SUSTech_CS207-DL_2024f_Project-a-kitchen-hood

小组成员：林鸿渝，吴嘉淇，冉明东

> **Caution**
>
> 源码托管于 GitHub，将在项目 ddl 结束后基于 MIT License 协议开源，访问链接：
>
> [Grimm57/CS207DigitalLogicFinalProject]

0. 项目工作相关内容说明

0.1 团队分工（括号内为贡献比）

林鸿渝（34%）：不同模式切换的状态机实现、开关机控制、手势开关使用及有效时间查询与修改、照明模式。

吴嘉淇（33%）：自清洁模式下的清除累计时间、智能提醒时间调节、分频器实现、动态时间显示与修改功能实现。

冉明东（33%）：油烟机模式下挡位切换与飓风模式控制、智能提醒、累计时间计时、手动自清洁。

0.2 开发计划日程安排和实施情况

0.2.1 开发计划日程安排

日期	计划完成内容
11.23 - 11.24	完成油烟机项目的要求梳理和任务分配，实现顶层模块初步设计。
11.25 - 12.1	各自完成分配到的任务，各自上板。工作期间找共同空余时间开会讨论下一步进展。
12.2 - 12.8	拼接不同模块，相聚进行上板测试以及相应部分代码讲解，减少重复冗余代码，修正代码漏洞。
12.9 - 12.13	基本实现项目所有功能。根据答疑文档和要求文档一一比对功能，添加缺失功能以及修改有误功能。
12.14 - 12.15	最后的项目功能校对，改进代码规范性。
12.16	提前一周提交项目文件，准备周三答辩。
12.18	顺利结束答辩，代码对应功能全部实现。

0.2.2 实际实施情况

日期	实际完成内容
11.24	选题确定油烟机，完成油烟机项目的要求梳理和任务分配。
12.5	倒计时、分频器小功能实现，顶层模块初步实现。

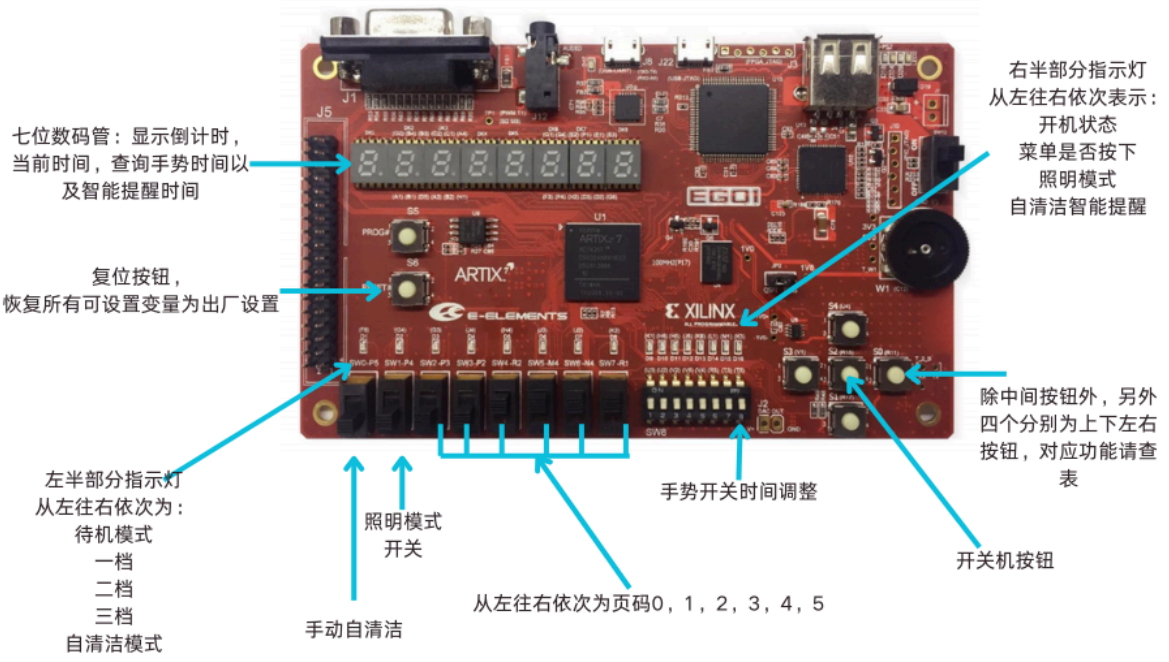
日期	实际完成内容
12.8	正常开关机功能实现，油烟机模式与自清洁初步实现。
12.11	手势开关机实现。初步实现 fsm 状态机。整理了 $github$ 的项目文件。开始整合项目不同模块。
12.18	动态时间显示实现，时间调整实现，油烟机模式基本功能实现。
12.21	自清洁智能提醒实现。添加照明功能。修复模块合并后产生的 bug 。
12.22 - 12.23	参照项目要求比对，添加缺失功能，更正有误功能。修复 bug。
12.23 晚	提交文件。
12.25	答辩，现场出现功能实现异常，当场修复。

1. 系统功能列表

本项目实现了一个由 EGO1 控制的抽油烟机控制电路，通过对不同按钮与拨码开关的调整，模拟实现了抽油烟机的油烟机、自清洁、开关机（包括手势开关机）、状态切换、查询/调整时间、照明、参数高级设置以及时间显示功能。

系统功能	具体说明	特别说明
开关机	短按开机键开机，初始化系统时间和累计工作时间，长按3s关机。	包括可以自定义时间的手势开关机功能
模式切换	通过不同的页码和菜单键，一二三挡键以及自清洁键的组合，实现不同的状态切换	/
油烟机	包含一二三挡的转化和累计时间的计算，倒计时的计算	与自清洁模式配合工作，传递自清洁提醒
自清洁	与油烟机模块配合工作，实现对累计时长的清零	/
查询/显示时间	一般情况下都显示系统时间，可以单独查询累计工作时间	/
照明	通过照明键实现照明功能	/
高级参数设置	可以自由调整系统时间，自清洁提醒时间和手势开关机时间	/

2. 系统使用说明



按钮	上按钮功能	下按钮功能	左按钮功能	右按钮功能	中按钮功能
页码 0	选择智能提醒/手势开关机时间调整位数	开启智能提醒/手势开关机时间调整模式	减少时间	增加时间	开关机操作
页码 1	菜单切换	进入自清洁模式	显示累计工作时间	显示手势开关机时间	开关机操作
页码 2	菜单切换	三档模式	一档模式	二档模式	开关机操作
页码 3	选择当前时间调整位数	开启当前时间调整模式	减少时间	增加时间	开关机操作
页码 4	菜单切换	/	查询当前智能提醒时间	/	开关机操作
页码 5	菜单切换	/	手势开关机左键	手势开关机右键	开关机操作

（在打开页码0的同时打开开启手势开关机时间调整开关可以实现手势开关机时间的调整）

3.系统结构说明

3.0 总体结构

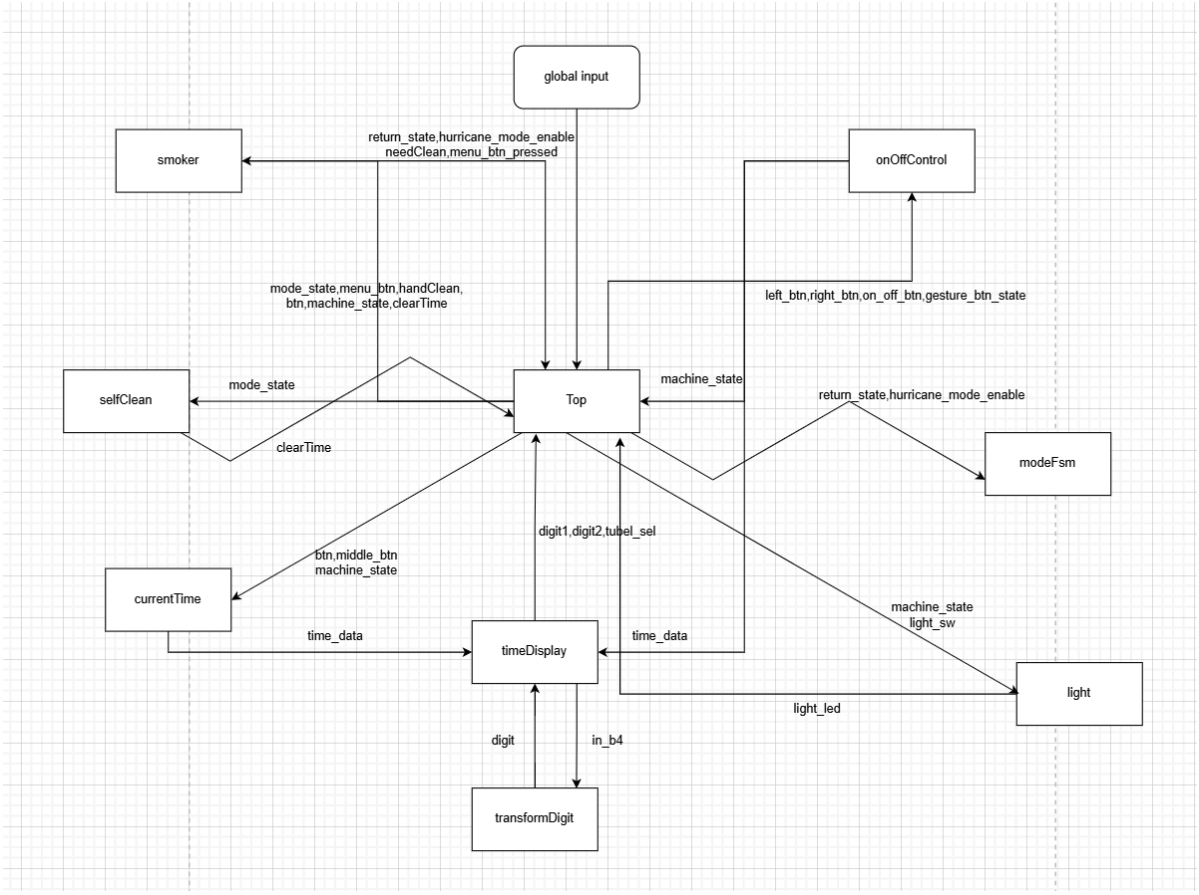
- 中心模块 `Top`
- 图的核心模块，负责与其他所有子模块交互。
- 接收并发送大量信号，包括模式状态（`mode_state`）、机器状态（`machine_state`）、时间数据（`time_data`）、按钮状态等。

3.1 主要模块及交互关系

1. `Top` **模块**：
   - 顶层模块，系统的全局输入接口。
   - 负责将各种状态信号（如返回状态、飓风模式启动信号、清洁需求信号、按钮状态等）传递给需要的对应模块。
2. `smoker` **模块**：
   - 与中心模块 `Top` 交互。
   - 提供与吸烟机相关的状态信息（如 `return_state`、`hurricane_mode_enable`、`needClean` 等）。
   - 涉及吸烟机的功能实现。
3. `selfClean` **模块**：
   - 用于自清洁功能的实现。
   - 接收 `mode_state` 信号，向 `Top` 提供 `clearTime` 信号。
4. `currentTime` **模块**：
   - 负责系统时间的维护。
   - 接收来自 `Top` 的按钮和机器状态，返回当前时间数据（`time_data`）。
5. `timeDisplay` **和** `transformDigit` **模块**：
   - `transformDigit`：处理数码管显示的数值转换。
   - `timeDisplay`：负责时间显示，接收数值输入以及转换信号，与 `Top` 协作完成显示功能。
6. `onOffControl` **模块**：
   - 控制系统的开关状态。
   - 接收多种按钮状态（如 `left_btn`、`right_btn`、`on_off_btn`、`gesture_btn_state`），并与 `Top` 交互管理系统的开关功能。
7. `modeFsm` **模块**：
   - 实现系统的状态机逻辑。
   - 根据输入信号（如 `return_state`、`hurricane_mode_enable`）来调整三档结束后的状态选择。
8. `light` **模块**：
   - 控制系统灯光功能。
   - 接收来自 `Top` 的机器状态信号（`machine_state`）和开关信号（`light_sw`）。
   - 返回灯光状态信号（`light_led`）。

3.3 数据交互特点

- **信号流动性**：
  - 信号在各模块之间双向流动，模块间关系紧密。
  - 大部分信号通过中心模块 `Top` 进行分发。
- **分层设计**：
  - 中心模块 `Top` 负责调度信号。
  - 下层功能模块（如 `smoker`、`selfClean`、`currentTime` 等）完成具体任务。

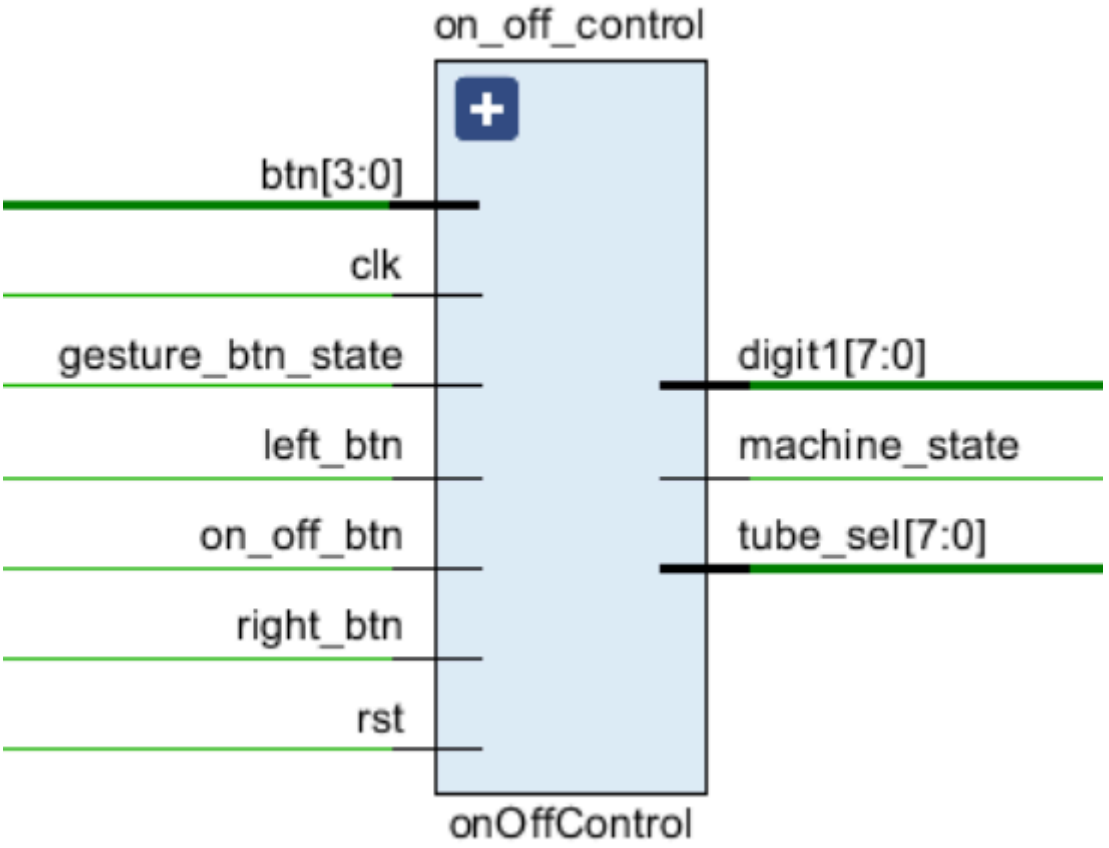


4. 子模块功能说明

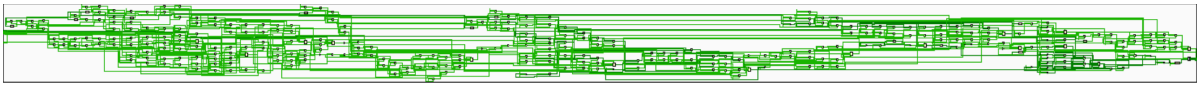
> **Note**
>
> 下面展示的部分 $schematic$ 并不通用于项目里的所有文件。例如对 $ClockDivider500Hz$ 进行了多次实例化，传入传出关系根据具体情况而定。对于这类 $schematic$ ，我们只展示功能实现的其中一张。

- OnOffControl

开关机控制模块用来模拟油烟机的正常开关机、手势开关机，同时可实现对于有效手势时间的调整以及显示。

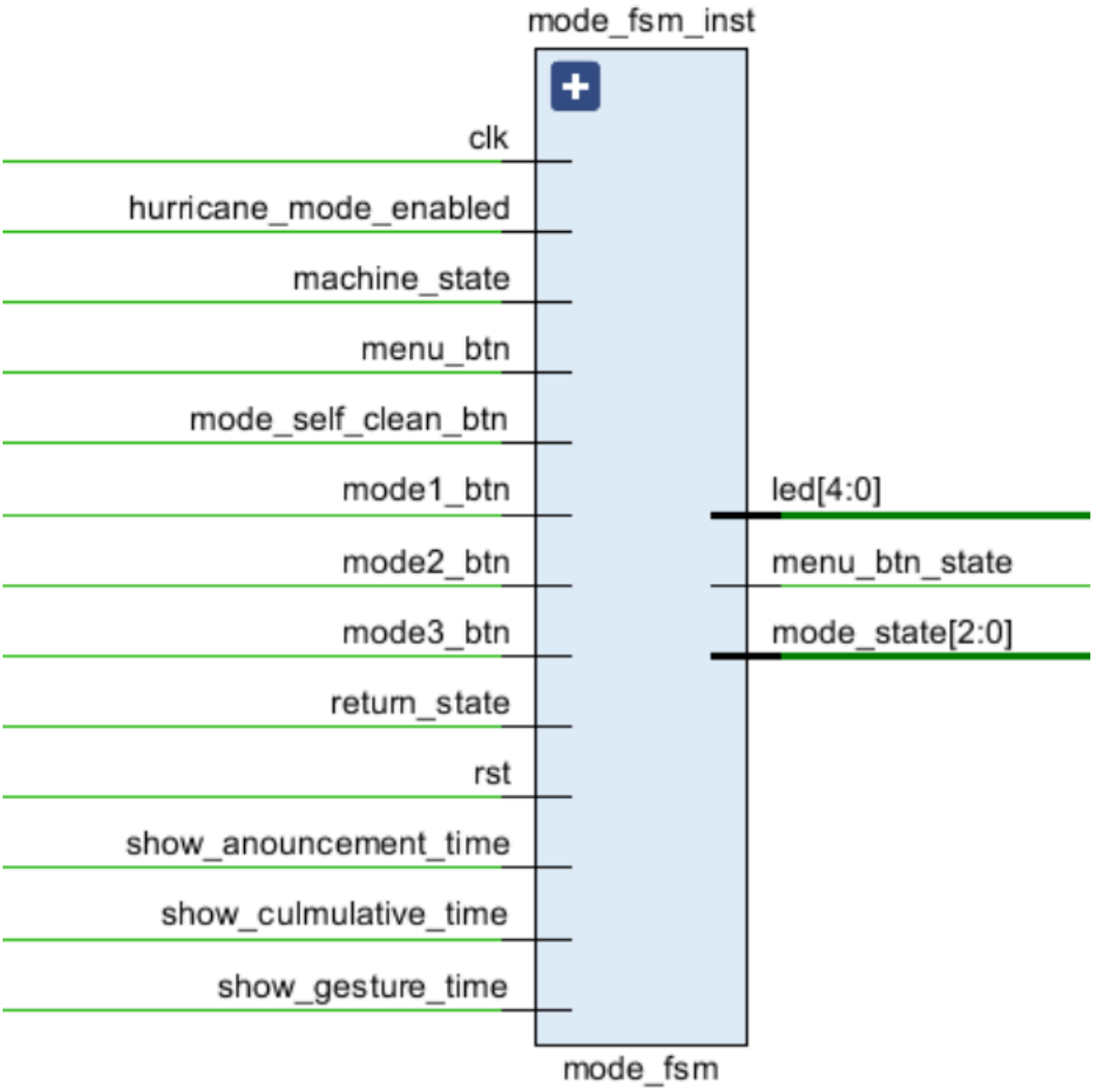


端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
btn[3:0]	I	进入编辑状态后，检测上下左右键的瞬时使用状态，在调整有效手势时间时使用
gesture_btn_state	I	表征按钮页是否匹配手势开关机页
left_btn	I	进入手势开关机页后，检测左键瞬时使用状态
right_btn	I	进入手势开关机页后，检测右键瞬时使用状态
on_off_btn	I	任何模式下的开关机按钮，单次按动实现开机，长按三秒实现关机
machine_state	O	表征开关机状态
tube_sel[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，在编辑有效手势时间以及查询有效手势时间时调整 / 显示时间
digit1[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，在编辑有效手势时间以及查询有效手势时间时调整 / 显示时间



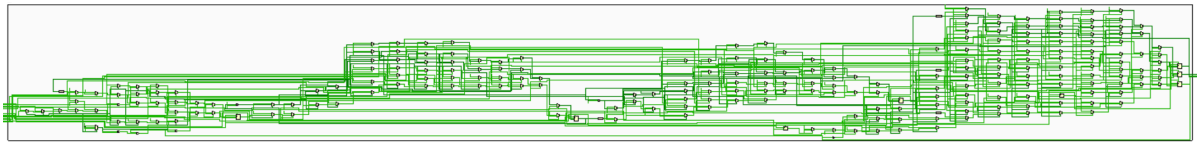
- ModeFSM

模式状态机模块实现了待机、一档、二档、三档、自清洁、显示提醒时间、显示累计工作时间、显示有效手势时长的当前工作 / 显示状态之间的相互转换。同时输出 $led[4:0], menu\_btn\_state$ 等来进行当前挡位的提醒。传出 $mode\_state[2:0]$ ，在 top 及其他模块中控制油烟机工作及显示。



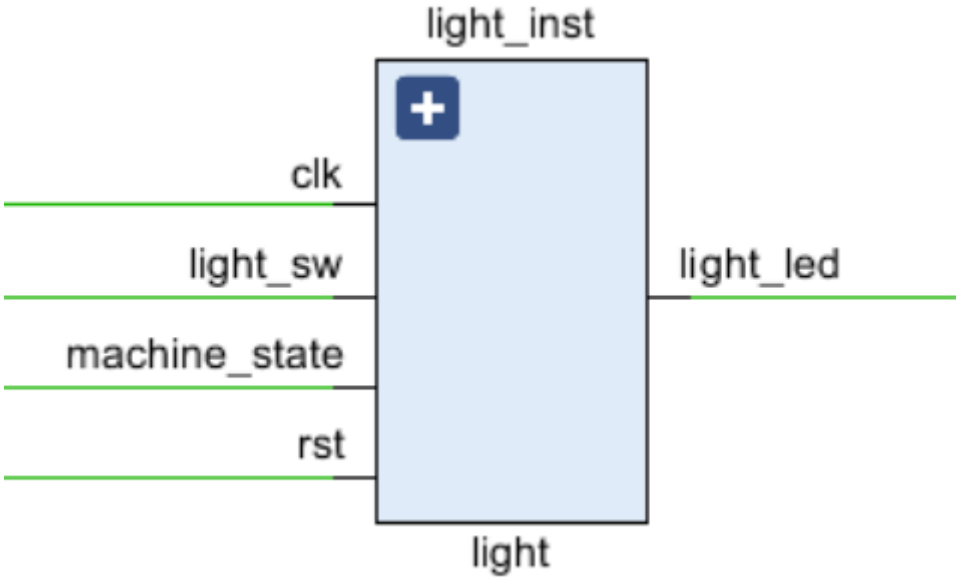
端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
machine_state	I	开关机状态。仅当开机状态下所有功能有效
menu_btn	I	菜单按钮瞬时使用状态，摁下后 $menu\_btn\_state$ 翻转
mode1_btn	I	一档按钮瞬时使用状态
mode2_btn	I	二档按钮瞬时使用状态
mode3_btn	I	三档按钮瞬时使用状态
mode_self_clean_btn	I	自清洁按钮瞬时使用状态

端口名称	I/O	功用描述
hurricane_mode_enabled	I	表征三档是否允许进入。一次开机仅能进入一次三档
return_state	I	在三档状态下，若检测到此端口为高电平，说明未摁下菜单键，倒计时结束回二档；否则说明摁下菜单键，倒计时结束回待机
show_anouncement_time	I	显示智能提醒时间的信号传入。检测到该信号，调整 $mode\_state$ 至显示智能提醒时间的模式
show_culmulative_time	I	显示累计工作时间的信号传入。检测到该信号，调整 $mode\_state$ 至显示累计工作时间的模式
show_gesture_time	I	显示有效手势时间的信号传入。检测到该信号，调整 $mode\_state$ 至显示有效手势时间的模式
led[4:0]	O	若 $mode\_state$ 在 $3'b000$ 到 $3'b011$ 之间，显示对应的挡位灯
menu_btn_state	O	表征菜单键是否已被摁下，同时绑定一个 led 进行显示
mode_state[2:0]	O	传出不同模式状态，控制其余模块的显示 / 工作



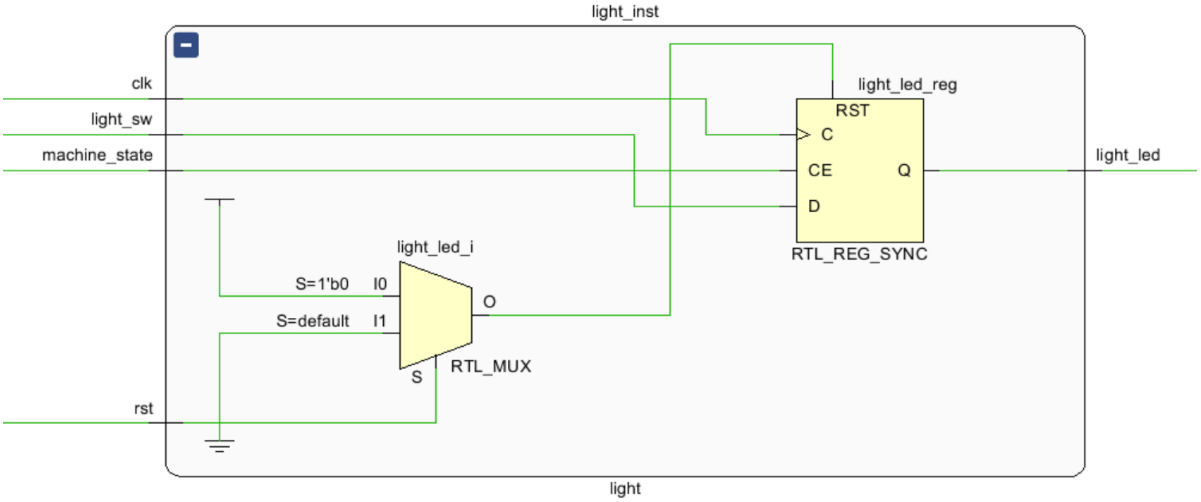
- Light

照明模块实现简单的照明功能，仅在开机状态中能使用。接受一比特照明开关输入，输出照明 LED 灯状态。



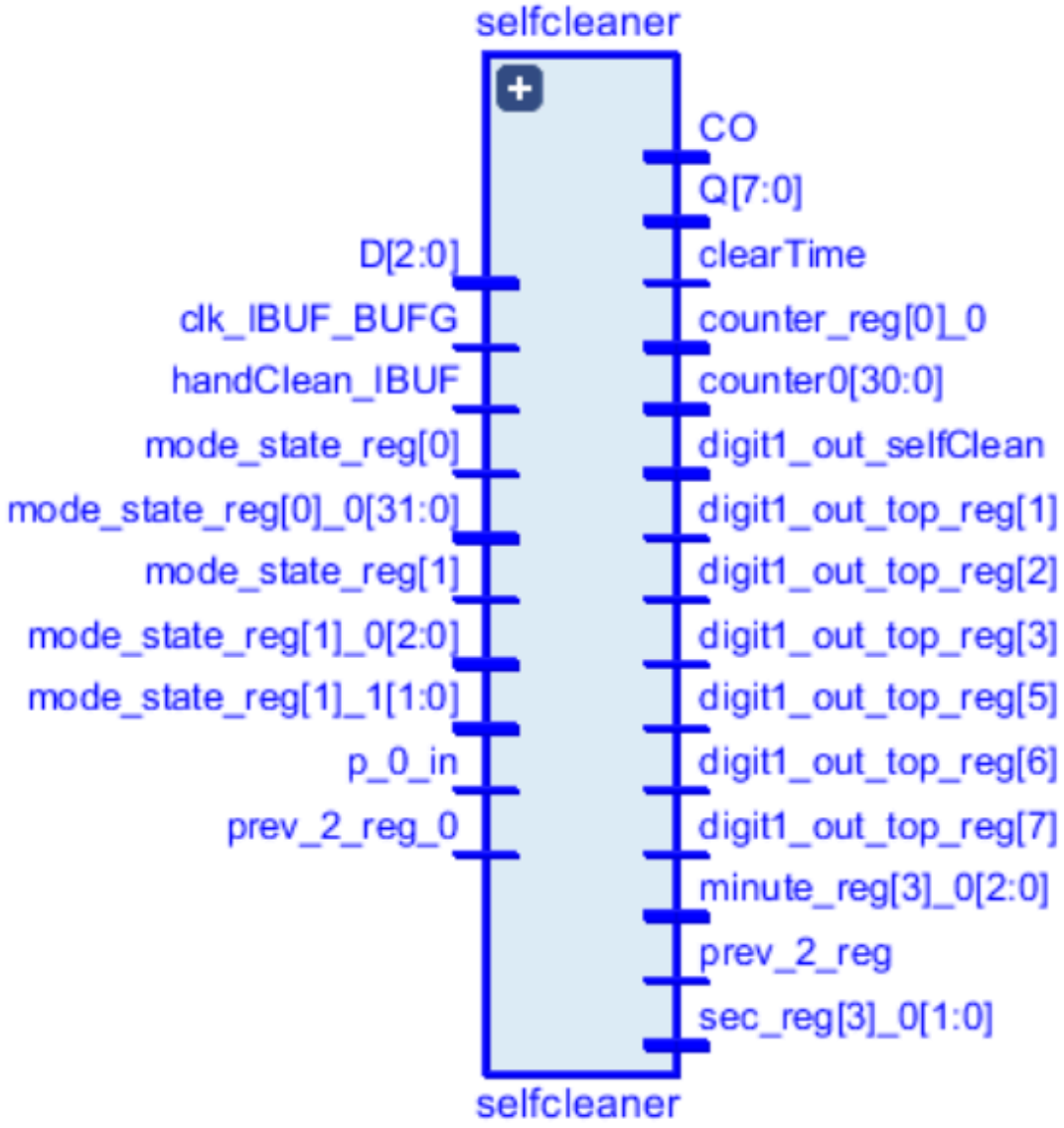
端口名称	I/O	功用描述
clk	I	系统时钟

端口名称	I/O	功用描述
rst	I	重置所有设置，参数变量全部初始化
light_sw	I	照明开关输入
machine_state	I	开关机状态输入。仅在其为高电平时能实现照明
light_led	O	照明 led 灯输出。当照明模式开启时为 1'b1，否则为 1'b0

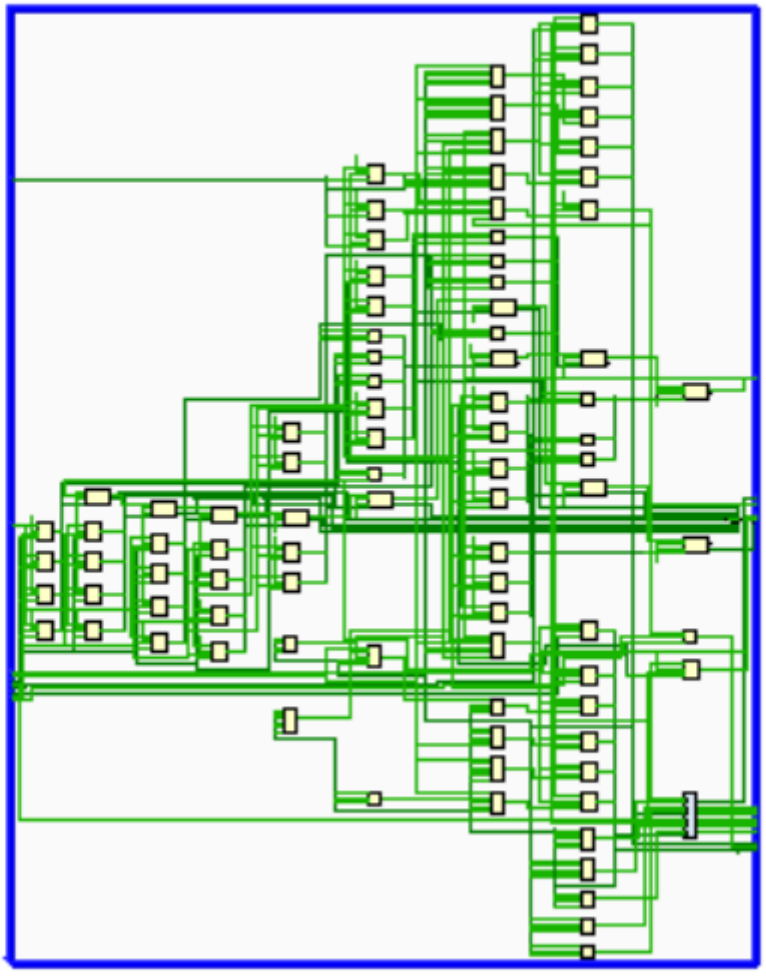


- SelfCleaner

  用于完成自清洁的倒计时功能,并且传出已完成自清洁的指示信号给抽油烟机

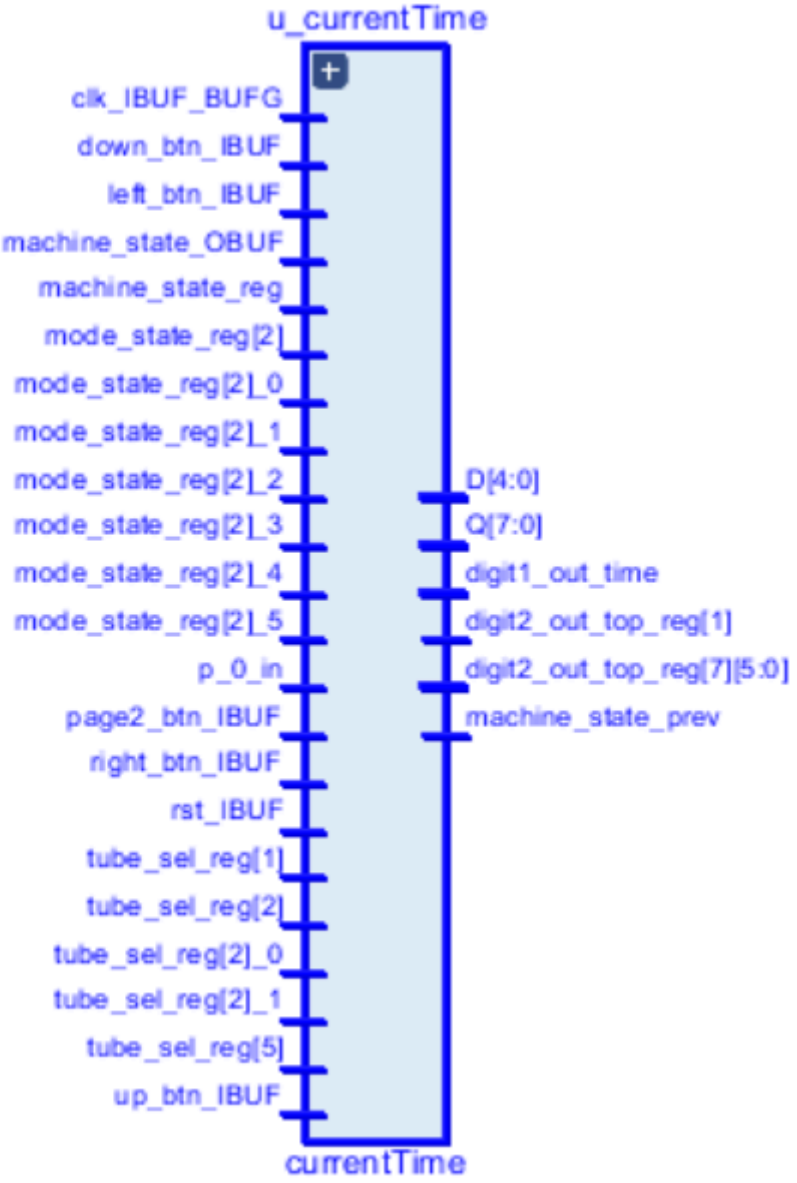


端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
btn[3:0]	I	进入编辑状态后，检测上下左右键的瞬时使用状态，在调整有效手势时间时使用
mode_state[2:0]	I	接收ModeFSM传出的指示信息,只有自清洁状态下才启动该模块
clear_accumulated_time	O	清空累计工作时长的指示信号,在每次结束自清洁后会反转
tube_sel[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，显示倒计时时间
digit1[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，显示倒计时时间
digit2[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，显示倒计时时间

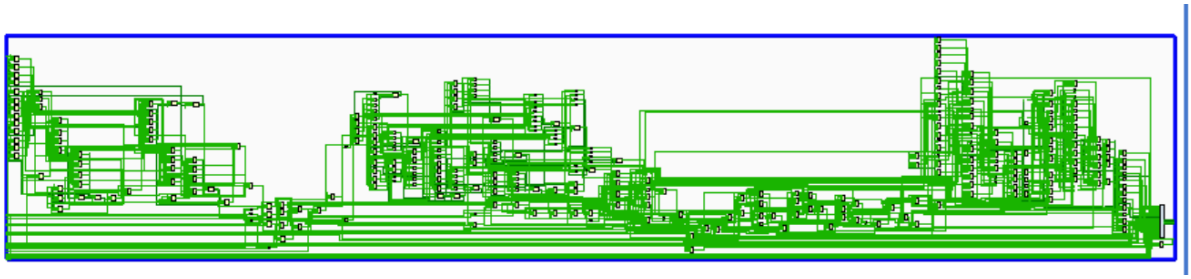


- CurrentTime

  用于显示当前时间和调整当前时间

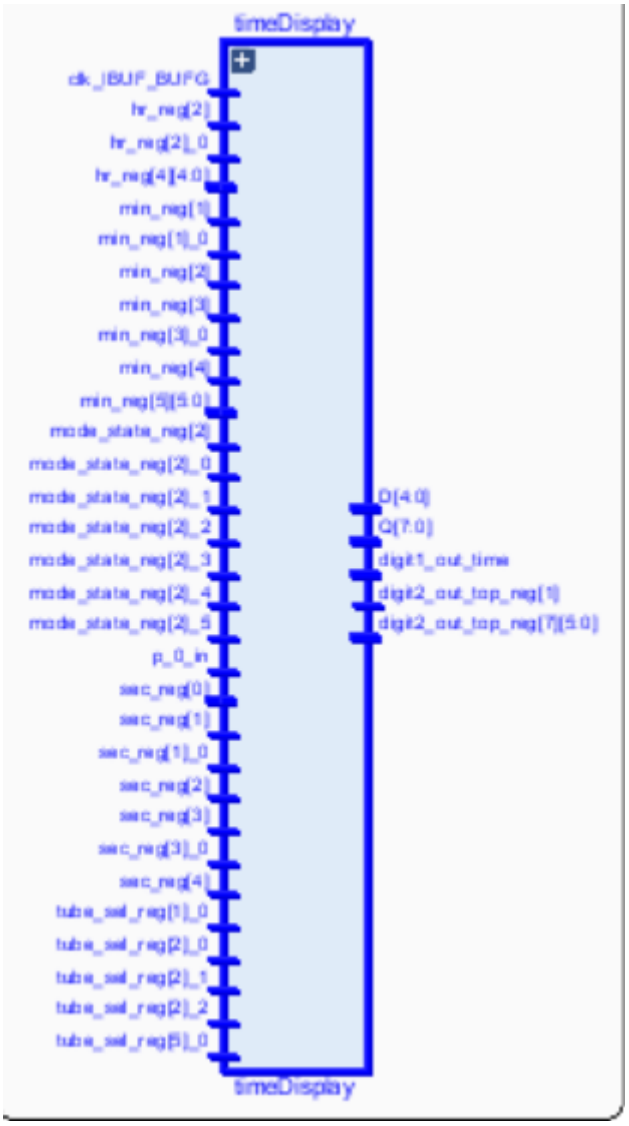


端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
btn[3:0]	I	进入编辑状态后，检测上下左右键的瞬时使用状态，在调整有效手势时间时使用
machine_state	I	表征开关机状态,只有开机状态才可以编辑和显示当前时间
time_adjust_led	O	表示是否开启时间调整模式,仅仅在测试中使用
location_led[5:0]	O	表示当前正在调整哪一位时钟的显示,仅仅在测试中使用
tube_sel[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，编辑/显示当前时间中会显示对应的内容
digit1[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，编辑/显示当前时间中会显示对应的内容
digit2[7:0]	O	作为 $TimeDisplay$ 模块实例化的传入信号，编辑/显示当前时间中会显示对应的内容



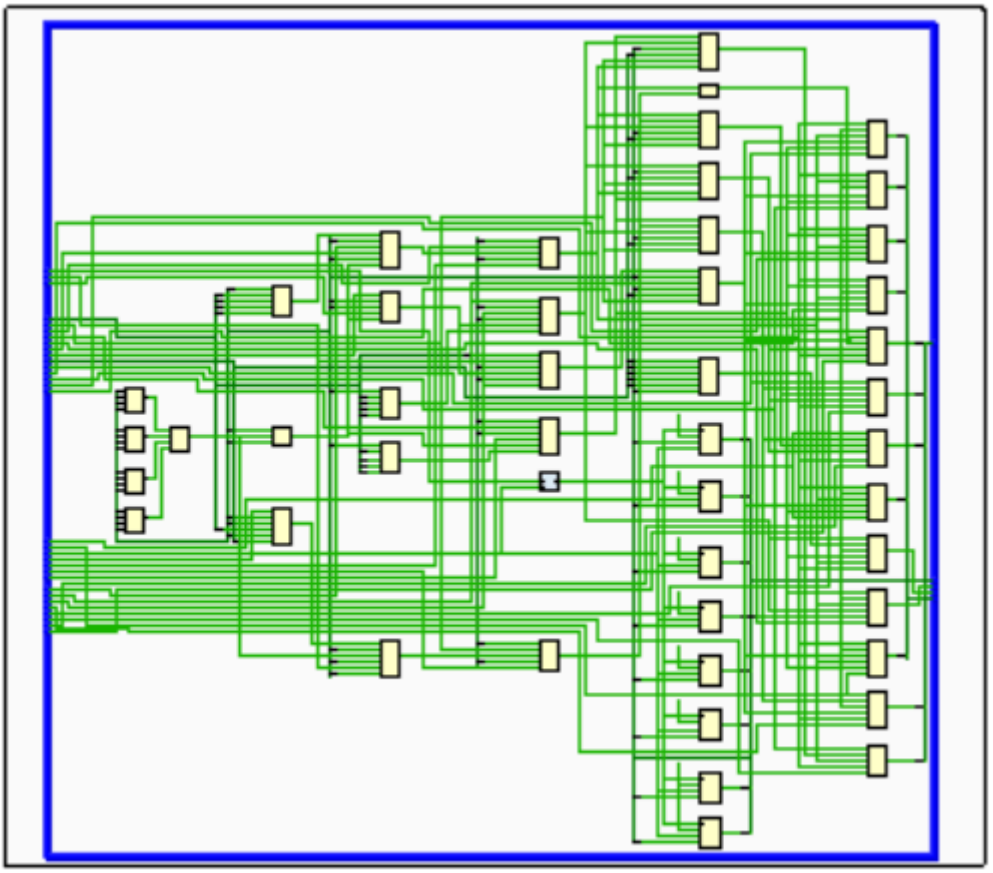
- TimeDisplay

  用于将接收到的32位时钟数据编码后显示在数码管上



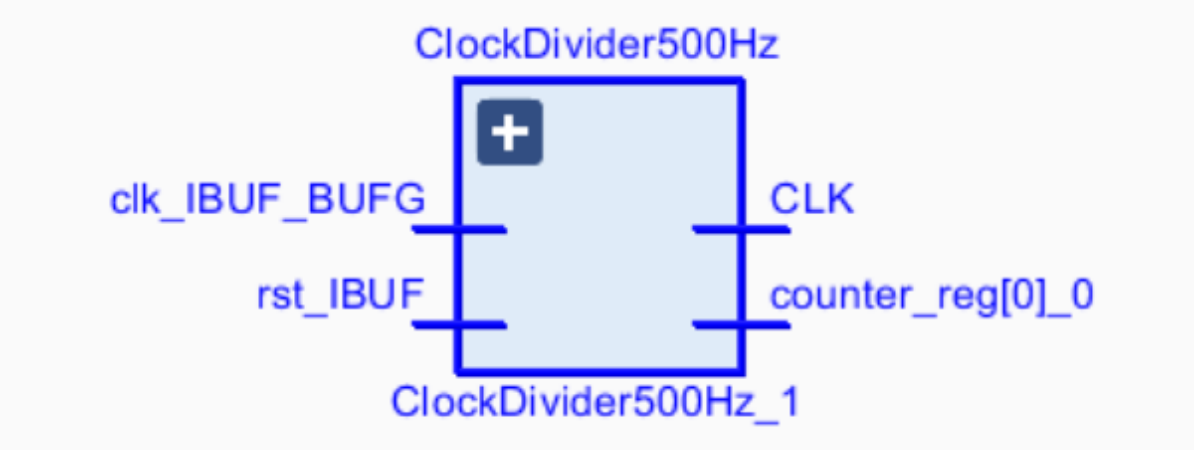
端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
time_data[31:0]	I	每四bit代表一个0到9的数字,共有8个可显示的数码管组合故共有32bits,对应相应的数码管显示数字(将编码转化为数字的功能在transformDigit模块中实现)
tube_sel[7:0]	O	输出当前模块需要显示的时间信息在数码管上
digit1[7:0]	O	输出当前模块需要显示的时间信息在数码管上

端口名称	I/O	功用描述
digit2[7:0]	O	输出当前模块需要显示的时间信息在数码管上

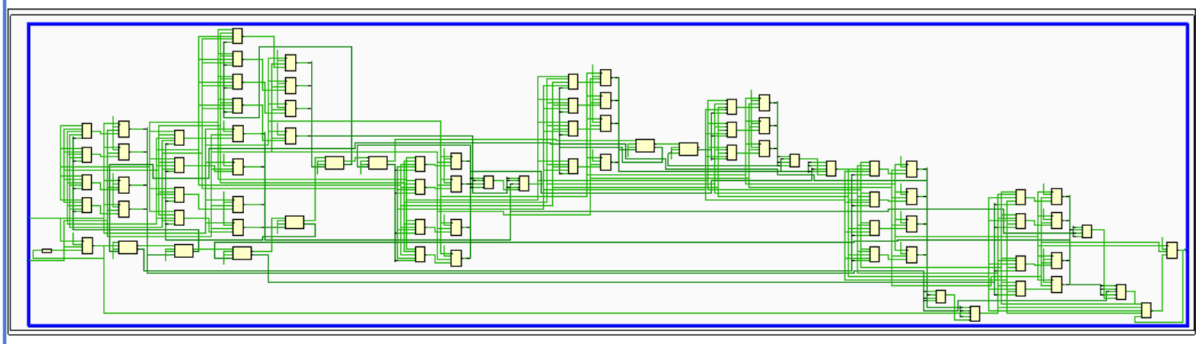


- ClockDivider500Hz

  用于得到数码管显示时流水灯时钟频率

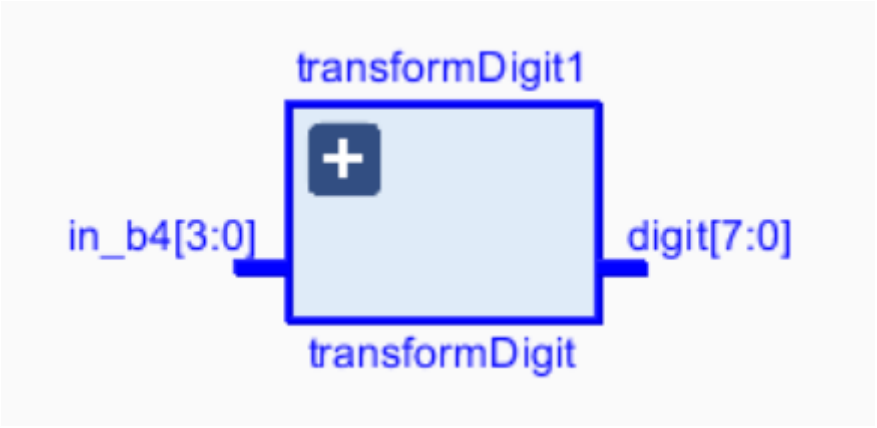


端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
clk_out	O	输出时钟用于控制流水灯显示的频率

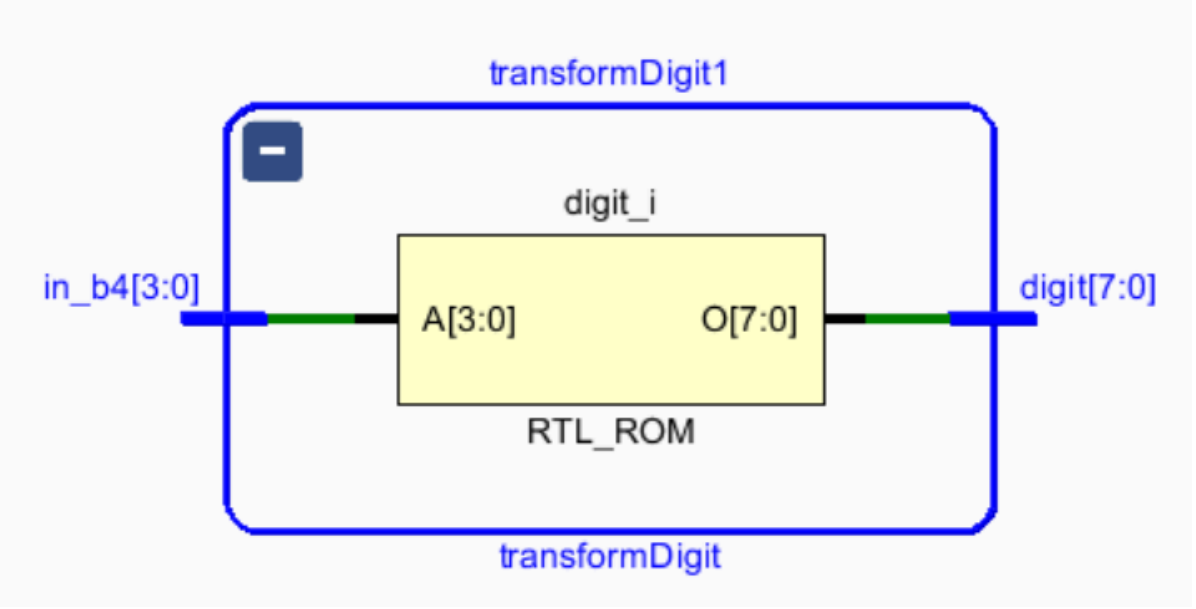


- transformDigit

接收四比特的一位十进制数字信息,把它转换成数码管显示格式,如果信息无效则转换成"横杠"进行显示

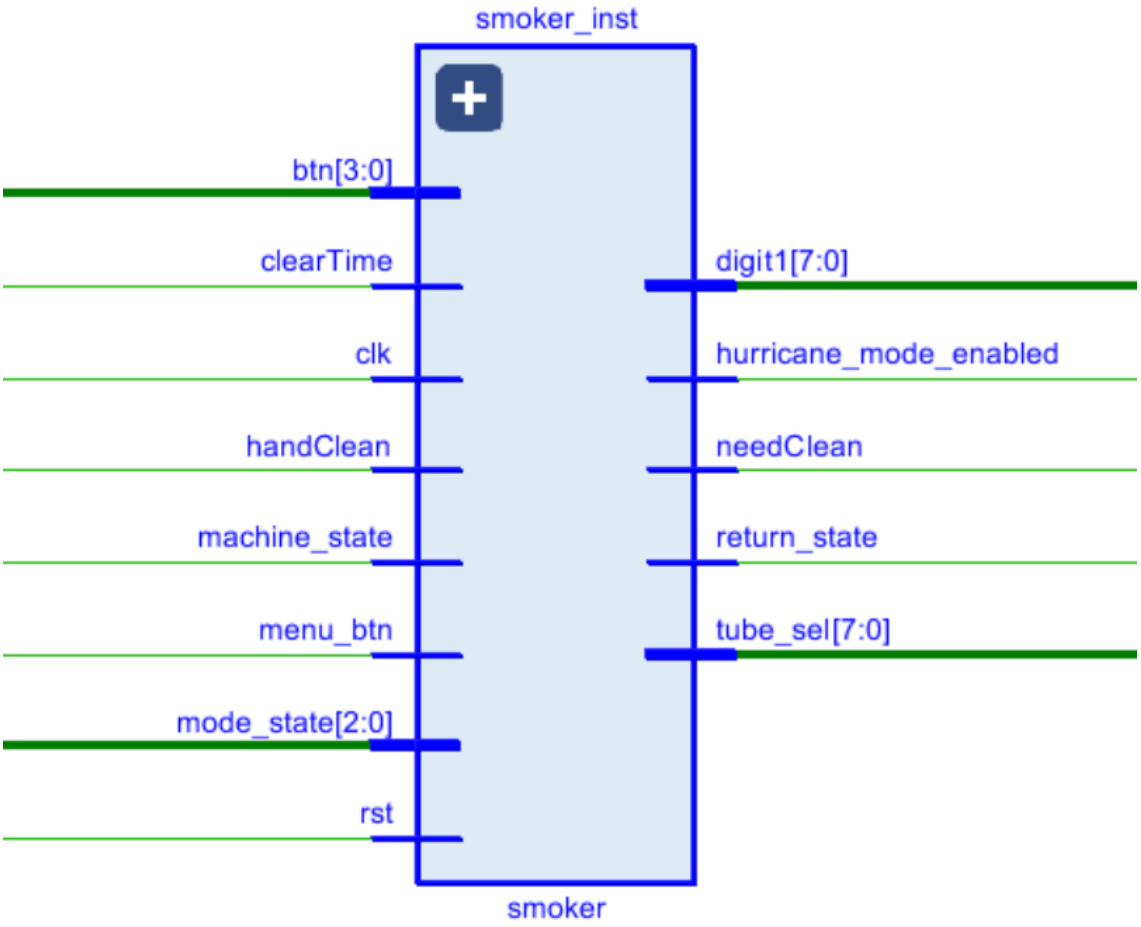


端口名称	I/O	功用描述
in_b4[3:0]	I	输入四比特二进制数字
digit[7:0]	O	输出显示在数码管上的编码



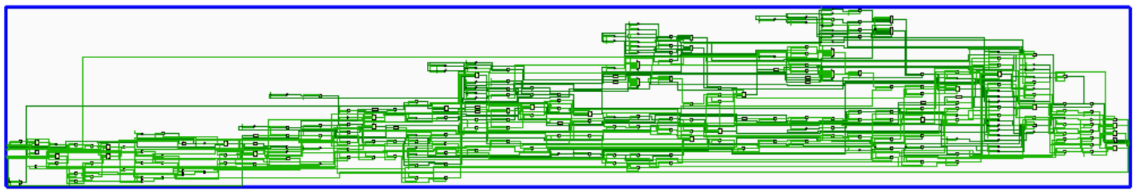
- smoker

用于实现一二三挡位的切换，累计时间的计数，倒计时的计数，接受自清洁模块和手动自清洁的信号重置累计时间以及输出自清洁提示，接受fsm里面的mode_state切换挡位



端口名称	I/O	功用描述
clk	I	系统时钟
rst	I	重置所有设置，参数变量全部初始化
mode_state[2:0]	I	检测fsm里面的状态信号切换挡位
menu_btn	I	检测三档状态下有无按下菜单键重置60s倒计时
handClean	I	手动自清洁信号
btn[3:0]	I	防抖实现
machine_state	I	检测是否开机
clearTime	I	自清洁完成信号，用于重置累计时间
digit1[7:0]	O	输出当前模块需要显示的时间信息在数码管上
digit2[7:0]	O	输出当前模块需要显示的时间信息在数码管上
tube_sel[7:0]	O	作为 TimeDisplay 模块实例化的传入信号，编辑/显示当前时间中会显示对应的内容
return_state	O	传入fsm决定三档结束后返回待机还是二档
hurricane_mode_enabled	O	决定一次开机只能进入一次三档
needClean	O	传出自清洁提示信号

端口名称	I/O	功用描述
meun_btn_pressed	O	查看三档情况下是否按下了菜单键



5. bonus实现说明

本项目为用开发板模拟油烟机。$bonus$ 部分为高级开关机方式，即手势开关机。

下面展示的代码为 $onOffControl$ 模块里 $always$ 里的部分相关代码。（$reset$ 低电平时重置状态，这里不再展示）

操作实现：在 $SW0-R1$ 拨码开关拨上时，表示当前按钮页面为手势开关机页面。（top 文件里实现按钮页面分配，与手势实际功能关系不大，这里不进行展示）按钮页码匹配手势开关机页面时，①先按动左键，即引脚为 $V1$ 的按钮，在有效手势时间（默认为 $5s$）内按动右键，即引脚为 $R11$ 的按钮，实现开机。②先按动右键，有效手势时间（默认为 $5s$）内按动左键，实现关机。③按下按钮开启计时，无论先按动左键或者按动右键，时间超过有效手势时间后，先按动的键失效，所有相关状态恢复初始状态，表现为开/关机失败。

相关变量说明：

变量	含义/表征意义
gesture_btn_state	手势开关机是否启用，表征当前按钮页面是否匹配手势按钮页
left_btn	左键瞬时状态，摁下时为高电平
right_btn	右键瞬时状态，摁下时为高电平
left_ges	左键启用状态，表示左键先前已摁下
right_ges	右键启用状态，表示右键先前已摁下
left_begin	左键先启用，进入开机准备，等待右键信号
right_begin	右键先启用，进入关机准备，等待左键信号
start	手势开关机操作开始执行
counter	正常开关机的计时器 (手势开关机的语句在与正常开关机相同的一个 $always$ 语句块里)
gesture_counter	手势开关机的计时器
second_counter	秒数计时器
over_gesture	效果类似于 $prev$，当手势操作结束后，避免其直接进入下一次开关机准备状态

代码解读：

$reset$ 低电平时清零所有状态。正常开关机和手势开关机在同一个 $always$ 语句下，跨越行数过多且实现简单，不作代码展示。

当 $gesture\_btn\_state = 1'b1$ 时可进行手势开关机操作，说明此时按钮页在手势页。

①所有手势按键状态和开始状态为 $1'b0$ 时 ($!right\_ges \ \& \ !left\_ges \ \& \ !start$)：a) 按左键进入开机准备状态，重置 $gesture\_counter$，$second\_counter$ 和 $counter$，将 $start$ 调整至高电平，$left\_begin <= 1'b1$ 表示进入开机准备。 b) 按右键进入关机准备状态，重置 $gesture\_counter$，$second\_counter$ 和 $counter$，将 $start$ 调整至高电平，$right\_begin <= 1'b1$ 表示进入关机准备。

②左右键都为低电平时将 $over\_gesture$ 恢复低电平，因为此时没有手势按钮摁下，无需避免在上一次手势开关机操作执行后直接进入下一次开关机准备状态。

③进入开机准备 ($left\_begin == 1'b1$) 时，每次 clk 上升沿使秒数计时器 $second\_counter + 1$，每当其到达 $1s$，即 $second\_counter == 10\_000\_000$ （***所有计数器均为 Integer***），$gesture\_counter + 1$。当 $gesture\_counter$ 小于设置的有效手势时间时，按下右键实现开机，除了开关机的所有状态参数恢复初始值；按下左键重新开启开机准备。若 $gesture\_counter$ 大于设置的有效手势时间，所有手势相关状态恢复初始值，开机失败，手势开机失效，状态还原。

④进入关机准备 ($right\_begin == 1'b1$) 时，每次 clk 上升沿使秒数计时器 $second\_counter + 1$，每当其到达 $1s$，即 $second\_counter == 10\_000\_000$ （***所有计数器均为 Integer***），$gesture\_counter + 1$。当 $gesture\_counter$ 小于设置的有效手势时间时，按下左键实现关机，除了开关机的所有状态参数恢复初始值；按下右键重新开启开机准备。若 $gesture\_counter$ 大于设置的有效手势时间，所有手势相关状态恢复初始值，关机失败，手势关机失效，状态还原。

```
//手势开关机
if (gesture_btn_state) begin
    //判断是左键先
    if (left_btn & ~right_ges & ~left_ges & ~start) begin
        if (~over_gesture) begin
            left_begin <= 1'b1;
            gesture_counter <= 0;
            second_counter <= 0;
            start <= 1'b1;
            left_ges <= 1'b1;
            counter <= 0;
        end
    end
    //判断是右键先
    if (right_btn & ~left_ges & ~right_ges & ~start) begin
        if (~over_gesture) begin
            right_begin <= 1'b1;
            gesture_counter <= 0;
            second_counter <= 0;
            start <= 1'b1;
            right_ges <= 1'b1;
            counter <= 0;
        end
    end
    //恢复过长时间摁下按钮的状态检验
    if (~left_btn & ~right_btn) begin
        over_gesture <= 1'b0;
    end

```

```
    if (left_begin) begin
        if (gesture_counter < gesture) begin
            second_counter <= second_counter + 1;
            if (second_counter == second) begin
                gesture_counter <= gesture_counter + 1;
                second_counter <= 0;
            end
            if (right_btn) begin
                //有效手势时间内，按动右键进入关机，进行相应状态变化
                gesture_counter <= 0;
                second_counter <= 0;
                machine_state <= 1'b1;
                left_ges <= 1'b0;
                right_ges <= 1'b0;
                left_begin <= 1'b0;
                right_begin <= 1'b0;
                start <= 1'b0;

                counter <= 0;
                over_gesture <= 1'b1;
            end else if (left_btn) begin
                //有效手势时间内，再次按动左键，重启手势操作
                left_begin <= 1'b1;
                gesture_counter <= 0;
                second_counter <= 0;
                start <= 1'b1;
                left_ges <= 1'b1;
                counter <= 0;

            end
        end else begin
            //超出有效手势时间，左键失效，所有手势相关状态归零
            gesture_counter <= 0;
            second_counter <= 0;
            left_ges <= 1'b0;
            right_ges <= 1'b0;
            left_begin <= 1'b0;
            right_begin <= 1'b0;
            start <= 1'b0;

            over_gesture <= 1'b0;
        end
    end
    if (right_begin) begin
        if (gesture_counter < gesture) begin
            second_counter <= second_counter + 1;
            if (second_counter == second) begin
                gesture_counter <= gesture_counter + 1;
                second_counter <= 0;
            end
            if (left_btn) begin
                //有效手势时间内，按动右键进入关机，进行相应状态变化
                gesture_counter <= 0;
                second_counter <= 0;
                machine_state <= 1'b0;
                left_ges <= 1'b0;
```

```
                right_ges <= 1'b0;
                left_begin <= 1'b0;
                right_begin <= 1'b0;
                start <= 1'b0;

                counter <= 0;
                over_gesture <= 1'b1;
            end else if (right_btn) begin
                //有效手势时间内，再次按动右键，重启手势操作
                right_begin <= 1'b1;
                gesture_counter <= 0;
                second_counter <= 0;
                start <= 1'b1;
                right_ges <= 1'b1;
                counter <= 0;

            end
        end else begin
            //超出有效手势时间，右键失效，所有手势相关状态归零
            gesture_counter <= 0;
            second_counter <= 0;
            left_ges <= 1'b0;
            right_ges <= 1'b0;
            left_begin <= 1'b0;
            right_begin <= 1'b0;
            start <= 1'b0;

            over_gesture <= 1'b0;
        end
    end
end
```

6. 项目总结

5.0 总体感悟与心得

小组三名成员都是第一次接触 $Verilog$ 这门语言，可以说是本次项目让我们真正入门了 $Verilog$ ，而不是像课堂练习或者课后作业那样有针对性地实现某一结构或者功能，小作业的完成颇有照猫画虎的随意感，本次大项目让我们真正认真地理解基础的 $Verilog$ ，实现其功能的综合。我们对于 $Verilog$ 实现项目的一个整体框架有了更好认识，有更丰富的设计文件、限制文件于上板调试经验。项目期间也发现了许多问题，值得从中汲取经验，下次改进。这部分将在后面部分继续详述。

5.1 团队合作

与以往的单人任务与双人任务不同，本次项目由三名组员共同完成，其间便出现了许多关于团队合作相关的问题。

首先，最主要的应该是项目文件管理。任务起初我们便创建了 $github$ 仓库，但是并未实现对于文件的管理。每当有一个模块更新，我们采取的方式是直接上传，这在项目中后期导致我们的仓库极其杂乱。这个问题在我们12月11日创建文件夹整理文件后有所好转。同时，还有一个与项目文件管理相关的问题，即各组员本地的模块文件更新不及时。当大部分功能实现，进行整合的阶段，我们耗费大量时间进行版本核对，代码逐行校对，甚至有时 $debug$ 一个小时发现是自己的版本落后导致的，白白耗费一小时时间。在往后的小组任务里，我们会在顶层设计时便做好文件管理规划，创建文件夹，在更新模块后及时通知队友更新文件。

其次，任务划分应在初步认识任务间联系后再进行，而不是根据分值或者任务量来简单粗暴分配。上一个问题所说的版本校对的其中一个很大原因就是因为我们小组在初期的任务划分时，并没有了解模块间的传入传出关系，只是简单的根据项目要求文档里的板块划分。导致在后期的功能整合环节发现很多功能是冗余重复的，出现 $multiple\ driver$ 等问题。也需要大量时间在其他组员的模块里进行功能添加与修改，这期间需要的代码理解环节也增大了整个项目的任务量，拖慢了项目进程。甚至当代码逻辑过于复杂，而编写该部分代码的同学有事无法脱身，会导致整个项目罢工，等待该同学讲解。

5.2 开发

项目功能开发环节出现的问题可以分为几个阶段。

初期，每个组员对于模块具体功能和整个项目的要求的理解不同，编写的代码没有实现彼此想要的功能。这导致在任务中期，小组整体的功能框架尚不清晰，我们又重新开会讨论了整个项目的要求，明确了整体逻辑和框架构建。为了避免这个问题，我们在下次项目中应当细化顶层模块的构建，在构建途中交流，让彼此明晰项目结构与彼此联系。

中期，项目文件出现版本差异，整合模块需删去大量重复功能，重新调整模块输入输出。版本差异这一点在 5.1 的团队合作中已经说明，这里不再赘述。整合模块需删去大量功能，是因为在早期的模块功能划分时，即顶层设计及下层模块功能划分的时候没有充分理解任务要求以及层级结构，导致大量时间用于调整模块，而不是修改 bug 或增加功能。

后期，核对项目文档要求时，发现部分代码没有按照评分标注编写，缺失功能。这一部分需要我们在写分配到的模块前对于其功能有一个整体认识，不是想到哪儿写到哪儿，有漏洞再加补丁。这样会把问题累计在项目后期，让本不富裕的末期调试时间雪上加霜。对于模块整体功能的理解也有利于整个代码的逻辑严密性以及可读性。

除了出现的问题，还有一些是可以优化或者改进的。

① 当任务越写越多或 $debug$ 时，小组成员开始停止写注释。注释应该坚持到底，而不是在项目后期添改。

② 有时为了功能测试，命名时会随意命名或不再遵照相同命名规则。应从始至终。

③ 开发板上的按钮分配或者模式是有重复的，可以进一步优化。$i.e.$ 在开发板上有两套操作可以进入编辑模式，但是修改的内容不同。这部分可以合并。

④ 在以按钮为主要操作元件时，应加入防抖功能。

5.3 测试

测试环节的主要问题是板子分配问题。 $FPGA$ 开发板只有一块，而在具体功能的实现验证时，每个小组成员都需要上板测试。尽管可以通过三个人找个讨论间一起敲代码的方式来避免小组同时只有一个人开工的尴尬情形，但依旧效率低下。而且在最后答辩的过程中，发现我们时间显示的小时位是有问题的，这个我们并没有发现，因为从来没有测试过小时部分的代码。写仿真文件便是一个很好的解决方法，可以高效找到代码 bug 所在，也让项目进展实现并行。

7. Project出题的想法和建议

6.0 尝试分析今年的出题思路

由于我们小组选择的是抽油烟机项目,我这里主要分析一下抽油烟机项目中我认为的难点主要有以下几个:

- **系统不同状态的管理**:
  抽油烟机项目的设计不仅仅是对单一功能的实现，更注重如何管理不同的状态和模式切换。
  例如在开机状态下包括待机、抽油烟、以及自清洁模式，这些模式之间的切换和不同的操作流程使得我们深刻的理解到了很多fsm的相关知识。

- **数码管的使用(时间显示,调整与倒计时功能的显示)**:
  项目要求抽油烟机在特定工作模式下实现计时功能，例如抽油烟的累计时长和自清洁模式的倒计时等。
  这个部分里头比较困难的有时间调整中的进位退位操作,top模块里时间显示的逻辑编写和FSM如何联动等等问题,所以我觉得数码管该如何使用也是本课程项目的难点之一。
- **其它输入输出设备的使用(bonus部分)**:
  题目要求学习使用除了开发板之外的其它输入输出设备进行操作和状态标识，这鼓励了同学们去阅读手册和查阅网络上的资料进行自学,虽然最后我们小组因为时间不够并没有实现好bonus部分,但是我认为这样的bonus可以激发同学们的学习热情,并且在时间充足的情况下也不会让人无从下手,所以应该继续沿用。

6.1 明年出题建议：密码锁设计

针对明年的出题建议，我们认为可以考虑设计一个 **密码锁系统**，这个系统不仅仅是考察基本的输入输出操作，还可以深入到安全性、状态管理、与外部设备的交互等多个方面。下面是一些可能的出题思路：

1. 基本功能

- **密码输入与验证**:
  系统要求用户通过按钮输入一个预设的密码，如果密码正确，打开锁门；如果密码错误，提示用户并重新输入。这可以通过7段数码管来实现显示。
- **密码更改功能**:
  除了基本的密码验证，还可以要求在一定条件下允许用户修改密码，例如输入正确的旧密码后，可以设置一个新密码。
  这个功能要求系统需要设置多个状态,可以提高学生的FSM设计能力。
- **输入错误次数限制**:
  系统可以设置限制，比如用户输入错误次数超过3次后自动锁定，并通过输出信号来进行警告。
  这个功能可以考察学生如何实现输入次数的累积，并处理错误限制的逻辑,也需要状态控制相关的内容。

2. 高级功能与优化

- **使用外部设备**:
  例如，在尝试过多次数后，触发警报功能使用蜂鸣器播放警告消息。
- **带有时间限制的密码输入**:
  我们必须要在指定的时间内完成密码的设置和输入,并且在超时后自动返回待机状态,这样就不用怕打开密码设置界面忘记关掉被别人篡改了。
- **密码锁的安全性增强**:
  密码锁不仅仅是输入数字，还可以设计多种输入方式（如按键顺序、组合键等），使其更具安全性。这里也可以作为bonus的使用其他设备输入的部分。

通过这一项目，我相信下一届的学弟学妹们也可以非常愉快而充满乐趣的学习verilog代码学习,提升自己的HDL编程能力。

项目报告到此结束，感谢您的阅读！