北京航空航天大學

BEIHANG UNIVERSITY

实验三、时序逻辑设计——三色灯开关

2018 年 11 月 21 日（版本 v01）

网络空间安全学院

目录

实验三、时序逻辑设计——三色灯开关（实验指导书部分）

1 实验指导

设计一种通过操作开关的时间控制灯光颜色的开关，采用硬件描述语言描述同步时序逻辑电路的方法，体会状态转换和计数器定时，对状态机及顶层可综合模块进行功能仿真，并利用 EELAB-FPGACORE2 实验硬件实验平台（以下简称“实验板”）调试并实现，完成较为完整的 EDA 设计实现过程。

1.1 设计需求

某灯具含有两组 LED 白光灯芯和两组 LED 黄光灯芯，由一个开关（switch）控制。设有电气转换装置将强电开关信号转换为数字逻辑电平输入信号 X——当开关断开，X=0，闭合，X=1；且将 4 个数字逻辑电平输出信号 W_1、W_0、Y_1 和 Y_0 分别转换为灯芯控制信号——W_i=1 为白光点亮，W_i=0 为白光熄灭，Y_i=1 为黄光点亮，Y_i=0 为黄光熄灭，其中 i=0, 1。

说明：如果白光和黄光同时点亮，则为日光色调，则如果在同等亮度条件下，该灯具具有白光、日光和黄光三种灯色。（考虑灯光的亮度，4 个灯芯的组合有更多种可能）

三色灯开关的功能是：

(1) 如果快速断开/闭合开关，可以切换灯色，即：

- 如果灯具发出白光时断开开关，且在 1 秒之内再次闭合开关，灯具发出日光，
- 如果灯具发出日光时断开开关，且在 1 秒之内再次闭合开关，灯具发出黄光，
- 如果灯具发出黄光时断开开关，再次闭合开关之后，灯具发出白光。

(2) 如果开关断开的时间超过 1 秒，则开关闭合后，灯具发出白光。

1.2 基本实验要求

基本实验要求为：

a) 采用 Verilog HDL 语言描述有限状态机（FSM），实现第 1.1 节所述的灯具控制电路的时序逻辑，用实验板上的 4 个 LED 灯分别代表现实中的 4 个灯芯；

b) 采用 ModelSim 功能仿真，对状态机、计数器等关键实例进行功能仿真测试；

c) 设计实现分为两种操作模式 MODE_RUN 和 MODE_DEMO，通过一个开关来控制，前者为正常操作模式（连续按开关获得不同灯色的定时值为 1 秒），后者为演示模式，逻辑不变但将时间放大 10 倍的模式；

d) 在 MODE_DEMO 模式下，将计数器的计数值即时显示在数码管上，以便直观地观察定时的运行效果；

e) 在 MODE_DEMO 模式下，利用数码管的小数点（“dp”位）显示状态机的状态，由于小数点只有 2 个，而状态机的状态应是超过 4 个，请合理地将状态机的状态编码转化为易于观察的状态符号，并配合小数点的闪动实现超过 4 种的状态显示；

f) 采用 FPGA 开发工具实现相应的逻辑功能并加以演示；

g) 在实验报告中，请绘制设计的框图（**此项为必选项**）；

h) 在实验报告中，请说明采用何种风格（一段式、二段式、三段式）实现时序逻辑电路的状态机，并切实地分析所采用的实现形式的优缺点。

本实验将采取**实验报告和现场汇报演示相结合**的形式，实验报告上交的具体**截止期限**请注意任课教师的通知，实验报告格式请参考本文档；**现场汇报演示**的时间和分组情况请注意任课老师的通知。特此广而告之。

1.3 扩展要求

扩展的实验要求（根据完成的工作量逐次提高评价的等级，但鼓励大家积极地去实验——即：只要至少完成其中一项，即可显著地获得好评，并可根据局部完成的质量酌情提高评价的等级。）

i) 开发时序逻辑电路，将按动开关的操作（“开”、“关”）以及按动开关之间的时间间隔（以 10ms 为单位，进行 0~127 编码，分别代表 0s 至 1.27s，超过 1.27s 秒均算作 1.27s）以数码的形式“录制”下来，记录深度至少为 16 条操作序列；

j) 按动实验板上的一个按键，可以将记录的内容用 UART 依次上传并显示到 PC 机终端上；

k) 开发新的时序逻辑电路功能，改动 MODE_DEMO 状态下的逻辑功能，当按动实验板上的另一个按键，可以根据“录制”的操作序列自动地“慢动作”演示上一次的操作（时间放大 10 倍）；

l) 可以利用 UART 工具，从 PC 机终端上输入的操作序列，自动地进行硬件操作，并回显电路状态的变化。

1.4 其它提示

实验板的开关非常脆弱，如果操作不便，请用按键代替开关，但需要加入防抖电路，并适当修改开灯、关灯信号的逻辑（由电平信号变为脉冲信号）。

“实验板”的 I/O 接口和人机接口资源请参考《digiC2018 课程实验实验指导书(01)》，或参考实验一手册后面的附录。

实际中的三色灯灯芯，以及基本功能效果的录像将发布在网盘上，网盘地址在 course.buaa.edu.cn 中另行通知。如果实在读不清楚设计需求，可以结合录像进行理解。

实验三、时序逻辑设计——三色灯开关（实验报告部分）

2 实验报告

2.1 需求分析

三色灯的三种颜色变换可以使用有限状态机来实现，不同的切换条件对应不同的次态。状态机采用一段式进行描述，编码简单，便于调试，也可以用三段式实现。

设置计时器记录间隔时间，间隔时间用数码管即时显示。用 LED 灯和小数点来表示当前状态，小数点可以分为长灭、长亮和闪烁三个状态，两个小数点最多可以表示九个状态。实验还要求两种不同的工作状态，设置了不同的时间常数来应对这两种情况。

拓展实验中要求记录 16 条操作序列，建立二维数组来储存。传输环节使用 UART 与 PC 端进行交互，在已有的基础上改进了串口发送程序。

2.2 系统设计

2.2.1 总体设计思路

首先我们需要实现作为主体功能的状态机模块，由于开发板的拨码开关较为脆弱，状态机换用加入防抖电路的按键来控制，以脉冲信号代替电平信号控制状态的转换。作为人机接口，电路分为两种工作模式，通过第一路拨码开关切换两种模式，MODE_RUN 为正常操作模式，MODE_DEMO 为演示模式，后者的时间单位为前者的十倍，通过一个按键来模拟开关，从而实现基本的状态机操作，用数码管来显示计数器的计数值，小数点显示状态机的状态，而四个 LED 灯两个用来模拟白灯灯芯，另外两个用来模拟黄灯灯芯，日光时只需各点亮一个即可。另设计一个模块用来记录操作序列，保存在一个寄存器堆中，当按下另一个按键时通过 UART 发送给终端。除此之外，我们还需要修改状态机的控制电路，使其能够实现依据已经保存的操作序列自动地演示录制的操作，进一步修改 UART 使其能够接收终端的控制信号，自动操作电路并且回显状态机的变化。系统组成框图如下：

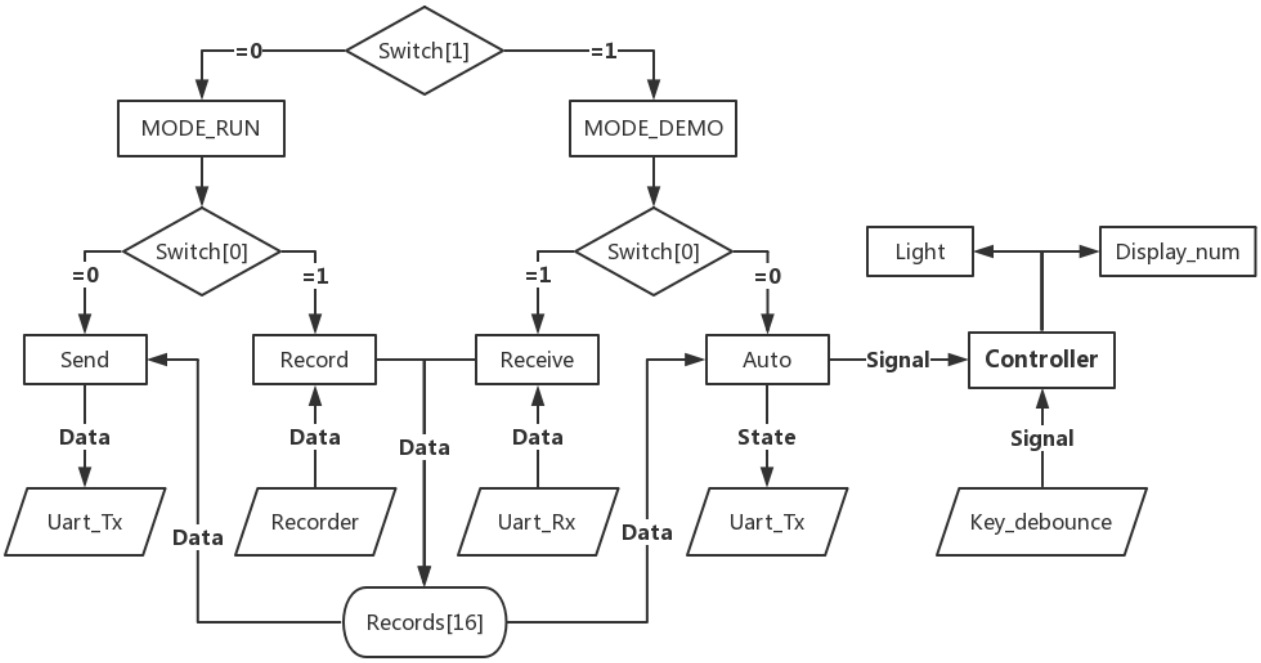

图 1 系统组成框图

2.2.2 接口设计

实验板提供的 IO 分别有系统时钟，复位，拨码开关，按键，数码管管脚，选通端，LED 灯和 UART 的发送端 Tx 以及接收端 Rx，这些接口必须先在顶层模块中定义，由于按键去抖模块和数码管显示模块均需要时钟信号控制，所以也将系统时钟作为他们的输入。除此之外，按键去抖模块还需要按键信号输入，输出去抖后的按键信号，同时也在必要的时候将复位信号分配给相应的模块。

灯具控制电路的核心状态机需要按键来进行操作，故分配一个按键信号给此模块，还需要输入当前电路的运行模式是正常模式还是演示模式，同时该模块输出灯芯信号，状态机的状态和计时器当前的数组供给其他模块使用。由于数码管显示模块需要显示计时器的状态和状态机的状态，所以以这两个信号作为输入，输出相应的管脚控制信号，LED 灯显示模块则以灯的信号作为输入。操作序列录制模块计时器在内部，故只需输入系统时钟信号和按键信号，同时输出当前一个操作的编码和序列长度下标传给顶层模块。由于 UART 在此电路中需要承担较复杂的功能，所以接口也较多，除了必需的时钟信号和 UART 端口信号外，还需要输入序列信号用于发送，以及输出从终端接收到的序列信号。

2.2.3 一段式状态机模块

初次观察问题时，我们将状态按实验要求粗略分为 WHITE、YELLOW、SUN 和 OFF 四种，经过进一步的深入分析，我们发现灯灭之后不同的等待时间对应不同的状态，于是我们将“OFF”状态进行了分割，根据状态机次态的灯色将其一分为三——“WHITE OFF”、“YELLOW OFF”、“SUN OFF”，

再加上之前的“WHITE”“YELLOW”“SUN”共六个状态。

状态图如下：

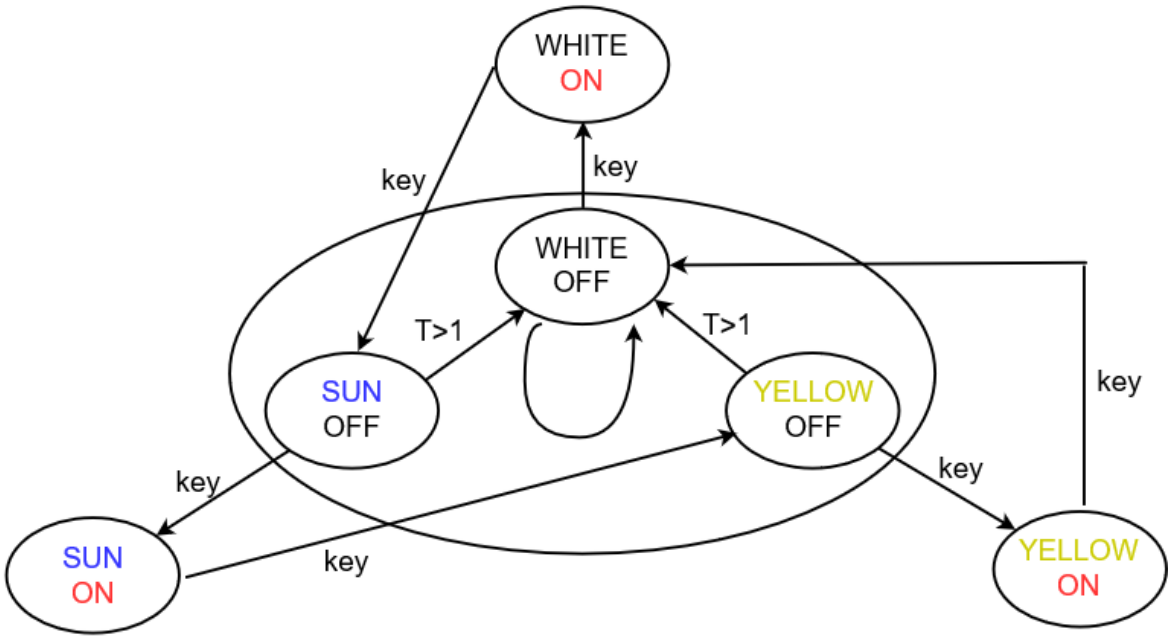

图 2 一段式状态机转换图

设置 WHITE OFF 为初始状态，按动开关，检测到 key 信号后，跳到 WHITE ON 状态，此时表示白灯亮。

再次按动开关，跳至 SUN OFF 状态，同时计数器开始计时，如果 T>1 时还未检测到 key 信号，跳至 WHITE OFF 状态，直到检测到 key 信号，再次跳到 WHITE ON 状态，否则将一直在 WHITE OFF 状态自循环；如果 key 信号在 T=1 之前到达，将会跳至 SUN ON 状态，日光灯亮。

同理再次按动开关将跳至 YELLOW OFF 状态，逻辑同 SUN OFF。

如果处于 YELLOW ON 状态，检测到 key 信号后就会直接跳至 WHITE OFF 状态。

综上，状态机的总体设计思路就是将 OFF 状态一分为三，在这三个状态中同时检测 key 信号和时间是否大于 1，两个条件中只要满足一个即会跳到下一个状态。

状态机采用一段式进行描述，编码简单，便于调试。但是可能会出现毛刺，稳定性相对较差。

2.2.4 三段式状态机模块

首先分析需求：根据开关的状态从而判断对应的输出。所以，可以将需求抽象出来，即 Mealy 型状态机——输出是由输入和现态决定的。而状态机的次态是由输入的变化和现态决定的。接下来，根据需求，配合 Mealy 型状态机和三段式状态机的特点继续完成设计即可！

易知，Mealy 型状态机的次态是由输入的变化和现态决定的。则我们首先要构建这个输入信号（开

关断开闭合状况以及间隔时间）。为此，我们首先构建一个使能信号 Q，依据按键输入的变化而改变，从而表示开关的断开或闭合；其次，我们在状态机外构建一个减法计数器 num（从 99->0,到 0 时保持不动），根据使能信号 Q 来进行工作——当 Q=0（开关断开时），先置为 99，再以 0.1s 为间隔计时；当 Q=1（开关闭合时），num 保持不变；

这样，我们就建立了 Mealy 型状态机的两个输入信号——Q 和 num，从而可以完全模拟需求中的各种输入情况。下来就是根据三段式状态机的特点按步骤构建状态机。

第一段：同步时序 always 模块，标准化描述次态寄存器迁移到现态寄存器。这一部分很简单，仅仅是将状态转移，所以主代码为

```
now <= next;
```

第二段：组合逻辑 always 模块，描述状态机状态转移的条件的判断。

这一部分我们只要成功描述状态机的流程，就可以最后转换为 Verilog HDL 程序。首先根据分析，我们定义 6 个状态 white_on、white_off、sun_on、sun_off、yellow_on、yellow_off;

当 now 处于开关断开状态时，如果 Q=0, 即开关仍断开，则 next 仍然为当下状态；如果 Q=1，即开关闭合了，则根据拨码开关和 num 判断 next 应该为哪一个亮灯状态即可；

当 now 处于开关闭合状态时，如果 Q=1，即开关仍闭合，则 next 仍然为当下状态；如果 Q=0，即开关断开，则 next 为对应灯色的关闭状态。

同步时序 always 模块，标准化描述次态寄存器输出。根据 now 按情况输出即可，此处不再赘述。

最终的框图如下：

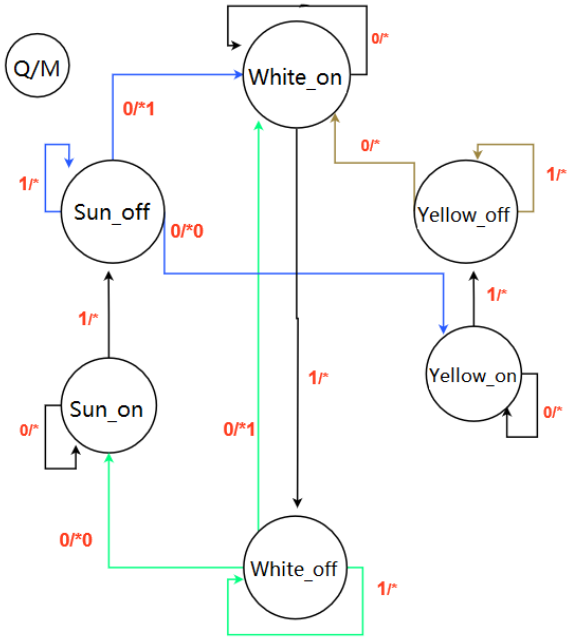

图 3 三段式状态机转换图

上图中，num 是时序电路的计数器（两位），当灯灭开始计时，灯亮停止计时，Q 为 Mealy 型状态机的一个输入，Q 是 0 时计数器停止，Q 是 1 时计数器开始计时。M[1:0]是第二个输入，M=2’b00 时处于 MODE_RUN 且小于 1s，M=2’b01 时处于 MODE_RUN 且长于 1s，M=2’b10 时处于 MODE_DEMO 且小于 10s，M=2’b11 时处于 MODE_DEMO 且长于 10s。

优点：

1. 消除了组合逻辑输出的不稳定与毛刺的隐患，而且更利于时序路径分组

因为对于三段式状态机来说，状态转移部分是同步时序电路，而状态的转移条件的判断是组合逻辑电路，则这样就可以较好的提高系统的稳定性，消除了毛刺的隐患。

2. 逻辑清晰易懂，便于阅读和维护

可以从前面的分析中开出来，第一段就是状态转移，第二段就是条件判断，第三段就是结果输出，每一部分功能清晰明了，从而便于阅读和修改。

缺点：代码块多，不简洁

2.2.5 数码管及 LED 灯显示模块

本实验中数码管数字显示模块与之前相同，这里主要说明小数点显示和 LED 灯显示功能的实现。

由于我们需要用两个小数点来显示超过 4 种状态，所以我们需要在亮和灭外再增加一种显示形式——闪动，这样每个小数点就有三种状态，两个即可配合显示九种状态，本实验中状态机只有六个状态，满足要求。

每个小数点只能表示三种状态，而状态机编码与其不一致，所以我们需要重新对小数点的状态进行编码，每两位表示一个小数点：

```
always @(state) begin
   case (state)
      3'b000: encoded_state <= 4'b0000;
      3'b001: encoded_state <= 4'b0001;
      3'b010: encoded_state <= 4'b0100;
      3'b011: encoded_state <= 4'b0010;
      3'b100: encoded_state <= 4'b1000;
      3'b101: encoded_state <= 4'b0101;
      default:encoded_state <= 4'b0000;
   endcase
end
```

为了实现小数点的闪动，我们建立了新的分频时钟，让小数点闪动信号每 0.1s 变化一次：

```
parameter dp_interval = 5000000; // 100ms
always @(posedge clk or negedge rst) begin
```

```
    if (!rst) begin
        dp_cnt <= 0;
        dp_state <= 0;
    end
    else begin
        dp_cnt <= dp_cnt + 1'd1;
        if (dp_cnt == dp_interval) begin
            dp_cnt <= 0;
            dp_state <= ~dp_state;
        end
    end
end
```

最后根据小数点的状态编码判断当前显示形式即可：

```
case (Light)
    2'b00: Led <= 4'b0000;
    2'b01: Led <= 4'b0011;
    2'b10: Led <= 4'b0101;
    2'b11: Led <= 4'b1100;
endcase
```

对于 LED 灯显示模块，我们需要根据三色灯的实际情况改变信号，这里假设 LED 灯低两位为白光灯芯，高两位为黄光灯芯，而日光为一白一黄，则很容易实现该模块：

```
always @(*)
    case (Light)
        2'b00: Led <= 4'b0000;
        2'b01: Led <= 4'b0011;
        2'b10: Led <= 4'b0101;
        2'b11: Led <= 4'b1100;
    endcase
```

2.2.6 操作序列录制模块

我们需要实现以 10ms 为单位录制操作序列，这里单独使用 Recorder 模块来实现单个操作序列的记录，然后传输到顶层模块中以寄存器堆进行存储。

录制时需要在模块内实现以 10ms 为时间单位的计数器，可以将记录的时间自然地编码为 0-127，共需要 6 位：

```
if (Key_In || !Recording) begin
    cnt <= 0;
    number <= 7'b0;
end
else begin
    if (cnt == unit_interval) begin
```

```
        if (number < max_num)
           number <= number + 1;
        cnt <= 0;
     end
     else
        cnt <= cnt + 1;
  end
```

然后每当按键信号到来时，将信号拼接发送给顶层模块，同时改变当前序列的长度，由于这里是按键脉冲信号，无法直接得知是开还是关，所以还需要配合灯的状态来判断是开灯还是关灯：

```
  if (Key_In) begin
     if (Light == 2'b00) begin
        opList[7] <= 1'b1;
     end
     else begin
        opList[7] <= 1'b0;
     end
     opList[6:0] <= number;
     Index <= Length + 1;
  end
```

最后在顶层模块中更新寄存器堆的内容：

```
  if (length != index && length < 16) begin
     length <= index;
     records[length] <= oplist;
  end
```

2.2.7 UART 上传模块

为了让 UART 在按键信号到来时将寄存器堆记录的操作序列按顺序发送给终端，我们给 Uart 的顶层模块加入了发送使能和发送空闲信号，当发送空闲且按键信号到来时，修改发送使能为高电平，如此按顺序将寄存器堆的内容发送即可：

```
  if (key_out[1] && !sending) begin // Send start
     sending <= 1'b1;
  end
  if (sending) begin // Sending
     if (send_idle && !send_enable) begin
        if (send_index == max_index) begin
           sending <= 1'b0;
           send_index <= 5'b0;
        end
        else begin
           send_enable <= 1'b1;
```

```
            send_data <= records[send_index];
            send_index <= send_index + 1;
         end
      end
      else if (!send_idle) begin
         send_enable <= 1'b0;
      end
   end
```

为了让程序更易读，我们利用 C 语言用打表的方法将记录的二进制序列转化为易读的信息，格式为 t.tts:on/of\r，C 语言打表程序如下：

```
void dec2bin(int x)
{
   if (x > 1)
      dec2bin(x >> 1);
   printf("%d", x & 1);
}
int main()
{
   for (int i = 0; i < 128; i++) {
      printf("7'b");
      dec2bin(i);
      printf(": begin ");
      printf("Print[0] <= 8'd%d; ", i / 100 + '0');
      printf("Print[2] <= 8'd%d; ", i / 10 % 10 + '0');
      printf("Print[3] <= 8'd%d; ", i % 10 + '0');
      printf("end // %d\n", i);
   }
}
```

2.2.8 自动演示模块

为了实现自动演示功能，我们修改了状态机的输入信号，将状态机的输入 Key 分为了两个信号：Key_In 和 Key_Auto，前者为真实的按键信号，后者为自动演示时模拟出的按键信号：

```
if (Auto)
   Key <= Key_Auto;
else
   Key <= Key_In;
```

模拟的按键信号通过一个计时器来判断，当计时器的数值等于录制的内容时，将 Key_Auto 设为高电平保持一个时钟周期即可。

```
if (Key_Auto) // Restrict to a cycle
   Key_Auto <= 1'b0;
```

```
if (running) begin
   if (auto_cnt == unit_interval) begin
      auto_cnt <= 0;
      if (auto_num == Auto_data) begin
         Key_Auto <= 1'b1;
         auto_num <= 7'b0;
         running <= 1'b0;
         Auto_idle <= 1'b1;
      end
      else
         auto_num <= auto_num + 1;
   end
   else
      auto_cnt <= auto_cnt + 1;
end
else if (Auto_enable) begin
   running <= 1'b1;
   Auto_idle <= 1'b0;
end
```

2.2.9 UART 接收及回显模块

根据实验要求，系统需要能够实现通过 UART 串口接受终端发送的操作序列并且自动操作，回显电路的状态，所以我们给 UART 顶层模块的状态机增加了一种接收状态和发送状态。

接收状态将当前的序列记录下来并且发送到顶层模块，保存在寄存器堆中：

```
if (!Error_Flag) begin
   case (Data_Rx)
      8'd48: Recv_Reg[7-Recv_Cnt] <= 1'b0;
      8'd49: Recv_Reg[7-Recv_Cnt] <= 1'b1;
   endcase
   Recv_Cnt <= Recv_Cnt + 1;
   State <= Wait_Rdsig;
end
else
   State <= Trans_Data;
```

而每当状态机状态改变时，顶层模块将状态发送给 UART 模块：

```
if (light != light_old) begin
   light_old <= light;
   if (send_idle && !send_enable)
      send_enable <= 1'b1;
end
if (!send_idle) begin
```

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

UART 模块来回显当前的状态：

```
if (Echo_Cnt == 4'd7) begin
    Echo_Cnt <= 4'd0;
    State <= State_Delay;
end
else begin
    if (cnt == 254) begin
        Data_Tx <= String[Light][Echo_Cnt];
        Wrsig <= 1'b1;
        cnt <= 8'd0;
        Echo_Cnt <= Echo_Cnt + 1;
    end
    else begin
        Wrsig <= 1'b0;
        cnt <= cnt + 8'd1;
    end
end
```

2.3 功能仿真测试

2.3.1 测试程序设计

为了真实地模拟 verilog 程序在 FPGA 上的运行过程，我们小组将仿真的时钟周期于 FGPA 设置为相同。这样通过 ISIM 显示出来的时间就是烧在 FPGA 上程序运行的时间。为了实现这一目的，首先要更改时钟信号的周期。由于程序的时延单位只能是十的倍数，而 FGPA 时钟默认频率为 50MHZ，所以不能简单地修改 timescale 中的参数来实现。50MHZ 说明一个时钟周期为 2×〖10〗^(-8) s,也就是 20ns。这样仍然取 1ns 为时延单位，每 20 个时延单位时钟信号翻转一次。这样仿真出来的时钟信号周期就和 FPGA 实际运行的时间一致了。

但是这样仿真又遇到了新的问题，之前按键去抖时用的计数器记 3ffff 个时钟信号，也就是 5242860ns（5.4ms）才会认为一次按键信号按下。所以在仿真时输入的 key 信号的延时也需要满足这一要求，才能模拟出按键按下。同时还要注意键入的 key 信号的延迟时间并不与 key_out 信号延迟时间对应。

首先仿真 1s 内转换的状态，首先产生一个按键为 1 的 90ms 的信号，然后再仿真 90ms 的低电平按键信号，这样产生一个按键信号，白灯亮。再次仿真一个按键信号，白灯灭；继续重复上述状态，实现了对日光灯亮、日光灯灭、黄灯亮、黄灯灭的仿真。

接下来实现对关灯超过 1s 的状态的仿真，在上面的仿真中当日关灯灭之后，key 低电平保持 1s,这样保证去抖之后的 key_out 大于 1s,然后再仿真一个 key_out 高电平，发现状态跳回到白灯。

继而类似上面的仿真，实现了从白灯到白灯的跳变。

至此状态机仿真完毕。

接下来对拓展功能 i、j、k 进行仿真，观察 Data 与 LED 两个信号，发现 Data 表示的 8bit 的数字与 LED 状态变换相符，再在 tb 文件中写入通过串口传输进去的数据，仿真后看到 LED 随数据变化，仿真成功。

2.3.2 功能仿真过程

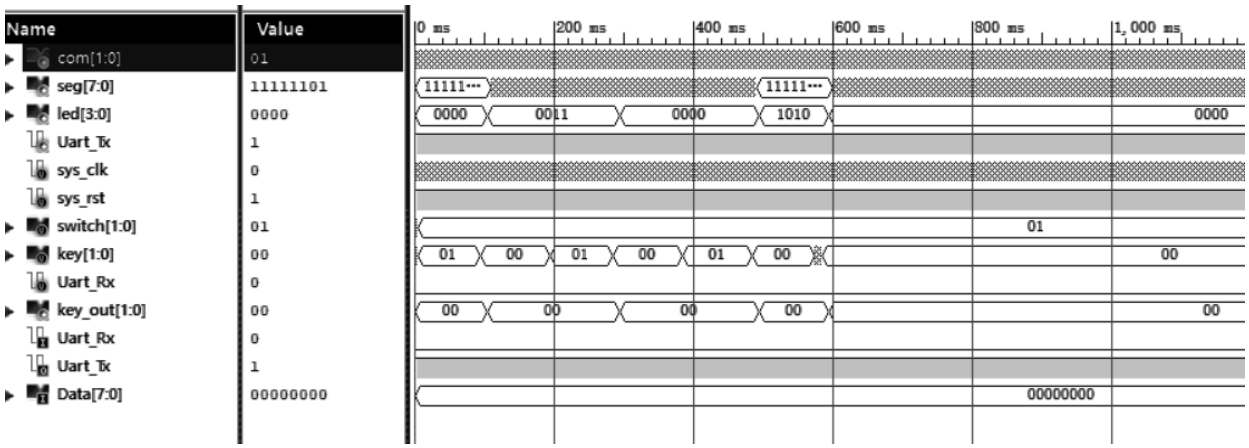

图 4 状态机仿真

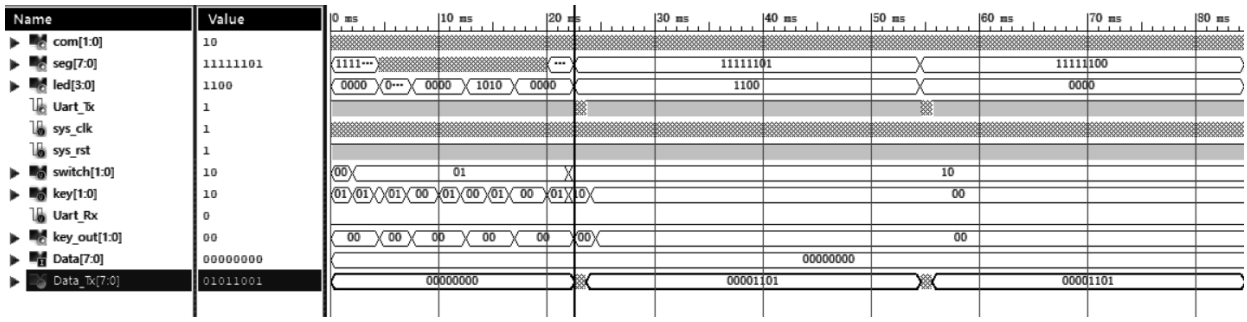

图 5 扩展 i, j, k 仿真

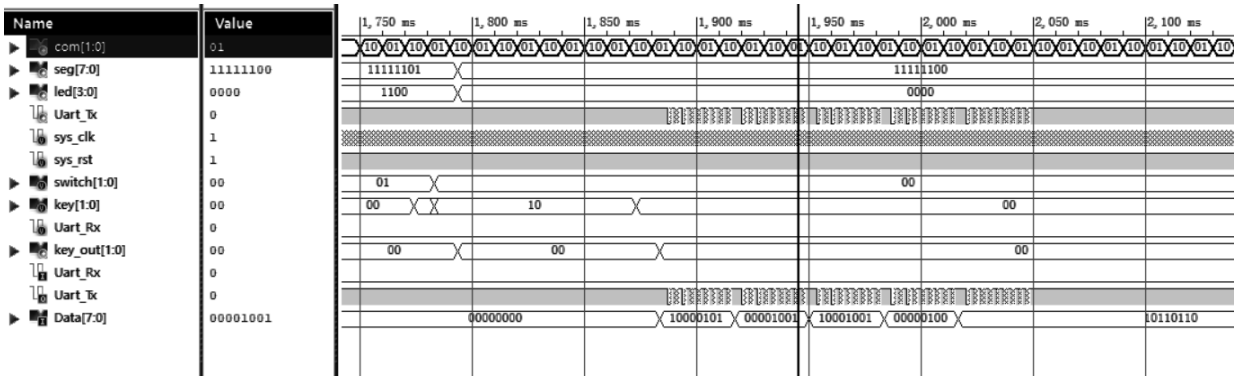

图 6 扩展 I 仿真

2.3.2 实验关键结果及其解释

从图一看出，通过对 key 的信号的仿真，实现了状态机白灯亮（LED 为 0011）、日光灯亮（LED 为 1010）、黄灯亮（LED 为 1100）几个状态的跳变，而且受时钟的控制。证明了基本功能已经完成。

从图二可以发现，当把 swich 设到 0，按下 key[1]后，FPGA 通过 Uart 把存储的 8bit 的操作信息传输出来，Data[7]是表明灯的开关状态，Data[6:0]表明计数器记的时间。比如第一组数据 10000101，表明之前有 50ms 的关闭状态，然后灯亮。证明拓展的 i、j、k 功能实现。

从图三可以看出，当通过串口把多组数据传入 FPGA 时，FGPA 分析数据后自动进行开关灯以及延时操作，证明拓展的 l 也实现。

2.4 设计实现

2.4.1 综合和下载过程

语法检验通过后，通过 PlanAhead 进行引脚分配。将系统用到的 8 个数码管引脚，2 个数码管片选信号引脚，2 个按键信号引脚，4 个 LED 引脚，2 个拨码开关引脚、UART 通信的 Tx 端、Rx 端和时钟信号、复位信号绑定到实验板上，分配好之后生成 Top.ucf 文件。

Name	Direction	Neg Diff Pair	Site	Fixed
All ports (20)				
com (2)	Output			
com[1]	Output		P27	☑
com[0]	Output		P26	☑
key (2)	Input			
led (4)	Output			
seg (8)	Output			
switch (2)	Input			
Scalar ports (2)				

图 7 引脚分配

然后在 ISE 中进行综合，观察电路原理图：

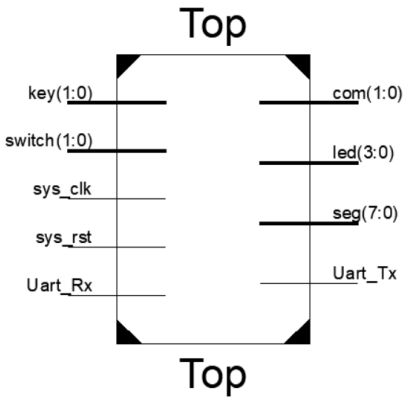

图 8 电路原理图

最后在 iMPACT 中将程序烧录到实验板中：

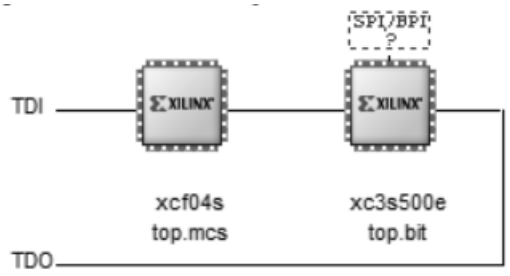

图 9 程序烧录

2.4.2 实验关键结果及其解释

接通电源后，在第一路拨码开关为“0”时，为 MODE_RUN 模式，按下按键对电路进行连续操作，观察到数码管显示当前计数器的数字，小数点也正确表示了当前状态，LED 灯发生与预期相同的变化，证明系统工作正常。接着把第一路拨码开关拨到“1”，观察 MODE_DEMO 模式下的系统工作状态，依旧正常。

接着我们使用 UART 测试拓展功能，首先记录下一系列操作后，将操作序列发送到终端：

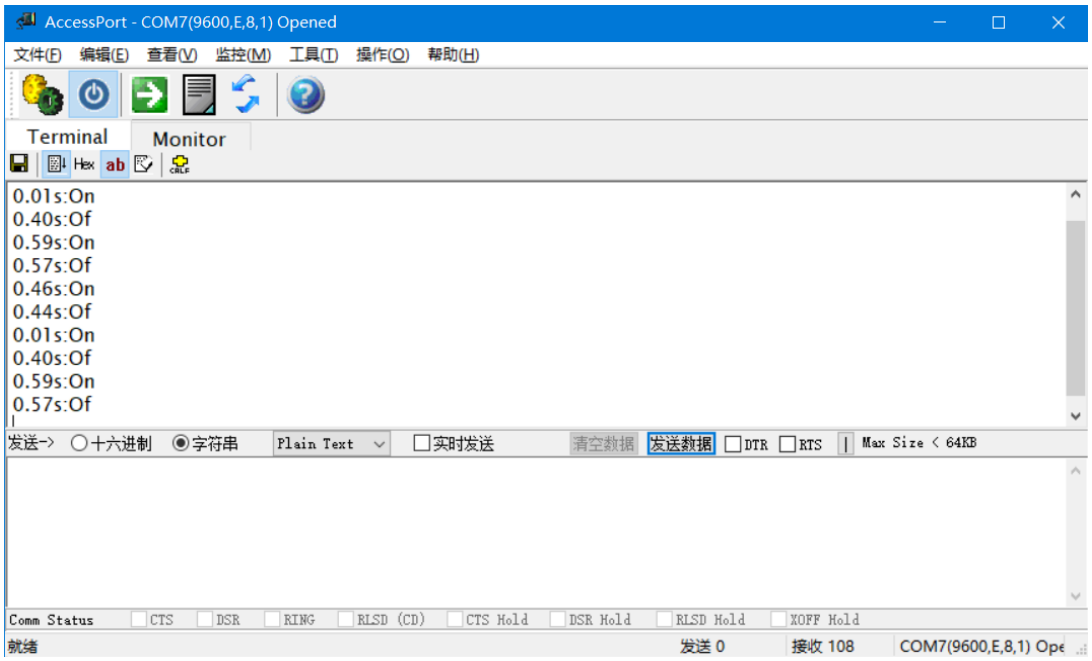

图 10 通过 UART 上传操作序列

接着从终端发送操作序列使系统自动操作并且反馈当前状态：

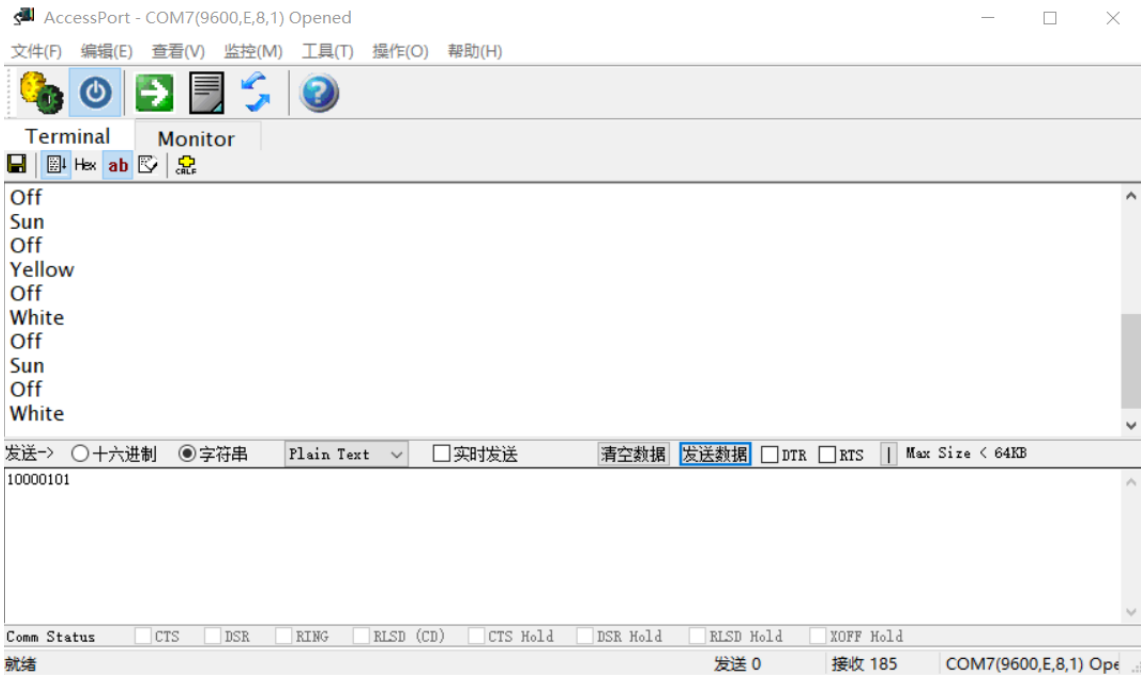

图 11 接收操作序列自动操作并反馈状态

2.5 小结

通过此次实验，我们学习到的知识有：

1. 有限状态机的基本概念和描述。
2. 利用一段式、二段式、三段式来描述状态机的利弊。
3. 利用状态机描述并解决一个实际问题。
4. 对于数字电路中的延迟有了进一步的认识。
5. 人机交互接口的综合。
6. 编写合理且全面的 testbench 对实验结果进行测试。

附注：本次实验所用代码已经上传到 Github 仓库：

https://github.com/hiyouga/digiC-experiment/tree/master/ex3