminder**
   - Tracks accumulated extraction time.
   - Provides reminders for self-cleaning or manual cleaning.
   - Resets tracking post-cleaning.
4. **Advanced Settings**
   - Configure parameters such as reminder thresholds and gesture control duration.
   - Restore factory defaults.
5. **Query Function**
   - Query accumulated working time and gesture control settings.

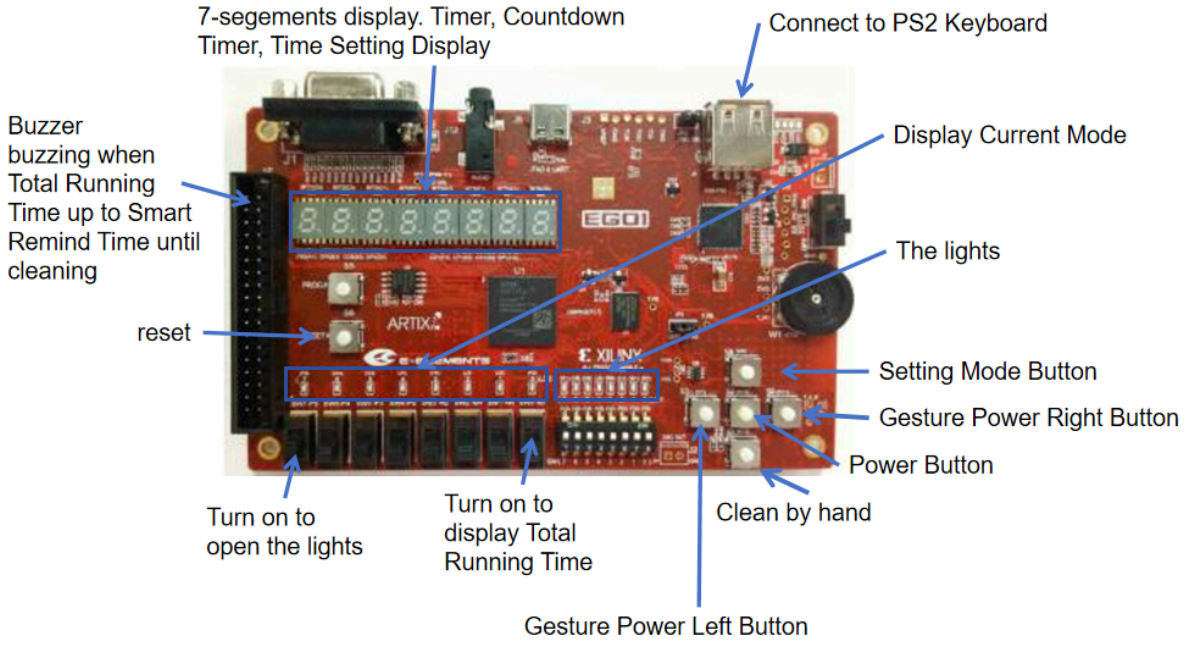
Bonus Features

1. Simulated gesture control for power on/off.
2. Use of alternative input devices (PS2 Keyboard).
3. Use of alternative output devices (A Buzzer).

System Usage Instructions *系统使用说明*

EGO1 Board

Schematic Diagram *示意图*



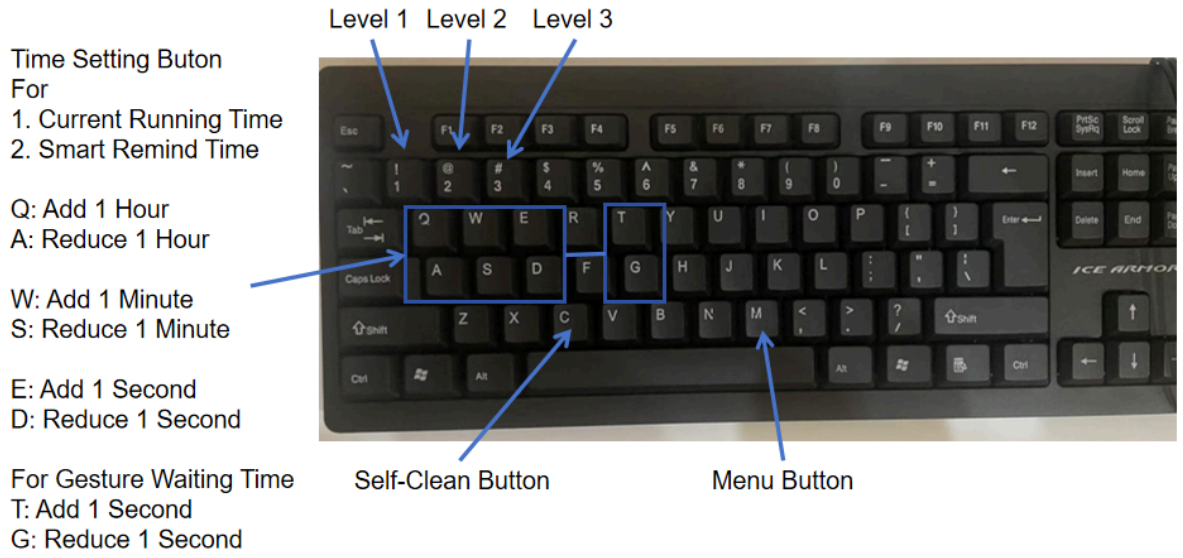
Input and Output Ports Instructions *端口说明*

Port Name	Port PIN	Port Method	Instruction
开关机按键	R15	Input Port	短按开机，长按关机
手势开关机左按键	V1	Input Port	先左后右，开机（默认等待时间5s，可调整）
手势开关机右按键	R11	Input Port	先右后左，关机（默认等待时间5s，可调整）
设置模式触发按键	U4	Input Port	待机模式按下后七段数码管显示智能提醒和手势时间并可供设置。
手动清洁按键	R17	Input Port	按下后累计工作时间清零
照明模式开关	P5	Input Port	上拨时照明功能开启
使用时间显示开关	R1	Input Port	上拨时检查抽油烟机使用时间
恢复出厂设置	P15	Input Port	恢复出厂设置，低电平有效
PS2键盘输入	K5, L4	Input Port	K5: 键盘时钟周期，L4: 键盘输入数据
蜂鸣器	M6, T1	Output Ports	控制蜂鸣器响动
模式显示Led组	Mode Display	Output Ports	显示当前模式[1]

Port Name	Port PIN	Port Method	Instruction
照明光Led组	*Shining Display*	Output Ports	照明功能开启时全亮
七段数码管组	*7-Segments Display*	Output Ports	显示所有 *时间* 元素

PS2 Keyboard

Schematic Diagram *示意图*



Input and Output Ports Instructions *端口说明*

> 键盘输入为单一数据端口，在keyboard.v中进行独热编码将不同键盘键入信号区分。

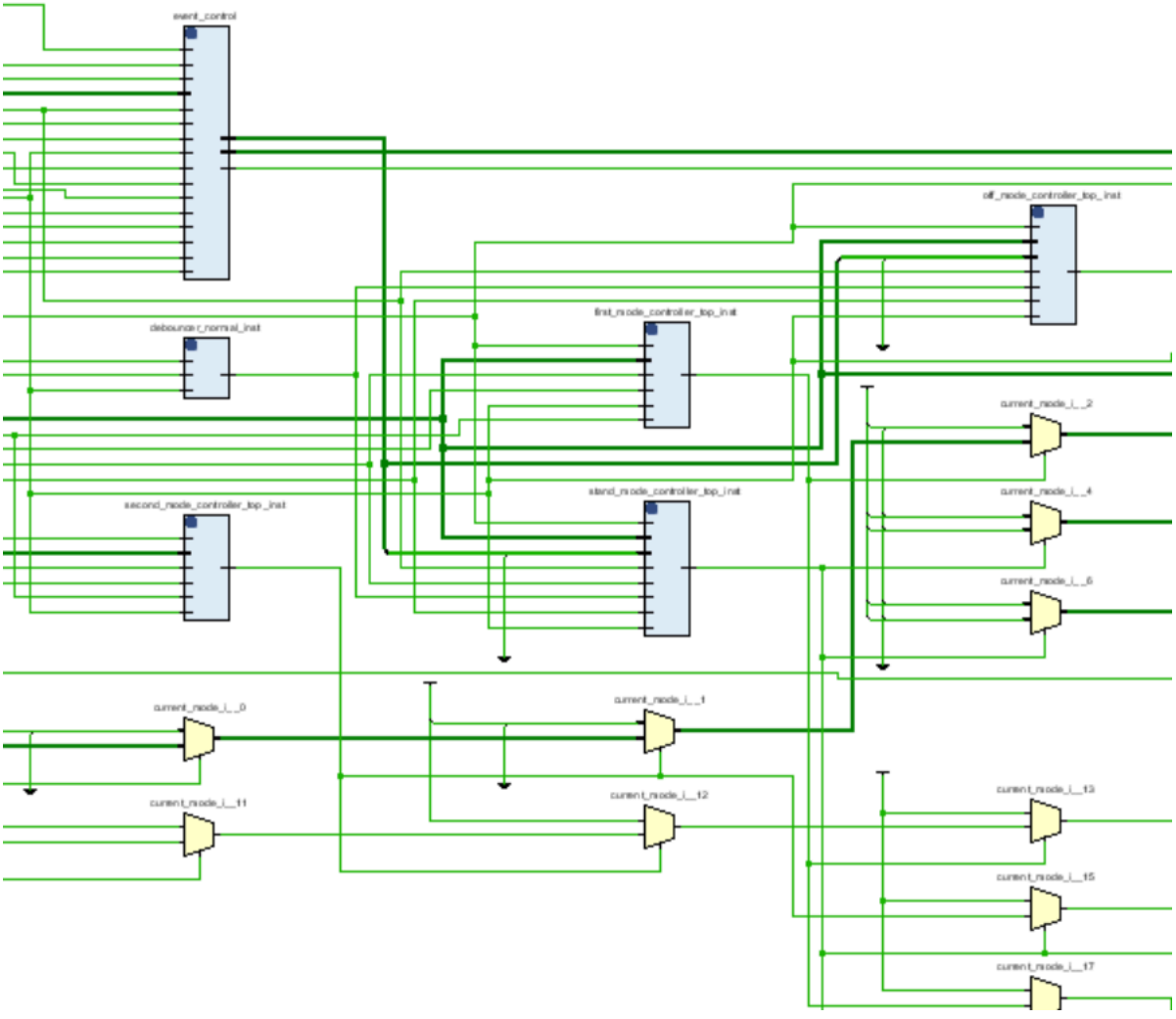
Port Name	Instruction
1	一档功能键
2	二档功能键
3	三档功能键
M	菜单键
C	自清洁键
Q	增加一小时（当前时间、提醒时间）
A	减少一小时（当前时间、提醒时间）
W	增加一分钟（当前时间、提醒时间）
S	减少一分钟（当前时间、提醒时间）
E	增加一秒钟（当前时间、提醒时间）
D	减少一秒钟（当前时间、提醒时间）

Port Name	Instruction
T	增加一秒钟（手势等待时间）
G	减少一秒钟（手势等待时间）

System Structure Description *系统结构说明*

The system is structured into two key components: **state transition controllers** and **event handlers**, each serving a distinct role in managing the kitchen exhaust hood's operation.

> 可以看到，左上角的长模块控制了所有events，和其它状态转移独立开来直接连接到输出。该架构保证了项目的可拓展性和安全性。



1. State Transition Controllers (`controller_top` Modules)

Purpose

Each `controller_top` module is responsible for handling **state transitions** between specific modes. These modules ensure that the system correctly transitions from one mode to another based on user inputs and internal conditions.

Modules and Functionality

1. `off_mode_controller_top`
    - Handles transitions out of the OFF state (e.g., to standby mode) via button presses or gestures.
2. `stand_mode_controller_top`
    - Manages transitions from standby mode to other modes (e.g., first, second, third, clean, or settings).
3. `first_mode_controller_top`
    - Governs transitions between standby mode, Level 1 (first mode), and Level 2 (second mode).
4. `second_mode_controller_top`
    - Controls transitions between standby mode, Level 1, and Level 2 extraction modes.
5. `third_mode_controller_top`
    - Handles transitions to/from standby mode and Level 3 (hurricane mode).
6. `clean_mode_controller_top`
    - Manages transitions between standby and self-cleaning modes.
7. `setting_mode_controller_top`
    - Governs transitions to/from the settings mode for advanced parameter configuration.

---

2. Event Control (`event_control` Module)

Purpose

The `event_control` module is independent of state transitions and focuses on **managing actions within a specific state**. It coordinates timekeeping, display updates, and operational events for each mode, offering high modularity and extensibility.

Modules and Functionality

1. **Mode-Specific Event Modules**
   Each mode has a dedicated event module for handling operations within that state:
    - `clean_mode_event_module`: Tracks self-cleaning time and completion.
    - `first_mode_event_module`: Manages time and events in Level 1.
    - `second_mode_event_module`: Handles Level 2-specific operations.
    - `third_mode_event_module`: Implements hurricane mode timing and output.
    - `stand_mode_event_module`: Tracks time in standby mode and processes gesture inputs.
    - `off_mode_event_module`: Tracks elapsed time when in OFF mode.
2. **Beep and Notification Handling**
    - The `beep_needing_event_module` generates audible alerts based on accumulated operation time and cleaning reminders.
3. **Display Management**

- Updates the `numbers` output to show mode-specific information, such as runtime, countdowns, or settings.
- Converts seconds to an `h:m:s` format for display on 7-segment outputs using the `hms888` function.

4. **Time and Cleaning Management**
   - Maintains total and per-mode running times.
   - Resets running time after cleaning, either automatically (self-cleaning mode) or manually (user-triggered).

---

3. Key Features

State Transition Controllers

- Modular design allows for clear separation of state-specific transition logic.
- Each module independently manages the logic for entering and exiting its assigned state, ensuring clean transitions.

Event Control

- Fully decoupled from state transition logic, making it highly extensible and easy to debug.
- Handles only the functional operations of the current state, including:
  - Time tracking for events.
  - Display and notification outputs.
  - Gesture and button-based interactions.

Interconnection Between Modules

- **Controllers** provide state transition signals to update `current_mode` in the top module.
- **Event handlers** operate based on the `current_mode` signal, ensuring correct functionality for the active state.

Advantages of the Design

1. **Modularity**
   - Clear separation between state transitions (controllers) and operational logic (event control) simplifies debugging and maintenance.
2. **Extensibility**
   - New states or features can be added by implementing additional controller and event modules without affecting existing functionality.
3. **Scalability**
   - The design supports adding complex event logic in specific modes without overloading state transition logic.

This structure ensures robust functionality, high modularity, and ease of debugging, making the system well-suited for real-world applications.

Sub Module Function Description *子模块功能说明*

Detailed Explanation of State Transition Controller Modules

Each `controller_top` module is responsible for managing state transitions into a specific mode. Below is a detailed breakdown of their inputs, outputs, functionality, and design considerations.

---

1. `clean_mode_controller_top`

Inputs

- `clk`, `rstn`: Clock and reset signals for synchronization.
- `current_mode`: Current operating mode of the system.
- `menu_signal`: Signal from the user input to activate the menu.
- `normal_signal`: Signal to toggle into clean mode.

Outputs

- `clean_mode_toggle`: Toggles high to signal the transition into clean mode.

Functionality

- Handles transitions from `STAND_MODE` to `CLEAN_MODE`.
- Internally calls `clean_mode_controller_normal` to determine if a valid toggle condition is met.
- The logic ensures the toggle signal is generated only when the system is in `STAND_MODE`.

---

2. `first_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard synchronization and state signals.
- `menu_signal`: Activates the menu for mode selection.
- `normal_signal`, `second_signal`: User inputs to toggle modes.

Outputs

- `first_mode_toggle`: Toggles high to signal entry into `FIRST_MODE`.

Functionality

- Transitions from `STAND_MODE` or `SECOND_MODE` to `FIRST_MODE`.
- Two submodules:
  - `first_mode_controller_normal`: Handles transitions from standby mode.
  - `first_mode_controller_from_second`: Manages transitions from second mode.

---

3. `off_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `normal_signal`: Button press to turn off the system.
- `left_gesture_signal`, `right_gesture_signal`: Gesture-based toggles.
- `gesture_counter_time`: Tracks gesture timing.

Outputs

- `off_mode_toggle`: Toggles high to indicate transition to OFF mode.

Functionality

- Transitions to `OFF_MODE` from any active mode.
- Submodules:
  - `off_mode_controller_normal`: Handles standard button toggles.
  - `off_mode_controller_gesture`: Processes gesture-based toggles with timing validation.

---

4. `second_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard synchronization and state signals.
- `menu_signal`, `normal_signal`, `first_signal`: User inputs for toggling modes.

Outputs

- `second_mode_toggle`: Toggles high to signal entry into `SECOND_MODE`.

Functionality

- Handles transitions from `STAND_MODE`, `FIRST_MODE`, or `THIRD_MODE` to `SECOND_MODE`.
- Submodules:
  - `second_mode_controller_normal`: Handles entry from standby.
  - `second_mode_controller_from_first`: Manages transitions from first mode.
  - `second_mode_controller_from_third`: Handles transitions from third mode.

---

5. `setting_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `toggle_signal`: Signal to enter settings mode.

Outputs

- `setting_mode_toggle`: Toggles high to signal entry into `SET_MODE`.

Functionality

- Handles transitions from `STAND_MODE` to `SET_MODE`.
- Simple logic within `setting_mode_controller_normal` validates the toggle signal.

---

6. `stand_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `normal_signal`, `menu_signal`: User inputs.
- `left_gesture_signal`, `right_gesture_signal`: Gesture inputs.
- `gesture_counter_time`: Tracks gesture timing.

Outputs

- `stand_mode_toggle`: Toggles high to signal entry into `STAND_MODE`.

Functionality

- Transitions from multiple modes (`OFF_MODE`, `FIRST_MODE`, etc.) to `STAND_MODE`.
- Submodules:
  - `stand_mode_controller_normal`: Standard toggle handling.
  - `stand_mode_controller_gesture`: Gesture-based transitions.
  - Additional submodules for transitions from other modes.

---

7. `third_mode_controller_top`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `menu_signal`, `normal_signal`: User inputs for toggling modes.

Outputs

- `third_mode_toggle`: Toggles high to signal entry into `THIRD_MODE`.

Functionality

- Transitions from `STAND_MODE` to `THIRD_MODE`.
- Ensures `THIRD_MODE` can only be entered once per power cycle using an internal state (`has_opened`).

Detailed Explanation of Event Modules

Each event module handles specific tasks related to operations within a particular mode. Below is the breakdown of their inputs, outputs, functionality, and extensibility.

---

1. `event_control`

Role

The `event_control` module is the central hub for managing transactions within the system. It delegates mode-specific tasks to dedicated event modules, ensuring modularity and clean separation of responsibilities.

Relationship with Submodules

1. **Delegation of Responsibilities**
   Based on the `current_mode`, `event_control` forwards control to specific submodules, such as:
   - `clean_mode_event_module` for `CLEAN_MODE` tasks.
   - `first_mode_event_module`, `second_mode_event_module`, etc., for other modes.
   - `setting_mode_event_module` for handling user-configurable parameters.
2. **Data Aggregation**
   - Gathers runtime data from all submodules to compute `total_running_time`.
   - Dynamically updates `numbers` to display relevant information from the active mode's module.
3. **Centralized Output**
   - Submodules handle mode-specific logic independently, but their outputs (e.g., running time, alerts) are managed and displayed centrally by `event_control`.
4. **Extensibility**
   - Adding a new mode only requires implementing its event module and linking it to `event_control`.
   - Submodule independence ensures minimal impact on existing logic.

Summary

`event_control` acts as the orchestrator, delegating operational tasks to submodules and consolidating their outputs, enabling a highly modular and scalable architecture.

---

2. `clean_mode_event_module`

Inputs

- `clk`, `rstn`: Clock and reset signals for synchronization.
- `current_mode`: Indicates the current mode of the system.

Outputs

- `current_running_time`: Tracks the elapsed time in clean mode.
- `total_running_time`: Accumulates the total runtime in clean mode.

Functionality

- Counts and updates the running time when the system is in `CLEAN_MODE`.
- Resets the counter and `current_running_time` when leaving `CLEAN_MODE`.

---

3. `first_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard synchronization and state signals.

Outputs

- `current_running_time`: Tracks the time elapsed in `FIRST_MODE`.
- `total_running_time`: Accumulates the total runtime in `FIRST_MODE`.

Functionality

- Similar to `clean_mode_event_module`, tracks and updates runtime while in `FIRST_MODE`.

---

4. `second_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard synchronization and state signals.

Outputs

- `current_running_time`: Tracks the runtime in `SECOND_MODE`.
- `total_running_time`: Accumulates total runtime for this mode.

Functionality

- Counts time when the system is in `SECOND_MODE`.
- Resets counters upon leaving the mode.

---

5. `third_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `menu_signal`: Triggers specific operations within `THIRD_MODE`.

Outputs

- `current_running_time`: Tracks the runtime in `THIRD_MODE`.
- `total_running_time`: Accumulates the total runtime.
- `third_output_time`: Computes the remaining runtime based on a predefined duration.

Functionality

- Tracks time elapsed and adjusts `third_output_time` dynamically based on user interactions (e.g., menu inputs).

---

6. `setting_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `tq, ta, tw, ts, te, td, tt, tg`: Keyboard inputs for time adjustment.

Outputs

- `numbers`: Displays adjusted times on the 7-segment display.
- `clean_remind_time`: Configurable reminder time for cleaning.
- `gesture_time`: Adjustable gesture duration for switching modes.

Functionality

- Adjusts `clean_remind_time` and `gesture_time` based on user inputs.
- Provides real-time feedback through the `numbers` output.

---

7. `stand_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `tq, ta, tw, ts`: Inputs for adjusting runtime.
- `first_toggle_signal`: Signal to toggle out of `STAND_MODE`.
- `counter_time`: Duration for countdown operations.

Outputs

- `running_time`: Tracks runtime in `STAND_MODE`.
- `stand_output_time`: Outputs adjusted time for the countdown.

Functionality

- Tracks runtime and adjusts it based on user inputs.
- Includes countdown logic for specific operations.

---

8. `off_mode_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `first_toggle_signal`: Signal to initiate the countdown.
- `counter_time`: Countdown duration.

Outputs

- `off_output_time`: Tracks the time left in the countdown.

Functionality

- Handles countdown logic for transitioning from `OFF_MODE`.

---

9. `beep_needing_event_module`

Inputs

- `clk`, `rstn`, `current_mode`: Standard signals.
- `total_running_time`: Accumulated runtime for all modes.
- `clean_remind_time`: Threshold for triggering cleaning reminders.
- `clean_by_hand`: User input to manually reset the reminder.

Outputs

- `play_beep`: Triggers a beep when conditions are met.

Functionality

- Generates a beep signal when the runtime exceeds the reminder threshold or manual cleaning is triggered.

Detailed Explanation of utils_modules Submodules

The `utils_module` contains utility submodules providing essential functionality, such as timing, signal processing, and user interaction support. Here's a detailed explanation:

---

1. `countdown_timer`

Inputs

- `clk`, `rstn`: Clock and reset signals for synchronization.
- `start`: Edge-triggered signal to begin the countdown.
- `load_value`: The initial countdown value in seconds.
- `reset_signal`: Signal to reset the countdown timer.

Outputs

- `count_reg`: The remaining seconds in the countdown.
- `done`: High when the countdown reaches zero.

Functionality

- Counts down from the `load_value` to zero, decrementing every second.
- Tracks whether the countdown is active and stops when completed.

---

2. `debouncer`

Inputs

- `clk`, `rstn`: Clock and reset signals.
- `button_in`: Raw button signal with potential noise.

Outputs

- `debounced_out`: Stable output signal after debouncing.

Functionality

- Filters noise from the input signal using a shift register and outputs a stable signal.

---

3. `keyboard`

Inputs

- `clk`, `rstn`: Clock and reset signals.
- `ps2_clk`, `ps2_data`: PS/2 keyboard clock and data signals.

Outputs

- `t1, t2, t3, tc, tm, tq, ta, tw, ts, te, td, tt, tg`: One-hot signals indicating key presses.

Functionality

- Decodes keypresses from a PS/2 keyboard into specific one-hot signals for further processing.

---

4. `light_7seg_ego1`

Inputs

- `sw`: 4-bit input representing a digit or symbol.

Outputs

- `seg_out`: 8-bit output to control a 7-segment display.

Functionality

- Maps the 4-bit input to the appropriate 7-segment display pattern.

---

5. `player`

Inputs

- `clk`, `rstn`: Clock and reset signals.
- `play`: Signal to start playing a sound.

Outputs

- `beep`: Square wave output for sound generation.
- `loud`: Constant high signal for compatibility with external devices.

Functionality

- Generates a beep sound at a fixed frequency when `play` is high.

---

6. `scan_seg`

Inputs

- `rstn`, `clk`: Clock and reset signals.
- `show`: 32-bit input representing the digits to be displayed.

Outputs

- `seg_en`: 8-bit signal to enable specific 7-segment displays.
- `seg_out_left`, `seg_out_right`: Outputs for the 7-segment display segments.

Functionality

- Cycles through each digit in the `show` input and displays it on a 7-segment display.

---

7. `shining_mode`

Inputs

- `clk`, `rstn`: Clock and reset signals.
- `islight_signal`: Signal to enable or disable lighting.
- `current_mode`: Indicates the active mode.

Outputs

- `shining`: 8-bit output controlling the light status.

Functionality

- Controls lighting based on `islight_signal` and the current mode.

Overall Summary of the System Design

The system is built on a well-structured architecture, comprising state machine modules, transaction handling modules, and utility modules, all designed with a focus on **modularity**, **scalability**, and **maintainability**. The architecture ensures seamless interaction between components while supporting future enhancements and debugging.

Key Attributes of the System

1. Highly Modular Architecture

- **State Machine Modules**: Each state is managed by a dedicated module, ensuring clear boundaries between state transitions and reducing interdependencies.
- **Transaction Handling Modules**: Operations within each state are isolated into independent modules, allowing state transitions and internal logic to function without interference.
- **Utility Modules**: Reusable components, such as debouncers and timers, provide essential functionality without duplicating logic across the system.

---

2. Strong Scalability

- **Adding New States**: The state machine framework allows for the effortless addition of new states by simply integrating additional state controllers.
- **Expanding Features**: Transaction modules can be enhanced or extended to include new functionality without impacting the existing system.
- **Reusable Utilities**: Core utility modules, like the `countdown_timer` and `keyboard`, are generic and can support a wide range of applications.

---

3. Debug-Friendly Design

- **Clear Signal Flow**: Each module has well-defined inputs and outputs, simplifying the process of tracking data and signals across the system.
- **Independent Testing**: Modules operate independently, enabling targeted testing and debugging without affecting other components.
- **State Traceability**: Registers and counters within modules provide detailed insights into their current operation, aiding in quick troubleshooting.

Conclusion

The system's architecture is both **flexible** and **robust**, ensuring it can adapt to evolving requirements with minimal changes. Its modular design principles and logical separation of responsibilities make it an ideal platform for complex applications, offering excellent extensibility and ease of maintenance.

Implementation Instructions for Bonus *附加分数的实现说明*

The implementation of the bonus features demonstrates a well-thought-out approach to system design, focusing on both innovation and practicality. Here's how each feature is implemented:

---

1. Simulate Gesture Control for Power On/Off Using Button Operations

Description:
We simulated gesture control functionality by mapping specific button sequences to power on/off operations. For example:

- **Power On**: Simulated by pressing a button to represent a "left gesture" followed by a "right gesture."
- **Power Off**: Simulated by pressing a button to represent a "right gesture" followed by a "left gesture."

Design Safeguards:
To prevent unintended behaviors, such as simultaneous power on and off, the system is designed to:

- **Ignore Invalid Sequences**: Inputs such as "left-right-left" or "right-left-right" are invalid and do not trigger any operation.
- **State-Dependent Responses**: The system ensures that power on/off gestures are processed only if they follow the correct sequence and match the current state (e.g., "power on" only works when the system is off).

Rationale:
This approach ensures:

- **No False Triggers**: Improves the reliability of gesture control by filtering out unintended or rapid button presses.
- **User-Centric Design**: Reflects real-world expectations where gesture-based systems are designed to minimize user error.

---

2. Use of Input Devices Other Than DIP Switches and Button Switches

Description:
We integrated a PS/2 keyboard to replace conventional touchpad operations found in modern range hoods. The keyboard handles non-critical inputs, such as:

- **Mode switching**.
- **Settings adjustment**.
- **Querying operational data**.

Design Decision:
Critical operations like power control, mode setting, and manual cleaning remain on the development board to ensure:

- **Stability**: These core functions require reliable and robust physical inputs.

- **Physical Assurance**: Tactile feedback from physical buttons provides confidence in critical operations.

Rationale:
This separation of input responsibilities aligns with real-world design practices, where critical functions prioritize stability, and auxiliary functions leverage flexible input options like touchpads or keyboards.

---

3. Use of Output Devices Other Than Seven-Segment Displays and LEDs

Description:
We added a buzzer as an additional output device to provide audio feedback. The buzzer emits continuous sound when:

- The accumulated operational time exceeds the preset maintenance reminder threshold.

Rationale:
The buzzer serves as an intuitive and practical alert mechanism, ensuring users are aware of maintenance requirements without the need for constant visual monitoring.

Summary

The implementation of these bonus features highlights our thoughtful and user-focused design:

1. **Gesture Simulation**: Provides reliable gesture-based functionality while preventing false triggers through sequence validation.
2. **PS/2 Keyboard Integration**: Balances flexibility and stability by assigning critical operations to physical inputs and convenience features to the keyboard.
3. **Buzzer Integration**: Enhances usability with clear, real-world audible alerts for maintenance reminders.

This comprehensive approach not only satisfies the bonus requirements but also aligns with practical appliance design principles, ensuring extensibility and reliability.

Project Summary *项目总结*

Teamwork Summary

Our team divided the workload effectively, but time management was a significant challenge. One team member had other major assignments during the project, which required adjustments to the task schedule. This led to delays in completing the initial design phase, especially when debugging the state machine. To address this, other members stepped in to support debugging while progressing with their own tasks.

Development Process

The development process faced several obstacles:

1. **State Machine Complexity**: Designing and coding the state machine was more time-consuming than anticipated. Issues with state transitions and unintended loops caused delays, requiring repeated debugging and revisions.

2. **Advanced Features**: Implementing gesture control was challenging due to the precise timing required. Initial simulations showed inconsistent behavior, leading to additional refinements in the timing logic.
3. **Coordination**: Some tasks, such as integrating auxiliary functions like lighting and time display, were delayed as priority was given to the state machine. This created pressure on the testing phase.

Testing and Validation

Testing uncovered several critical issues:

1. **Mode Switching Errors**: Transitioning between modes, particularly from extraction to standby, occasionally failed due to incomplete state updates. This required extensive debugging of the state machine logic.
2. **Timing Issues**: On-board tests revealed timing mismatches in gesture control, where the countdown was either too short or failed to reset properly.
3. **Unexpected Bugs**: The query function intermittently provided incorrect results for accumulated time tracking, which was traced back to a minor oversight in variable initialization.

Conclusion

Although the project faced challenges in managing schedules, debugging complex state transitions, and ensuring the advanced features worked smoothly, teamwork and determination helped overcome these issues. The final system was completed on time and met all functionality requirements, providing a valuable learning experience for the team.

Ideas and Suggestions for Projects *项目出题的想法和建议*

Idea 确保为个人想法（写的时候抬头看到了饮水机）

> # 基于 EGO1 开发板的简易饮水机设计
>
> ## 一、项目概述
>
> 本项目旨在设计并实现一个基于 **EGO1 开发板** 的简易饮水机控制系统，模拟日常饮水机的典型工作流程。通过该项目，学习者能够掌握常见外设（如拨码开关、按钮、七段数码管、LED 指示灯等）的使用方法，并通过有限状态机（FSM）设计实现逻辑控制。项目的功能包含温度设定、加热控制、取水操作、显示与指示等关键模块，最终形成一个从“加热”到“出水”的完整控制流程，适用于教学和实验。
>
> ---
>
> ## 二、设计目标
>
> 1. 实现功能模块
>   - 温度设定：通过拨码开关选择目标温度。
>   - 加热控制：基于当前温度与目标温度的比较，实现加热与停止逻辑。
>   - 取水操作：检测用户需求并控制出水流程。

- 显示与指示：利用七段数码管和LED指示灯提供实时反馈。

2. 教学价值
   - 提供对单片机 GPIO、定时器、中断和状态机编程的实践机会。
   - 加强对硬件资源和控制逻辑之间关系的理解。
3. 可扩展性
   - 支持添加高级功能（如水位检测、过温保护、节能模式等）。

三、系统硬件结构

1. 硬件资源需求

硬件组件	功能描述
EGO1 开发板	提供基础计算与接口能力。
拨码开关	用户输入目标温度选择。
按钮	控制加热、取水与系统重置操作。
七段数码管	实时显示当前水温或操作提示。
LED 指示灯	指示加热状态与温度达标状态。
温度传感器(模拟)	获取真实温度数据（或模拟温度变化）。

2. 硬件连接示意

- **拨码开关（DIP Switch）**：连接到 GPIO，用于目标温度输入。
- **七段数码管**：连接到 GPIO，通过定时器刷新显示当前水温或操作状态。
- **LED 指示灯**：通过 GPIO 显示系统加热状态与目标温度达标状态。
- **模拟温度传感器**：通过变量递增/递减模拟温度变化，或连接热敏电阻等实际传感器。

四、系统功能描述

1. 温度设定

通过拨码开关读取目标温度值并映射到特定的温度范围（例如 50℃ 至 90℃，以 10℃ 为步长）。

2. 加热控制

- 在“开始加热”按钮触发后，系统判断当前温度是否低于目标温度。
- 若低于目标值，启动加热；若已达目标值，则指示温度达标。

3. 取水操作

- 检测“取水”按钮输入，当温度满足目标值时允许出水；否则提示温度不足。

4. 显示与指示

- 七段数码管用于显示实时水温或提示信息。
- LED 指示灯提供加热与温度达标的视觉反馈。

5. 系统重置

- 按下“重置”按钮后，系统恢复初始状态，重新加载目标温度值并重置所有显示。

五、状态机设计

通过有限状态机（FSM）定义项目的工作逻辑，包括以下状态：

1. S0：初始/重置
    - 读取拨码开关值并映射为目标温度。
    - 重置当前温度为默认值（如 25℃）。
    - 转入待机状态。
2. S1：待机
    - 等待用户操作，显示当前温度。
    - 根据用户输入切换至加热或重置状态。
3. S2：加热中
    - 模拟水温逐步上升。
    - 达到目标温度后停止加热，转入温度达标状态。
4. S3：温度达标
    - 允许用户取水并检测温度变化。
    - 温度下降到目标值以下时，重新进入加热状态。
5. S4：出水
    - 模拟出水流程，根据水量或时间控制返回待机状态。

六、总结与扩展

本项目通过实现一个模拟饮水机的控制系统，系统性地展示了如何使用外设与 FSM 实现完整逻辑控制。通过本项目，学习者可以掌握从硬件资源操作到软件逻辑设计的全流程。此外，系统具备高度可扩展性，能够添加更多功能以增强系统实用性与智能化程度。