浙江大學

《EDA技术应用》实验报告

项目名称：Lcd 万年历

姓　　名：

学　　号：

学　　院：生物医学工程与仪器科学学院

专　　业：生物医学工程

指导老师：黄添添、裘利坚

2026 年 4 月 21 日

浙江大学 实验报告

课程名称：EDA 技术应用　实验类型：EDA 实验　指导老师：黄添添、裘利坚

实验名称：Lcd 万年历

姓　　名：　　学　　号：　　专　　业：生物医学工程

实验地点：玉泉教六 204　日　　期：2026 年 4 月 21 日

目　录

一、实验目的

(1) 了解 Quartus II 开发环境，熟悉 Verilog 语言基本语法。
(2) 实现 Lcd 显示时间功能。
(3) 实现一个万年历的功能程序，内容包括计时，修改时间，万年历正确显示，并且修改时不出现非法月（如 04 月-14 月）日时分秒，每月按 30 日，不考虑闰年闰月。

二、实验内容

根据实验指导资料 Lcd_Date 工程与相关指导手册，完善缺失的代码逻辑，完成基于 FPGA 和 LCD1602 的万年历设计。实现时间在屏幕上的实时刷新、自然走时以及无 Bug 的按键状态切换和数值修改功能。

三、主要仪器设备

Quartus ii 13.1, DE2-115 开发板。

四、软件流程图

4.1 时钟分频与按键消抖

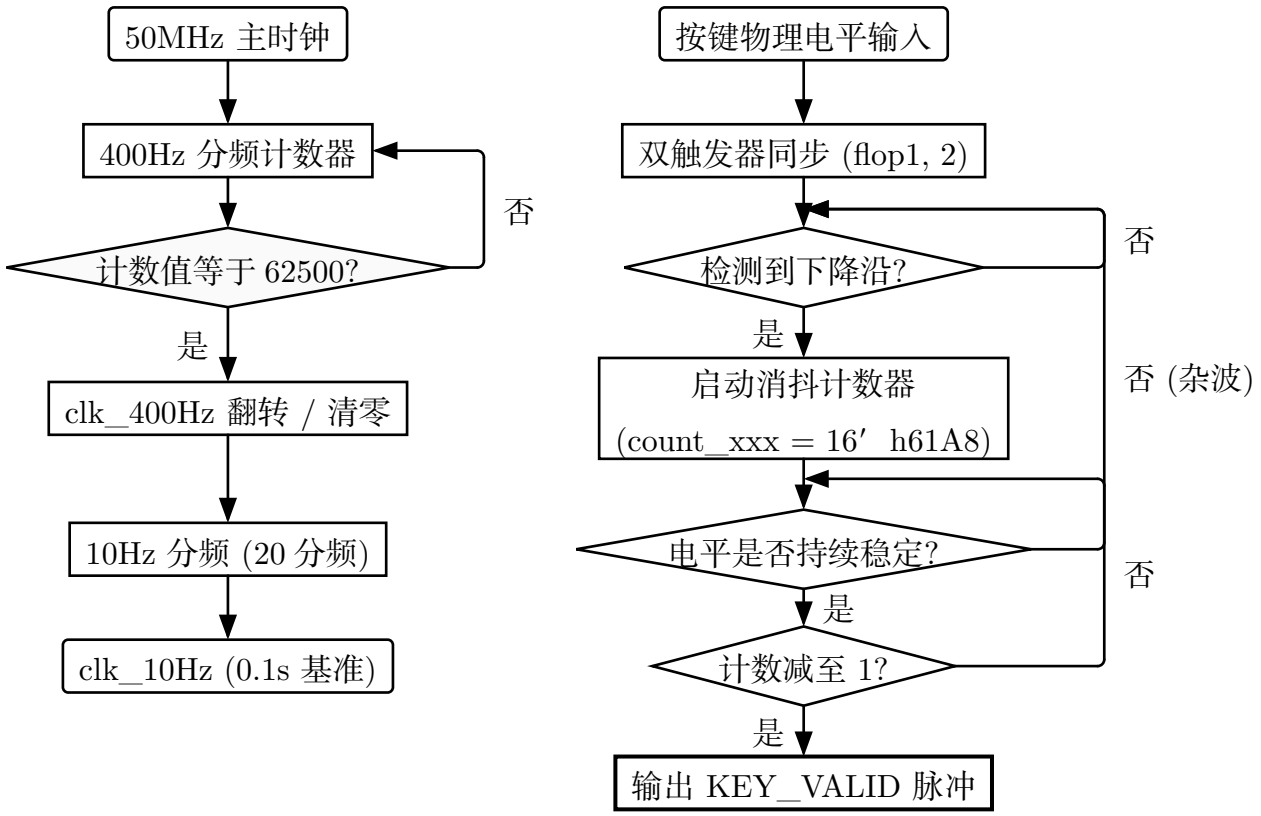

图 4.1 – 时钟分频与按键消抖逻辑流程图

4.2 手动调时逻辑与非法态处理

4.2.1 月份调节逻辑

手动调节月份时，需考虑 [01-12] 的合法范围。当修改月份十位时，若个位已大于等于 3（如 3 月-9 月），十位加 1 将导致 13-19 月的非法态，此时十位必须强制复位为 0。

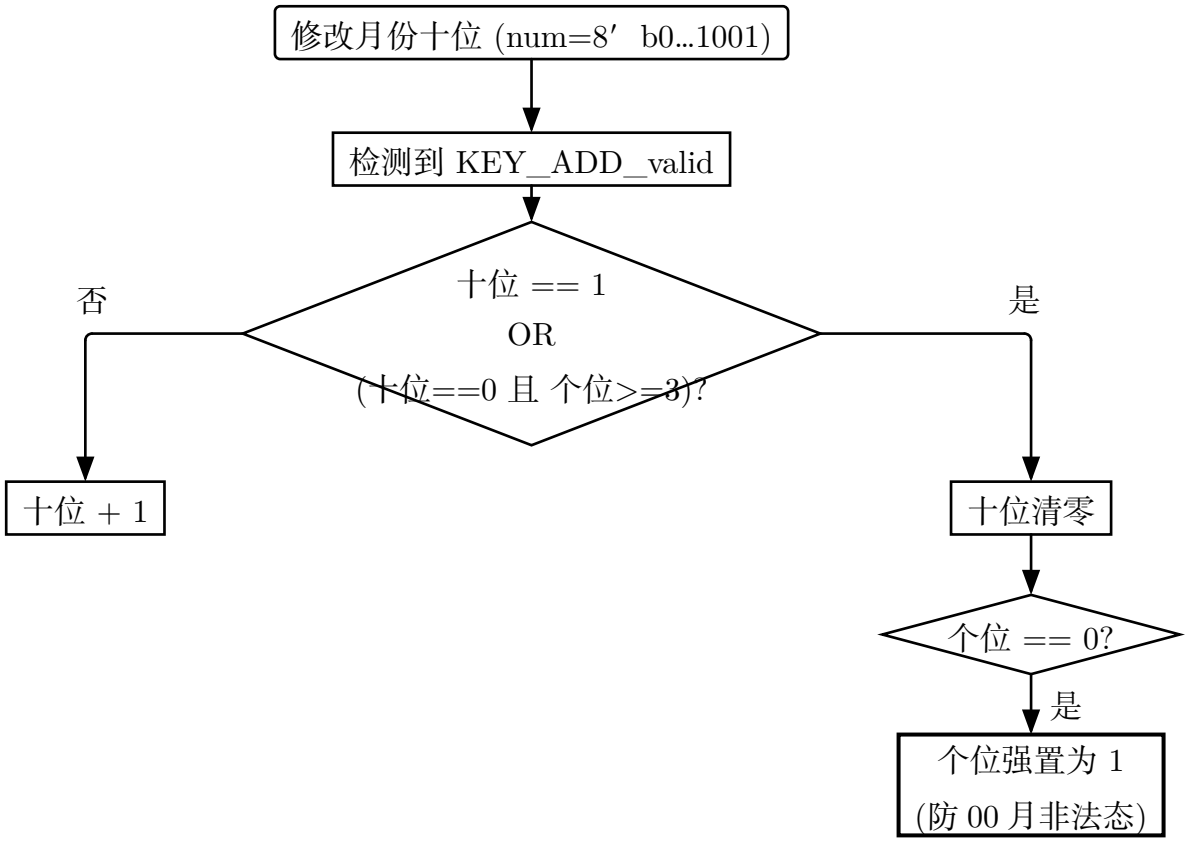

图 4.2 – 月份十位调节防溢出逻辑图

4.2.2 日期调节逻辑

日期调节遵循每月 30 天的简化逻辑。修改日期十位时，若当前日期已处于 21-29 日，十位加 1 会产生 31-39 日的非法态，需进行回弹处理。

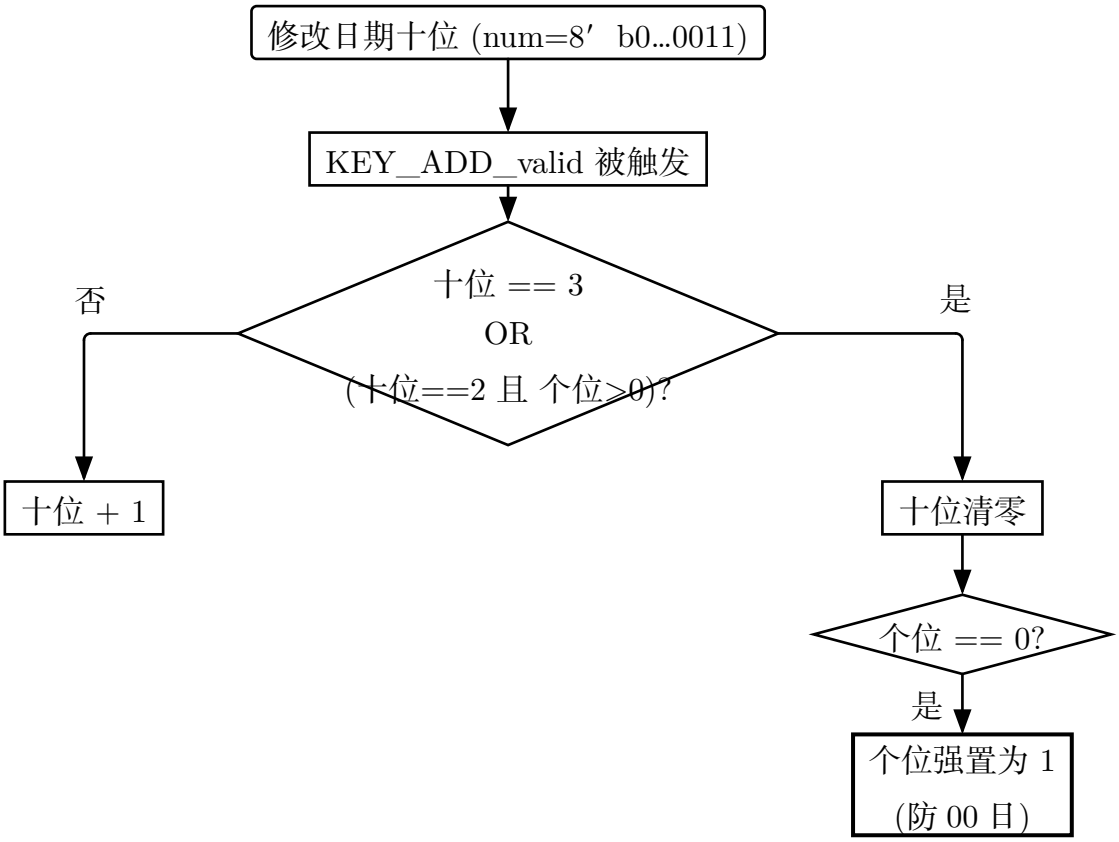

图 4.3 – 日期十位调节逻辑图

4.2.3 小时调节防溢出逻辑分析

小时位采用 24 进制（00-23）。手动调节逻辑必须严密防止产生 24-29 或 30-39 这样的非法时间。如流程图所示，逻辑通过检测高低位的组合状态来决定是执行加法还是强制归零。

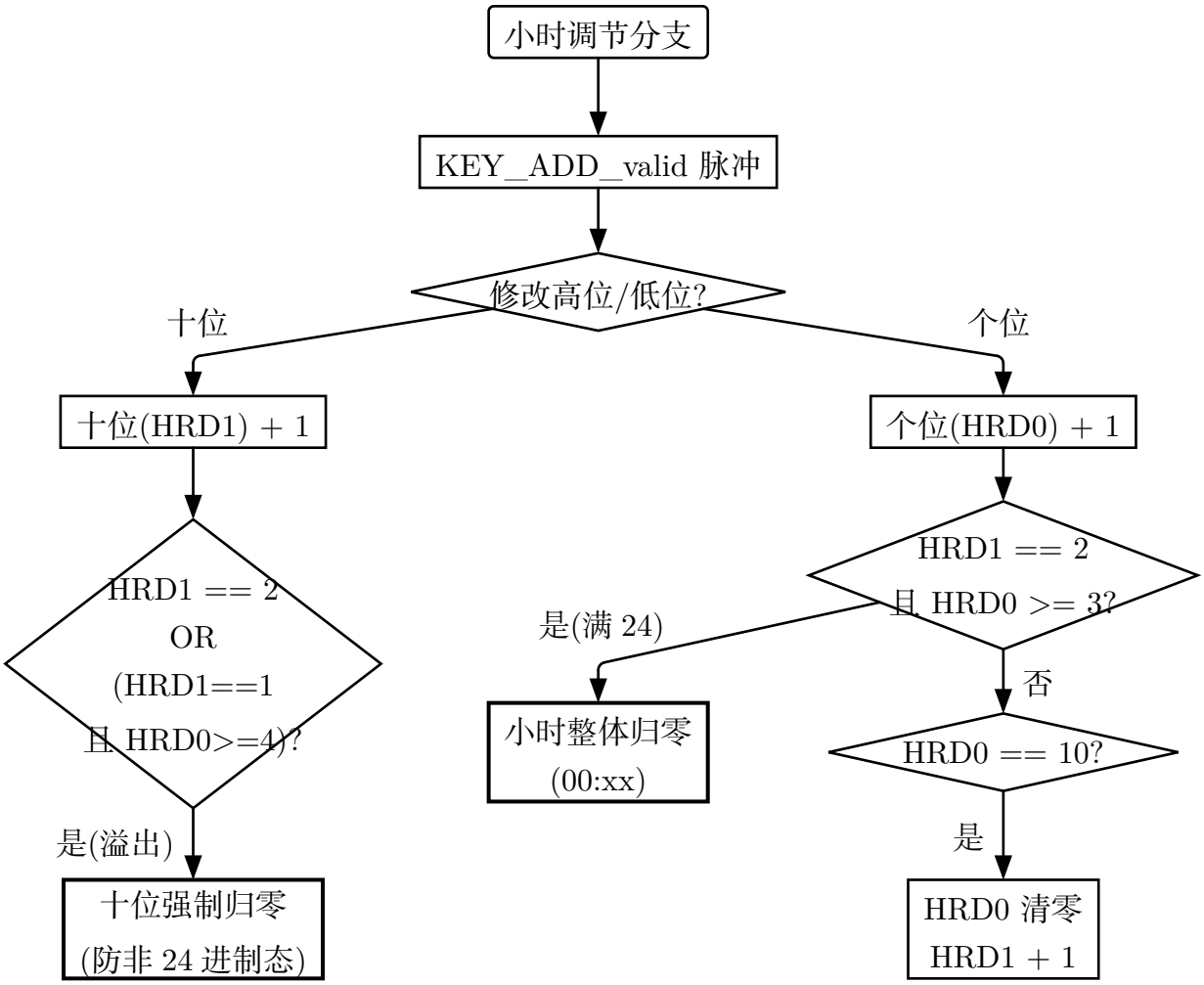

图 4.4 – 小时 (24 进制) 手动调节流程图

4.2.4 分钟调节逻辑分析

用按键增加万年历的时分秒的逻辑基本一致，所以使用分钟为例进行说明。

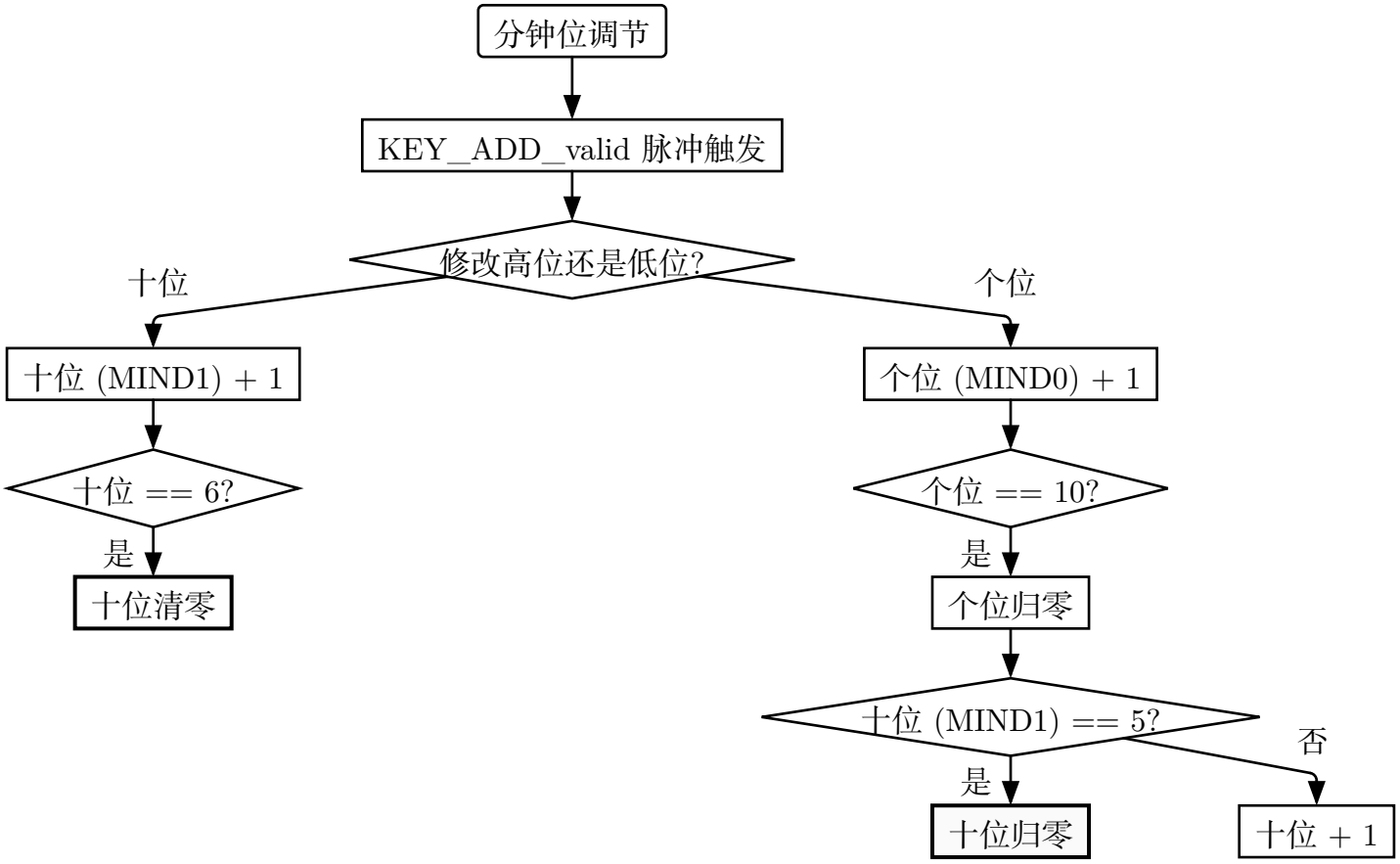

图 4.5 – 分钟手动调节流程图

4.3 万年历自然计时级联进位逻辑

当系统处于非调节状态时，时间根据 10Hz 时钟脉冲（0.1s）自然流逝，产生从低位向高位的级联进位。

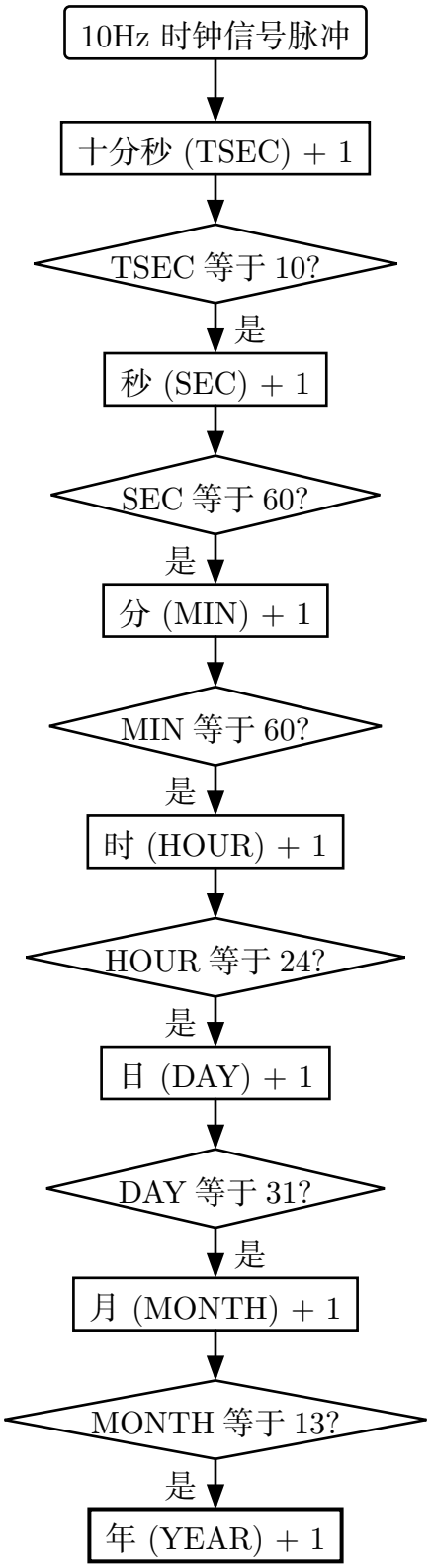

图 4.6 – 万年历级联计时逻辑流程图

五、操作方法和实验步骤

本实验涉及分频器、按键消抖、状态机驱动和复杂时序逻辑四个主要部分。下面将分模块对核心实现逻辑进行分析。

5.1 时钟分频与按键消抖模块

由于板载系统主时钟高达 50MHz（周期极短），不适合直接驱动 LCD 刷新，更不适合作为人类感知的时间基准。因此，首要任务是将 50MHz 时钟降频为 400Hz（用于 LCD 状态机刷新）和 10Hz（即 0.1 秒，用于万年历走时与按键检测）。

按键消抖逻辑与实验二类似，采用双寄存器打拍（消除亚稳态）结合倒数计数器（过滤机械颤抖）的方案。

以下为系统中的时钟分频的 verilog 代码。

```
// 产生 400Hz 时钟
always @ ( posedge clk or negedge rst ) begin
  if ( !rst ) begin
    clk_400Hz <= 1'b0;
    clk_count_400Hz <= 16'h0;
  end
  // 50MHz 时钟下，数 62500 次刚好是一半的周期（十六进制 0xF424 就是十进制 62500）
  else if ( clk_count_400Hz < 16'hF424 ) begin
    clk_count_400Hz <= clk_count_400Hz + 1'b1;
  end
  else begin
    clk_400Hz <= !clk_400Hz; // 数满了，时钟电平翻转一次
    clk_count_400Hz <= 16'h0; // 计数器清零，重新开始数
  end
end

// 产生 10Hz 时钟（周期为0.1秒）
always @ ( posedge clk_400Hz or negedge rst ) begin
  if ( !rst ) begin
    clk_count_10Hz <= 5'b0;
    clk_10Hz <= 1'b0;
  end
  // 在 400Hz 的基础上，数 20 次（从0数到19）翻转一次，就得到了 10Hz
  else if ( clk_count_10Hz < 5'b10011) begin // 5'b10011 就是十进制 19
    clk_count_10Hz <= clk_count_10Hz + 1'b1;
  end
  else begin
    clk_count_10Hz <= 5'b0;
    clk_10Hz <= !clk_10Hz;
  end
end
```

以下为按键消抖的 verilog 代码。

```
// 对右移按键（KEY_RIGHT）的信号进行打拍同步，防止亚稳态干扰系统
  always @ ( posedge clk or negedge rst ) begin
    if ( !rst ) begin
      KEY_RIGHT_flop1 <= 1'b1;
      KEY_RIGHT_flop2 <= 1'b1;
    end
    else begin
      KEY_RIGHT_flop1 <= KEY_RIGHT;
      KEY_RIGHT_flop2 <= KEY_RIGHT_flop1; // 延迟一拍，过滤掉极其微小的毛刺
    end
```

```
  end

  // 右移按键的延时检测器
  always @ ( posedge clk or negedge rst ) begin
    if ( !rst ) count_right <= 16'h0;
    else if ( KEY_RIGHT_flop1 ) begin // 如果当前引脚电平是高电平（即按键处于弹起/未按下状态）
      if ( !KEY_RIGHT_flop2 ) count_right <= 16'h61A8; // 刚检测到从低变高（边沿），赋初值开始倒计时
      else count_right <= count_right ? count_right - 1'b1 : count_right; // 如果一直保持，倒数计数器递减
    end else count_right <= 16'h0;
  end

  // 输出最终干净的按键有效脉冲
  always @ ( posedge clk or negedge rst ) begin
    if ( !rst ) KEY_RIGHT_valid <= 1'b0;
    // 只有当倒数计数器减到了1，才说明按键电平已经稳定了很久，输出高电平脉冲
    else if ( 16'h0001 == count_right ) KEY_RIGHT_valid <= 1'b1;
    else KEY_RIGHT_valid <= 1'b0;
  end

  // 对加一按键（KEY_ADD）进行相同的消抖处理，这里利用慢速的 10Hz 时钟进行采样
  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) begin KEY_ADD_flop1 <= 1'b1; KEY_ADD_flop2 <= 1'b1; end
    else begin KEY_ADD_flop1 <= KEY_ADD; KEY_ADD_flop2 <= KEY_ADD_flop1; end
  end

  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) count_add <= 16'h0;
    else if ( KEY_ADD_flop1 ) begin
      if ( !KEY_ADD_flop2 ) count_add <= 16'b0001;
      else count_add <= 16'h0;
    end else count_add <= 16'h0;
  end

  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) KEY_ADD_valid <= 1'b0;
    else if ( 16'h0001 == count_add ) KEY_ADD_valid <= 1'b1;
    else KEY_ADD_valid <= 1'b0;
  end
```

5.2 游标位置切换逻辑

万年历有多个可修改的时间位（年、月、日、时、分、秒），我们需要一个变量 num 来记录当前按下“加一键”时，究竟要修改哪一位。当检测 KEY_RIGHT_valid 脉冲时，通过 case 语句按照固定的流水线顺序循环切换目标寄存器。

下表展示了 num 的取值与 LCD 显示位及内部 BCD 寄存器的对应关系：

表 5.1 – 游标控制变量 num 与硬件寄存器对应表

num	BCD 寄存器	LCD 状态	功能说明
8′ b1000_0000	BCD_HRD1	WRITE_CHAR1	小时十位
8′ b0100_0000	BCD_HRD0	WRITE_CHAR2	小时个位
8′ b0010_0000	BCD_MIND1	WRITE_CHAR4	分钟十位
8′ b0001_0000	BCD_MIND0	WRITE_CHAR5	分钟个位
8′ b0000_1000	BCD_SECD1	WRITE_CHAR7	秒钟十位
8′ b0000_0100	BCD_SECD0	WRITE_CHAR8	秒钟个位
8′ b0000_0010	BCD_TSEC	WRITE_CHAR10	0.1 秒位
8′ b1000_0001	BCD_YEAR3	WRITE_CHAR12	年份千位
8′ b0100_0001	BCD_YEAR2	WRITE_CHAR13	年份百位
8′ b0010_0001	BCD_YEAR1	WRITE_CHAR14	年份十位
8′ b0001_0001	BCD_YEAR0	WRITE_CHAR15	年份个位
8′ b0000_1001	BCD_MONTH1	WRITE_CHAR17	月份十位
8′ b0000_0101	BCD_MONTH0	WRITE_CHAR18	月份个位
8′ b0000_0011	BCD_DAY1	WRITE_CHAR20	日期十位
8′ b0000_0001	BCD_DAY0	WRITE_CHAR21	日期个位

以下为游标位置切换的 verilog 代码。

```
always @ ( posedge clk or negedge rst ) begin
  if ( !rst ) num <= 8'b10000000; // 系统复位时，默认光标停留在小时的十位上
  else if ( KEY_RIGHT_valid ) begin
    // 按照 时 -> 分 -> 秒 -> 0.1秒 -> 年 -> 月 -> 日 的流水线顺序切换游标
    case ( num )
      8'b10000000 : num <= 8'b01000000; // 小时十位 -> 小时个位
      8'b01000000 : num <= 8'b00100000; // 小时个位 -> 分钟十位
      8'b00100000 : num <= 8'b00010000; // 分钟十位 -> 分钟个位
      8'b00010000 : num <= 8'b00001000; // 分钟个位 -> 秒钟十位
      8'b00001000 : num <= 8'b00000100; // 秒钟十位 -> 秒钟个位
      8'b00000100 : num <= 8'b00000010; // 秒钟个位 -> 0.1秒位
      8'b00000010 : num <= 8'b10000001; // 0.1秒位  -> 年份千位
      8'b10000001 : num <= 8'b01000001; // 年份千位 -> 年份百位
      8'b01000001 : num <= 8'b00100001; // 年份百位 -> 年份十位
      8'b00100001 : num <= 8'b00010001; // 年份十位 -> 年份个位
      8'b00010001 : num <= 8'b00001001; // 年份个位 -> 月份十位
      8'b00001001 : num <= 8'b00000101; // 月份十位 -> 月份个位
      8'b00000101 : num <= 8'b00000011; // 月份个位 -> 日期十位
      8'b00000011 : num <= 8'b00000001; // 日期十位 -> 日期个位
```

```
        8'b00000001 : num <= 8'b10000000; // 日期个位循环回到 -> 小时十位
        default : num <= 8'b10000000;     // 遇到未知错误强制复原
      endcase
    end
end
```

5.3 LCD 状态机驱动刷新逻辑

LCD1602 屏幕无法一次性接收一整段字符串，必须通过状态机按照特定的时序要求逐个字符发送。

状态机运行在 400Hz 时钟下，整体分为四个阶段：系统初始化 → 打印第一行（时分秒）→ 发送换行指令 (0xC0) → 打印第二行（年月日）→ 光标归位 (0x80)。

值得注意的是，LCD 接收的是 ASCII 码，而时间寄存器存放的是 BCD 码（0 9）。代码中通过拼接操作 `{4'h3, BCD_xxx}`，自动在数据前加上 `0011`，将数字转换为了对应的 ASCII 码（如 `5` 变为 `0x35`，即字符 `'5'`）。

以下面的核心驱动代码为例。当状态机进入 `WRITE_CHAR1`（打印小时的十位）时，首先会将 `LCD_RS` 置为高电平以声明传输的是字符数据，并将转换好的 ASCII 码送入数据总线 `DATA_BUS_VALUE`。随后，系统跳转至专门设计的 `TOGGLE_E` 状态，强行将使能引脚 `LCD_E` 拉低，正是这个人为制造的下降沿，触发了 LCD 屏幕的硬件锁存机制，让屏幕真正将总线上的字符显示出来。

```
// LCD状态机发送第一行小时位的数据示例
WRITE_CHAR1 : begin
    LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
    DATA_BUS_VALUE <= {4'h3, BCD_HRD1};
    state <= TOGGLE_E; next_cmd <= WRITE_CHAR2;
end
// 执行指令时的"下降沿动作"触发
TOGGLE_E : begin
    LCD_E <= 1'b0; // 使能信号拉低，LCD吞入数据
    state <= HOLD;
end
```

5.4 万年历核心走时与调时逻辑

这是整个实验中最核心也是最复杂的模块，运行在 10Hz 时钟下。它包含两条互斥的分支：自然推进时间与手动干预调时。

在自然走时分支中，代码采用了层层嵌套的 `if-else` 语句。当 0.1 秒满 10 次，秒钟加 1；秒钟满 60 次，分钟加 1；小时满 24，日期加 1。逻辑遵循物理世界的时间流逝规则。

在按键调时分支中，由于时间单位各不相同（秒/分是 60 进制，小时是 24 进制，月是 12 进制），若仅采用简单的加法器，极易发生溢出错误。

以月份十位的单独调节为例，正常的月份最高只能是 12 月。如果当前月份的个位是 3 到 9（比如 3 月），此时若按下“修改十位”键，十位简单加 1 就会直接变成非法的 13 月至 19 月；同理，若当前已经是 10 月、11 月，十位再加 1 就会变成 20 月、21 月。

因此，在代码中，我加入了严格的边界判定：只要检测到当前已经是十位数（`BCD_MONTH1 == 1`），或者在个位数 ≥ 3 的前提下想要增加十位（`BCD_MONTH0 >= 3`），系统就会拦截加法操作，强制将十位清零。

同时，代码还加入了一个防止显示 00 的操作。假如原本是 10 月，十位被强制清零后会变成非法的“00 月”，此时代码会敏锐地捕捉到个位为 0 的状态，强行将个位拉升至 1，确保月份安全回弹至合法的 01 月。

对应的代码如下。

```
// 修改月的十位(范围: 01-12)
8'b00001001 : begin
    // 拦截非法跨度: 防止产生 13~19月 或 20~22月
    if (BCD_MONTH1 == 4'd1 || (BCD_MONTH1 == 4'd0 && BCD_MONTH0 >= 4'd3)) begin
        BCD_MONTH1 <= 4'd0; // 判定为非法越界，十位强制清零回弹

        // 防止出现绝对非法的 "00月", 将00月强行纠正为01月
        if (BCD_MONTH0 == 4'd0) BCD_MONTH0 <= 4'd1;
    end
    else BCD_MONTH1 <= BCD_MONTH1 + 1'b1; // 安全范围内（如01、02月）正常加十位
end
```

六、实验结果与分析

将代码编译并烧录至 DE2-115 开发板后，LCD1602 屏幕成功点亮。屏幕第一行清晰稳定地显示 `HH:MM:SS.t`（精确到 0.1 秒），第二行显示 `YYYY-MM-DD`。

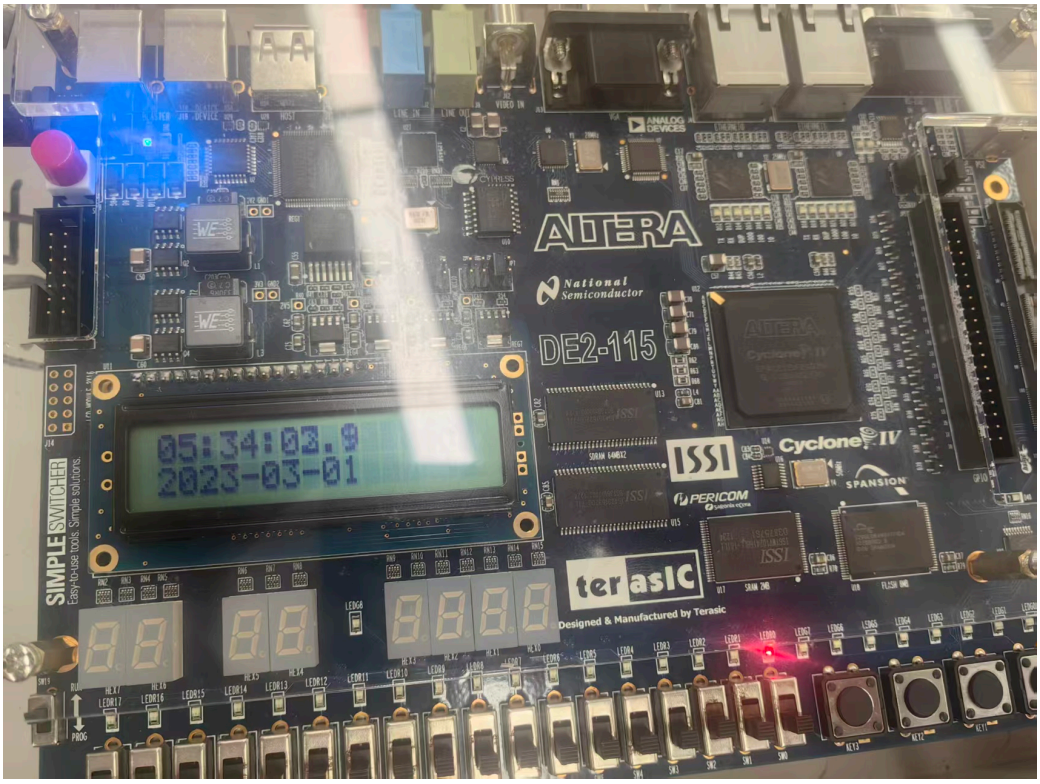

图 6.1 – LCD 屏幕运行结果图

6.1 自动走时测试

系统能精准地实现逢 60 进 1、逢 24 进 1 以及月末进位，秒钟指示灯以 1Hz 频率准确闪烁。

6.2 按键修改测试

按下右移键，能够顺畅地切换想要修改的数字位；按下加一键，时间增加逻辑正确。特别是在测试边界值（如 23 点、12 月、30 日）的单独进位时，系统成功阻断了非法字符的产生，自动回弹至合法区间（如 10 月 $\rightarrow$ 01 月），防溢出逻辑完美生效。

七、讨论与心得

本次基于 FPGA 和 LCD1602 的万年历设计实验，不仅是对 Verilog 语法和 Quartus II 工具链的综合应用，更是对我底层硬件思维和系统级工程设计能力的一次全面重塑。在代码的编写、逻辑的排错以及最终的软硬件联调过程中，我收获了以下几点深刻的体会。

7.1 硬件并行驱动的思维形成

在本次实验中，我深刻体会到了硬件描述语言的本质是用代码画电路。万年历系统实际上是由多个并行运作的逻辑模块构成的：分频器不断输出 400Hz 和 10Hz 的时钟节拍；按键消抖模块拦截物理弹片的毛刺；LCD 状态机在 400Hz 的驱动下高速刷新屏幕；而底层的计时模块则在 10Hz 的节拍下默默进行级联进位。

7.2 非法日期和时间的处理

在实现手动调时功能时，我遇到了极具挑战性的边界溢出问题。起初，我认为只要在月份或日期的十位溢出时（例如尝试调出 13 月或 39 日）将其清零即可。

但我发现了隐藏的零值非法态漏洞。如果当前是 10 月，强行清零十位会产生现实中不存在的“00 月”。为此，我设计并加入了相关代码，实现在十位清零的瞬间侦测个位状态，若双双为零则强拉至 01。这让我切实体会到，优秀的数字电路设计绝不能假设用户的输入永远合法，必须在底层逻辑上堵死所有引发死机的可能性。

7.3 手动调时与自然走时的独立设计

在软硬件联调时，我曾对一个现象产生过思考。为什么在手动调节分钟，使其从 59 跳回 00 时，小时位没有发生联动进位？自然走时与手动调表必须是两套相互隔离的逻辑体系。如果在校准模式下保留级联进位，用户在微调分钟时就会无可避免地破坏已校准好的小时数。因此，在代码中特意切断手动按键引发的跨单位进位脉冲，这绝非逻辑漏洞，而是符合人类直觉的最佳交互设计。

7.4 总结

这次实验不仅让我的专业技能有了质的飞跃，更培养了我面对复杂系统时模块化拆解、严防边界漏洞的工程素养，为后续更复杂的电子系统开发打下了坚实的基础。同时也感谢老师和助教对我的悉心指导！

八、附录

8.1 实验完整代码

```
module Lcd_Date(
  // --- 1. 输入端口定义（相当于芯片的输入引脚） ---
  input clk,              // 系统主时钟：通常由开发板上的晶振提供，频率为 50MHz（每秒震荡5000万次）
  input rst,              // 复位按键：低电平有效，按下后时间会重置到初始值
  input KEY_RIGHT,        // 右移按键：用于切换当前想要修改的时间位置
  input KEY_ADD,          // 加一按键：用于对当前选中的时间位置进行数值累加

  // --- 2. 输出端口定义（相当于芯片连接外部硬件的引脚） ---
  output reg LCD_RS,      // LCD 寄存器选择引脚：输出高电平(1)表示写字符数据，低电平(0)表示写控制指令
  output reg LCD_RW,      // LCD 读写控制引脚：1为读，0为写。本实验只需要向屏幕写数据，所以一直固定为0
  output reg LCD_E,       // LCD 使能引脚：只有当它从高电平变成低电平（下降沿）时，屏幕才会读取数据
  output LCD_ON,          // LCD 背光/电源开启引脚：一直输出高电平，保持屏幕点亮
  inout [7:0] DATA_BUS,   // 8位双向数据总线：芯片就是通过这8根线，把指令或字符数据传给 LCD 屏幕的

  output RESET_LED,       // 复位指示灯：当按下复位键时，这个灯会亮起
  output SEC_LED          // 秒闪烁指示灯：利用秒钟的单双数变化，实现一秒亮、一秒灭的闪烁效果
);

  //=============================================================================
  // 状态机参数定义
  //=============================================================================
  localparam RESET1        = 5'b00000; // 屏幕初始化步骤1
  localparam RESET2        = 5'b00001; // 屏幕初始化步骤2
  localparam RESET3        = 5'b00010; // 屏幕初始化步骤3
  localparam TOGGLE_E      = 5'b00011; // 翻转使能信号E的状态：这是状态机的中转站，专门用来产生LCD_E的下降沿
  localparam FUNC_SET      = 5'b00100; // 设置功能：告诉屏幕我们用8位数据线，显示两行
  localparam DISPLAY_OFF   = 5'b00101; // 关显示：配置期间先关掉屏幕防止乱码
  localparam DISPLAY_CLEAR = 5'b00110; // 清屏：清空屏幕上原有的内容
  localparam DISPLAY_ON    = 5'b00111; // 开显示：配置完毕，正式打开显示
  localparam MODE_SET      = 5'b01000; // 设置输入模式：告诉屏幕每写一个字符，光标自动向右移动一格

  // 第一行打印状态：准备依次将时、分、秒字符送到屏幕
  localparam WRITE_CHAR1   = 5'b01001;
  localparam WRITE_CHAR2   = 5'b01010;
  localparam WRITE_CHAR3   = 5'b01011;
  localparam WRITE_CHAR4   = 5'b01100;
  localparam WRITE_CHAR5   = 5'b01101;
  localparam WRITE_CHAR6   = 5'b01110;
  localparam WRITE_CHAR7   = 5'b01111;
  localparam WRITE_CHAR8   = 5'b10000;
  localparam WRITE_CHAR9   = 5'b10001;
  localparam WRITE_CHAR10  = 5'b10010;

  // 第二行打印状态：先发送换行指令，然后依次发送年、月、日字符
  localparam WRITE_CHAR11  = 5'b10101;
  localparam WRITE_CHAR12  = 5'b10110;
  localparam WRITE_CHAR13  = 5'b10111;
  localparam WRITE_CHAR14  = 5'b11000;
  localparam WRITE_CHAR15  = 5'b11001;
  localparam WRITE_CHAR16  = 5'b11010;
  localparam WRITE_CHAR17  = 5'b11011;
```

```
52    localparam WRITE_CHAR18  = 5'b11100;
53    localparam WRITE_CHAR19  = 5'b11101;
54    localparam WRITE_CHAR20  = 5'b11110;
55    localparam WRITE_CHAR21  = 5'b11111;
56
57    localparam RETURN_HOME   = 5'b10011; // 循环归位：两行都打印完了，让光标回到屏幕最左上角，准备下一轮刷新
58    localparam HOLD          = 5'b10100; // 延时等待状态：等待屏幕把数据处理完
59
60    //=============================================================================
61    // 内部寄存器（变量）定义
62    //=============================================================================
63    // --- 分频器相关变量 ---
64    reg [15:0] clk_count_400Hz; // 用来数时钟震荡次数的计数器，16位宽，最大能数到65535
65    reg clk_400Hz;              // 从 50MHz 降速得到的 400Hz 时钟，专门用来驱动屏幕刷新
66
67    reg [4:0] clk_count_10Hz;   // 用来进一步计数的计数器
68    reg clk_10Hz;               // 从 400Hz 降速得到的 10Hz 时钟（0.1秒跳动一次），用来计时和检测按键
69
70    // --- KEY_RIGHT 按键消抖相关变量 ---
71    reg KEY_RIGHT_flop1, KEY_RIGHT_flop2; // 两个触发器，用于消除跨时钟域产生的亚稳态（不稳定电平）
72    reg KEY_RIGHT_valid;                  // 经过消抖处理后，产生的一个干净、绝对有效的一次性脉冲信号
73    reg [15:0] count_right;               // 倒数计数器，用于过滤掉人手按下按键瞬间金属弹片的机械颤抖
74
75    // --- KEY_ADD 按键消抖相关变量 ---
76    reg KEY_ADD_flop1, KEY_ADD_flop2;
77    reg KEY_ADD_valid;
78    reg [15:0] count_add;
79
80    // --- 系统控制与屏幕相关变量 ---
81    reg [7:0] num;              // 游标指示器：记录当前我们按下加一键时，想要修改的是哪一位时间（独热码存储）
82    reg [4:0] state;            // 记录打字员（屏幕状态机）目前进行到了哪一步
83    reg [4:0] next_cmd;         // 记录打字员（屏幕状态机）下一步应该去干什么
84    reg [7:0] DATA_BUS_VALUE; // 数据缓存区：准备通过总线发给屏幕的 8 位数据
85
86    // --- BCD 时间寄存器（每个寄存器只存 0-9 之间的数字，方便转化为屏幕字符）---
87    reg [3:0] BCD_YEAR3, BCD_YEAR2, BCD_YEAR1, BCD_YEAR0; // 存储年份的千、百、十、个位
88    reg [3:0] BCD_MONTH1, BCD_MONTH0;                     // 存储月份的十、个位
89    reg [3:0] BCD_DAY1, BCD_DAY0;                         // 存储日期的十、个位
90    reg [3:0] BCD_HRD1, BCD_HRD0;                         // 存储小时的十、个位
91    reg [3:0] BCD_MIND1, BCD_MIND0;                       // 存储分钟的十、个位
92    reg [3:0] BCD_SECD1, BCD_SECD0;                       // 存储秒钟的十、个位
93    reg [3:0] BCD_TSEC;                                   // 存储 0.1 秒的位
94
95    //=============================================================================
96    // 连线逻辑
97    //=============================================================================
98    assign LCD_ON = 1'b1;          // 用导线把高电平(1)直接接到屏幕电源，确保屏幕一直亮着
99    assign RESET_LED = !rst;       // 因为复位按键是按下时变成低电平(0)，所以取反(!rst)变成高电平去点亮指示灯
100   assign SEC_LED = BCD_SECD0[0];// BCD_SECD0 是秒的个位。提取它的最低位(0和1交替)，就能实现指示灯每秒闪烁一次
101
102   // 三态门控制：当 LCD_RW 为低电平时（写模式），把缓存里的数据推向数据线；否则让引脚处于高阻态(ZZ)，不干扰外部电路
103   assign DATA_BUS = ( !LCD_RW ) ? DATA_BUS_VALUE : 8'hZZ;
104
105   //=============================================================================
```

```
106   // 模块一：时钟分频器
107   //============================================================================
108   // 产生 400Hz 时钟
109   always @ ( posedge clk or negedge rst ) begin
110     if ( !rst ) begin
111       clk_400Hz <= 1'b0;
112       clk_count_400Hz <= 16'h0;
113     end
114     // 50MHz 时钟下，数 62500 次刚好是一半的周期（十六进制 0xF424 就是十进制 62500）
115     else if ( clk_count_400Hz < 16'hF424 ) begin
116       clk_count_400Hz <= clk_count_400Hz + 1'b1;
117     end
118     else begin
119       clk_400Hz <= !clk_400Hz; // 数满了，时钟电平翻转一次
120       clk_count_400Hz <= 16'h0; // 计数器清零，重新开始数
121     end
122   end
123
124   // 产生 10Hz 时钟（周期为0.1秒）
125   always @ ( posedge clk_400Hz or negedge rst ) begin
126     if ( !rst ) begin
127       clk_count_10Hz <= 5'b0;
128       clk_10Hz <= 1'b0;
129     end
130     // 在 400Hz 的基础上，数 20 次（从0数到19）翻转一次，就得到了 10Hz
131     else if ( clk_count_10Hz < 5'b10011) begin // 5'b10011 就是十进制 19
132       clk_count_10Hz <= clk_count_10Hz + 1'b1;
133     end
134     else begin
135       clk_count_10Hz <= 5'b0;
136       clk_10Hz <= !clk_10Hz;
137     end
138   end
139
140   //============================================================================
141   // 模块二：按键消抖门卫
142   //============================================================================
143   // 对右移按键（KEY_RIGHT）的信号进行打拍同步，防止亚稳态干扰系统
144   always @ ( posedge clk or negedge rst ) begin
145     if ( !rst ) begin
146       KEY_RIGHT_flop1 <= 1'b1;
147       KEY_RIGHT_flop2 <= 1'b1;
148     end
149     else begin
150       KEY_RIGHT_flop1 <= KEY_RIGHT;
151       KEY_RIGHT_flop2 <= KEY_RIGHT_flop1; // 延迟一拍，过滤掉极其微小的毛刺
152     end
153   end
154
155   // 右移按键的延时检测器
156   always @ ( posedge clk or negedge rst ) begin
157     if ( !rst ) count_right <= 16'h0;
158     else if ( KEY_RIGHT_flop1 ) begin // 如果当前引脚电平是高电平（即按键处于弹起/未按下状
    态）
159       if ( !KEY_RIGHT_flop2 ) count_right <= 16'h61A8; // 刚检测到从低变高（边沿），赋初
    值开始倒计时
160       else count_right <= count_right ? count_right - 1'b1 : count_right; // 如果
    一直保持，倒数计数器递减
161     end else count_right <= 16'h0;
162   end
163
164   // 输出最终干净的按键有效脉冲
165   always @ ( posedge clk or negedge rst ) begin
166     if ( !rst ) KEY_RIGHT_valid <= 1'b0;
```

```
    // 只有当倒数计数器减到了1，才说明按键电平已经稳定了很久，输出高电平脉冲
    else if ( 16'h0001 == count_right ) KEY_RIGHT_valid <= 1'b1;
    else KEY_RIGHT_valid <= 1'b0;
  end

  // 对加一按键（KEY_ADD）进行相同的消抖处理，这里利用慢速的 10Hz 时钟进行采样
  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) begin KEY_ADD_flop1 <= 1'b1; KEY_ADD_flop2 <= 1'b1; end
    else begin KEY_ADD_flop1 <= KEY_ADD; KEY_ADD_flop2 <= KEY_ADD_flop1; end
  end

  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) count_add <= 16'h0;
    else if ( KEY_ADD_flop1 ) begin
      if ( !KEY_ADD_flop2 ) count_add <= 16'b0001;
      else count_add <= 16'h0;
    end else count_add <= 16'h0;
  end

  always @ ( posedge clk_10Hz or negedge rst ) begin
    if ( !rst ) KEY_ADD_valid <= 1'b0;
    else if ( 16'h0001 == count_add ) KEY_ADD_valid <= 1'b1;
    else KEY_ADD_valid <= 1'b0;
  end

  //============================================================================
  // 模块三：游标位置切换器
  // 原理：当收到 KEY_RIGHT 脉冲时，让记录修改位置的变量 num 进行循环移位
  //============================================================================
  always @ ( posedge clk or negedge rst ) begin
    if ( !rst ) num <= 8'b10000000; // 系统复位时，默认光标停留在小时的十位上
    else if ( KEY_RIGHT_valid ) begin
      // 按照 时 -> 分 -> 秒 -> 0.1秒 -> 年 -> 月 -> 日 的流水线顺序切换游标
      case ( num )
        8'b10000000 : num <= 8'b01000000; // 小时十位 -> 小时个位
        8'b01000000 : num <= 8'b00100000; // 小时个位 -> 分钟十位
        8'b00100000 : num <= 8'b00010000; // 分钟十位 -> 分钟个位
        8'b00010000 : num <= 8'b00001000; // 分钟个位 -> 秒钟十位
        8'b00001000 : num <= 8'b00000100; // 秒钟十位 -> 秒钟个位
        8'b00000100 : num <= 8'b00000010; // 秒钟个位 -> 0.1秒位
        8'b00000010 : num <= 8'b10000001; // 0.1秒位  -> 年份千位
        8'b10000001 : num <= 8'b01000001; // 年份千位 -> 年份百位
        8'b01000001 : num <= 8'b00100001; // 年份百位 -> 年份十位
        8'b00100001 : num <= 8'b00010001; // 年份十位 -> 年份个位
        8'b00010001 : num <= 8'b00001001; // 年份个位 -> 月份十位
        8'b00001001 : num <= 8'b00000101; // 月份十位 -> 月份个位
        8'b00000101 : num <= 8'b00000011; // 月份个位 -> 日期十位
        8'b00000011 : num <= 8'b00000001; // 日期十位 -> 日期个位
        8'b00000001 : num <= 8'b10000000; // 日期个位循环回到 -> 小时十位
        default : num <= 8'b10000000;     // 遇到未知错误强制复原
      endcase
    end
  end

  //============================================================================
  // 模块四：LCD 状态机驱动
  // 原理：以极快的速度（400Hz），将当前所有的数字转化为 ASCII 码，并严格按照时序送到屏幕总线
  //============================================================================
  always @ ( posedge clk_400Hz or negedge rst ) begin
    if ( !rst ) begin
      state <= RESET1;
      DATA_BUS_VALUE <= 8'h38;
      next_cmd <= RESET2;
      LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0; // 初始化所有总线控制脚
    end
```

```
232     else begin
233       case ( state )
234         // --- 步骤块1: 发送屏幕初始化指令 (注意 RS 都是 0, 代表发指令) ---
          RESET1 : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
235 DATA_BUS_VALUE <= 8'h38; state <= TOGGLE_E; next_cmd <= RESET2; end
          RESET2 : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
236 DATA_BUS_VALUE <= 8'h38; state <= TOGGLE_E; next_cmd <= RESET3; end
          RESET3 : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
237 DATA_BUS_VALUE <= 8'h38; state <= TOGGLE_E; next_cmd <= FUNC_SET; end
          FUNC_SET : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
238 DATA_BUS_VALUE <= 8'h38; state <= TOGGLE_E; next_cmd <= DISPLAY_OFF; end
          DISPLAY_OFF : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
239 DATA_BUS_VALUE <= 8'h08; state <= TOGGLE_E; next_cmd <= DISPLAY_CLEAR; end
          DISPLAY_CLEAR : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
240 DATA_BUS_VALUE <= 8'h01; state <= TOGGLE_E; next_cmd <= DISPLAY_ON; end
          DISPLAY_ON : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
241 DATA_BUS_VALUE <= 8'h0C; state <= TOGGLE_E; next_cmd <= MODE_SET; end
          MODE_SET : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
242 DATA_BUS_VALUE <= 8'h06; state <= TOGGLE_E; next_cmd <= WRITE_CHAR1; end
243
          // --- 步骤块2: 往屏幕第一行发送 时:分:秒.十分秒 (注意 RS 变成 1 了, 代表发可视字符)
244 ---
          // 技巧解析: 纯数字0-9屏幕是不认识的。我们在前面拼上一个二进制的 0011 (即4'h3),
          // 比如把数字 5(0101) 拼成了 00110101, 这就是十六进制的 0x35, 刚好对应 ASCII 码表里
246 的字符 '5'。
          WRITE_CHAR1 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
247 DATA_BUS_VALUE <= {4'h3, BCD_HRD1}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR2;
    end
          WRITE_CHAR2 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
248 DATA_BUS_VALUE <= {4'h3, BCD_HRD0}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR3;
    end
          WRITE_CHAR3 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
249 DATA_BUS_VALUE <= 8'h3A; /* 这里直接发十六进制 0x3A, 对应 ASCII 的冒号 ":" */ state <=
    TOGGLE_E; next_cmd <= WRITE_CHAR4; end
          WRITE_CHAR4 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
250 DATA_BUS_VALUE <= {4'h3, BCD_MIND1}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR5;
    end
          WRITE_CHAR5 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
251 DATA_BUS_VALUE <= {4'h3, BCD_MIND0}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR6;
    end
          WRITE_CHAR6 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
252 DATA_BUS_VALUE <= 8'h3A; /* 发送冒号 ":" */ state <= TOGGLE_E; next_cmd <=
    WRITE_CHAR7; end
          WRITE_CHAR7 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
253 DATA_BUS_VALUE <= {4'h3, BCD_SECD1}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR8;
    end
          WRITE_CHAR8 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
254 DATA_BUS_VALUE <= {4'h3, BCD_SECD0}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR9;
    end
          WRITE_CHAR9 : begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
255 DATA_BUS_VALUE <= 8'h2E; /* 0x2E 对应 ASCII 的小数点 "." */ state <= TOGGLE_E;
    next_cmd <= WRITE_CHAR10; end
          WRITE_CHAR10: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
256 DATA_BUS_VALUE <= {4'h3, BCD_TSEC}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR11;
    end
```

```
257
258         // --- 步骤块3：告诉屏幕我们要换行了 ---
            WRITE_CHAR11: begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
259 DATA_BUS_VALUE <= 8'hC0; /* 发送控制指令 0xC0，光标强行跳到第二行行首 */ state <=
    TOGGLE_E; next_cmd <= WRITE_CHAR12; end
260
261         // --- 步骤块4：往屏幕第二行发送 年-月-日 ---
            WRITE_CHAR12: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
262 DATA_BUS_VALUE <= {4'h3, BCD_YEAR3}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR13;
    end
            WRITE_CHAR13: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
263 DATA_BUS_VALUE <= {4'h3, BCD_YEAR2}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR14;
    end
            WRITE_CHAR14: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
264 DATA_BUS_VALUE <= {4'h3, BCD_YEAR1}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR15;
    end
            WRITE_CHAR15: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
265 DATA_BUS_VALUE <= {4'h3, BCD_YEAR0}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR16;
    end
            WRITE_CHAR16: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
266 DATA_BUS_VALUE <= 8'h2D; /* 0x2D 对应 ASCII 的横杠 "-" */ state <= TOGGLE_E;
    next_cmd <= WRITE_CHAR17; end
            WRITE_CHAR17: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
267 DATA_BUS_VALUE <= {4'h3, BCD_MONTH1}; state <= TOGGLE_E; next_cmd <=
    WRITE_CHAR18; end
            WRITE_CHAR18: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
268 DATA_BUS_VALUE <= {4'h3, BCD_MONTH0}; state <= TOGGLE_E; next_cmd <=
    WRITE_CHAR19; end
            WRITE_CHAR19: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
269 DATA_BUS_VALUE <= 8'h2D; /* 发送横杠 "-" */ state <= TOGGLE_E; next_cmd <=
    WRITE_CHAR20; end
            WRITE_CHAR20: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
270 DATA_BUS_VALUE <= {4'h3, BCD_DAY1}; state <= TOGGLE_E; next_cmd <= WRITE_CHAR21;
    end
            WRITE_CHAR21: begin LCD_E <= 1'b1; LCD_RS <= 1'b1; LCD_RW <= 1'b0;
271 DATA_BUS_VALUE <= {4'h3, BCD_DAY0}; state <= TOGGLE_E; next_cmd <= RETURN_HOME;
    end
272
273         // --- 步骤块5：一轮刷新完成，准备循环起手式 ---
            RETURN_HOME : begin LCD_E <= 1'b1; LCD_RS <= 1'b0; LCD_RW <= 1'b0;
274 DATA_BUS_VALUE <= 8'h80; /* 发送指令 0x80 让光标强行回到第一行最开头 */ state <=
    TOGGLE_E; next_cmd <= WRITE_CHAR1; end
275
276         // --- 最核心的触发机制：下降沿动作 ---
            // LCD 硬件规定，光把电线接好没用，必须让 LCD_E 产生一次从高电平到低电平的突变（下降
277 沿），数据才会被真正写入屏幕。
            TOGGLE_E : begin LCD_E <= 1'b0; state <= HOLD; end
278
            HOLD : begin state <= next_cmd; end
279
        endcase
280
      end
281
    end
282
283
    //======================================================================
284
    // 模块五：万年历核心算力中心
285
    // 原理：由 10Hz 时钟驱动，也就是每过 0.1 秒，就会执行一次这里的代码
286
    //======================================================================
287
    always @ ( posedge clk_10Hz or negedge rst ) begin
288
```

```
    if ( !rst ) begin
      // 按下复位键时，强制系统回到 2023-01-01 00:00:00.0 的出厂设置
      BCD_YEAR3 <= 4'd2; BCD_YEAR2 <= 4'd0; BCD_YEAR1 <= 4'd2; BCD_YEAR0 <= 4'd3;
      BCD_MONTH1 <= 4'd0; BCD_MONTH0 <= 4'd1;
      BCD_DAY1 <= 4'd0;   BCD_DAY0 <= 4'd1;
      BCD_HRD1 <= 4'd0;   BCD_HRD0 <= 4'd0;
      BCD_MIND1 <= 4'd0;  BCD_MIND0 <= 4'd0;
      BCD_SECD1 <= 4'd0;  BCD_SECD0 <= 4'd0;
      BCD_TSEC <= 4'd0;
    end

    // ------------------------------------------------------------
    // 情况A：人为干预（手动按下调时键），时间停止自然流动，进入调表逻辑
    // ------------------------------------------------------------
    else if ( KEY_ADD_valid ) begin
      case ( num )
        // ================= 小时修改保护逻辑 =================
        // 修改小时的十位 （最大只能是23）
        8'b10000000 : begin
          // 防溢出保护：如果当前已经是20点以上，再加就会变成非法时间(30多)；
          // 如果当前是14~19点，十位加一会变成24~29点也是非法的。触发这些条件时，十位强制归零。
          if (BCD_HRD1 == 4'd2 || (BCD_HRD1 == 4'd1 && BCD_HRD0 >= 4'd4))
            BCD_HRD1 <= 4'd0;
          else
            BCD_HRD1 <= BCD_HRD1 + 1'b1;
        end
        // 修改小时的个位
        8'b01000000 : begin
          if (BCD_HRD1 == 4'd2 && BCD_HRD0 >= 4'd3) begin BCD_HRD0 <= 4'd0;
BCD_HRD1 <= 4'd0; end // 到达23点时，再加一强制变成 00 点
          else if (BCD_HRD0 == 4'd9) begin BCD_HRD0 <= 4'd0; BCD_HRD1 <= BCD_HRD1
+ 1'b1; end // 普通状况满9进1
          else BCD_HRD0 <= BCD_HRD0 + 1'b1;
        end

        // ================= 分钟和秒钟修改逻辑 =================
        // 修改分钟的十位
        8'b00100000 : begin
          if (BCD_MIND1 == 4'd5) BCD_MIND1 <= 4'd0; // 分钟最大是59，如果十位已经是5，
再加就归0
          else BCD_MIND1 <= BCD_MIND1 + 1'b1;
        end
        // 修改分钟的个位 （注意：这里做了阻断机制，满59归零，绝不进位到小时，防止用户调表时产生连
锁影响）
        8'b00010000 : begin
          if (BCD_MIND0 == 4'd9) begin
            BCD_MIND0 <= 4'd0;
            if (BCD_MIND1 == 4'd5) BCD_MIND1 <= 4'd0;
            else BCD_MIND1 <= BCD_MIND1 + 1'b1;
          end
          else BCD_MIND0 <= BCD_MIND0 + 1'b1;
        end
        // 修改秒钟的十位
        8'b00001000 : begin
          if (BCD_SECD1 == 4'd5) BCD_SECD1 <= 4'd0;
          else BCD_SECD1 <= BCD_SECD1 + 1'b1;
        end
        // 修改秒钟的个位 （同样设置阻断机制，不进位给分钟）
        8'b00000100 : begin
          if (BCD_SECD0 == 4'd9) begin
            BCD_SECD0 <= 4'd0;
            if (BCD_SECD1 == 4'd5) BCD_SECD1 <= 4'd0;
            else BCD_SECD1 <= BCD_SECD1 + 1'b1;
```

```
348             end
349             else BCD_SECD0 <= BCD_SECD0 + 1'b1;
350           end
351           // 修改0.1秒
352           8'b00000010 : begin
353             if (BCD_TSEC == 4'd9) BCD_TSEC <= 4'd0;
354             else BCD_TSEC <= BCD_TSEC + 1'b1;
355           end
356
357           // ================= 年份修改逻辑 =================
358           // 修改年（每一位满9归0，最基础的十进制加法）
359           8'b10000001 : begin if (BCD_YEAR3 == 4'd9) BCD_YEAR3 <= 4'd0; else BCD_YEAR3 <= BCD_YEAR3 + 1'b1; end
360           8'b01000001 : begin if (BCD_YEAR2 == 4'd9) BCD_YEAR2 <= 4'd0; else BCD_YEAR2 <= BCD_YEAR2 + 1'b1; end
361           8'b00100001 : begin if (BCD_YEAR1 == 4'd9) BCD_YEAR1 <= 4'd0; else BCD_YEAR1 <= BCD_YEAR1 + 1'b1; end
362           8'b00010001 : begin if (BCD_YEAR0 == 4'd9) BCD_YEAR0 <= 4'd0; else BCD_YEAR0 <= BCD_YEAR0 + 1'b1; end
363
364           // ================= 月份修改保护逻辑 =================
365           // 修改月的十位（范围：01 到 12）
366           8'b00001001 : begin
367             // 防溢出判断：如果当前是 03~09月，加十位变13~19月是非法的。或者当前是10~12月，加十位变成20多也是非法的。
368             if (BCD_MONTH1 == 4'd1 || (BCD_MONTH1 == 4'd0 && BCD_MONTH0 >= 4'd3)) begin
369               BCD_MONTH1 <= 4'd0; // 如果非法，十位强制清零
370               // 假如原来是10月，清零十位后变成了绝对非法的 "00月"。所以我们必须强行把个位拉高到1，保证最小月份是01月。
371               if (BCD_MONTH0 == 4'd0) BCD_MONTH0 <= 4'd1;
372             end
373             else BCD_MONTH1 <= BCD_MONTH1 + 1'b1;
374           end
375           // 修改月的个位
376           8'b00000101 : begin
377             if (BCD_MONTH1 == 4'd1 && BCD_MONTH0 == 4'd2) begin BCD_MONTH0 <= 4'd1; BCD_MONTH1 <= 4'd0; end // 12满直接变01
378             else if (BCD_MONTH0 == 4'd9) begin BCD_MONTH0 <= 4'd0; BCD_MONTH1 <= BCD_MONTH1 + 1'b1; end // 满9正常进位
379             else BCD_MONTH0 <= BCD_MONTH0 + 1'b1;
380           end
381
382           // ================= 日期修改保护逻辑 =================
383           // 修改日的十位（按照普通历法简化处理：每月最大只能到30天）
384           8'b00000011 : begin
385             // 防溢出判断：和月份类似，如果遇到超过30的日期，直接将十位清零。
386             if (BCD_DAY1 == 4'd3 || (BCD_DAY1 == 4'd2 && BCD_DAY0 > 4'd0)) begin
387               BCD_DAY1 <= 4'd0;
388               // 同样防止出现非法的 "00日"，确保日期最低为 01日。
389               if (BCD_DAY0 == 4'd0) BCD_DAY0 <= 4'd1;
390             end
391             else BCD_DAY1 <= BCD_DAY1 + 1'b1;
392           end
393           // 修改日的个位
394           8'b00000001 : begin
395             if (BCD_DAY1 == 4'd3 && BCD_DAY0 == 4'd0) begin BCD_DAY0 <= 4'd1; BCD_DAY1 <= 4'd0; end // 30号满，回弹到01号
396             else if (BCD_DAY0 == 4'd9) begin BCD_DAY0 <= 4'd0; BCD_DAY1 <= BCD_DAY1 + 1'b1; end
397             else BCD_DAY0 <= BCD_DAY0 + 1'b1;
```

```
398           end
399         endcase
400       end
401
402       // ----------------------------------------------------------------
403       // 情况B: 没人按键，钟表系统依赖 0.1 秒的节拍自然向前推进
404       // ----------------------------------------------------------------
405       else begin
406         if ( BCD_TSEC < 4'b1001 ) BCD_TSEC <= BCD_TSEC + 1'b1; // 0.1秒正常累加
407         else begin
408           BCD_TSEC <= 4'b0; // 0.1秒满9归零，触发连环向上进位
409
410           // 以下就是标准的十进制和六十进制嵌套进位体系
411           if ( BCD_SECD0 < 4'b1001 ) BCD_SECD0 <= BCD_SECD0 + 1'b1; // 秒个位进位
412           else begin
413             BCD_SECD0 <= 4'b0;
414             if ( BCD_SECD1 < 4'b0101 ) BCD_SECD1 <= BCD_SECD1 + 1'b1; // 秒十位进位（满59进入下一级）
415             else begin
416               BCD_SECD1 <= 4'b0;
417               if ( BCD_MIND0 < 4'b1001 ) BCD_MIND0 <= BCD_MIND0 + 1'b1; // 分个位进位
418               else begin
419                 BCD_MIND0 <= 4'b0;
420                 if ( BCD_MIND1 < 4'b0101 ) BCD_MIND1 <= BCD_MIND1 + 1'b1; // 分十位进位（满59进入下一级）
421                 else begin
422                   BCD_MIND1 <= 4'b0;
423
424                   // 小时进位（满23进入天数计算）
425                   if (BCD_HRD1 == 4'd2 && BCD_HRD0 == 4'd3) begin
426                     BCD_HRD0 <= 4'd0; BCD_HRD1 <= 4'd0;
427
428                     // 天数进位（为了降低硬件复杂度，统一按每月30天计算，满30进入月份）
429                     if (BCD_DAY1 == 4'd3 && BCD_DAY0 == 4'd0) begin
430                       BCD_DAY0 <= 4'd1; BCD_DAY1 <= 4'd0; // 天数回落到1号
431
432                       // 月份进位（满12进入年份）
433                       if (BCD_MONTH1 == 4'd1 && BCD_MONTH0 == 4'd2) begin
434                         BCD_MONTH0 <= 4'd1; BCD_MONTH1 <= 4'd0; // 月份回落到1月
435
436                         // 年份进位系统（9999为上限）
437                         if (BCD_YEAR0 == 4'd9) begin
438                           BCD_YEAR0 <= 4'd0;
439                           if (BCD_YEAR1 == 4'd9) begin
440                             BCD_YEAR1 <= 4'd0;
441                             if (BCD_YEAR2 == 4'd9) begin
442                               BCD_YEAR2 <= 4'd0;
443                               if (BCD_YEAR3 == 4'd9) BCD_YEAR3 <= 4'd0; // 到达9999年时，整个年份归零重启
444                               else BCD_YEAR3 <= BCD_YEAR3 + 1'b1;
445                             end else BCD_YEAR2 <= BCD_YEAR2 + 1'b1;
446                           end else BCD_YEAR1 <= BCD_YEAR1 + 1'b1;
447                         end else BCD_YEAR0 <= BCD_YEAR0 + 1'b1;
448
449                       end else if (BCD_MONTH0 == 4'd9) begin BCD_MONTH0 <= 4'd0; BCD_MONTH1 <= BCD_MONTH1 + 1'b1; end
450                       else BCD_MONTH0 <= BCD_MONTH0 + 1'b1;
451
452                     end else if (BCD_DAY0 == 4'd9) begin BCD_DAY0 <= 4'd0; BCD_DAY1 <= BCD_DAY1 + 1'b1; end
453                     else BCD_DAY0 <= BCD_DAY0 + 1'b1;
454
```

```
                    end else if (BCD_HRD0 == 4'd9) begin BCD_HRD0 <= 4'd0; BCD_HRD1
455 <= BCD_HRD1 + 1'b1; end
456                     else BCD_HRD0 <= BCD_HRD0 + 1'b1;
457                 end
458               end
459             end
460           end
461         end
462       end
463     end
464 endmodule
```

8.2 RTL 电路视图

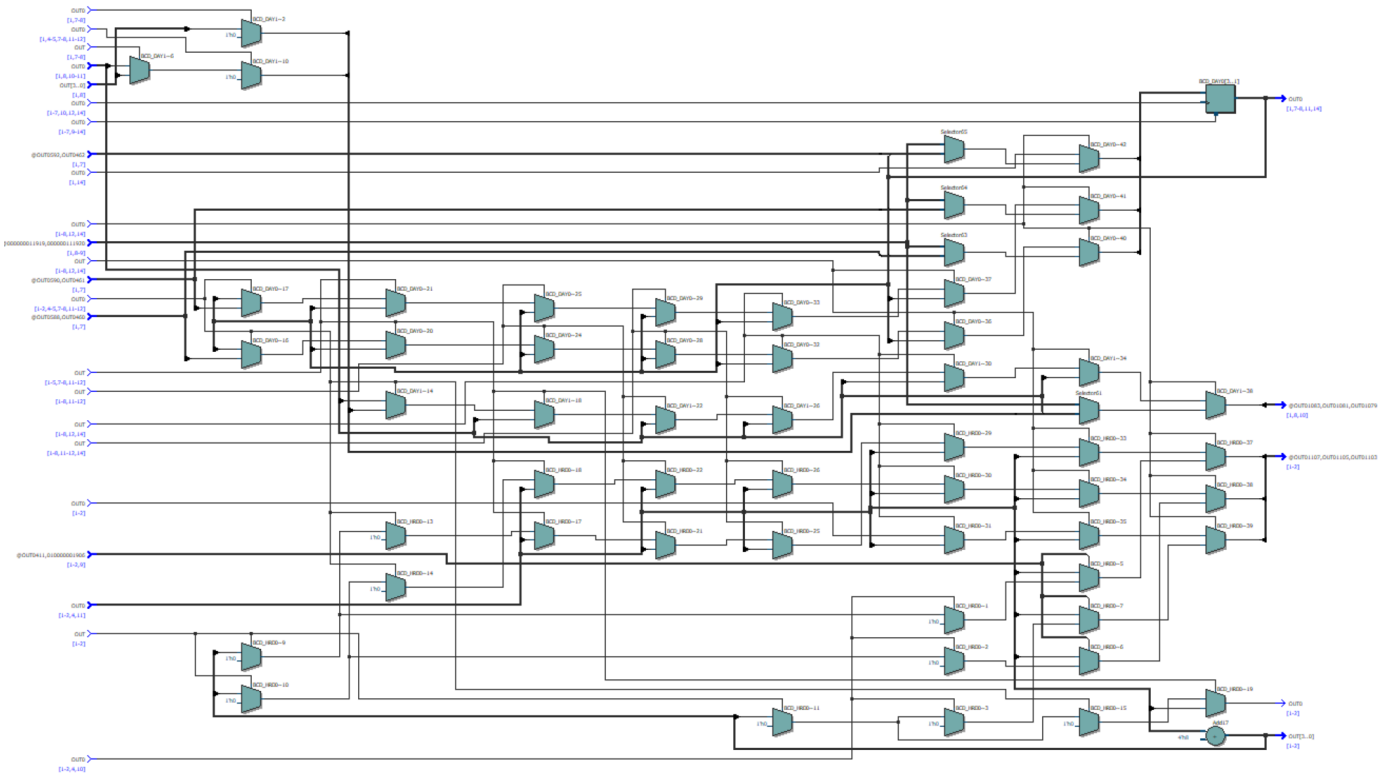

图 8.1 – RTL 电路视图