

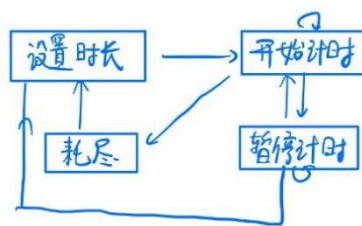
数字系统课程设计报告

——多功能时钟

一、设计过程及原理

1. 计时模块

最基本的倒计时模块要有以下过程：（当然，正计时模块类似，不过需要注意 24h 的进位问题，时钟模块直接调用正计时模块并将计时使能置 1 即可。）



这个过程很适合用状态机来写，不过我们先考虑最核心模块：

接口：毫无疑问，我们需要时钟和 reset（时钟要接 1s 分频器）；然后是小
时、分钟、秒的个位和十位（共六个），这一部分分为输入设置时间和输出当前
倒计时时间，输出可以直接接数码管或 OLED 屏模块；还有计时结束的标志
timeout；最后还有两个设置使能，包括设置使能和计时使能（开始/暂停计时）。

模块功能描述：

- 1、当设置使能 $\text{SetEN}=1$ 时，将输入设置值赋给当前时间。
- 2、当计时使能 $\text{EN}=1$ 时，开始计时； $\text{EN}=0$ 时，暂停计时。
- 3、 $\text{reset}=1$ 时，当前时间置 00: 00: 00。
- 4、当时间耗尽（时间各位全 0），timeout 置 1。

关于计时功能，最基本的是秒个位上的（反向）计数器。即在时钟上升沿，秒个位减去 1，若已达到 0 则下一时刻赋 9，同时向秒十位发送退位（-1）信号。下面给出了局部代码。以此类推，我们便可写出核心模块的代码。

```

always @(posedge clk_1s or posedge reset) begin
    if (EN) begin
        if (timer_sec_ones > 0) begin
            timer_sec_ones <= timer_sec_ones - 1;
        end
        else begin
            if (timer_sec_tens > 0) begin
                timer_sec_tens <= timer_sec_tens - 1;
                timer_sec_ones <= 4'd9;
            end
        end
    end
end

```

然后是设置时间模块；这里使用五向按键中的四个方向键（中间按键用来后续的状态机跳转），左右调整“光标”位置，上下调整数值大小。

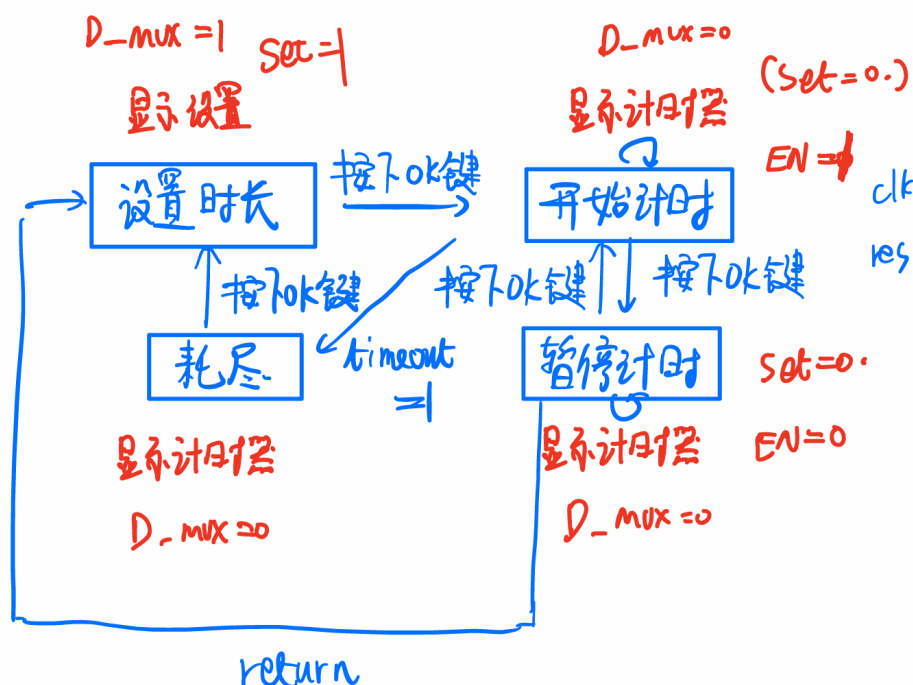
```

// 选择位移动逻辑
always @(posedge clk or posedge reset) begin
    //.....
    else begin
        if (left) begin
            sel <= (sel == 3'd6) ? 3'd0 : sel + 1;
        end
        else if (right) begin
            sel <= (sel == 3'd0) ? 3'd6 : sel - 1;
        end
    end
end
// 数值调整逻辑
always @(posedge clk or posedge reset) begin
    //.....
    else begin
        case (sel)
            3'd0: begin // 秒个位(s0)
                if (up) s0 <= (s0 == 4'd9) ? 4'd0 : s0 + 1;
                if (down) s0 <= (s0 == 4'd0) ? 4'd9 : s0 - 1;
            end
            3'd1: begin // 秒十位(s1)
                if (up) s1 <= (s1 == 4'd5) ? 4'd0 : s1 + 1;
                if (down) s1 <= (s1 == 4'd0) ? 4'd5 : s1 - 1;
            end
        end
    end
end

```

接下来处理数据通路：除了前述模块的输入（设置时间输入、计时/设置使

能、方向键)和输出(当前时间输出、时间耗尽标志)等,还会有若干个数据选择器,在设置时间和当前时间之间做出选择,赋给最终的输出(这个输出可以直接接数码管或 OLED 屏幕)。其中,数据通路的信号输入/输出与状态机一一对应。

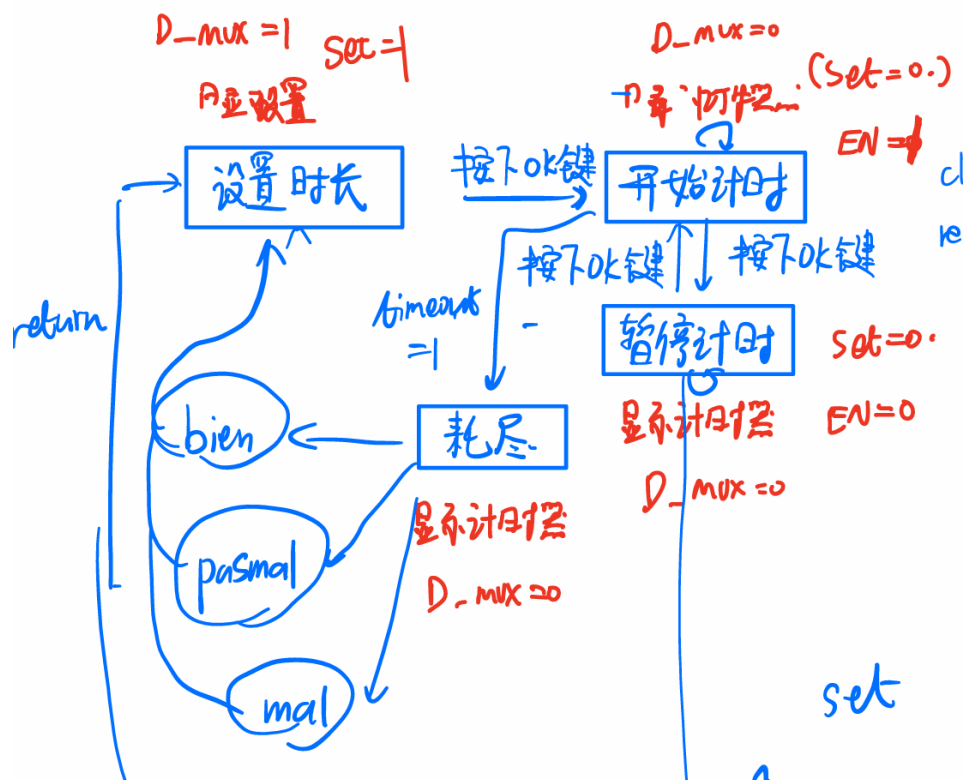


状态机部分,状态的转换主要通过五向按键的中间键来实现,除此之外还有时间耗尽的 `timeout` 标志,重新设置的按键。只需要按照三段式的顺序,状态机模块也很容易就能写出来。

2. 专注模块

在倒计时结束后,系统等待用户的反馈,左、确认、右键代表了完成任务的专注度/完成进度。根据用户的不同选择,将前面设置的这段倒计时存入不同的寄存器。当“bien”(特别好)时长达到一定值时,触发随机奖励。

将时间根据反馈存入不同寄存器可以在前述倒计时模块状态机添加三个并行状态,并添加对应的控制字(对应寄存器写入使能为1),之后立即转到设置状态。



至于随机奖励，我们采用了线性反馈移位寄存器（Linear Feedback Shift Register, LFSR）来生成奖励编号。所谓反馈就是在移位后，以寄存器中已有的某些序列作为反馈函数的输入，在函数中经过一定的运算后，将反馈函数输出的结果填充到移位寄存器的最左端。线性则是说反馈函数是线性的。以下给出本次使用的 LFSR 的参数：

(1) 寄存器宽度：4 位 ($out[3:0]$)

(2) 反馈多项式： $x^4 + x^3 + 1$ （反馈位计算： $out[3] \oplus out[2]$ （最高位与次高位异或））

(3) 初始种子值：4'b1010（十进制 10）

工作原理：

(1) 复位(reset)时：寄存器被初始化为 4'b1010

(2) 正常工作时(每个时钟上升沿)：寄存器左移 1 位 → 由反馈函数计算得出最低位（具体操作： $out \leftarrow \{out[2:0], feedback\}$ ）

下面是 LFSR 的代码：

```

module LFSR(
    input clk,
    input reset,
    output reg [3:0] out
);
wire feedback = out[3] ^ out[2];
always @(posedge clk or posedge reset) begin
    if (reset) begin
        out <= 4'b1010;
    end
    else begin
        out <= {out[2:0], feedback};
    end
end
end
endmodule

```

目前的奖励触发机制是：专注时长（timeB）每达到一固定值（比如 5 小时），就触发随机奖励。下面给出部分代码：

```

always @(posedge clk) begin
    if (reset) begin
        reward_available <= 0;
        reward_id <= 0;
        gain <= 0 ;
    end
    else if (daily_total_time >= 300*(gain+1)) begin
        reward_available <= 1;
        reward_id <= random_num;
        gain <= gain + 1;
    end
    else if (reward_available==1) begin
        reward_available <= 0;
    end
end
end
endmodule

```

3. 事项提醒模块

这个模块需要以下功能：

- (1) 事项设置：同时间设置模块，左右控制光标，上下控制数值大小。
- (2) 事项查看：上下浏览各个事项（还有+10 键，按下后可以更快速地翻阅）。
- (3) 事项提醒：当当前时间和存储的时间相同时，提醒并输出事项编号。

毫无疑问我们需要一个存储模块。我们可以指定若干个寄存器，但这无疑会占用大量资源（特别是后续要存储字模数据的情况下）。于是我们选用 BRAM 来存储。

```
reg [3:0] schedule_rom [0:499];
```

这样写综合工具会自动推断为 BRAM。事项编号可由 BRAM 位单元位置给出，只需存储小时、分钟的个位和十位共 4 个 4 位二进制数（后期又增加了星期几的存储）。

首先先对所有 BRAM 位单元初始化（时间全 8 是为了避免大半夜吵醒你）。

```
integer i;
initial begin
    for (i = 0; i < 100; i = i + 1)
    begin
        schedule_rom[5*i+0] <= 4'h8;
        schedule_rom[5*i+1] <= 4'h8;
        schedule_rom[5*i+2] <= 4'h8;
        schedule_rom[5*i+3] <= 4'h8;
        schedule_rom[5*i+4] <= 4'h0;
    end
end
```

然后写入预置事项（如上课、浇花、运动等）。

```
begin
    //课程 1-1
    schedule_rom[5*60+0] <= 4'h0;
    schedule_rom[5*60+1] <= 4'h7;
    schedule_rom[5*60+2] <= 4'h3;
    schedule_rom[5*60+3] <= 4'h0;
    schedule_rom[5*60+4] <= 4'h1;
    //课程 1-2
    schedule_rom[5*61+0] <= 4'h0;
    schedule_rom[5*61+1] <= 4'h9;
    schedule_rom[5*61+2] <= 4'h3;
    schedule_rom[5*61+3] <= 4'h5;
    schedule_rom[5*61+4] <= 4'h1;

    //.....
end
```

接下来设置事项：

```

if (setEN) begin
    schedule_rom[5*(10*ss1 + ss0) + 0] <= sh1;
    schedule_rom[5*(10*ss1 + ss0) + 1] <= sh0;
    schedule_rom[5*(10*ss1 + ss0) + 2] <= sm1;
    schedule_rom[5*(10*ss1 + ss0) + 3] <= sm0;
    schedule_rom[5*(10*ss1 + ss0) + 4] <= sweek;
end

```

下面是事项匹配：先检查时间是否有效，然后逐位比较，当时间、星期完全匹配时（也存在某事项一周七天重复提醒的可能，例如运动），输出匹配标准，并输出事项编号。

```

always @(posedge clk) begin
    for (i = 0; i < 90; i = i + 1) begin
        if ((schedule_rom[5*i + 0] != 4'h8) ||
            (schedule_rom[5*i + 1] != 4'h8) ||
            (schedule_rom[5*i + 2] != 4'h8) ||
            (schedule_rom[5*i + 3] != 4'h8))
            begin
                if ((schedule_rom[5*i + 0] == th1) &&
                    (schedule_rom[5*i + 1] == th0) &&
                    (schedule_rom[5*i + 2] == tm1) &&
                    (schedule_rom[5*i + 3] == tm0) &&
                    ((schedule_rom[5*i + 4] == tweek)
                     || (schedule_rom[5*i + 4] == 0)))
                    begin
                        ontime <= 1;
                        schnum <= i;
                    end
            end
    end
end

```

最后就是显示事项。直接由显示编号给出。

```

dh1 <= schedule_rom[5*disnum + 0];
dh0 <= schedule_rom[5*disnum + 1];
dm1 <= schedule_rom[5*disnum + 2];
dm0 <= schedule_rom[5*disnum + 3];
dweek <= schedule_rom[5*disnum + 4];

```

```

always @(posedge clk) begin
    for (i = 0; i < 90; i = i + 1) begin
        if ((schedule_rom[5*i + 0] != 4'h8) ||
            (schedule_rom[5*i + 1] != 4'h8) ||
            (schedule_rom[5*i + 2] != 4'h8) ||
            (schedule_rom[5*i + 3] != 4'h8))
        begin
            if ((schedule_rom[5*i + 0] == th1) &&
                (schedule_rom[5*i + 1] == th0) &&
                (schedule_rom[5*i + 2] == tm1) &&
                (schedule_rom[5*i + 3] == tm0) &&
                ((schedule_rom[5*i + 4] == tweek)
                 || (schedule_rom[5*i + 4] == 0)))
            begin
                ontime <= 1;
                schnum <= i;
            end
        end
    end
end

```

状态机则包括设置、查看、提醒三个状态。

4. 天气/穿衣提醒模块

该模块基于 DHT11 温湿度传感器采集环境数据，并结合温度高低与湿度大小进行判断，给出穿衣建议与是否需要携带雨具的提醒，同时通过 OLED 屏幕实时显示相关信息。

(1) 温湿度传感器采集模块：DHT11 模块内部集成温度传感器、湿度传感器及 ADC 转换器，可以直接输出处理后的温湿度数据，其中包括 8 bit 湿度整数值、8 bit 湿度小数值、8 bit 温度整数值以及 8 bit 温度小数值。

(2) main 控制模块：以 1s 为周期，定时触发采集请求，并在 dht11_done 信号拉高时锁存有效数据，向后级模块提供稳定的温湿度输入。


```
// 采样间隔控制(约 1 秒一次)
always @(posedge sys_clk or negedge rst_n) begin
    if (!rst_n) begin
        sample_timer <= 30'd0; //原: 24
        dht11_req <= 1'b0;
    end else begin
        // 50MHz 时钟下, 50,000,000 个周期=1 秒; 100MHZ
        if (sample_timer == 30'd100_000_000) begin
            dht11_req <= 1'b1; // 发出采集请求
            sample_timer <= 30'd0; // 重置计数器
        end else begin
            dht11_req <= 1'b0; // 保持低电平
            sample_timer <= sample_timer + 1'b1; //
计数器递增
        end
    end
end
```

```
// 数据锁存逻辑
always @(posedge sys_clk or negedge rst_n) begin
    if (!rst_n) begin
        // 复位时清零
        TempH <= 8'd0;
        TempL <= 8'd0;
        HumidH <= 8'd0;
        HumidL <= 8'd0;
    end else if (done) begin
        // 采集完成时更新数据
        TempH <= TempH0;
        TempL <= TempL0;
        HumidH <= HumidH0;
        HumidL <= HumidL0;
    end
end
end
```

(3) clothing_advice 模块: 根据温度值和湿度值, 输出穿衣建议, 并判断是否需要携带雨具。

①温度判断穿衣建议:

≥30° C: 短袖; 21~30° C: 长袖; 16~20° C: 薄外套; 6~15° C: 厚外套;
≤5° C: 羽绒服

②湿度判断带伞建议:

湿度 $\geq 75\%$: 提醒带伞

(4) temp_to_digits 模块: 将 8 bit 二进制形式的温、湿度整数部分的数值, 拆分为十进制的十位 (tens) 与个位 (ones), 用于显示驱动。

```
//温/湿度转换为数码管显示输出
module temp_to_digits(
    input [7:0] temperature,
    output reg [3:0] tens,
    output reg [3:0] ones
);
    always @(*) begin
        tens = temperature / 10;
        ones = temperature % 10;
    end
endmodule
```

(5) clothes_system 顶层模块: 集成数据采集、穿衣建议判断、显示控制三大核心功能。

(6) OLED 显示模块: 实时控制 OLED 屏显示内容包括温湿度的数值、单位, “穿衣建议” 固定标题, 并根据传感器输入的数据动态显示“短袖”“伞”等字符。

该模块设计优势有如下两点:

(1) 模块化设计结构清晰、功能独立, 便于调试与后续功能拓展。

(2) DHT11 采用数字信号输出方式, 避免了模拟信号带来的采样误差与外设复杂性问题, 提升了温湿度采集的可靠性与实时性, 适用于实际生活场景。

5. OLED 显示模块

使用 1.3 寸 128×64OLED 显示屏, 驱动芯片为 SSD1306。

OLED 的存储区域分为 8 个 page, 每个 page 下面共有 128bit, 如下图所示:

	列 (COL0~127)						
	SEG0	SEG1	SEG2	SEG125	SEG126	SEG127
行 (COM0~63)	PAGE0						
	PAGE1						
	PAGE2						
	PAGE3						
	PAGE4						
	PAGE5						
	PAGE6						
	PAGE7						

CSDN @ValentineHP

OLED 的数据存储模式：当我们向 OLED 的 GRAM 发送显示数据的过程中，OLED 内部共有三种处理模式：

模式一：当我们指定了一个 page 和某一列的时候，每写一个数据，列会自动加一，当写到 page 的最后一列的时候，列数，会自动跳转到第零列，需要手动的切换 page。

模式二：与模式一不同的时候，当写达到 page 的最后一列的时候，列数跳转到第零列的同时，page 数也会加一。

模式三：与前两个模式不同，模式三，是一列一列的写，每写完一个数据，page 数加一，当加到最后一个 page 的时候，列数加一，page 跳转到第一个 page。

OLED 关键命令介绍：

- (1) 0xAE/0xAF：对应着开启 OLED 显示和关闭 OLED 显示
- (2) 0x20-0x22：对应着上面的三种 OLED 数据存储模式，默认为 0x22，模式一
- (3) 0x00-0x0F：设置列地址的低四位，默认为 0x00，
- (4) 0x10-0x1F：设置列地址的高四位，默认为 0x10，
- (5) 0xB0-0xB7：设置 page，第四位表示 page。

IIC 驱动 OLED 数据格式：

驱动 OLED 分为写数据和写命令(读暂时不考虑)。写命令就是配置的过程，写数据就是写入 GRAM 中进行显示的过程。

IIC 数据格式：OLED 地址 + 命令 / 数据 + 值

- (1) OLED 地址，就是 IIC 协议中的从机地址。
- (2) 命令/数据中，0x00 表示接下来的值代表命令，0x40 表示接下的值表示

数据，存入 GRMA。

(3) 值，具体的命令或者数据。

也就是说每一次 IIC 需要传输 24bit，3 个字节的数据。

OLED 初始化：共需要配置 26 个命令，第一个 0xAE 命令，关闭 OLED 显示。最后一个是 0xAF 命令，开启 OLED 显示，没有配置模式，直接使用的默认的模式一，配置完成后，可以看到 OLED 屏被点亮，内容是杂乱的。

初始化后，就可以显示内容，仅需要配置 page 和起始列三个命令，然后写入数据即可，配置页和列地址，只有在模式一有效，也就是上电默认的模式。

简而言之，显示原理就是：一列一列从上至下显示。

以上这些内容，包括显示 8 行 16 列的点阵，在网上都可以找到现成的代码。接下来要做的事情，就是为了让 OLED 显示屏模块可以跟数码管一样，指哪打哪，让它在 8 行 4 列共 32 个空位显示 16×16 大小的字符。问题是：如何确实显示什么，以及显示在哪里。

确定显示位置的关键变量即下面代码中的 page、column 和 shift（shift 控制向右偏移的像素点）。

```
begin
  case(showfont_index)
    'd0:showfont_data_reg<={8'h78,8'h00,8'hB0 + page};
    'd1:showfont_data_reg<={8'h78,8'h00,8'h00 + shift};
    'd2:showfont_data_reg<={8'h78,8'h00,8'h10 + column};
    default:showfont_data_reg<={8'h78,8'h40,fontdata};
  endcase
end
```

一个字符有 16x16 个像素点，即 16 行，16 列。行好办，但是列呢？一个 page 只有 8 行，所以为了完整显示一个字必须得切换 page。这里引入变量 up。up=0 时取字的上半部分，up=1 时取字的下半部分。不难看出 up 是在 0、1 之间跳变的（0、1、0、1……）。

接下来，当 up=0 时，说明该显示下半部分了，page+1。当 up=1 时，该显示下一个字的上半部分了，这里还牵扯到是不是显示到本行的最右边，也就是要不要换行。当没有达到最右边时（page<7），接下来 page+1，column+1，num+1（表明要显示下一个字）；到最右边时（page=7），page+1，column 归 0，num+1。

当显示到最后一个字的下半部分时，全部归 0。

接下来，如何同时显示 32 个字符呢？原理类似于数码管在极短的时间内分别点亮不同的数码管，利用人眼视觉暂留效应呈现同时显示的效果。只不过我们在刷新完成一个字符后，它直到再次刷新前都不会再发生变化，而这种变化的最小单位是 1s，于是我们可以有一个 num 变量，显示不同字符时从 0 不断+1，最终回到 0，同时在 32 个待显示的字符里找到对应要显示的字（这里需要数据选择器）。

```

else if (EN) begin
    up <= ~up;
    if (up == 1'b0) begin
        page <= page + 1'b1;
    end
    else begin
        if (column < 3'd7) begin
            page <= page - 1'b1;
            column <= column + 1'b1;
            num <= num + 1'b1;
        end
        else begin
            page <= page + 1'b1;
            column <= 3'd0;
            num <= num + 1'b1;
        end
    end
    if ((page == 3'd7) && (column == 3'd7) && (up == 1'b1))begin
        page <= 3'd0;
        column <= 3'd0;
        up <= 1'b0;
        num <= 7'd0;
    end
end
end

```

至于要显示什么，就需要字符存储模块了。和上述事项存储一样，都使用了 BRAM。但是存些什么东西呢？存的是像素点亮不亮的信息。如上文所述，屏幕是一列一列从上至下显示的。比如 1111111 就是这一列全亮，000110000 就是这一列第四个和第五个点亮。另外由于取字模软件的输出是 16 进制，所以 Verilog 代码最好也是 16 进制的。

```

    reg [7:0] font_rom [0:8191];
initial begin
    font_rom[32*0 + 0] = 8'h00;
    font_rom[32*0 + 1] = 8'hE0;
    font_rom[32*0 + 2] = 8'h10;
    font_rom[32*0 + 3] = 8'h08;
    font_rom[32*0 + 4] = 8'h08;
    font_rom[32*0 + 5] = 8'h10;
    font_rom[32*0 + 6] = 8'hE0;
    font_rom[32*0 + 7] = 8'h00;
    font_rom[32*0 + 8] = 8'h00;
    font_rom[32*0 + 9] = 8'hE0;
    font_rom[32*0 + 10] = 8'h10;
    font_rom[32*0 + 11] = 8'h08;
    font_rom[32*0 + 12] = 8'h08;
    font_rom[32*0 + 13] = 8'h10;
    font_rom[32*0 + 14] = 8'hE0;
    font_rom[32*0 + 15] = 8'h00;
    font_rom[32*0 + 16] = 8'h00;
    font_rom[32*0 + 17] = 8'h0F;
    font_rom[32*0 + 18] = 8'h10;
    font_rom[32*0 + 19] = 8'h20;
    font_rom[32*0 + 20] = 8'h20;
    font_rom[32*0 + 21] = 8'h10;
    font_rom[32*0 + 22] = 8'h0F;

```

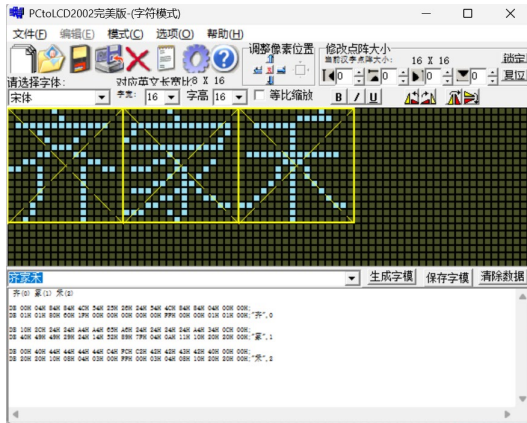
下面就是从 ROM 取出显示数据了。

```

    wire [12:0] addr = {font, 5'b0} + (up ? 16 : 0) +
index[4:0];
    reg [7:0] data_reg;
    always @(posedge clk) begin
        data_reg <= font_rom[addr];
    end

```

接下来就是通过软件获取字模数据并转换为 Verilog 代码，这里可以使用 python 辅助完成。



结果已追加到 font_rom_init.txt

Verilog字体ROM生成工具 (输入q退出)

请输入字符编号(0-255): 200
请输入字模数据(DB格式, 以空行结束输入):
DB 00H 04H 84H 84H 4CH 54H 25H 26H 24H 54H 4CH 84H 84H 04H 00H 00
H
DB 01H 01H 80H 60H 1FH 00H 00H 00H 00H 00H FFH 00H 00H 01H 01H 00
H

生成的Verilog代码:
font_rom[32*200 + 0] = 8'h00;
font_rom[32*200 + 1] = 8'h04;
font_rom[32*200 + 2] = 8'h84;
font_rom[32*200 + 3] = 8'h84;
font_rom[32*200 + 4] = 8'h4C;
font_rom[32*200 + 5] = 8'h54;
font_rom[32*200 + 6] = 8'h25;
font_rom[32*200 + 7] = 8'h26;
font_rom[32*200 + 8] = 8'h24;
font_rom[32*200 + 9] = 8'h54;
font_rom[32*200 + 10] = 8'h4C;

取字模软件

转换成 Verilog 代码的 Python 程序



128	基	189	奶
129	础	190	茶
130	法	191	黄
131	语	192	烟
132	视	193	鸡
133	听	194	汉
134	说	195	堡
135	电	196	香
136	子	197	酥
137	路	198	披

字模选项

最终字符库 (部分)

最终得到的 main 模块, 直接在 F0~F31 端口输入字符编号即可实现字符显示。

```
main u_main (
    .sys_clk(clk),
    .rst_n(reset),
    .OLED_SCL(OLED_SCL),
    .OLED_SDA(OLED_SDA),
    .F0(p2), .F1(p1), .F2(63), .F3(p0), .F4(k0), .F5(st1),
    .F6(st2), .F7(st0), .F8(70+Dis5),
    .F9(70+Dis4), .F10(98+clk_1s), .F11(70+Dis3),
    .F12(70+Dis2), .F13(98+clk_1s), .F14(70+Dis1),
    .F15(70+Dis0), .F16(80+Dis5), .F17(80+Dis4),
    .F18(98+clk_1s), .F19(80+Dis3), .F20(80+Dis2),
    .F21(98+clk_1s), .F22(80+Dis1), .F23(80+Dis0),
    .F24(b10), .F25(b11), .F26(b12), .F27(b13),
    .F28(type2), .F29(type1), .F30(type0), .F31(10*s1+s0)
);
```

6. 课程表模块

课程表模块只涉及课程设置和查看, 课程提醒由之前的事项提醒来做 (事项 60~84, 并通过额外模块对事项编号和课程名称、上课地点进行映射)。还是像

之前的模块那样，一个设置模块，一个存储模块。不过这里是在浏览的时候原地修改某一时段的课程（浏览到哪一个就设置哪一个），而不是之前还需要额外指定位置，原因是课程数量和上课时间都不算太多。

课程存储使用 25 个寄存器，每个寄存器（对应一个时间段）里存储课程编号。

```
// 课程存储 ROM（25 个时间段，每个时间段存储一个课程编号）
reg [4:0] course_rom [0:24];
integer i; // 循环变量
initial begin
    course_rom[0] <= 5'd6;
    course_rom[1] <= 5'd5;
    //course_rom[2] <= 5'd;
    course_rom[3] <= 5'd1;
    //.....
end
always @(posedge clk or posedge reset) begin
    if (reset) begin // 异步复位
        for (i = 0; i < 25; i = i + 1) begin
            course_rom[i] <= 5'd0; // 复位所有时间段
        end
    end
    else begin
        if (setEN) begin
            course_rom[disnum] <= set_course; // 存储课程编号
        end
    end
end
end
```

接下来要对事项编号和课程名称、上课地点进行映射。这个模块也类似于前面写的所有存储模块（前面的奖励显示模块也和这个模块类似）。


```

reg [9:0] coursedis_rom [0:319];
initial begin
//课程名称 1 基础法语
coursedis_rom [10*1+0] = 128;
coursedis_rom [10*1+1] = 129;
coursedis_rom [10*1+2] = 130;
coursedis_rom [10*1+3] = 131;
coursedis_rom [10*1+4] = 99;
coursedis_rom [10*1+5] = 99;
coursedis_rom [10*1+6] = 99;
coursedis_rom [10*1+7] = 99;
//上课地点
coursedis_rom [10*1+8] = 138;
coursedis_rom [10*1+9] = 141;
end
// 课程名称显示输出
always @(*) begin
if (disEN)begin
    course7 = coursedis_rom [disnum*10+0];
    course6 = coursedis_rom [disnum*10+1];
    course5 = coursedis_rom [disnum*10+2];
    course4 = coursedis_rom [disnum*10+3];
    course3 = coursedis_rom [disnum*10+4];
    course2 = coursedis_rom [disnum*10+5];
    course1 = coursedis_rom [disnum*10+6];
    course0 = coursedis_rom [disnum*10+7];
    classroom1 = coursedis_rom [disnum*10+8];
    classroom0 = coursedis_rom [disnum*10+9];
end
end
end

```

7. 提醒模块

提醒方法采用了蜂鸣器和屏幕闪烁“到点儿了”，并给出具体事件（如倒计时、上课、事项等）。

倒计时耗尽或事项时间匹配等触发条件会触发提醒，当用户反馈后停止提醒。下面模块的输出是提醒的使能，可以连接到蜂鸣器输入。但是屏幕闪烁会麻烦一点。闪烁无非就是在两个字符（提醒+空白）之间横跳，可以用数据选择器和 1s 时钟来实现，时钟的 01 跳变，刚好对应数据选择器在这两个字符之间的交替显示。但是不需要提醒的时候就不能显示字符（空白）。所以要把空白字符序号放在选择器 0 端，并接一个三态门。输入是时钟，输出接选择器的选择端，使能接

提醒使能。由此可以实现以下功能，不提醒时，三态门高阻，选择器选择 0（空白）；提醒时，三态门导通，选择器选择在 0、1 之间随着时钟交替变化，也就实现了闪烁。

```
always @(posedge clk) begin
    ontime_prev <= ontime;
end
always @(posedge clk) begin
    if (know) begin
        werwer <= 1'b0; // know 为高时清除输出
    end
    else if (ontime && !ontime_prev) begin
        werwer <= 1'b1; // 检测到 ontime 上升沿时置位输出
    end
end
end
```

关于蜂鸣器，我们设计了静音开关。当开关拨 1 时，只有屏幕提醒而蜂鸣器不响，此时，蜂鸣器恒不触发（我们使用的蜂鸣器是低电平触发，所以要想不触发就要置 1）。

```
module mute(
    input in,
    input EN,
    output reg out
);
    always @(*) begin
        if (EN)
            out = 1'b1;
        else
            out = in;
        end
    end
endmodule
```

8. 顶层模块整合

至此我们就得到了若干大模块：倒计时-专注模块、时钟模块、事项提醒模块、课程表模块等等，现在要做的是把他们整合到一个顶层模块里。这里主要涉及到显示屏要显示哪个功能的信息，以及按钮要控制哪个功能（按钮复用）。于是我们就需要若干数据选择器和多路开关（根据地址选择不同的输出）。地址则用功能编号切换（可以简单用按钮来递增，或是使用状态机，可以实现跳转但略微麻烦）。

```

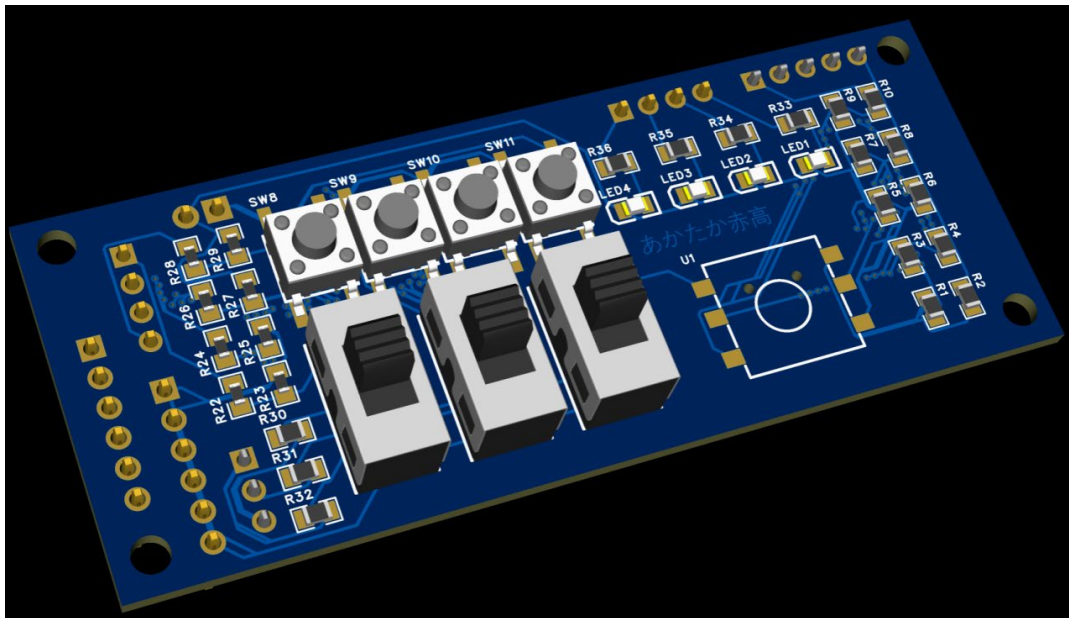
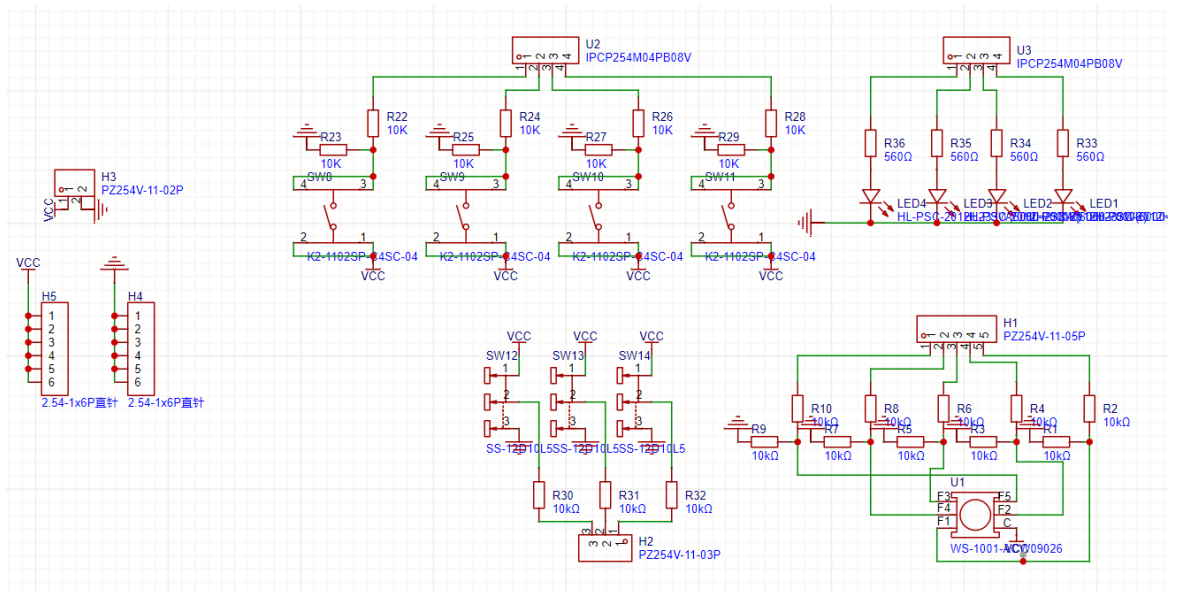
module XUM1(
    input in,
    input [2:0]S,
    output reg out0,
    output reg out1,
    output reg out2,
    output reg out3,
    output reg out4
);
always @(*) begin
    case (S)
        3'd0: begin
            out0 = in;
            out1 = 1'b0;
            out2 = 1'b0;
            out3 = 1'b0;
            out4 = 1'b0;
        end
        3'd1: begin
            out0 = 1'b0;
            out1 = in;
            out2 = 1'b0;
            out3 = 1'b0;
            out4 = 1'b0;
        end
        //.....
    endcase
end
endmodule

```

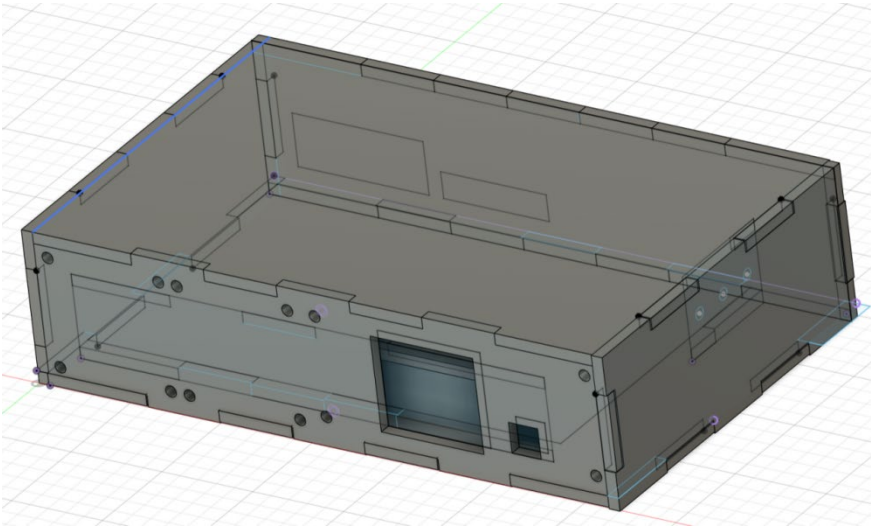
9. 按键开关移设及外壳制作

本项目涉及众多外设，连线混乱，如果就这么一块板子拖着一众外设，将会极不美观，同时外接按钮需要额外接入电阻，众多的杜邦线和大个的面包板，不利于我们控制时钟的大小。所以我们需要制作一个外壳，同时需要在时钟的正面设置单独的开关和按钮（否则为了碰到板子上的开关和按键把整个前面做空也是不美观的）。

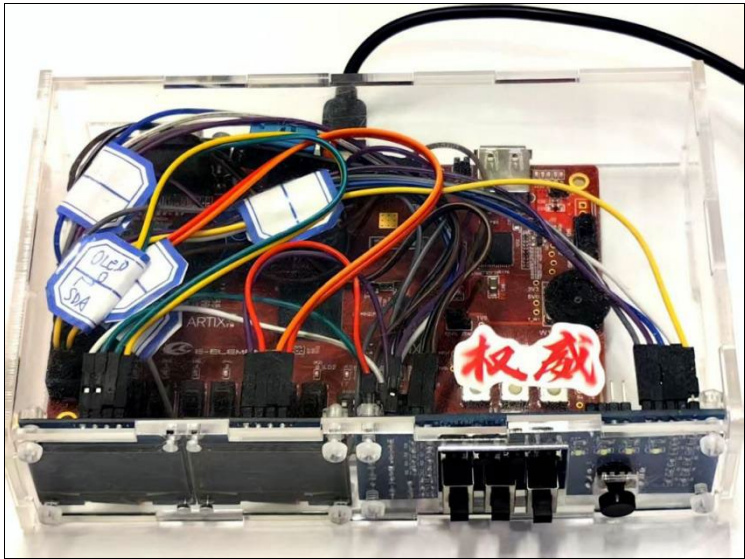
为此，我们设计了 PCB 电路板，内含三个开关，四个按键，一个五向按键，四个 LED 灯，同时在电路板背面提供了各种接口，可以实现电路板和板子直连（元件接法参照了板子说明书）。



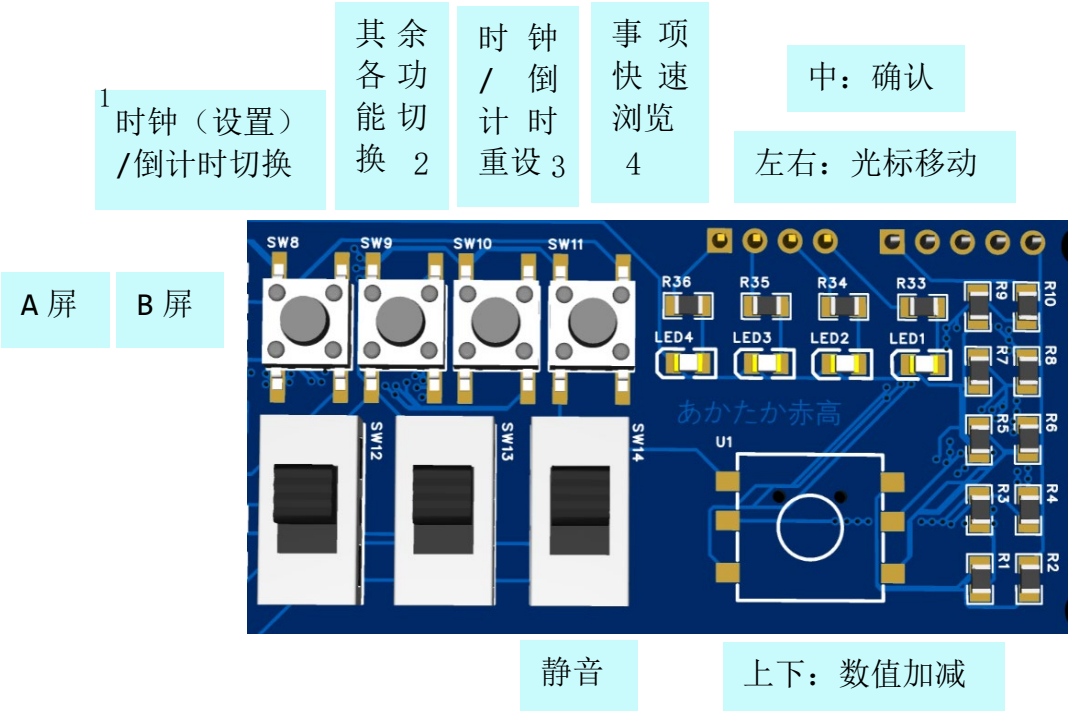
同时，我们设计了拼插式的亚克力外壳，在正面预留了显示屏和电路板的固定孔并为开关和按键开口，在后面为电源接口开口。



最终效果：





二、多功能时钟使用教程（面向用户）



1. 倒计时-专注

按下按钮 1，此时 A 屏左上角显示倒计时，右上角显示当前时间。用五向开关设置倒计时，左右选择要设置的位，上下加减该位数值。按下确定键，开始计时。途中可以按确定键暂停/继续计时。暂停计时时，可按下按钮 3 重新设置倒计时。当倒计时耗尽时，可以通过左/确认/右键来评估这一段时间是否专注/圆满完成任务，分别对应好/一般/不好。好的时间会被累计，这时可以按按钮 2 将 B 屏切换到专注时长查看专注总的时长。当专注时长累积到一定程度，则触发随机奖励。



2. 时钟

按下按钮 1，A 屏显示时钟，校对标志出现时，可用五向开关设置时间。时钟运行时，也可按下按钮 3 重新设置时间。按下按钮 1，A 屏显示时钟，校对标志消失时，可以按按钮 2 切换 B 屏上显示的各项功能（使用以下各功能时，请确实时钟、校对状态）。

3. 事项提醒

此时 B 屏显示事项提醒。可以用上下键来浏览各个事项，也可按下按钮 4 来快速浏览。按下左键，进入事项设置模式，用方向键调节时间和事项编号，并按确认键确定。当当前时间和事项时间匹配时，便会进行提醒：蜂鸣器响并闪烁“到点儿了”，并显示事项类型：上课/运动/浇花等。若需要静音，也可拨动静音开关静音。

4. 温湿度显示、穿衣建议

此时 B 屏会显示温湿度以及穿衣建议。特别的，当温度过高，指示显示，提醒注意防晒；湿度过高，指示显示，提醒带伞。

5. 课程表

此时 B 屏显示课程表。可以用上下键来按顺序浏览一周的课程，课程信息包括课程名称和上课地点。按下确认键可对课程进行修改，按上下键来选择新的课程。

三、实验总结

通过本次实践，我们对数字系统的整体设计思想有了更加全面而深刻的理解，认识到所有电路设计的本质都是建立在“0”和“1”这两个基本逻辑状态上的。

在项目推进的过程中，我们不断提升了使用 Verilog 语言进行模块开发的能力，也加深了对有限状态机工作机制的掌握与运用。同时，我们还深入体会到了“模块化设计”这一核心思想的重要性：将复杂的系统功能进行合理拆分，每一个子模块各司其职，既有利于提升设计效率，又方便后期的调试与功能拓展。通过模块的层次化组织，我们能够清晰地管理每一部分功能逻辑，并有效地协调各模块间的接口，确保整个系统的稳定运行。

此外，我们也深刻体会到了团队协作的力量及其在工程项目中的重要作用。尽管每位成员在任务安排上各有分工，例如有人负责具体功能模块的设计，有人专注于顶层模块的整合，有人负责外设的接入，但在实际推进过程中我们发现，单打独斗往往难以解决复杂的问题，而团队成员之间积极讨论、互相启发，激发出了很多新颖而有效的思路。正是这种开放、协作的氛围，使我们不断克服挑战，推动项目向前发展。

最后一点是关于时间规划和项目管理的感悟，在项目的启动阶段，我们小组提出了三个备选的设计方向：多功能时钟、智能宿舍控制装置，以及智能垃圾桶。经过充分的讨论与可行性分析后，我们最终确定以“多功能时钟”作为本次课程设计的主题。之后我们制定了详细的时间规划，将整个项目分为若干阶段，从功能划分、模块设计、顶层状态机整合、外设接入，到最终的外观设计实现，每一阶段都有明确的时间节点与责任分工。我们在一个半月的周期中有序推进项目，最终成功实现了所有预期功能，如时间显示、倒计时、穿衣建议、专注反馈与奖励等，并在此基础上进一步扩展了课程提醒功能，使其能够提供更多个性化的课程信息，也使得系统更加实用和智能。在李宇波老师和助教徐楚佳哥哥的指导下成功完成了一个集多功能于一体的智能数字时钟系统，在此感谢老师们的付出！

这次课程设计不仅提升了我们的专业技能，更增强了我们的团队合作能力与项目规划意识。相信这段经历将对我们今后的学习产生重要影响，为进一步深入数字系统设计奠定坚实基础。

四、成员及分工

姓名	专业	分工	贡献比
张赫	法语+电子科学与技术	状态机设计，OLED 屏代码，顶层代码整合，设计电路板，设计亚	40%

		克力板，实验报告撰写	
刘昕语	法语+电子科学与技术	状态机设计，模块代码，视频剪辑，PPT 制作	30%
闫佳晟	法语+电子科学与技术	状态机设计，模块代码，温湿度传感器接入，实验报告撰写	30%

五、完整代码(见文件“多功能时钟完整代码”)