一. 实验目的

- 有限状态机概念；
- 掌握分层次电路设计方法；
- 熟练掌握数字钟的设计与调试方法；
- 掌握用 Verilog HDL 描述数字逻辑电路与系统的方法；；
- 掌握用 Verilog HDL 描述有限状态机的方法；
- 掌握基于 Verilog HDL 语言的分模块、分层次小型数字系统设计方法；
- 熟练掌握 Vivado 集成的仿真工具的使用；
- 熟练掌握 Vivado 集成的调试工具的使用 ；
- 掌握 Verilog HDL 语言数字钟实现原理 。

二. 实验任务

（1） 步进电机脉冲分配器：

采用 Verilog HDL 语言设计一个能够自启动、具有正反转功能的三相六拍步进电机脉冲分配器电路，电路的状态图如图 1 所示：

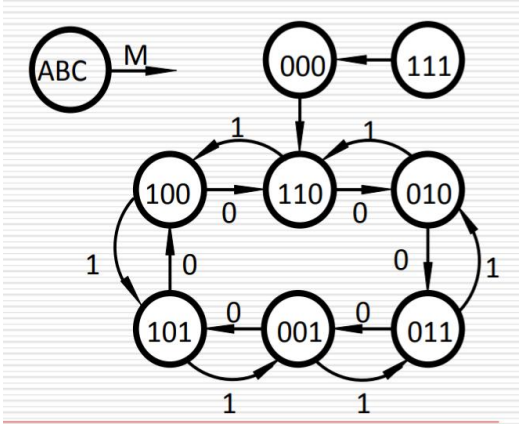


图 1 步进电机电路状态图

输入输出如图 2 所示。M=0 时，按顺时针方向转；M=1 时，按逆时针方向转。

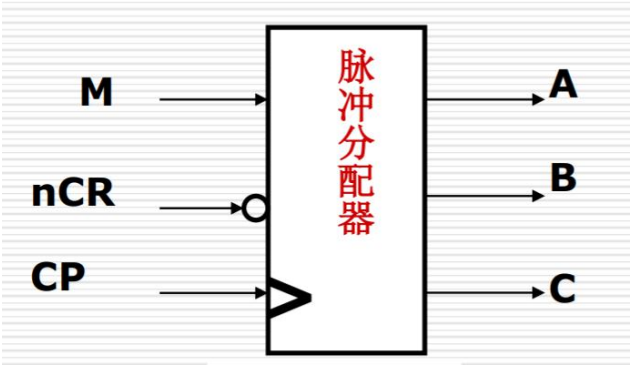


图 2 步进电机输入输出

（2） 多功能数字钟：

- 基本要求

a） 数字形式显示时、分、秒的时间；
b）小时计数器为同步 24 进制；
c）可手动校时、校分。

- 提高要求：

a）任意时刻闹钟；
b）小时显示（12/24）切换电路；
c）仿广播电台正点报时且自动报整点时数。

三.设计平台

嵌入式软件：Vivado 2018.1
仿真软件：Vivado
硬件平台：Nexys4

四.电路设计与实验仿真

（1） 开发流程：

·层次化、模块化

对于一个复杂的数字系统，运用层次化设计方法，使设计课题进一步细化，分块设计，条理清晰。另外，在调试时可采用逆向调试方式，即从模块调试向总体调试方向开展调试工作，使设计中出现的问题在模块级就能发现，及时处理，这样就会使一个复杂的设计变得容易调试，缩短了设计时间。

·自顶向下

先设计顶层总框图，该框图由若干个具有特定功能的源模块组成。下一步针对这些具有不同功能的模块进行设计，对于有些功能复杂的模块，还可以将该模块分为子模块，这样就形成模块套模块的层次化设计方法。

（2） 步进电机脉冲分配器：

步进电机脉冲分配器属于有限状态机，状态机中所有触发器的时钟输入端被连接到一个公共时钟脉冲源上，其状态的转换是在同一时钟源的同一脉冲边沿同步进行的，所以它也被称作时钟同步状态机。
根据电路的状态图，得到该电路的八个状态并对应图中的编码。采用 always 语句块描述状态触发器实现状态存储。使用 case 语句描述状态的转换逻辑，最后描述状态机的输出逻辑。

① 实现代码：

定义模块：输入 M 正反转控制，nCR 清零，clk 时钟周期；输出下一次的状态。

```
module StepMotorPorts(
    input M,
    input nCR,
    input clk,
    output reg A,
    output reg B,
    output reg C
    );
```

分频计数:步进电机的状态交替需要频率更低的时钟周期，在代码钟定义计数器以及计数阈值，当计数器达到计数阈值，就对步进电机的状态进行相应的改变同时清零计数器。

```
    reg[31:0] StepCounter = 32'b0;
    parameter[31:0] StepLockOut = 32'd100000000;            //250HZ
    reg InternalStepEnable;
    parameter S0=3'b111,S1=3'b000,S2=3'b110,S3=3'b010,S4=3'b011,S5=3'b001,S6=3'b101,S7=3'b100;

    always @(posedge clk or negedge nCR)
    begin
       if ( !nCR)
        begin
            {A,B,C} <= 3'b000;
            StepCounter <= 32'b0;
        end
```

状态切换：根据 M 的值与当前值，进行状态切换。

```
        else
        begin
           StepCounter <= StepCounter + 31'b1 ;
           if (StepCounter >= StepLockOut)
           begin
               StepCounter <= 32'b0;
               case({A,B,C})
                  S0:{A,B,C} <= S1;
                  S1:{A,B,C} <= S2;
                  S2:{A,B,C} = (M)?S7:S3;
                  S3:{A,B,C} = (M)?S2:S4;
                  S4:{A,B,C} = (M)?S3:S5;
                  S5:{A,B,C} = (M)?S4:S6;
                  S6:{A,B,C} = (M)?S5:S7;
                  S7:{A,B,C} = (M)?S6:S2;
               endcase
           end
        end
    end
endmodule
```

② 实验仿真:

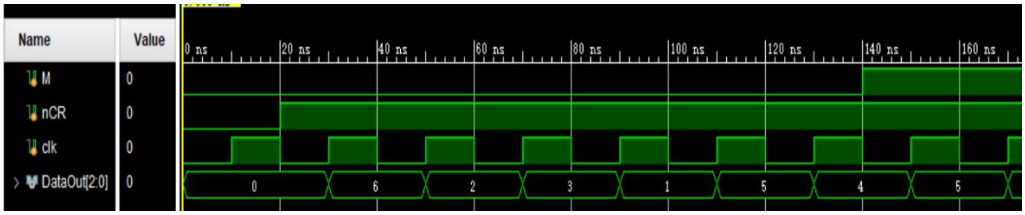
使用 Vavido 对该模块进行功能仿真。

设置仿真文件，代码如下:

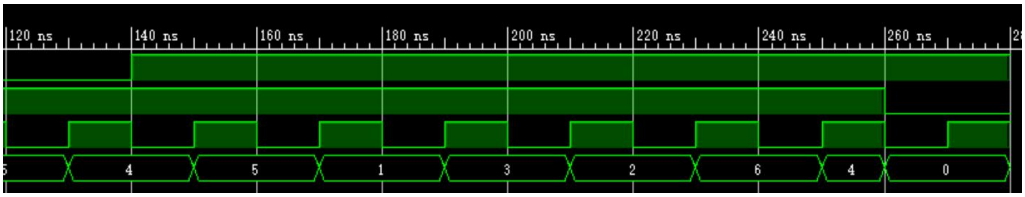
```
module sim_1(

    );
    reg M,nCR,clk;
    wire [2:0] DataOut;
    StepMotorPorts motor(M,nCR,clk,DataOut);

    initial begin
    clk = 0;
    forever #10 clk = ~clk;
    end

    initial begin
    M=1'b0;
    nCR=1'b0;
    #20;
    M=1'b0;
    nCR=1'b1;
    #20;
    M=1'b0;
    nCR=1'b1;
    #20;
     M=1'b0;
     nCR=1'b1;
     #20;
     M=1'b0;
     nCR=1'b1;
     #20;
     M=1'b1;
     nCR=1'b1;
     #20;
     M=1'b1;
     nCR=1'b1;
     #20;
     M=1'b1;
     nCR=1'b1;
     #20;
      M=1'b1;
      nCR=1'b0;
    #20;
    $stop;
    end
endmodule
```

设置六个时钟周期的正转以及六个时钟周期的反转，仿真分析结果如下：

可以看到，前面几个状态，当 M=0 时，步进电机进行正转。

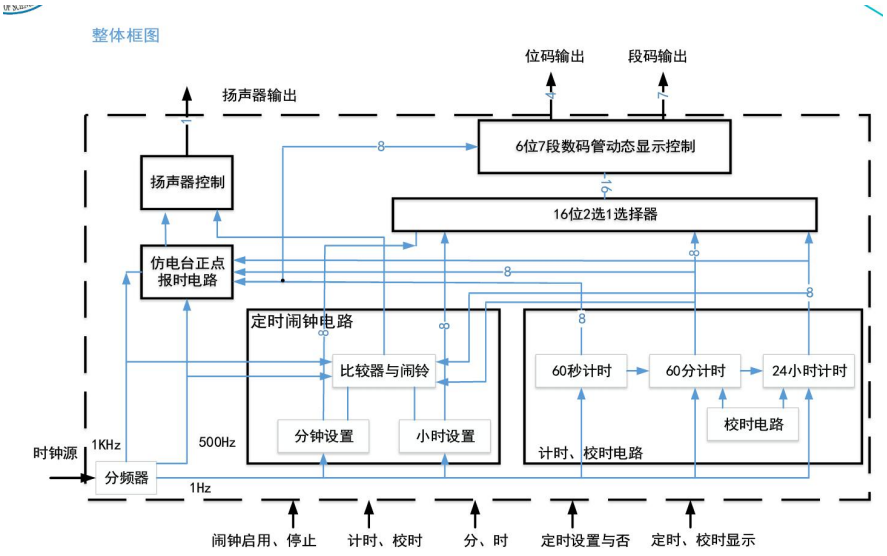


接下来 M=1，步进电机进行反转，当 nCR 跳变至低电平，DataOut 清零。



（3） 多功能数字钟：

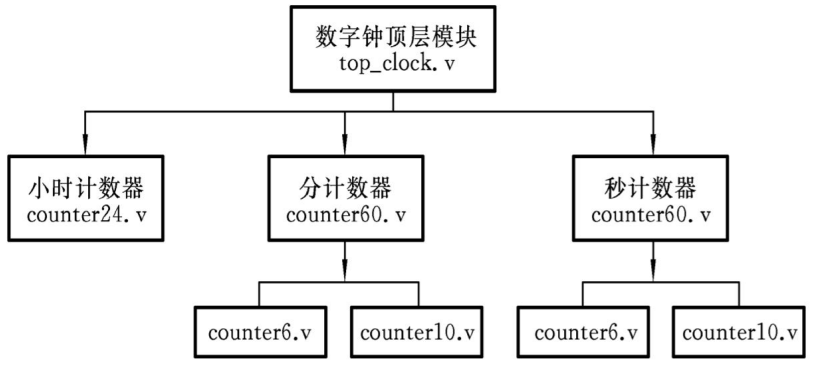
① 总体框图（参考 MOOC）：



② 模块实现：

a） 计时模块（主体）：

模块框图：



实现代码：

Top_clock：

```
module jishi_jiaoshi(
input clk,
input adjust,
input cr,
input min_hour,
output [3:0] bcd_su,
output [3:0] bcd_st,
output [3:0] bcd_mu,
output [3:0] bcd_mt,
output [3:0] bcd_hu,
output [3:0] bcd_ht
    );

 wire m_en,h_en,rco_s,rco_m;
 assign m_en = rco_s ||(adjust&&min_hour);
 assign h_en= (rco_s&&rco_m)|(adjust&&~min_hour);
 c60 second(clk,1'b1,cr,rco_s,bcd_st,bcd_su);
 c60 minute (clk,m_en,cr,rco_m,bcd_mt,bcd_mu);
 c24 hour(clk,h_en,cr,bcd_hu,bcd_ht);
endmodule
```

Counter（以 counter10 为例）：

```
module c60(
input clk,
input en,
input cr,
output rco,
output [3:0] bcd_t,
output [3:0] bcd_u
    );
wire rco_u,rc_t;
assign rco=rco_u&rco_t;
c10 units(clk,en,cr,rco_u,bcd_u);
c6 tens(clk,cr,rco_t,rco_u&&en,bcd_t);
endmodule
```

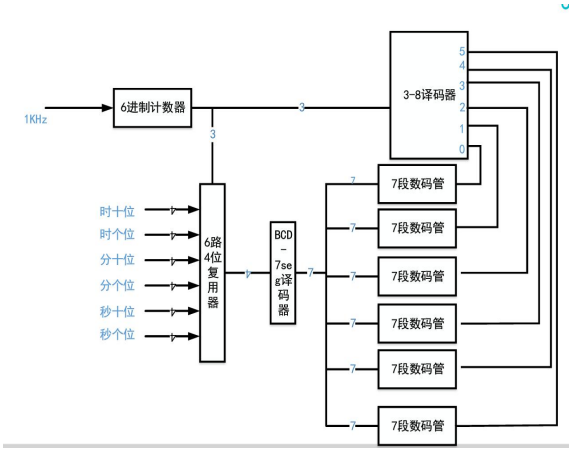
b） 分频器模块：

实现代码：

```
module clock_clk(
    input clk_100m,
    output clk_1k,
    output clk_5h,
    output clk_1hz,
    input cr
    );
    reg [15:0] count1k;
    reg [8:0]  count1hz;
    reg clk_1kr,clk_5hr,clk_1hzr;
    assign clk_1k = clk_1kr;
    assign clk_5h = clk_5hr;
    assign clk_1hz = clk_1hzr;
    always @(posedge clk_100m or negedge cr)
        if(!cr)
        begin
        count1k <= 16'h0000;
        clk_1kr <= 1'b0;
        end
            else begin
              if (count1k == 16'd49999)
              begin
              count1k <= 0;
              clk_1kr <= ~clk_1kr;
              end
              else count1k <= count1k+1'b1;
              end
    always @(posedge clk_1k or negedge cr)
    if(!cr)
    begin
    count1hz<= 9'h000;
    clk_1hzr <= 1'b0;
    end
       else begin
         if (count1hz == 9'd124)
         begin
         count1hz <= 0;
         clk_1hzr <= ~clk_1hzr;
         end
         else count1hz<= count1hz+1'b1;
         end
  always @(posedge clk_1k or negedge cr)
    if(!cr)
    begin
```

c） 数码管显示模块：

模块框图：



实现代码：

```
module posdecode(
    input [3:0] bcd6,
    output [7:0] pos
    );
    reg [7:0] posr;
    assign pos = posr;
    always@(bcd6)
    case(bcd6)
    4'b0000: posr <= 8'b11011111;
    4'b0001: posr <= 8'b11101111;
    4'b0010: posr <= 8'b11110111;
    4'b0011: posr <= 8'b11111011;
    4'b0100: posr <= 8'b11111101;
    4'b0101: posr <= 8'b11111110;
    default: posr <= 8'b11111111;
    endcase
endmodule
```

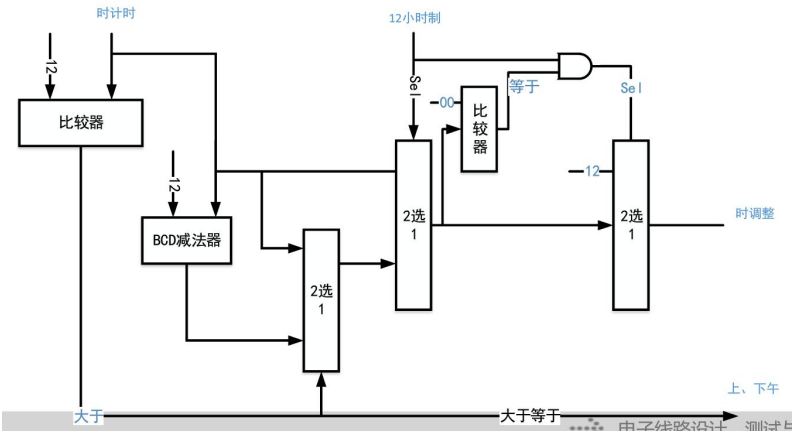
```
module bcd_8seg(
    input [3:0] bcd,
    output [7:0] seg
    );
    reg [7:0] seg1;
    assign seg = seg1;
    always@(bcd)
    case(bcd)
    4'b0000: seg1 <= 8'b11000000;
    4'b0001: seg1 <= 8'b11111001;
    4'b0010: seg1 <= 8'b10100100;
    4'b0011: seg1 <= 8'b10110000;
    4'b0100: seg1 <= 8'b10011001;
    4'b0101: seg1 <= 8'b10010010;
    4'b0110: seg1 <= 8'b10000010;
    4'b0111: seg1 <= 8'b11111000;
    4'b1000: seg1 <= 8'b10000000;
    4'b1001: seg1 <= 8'b10010000;
    default: seg1 <= 8'b11111111;
    endcase
endmodule
```

```
module mux6_1(
    input [3:0] ch0, ch1, ch2, ch3, ch4, ch5,
    input [3:0] sel,
    output [3:0] bcd
    );
    reg[3:0]bcdr;
    assign bcd = bcdr;
    always@(sel)
    case(sel)
    4'b0000:bcdr <= ch0;
    4'b0001:bcdr <= ch1;
    4'b0010:bcdr <= ch2;
    4'b0011:bcdr <= ch3;
    4'b0100:bcdr <= ch4;
    4'b0101:bcdr <= ch5;
    default:bcdr <= 4'b1111;
    endcase
endmodule
```

```
module scan_disp(
    input clk,
    input cr,
    input en,
    input [3:0] ch0,
    input [3:0] ch1,
    input [3:0] ch2,
    input [3:0] ch3,
    input [3:0] ch4,
    input [3:0] ch5,
    input [7:0] seg,
    output [7:0] pos
    );
    wire rco;
    wire [3:0] bcdsel,bcd_data;
    c6 u_count(clk,cr,rco,en,bcdsel);
    mux6_1 u_mux(ch0,ch1,ch2,ch3,ch4,ch5,bcdsel,bcd_data);
    posdecode u_pos(bcdsel,pos);
    bcd_8seg u_seg(bcd_data,seg);
endmodule
```

d） 24/12 小时制切换模块：

模块框图：

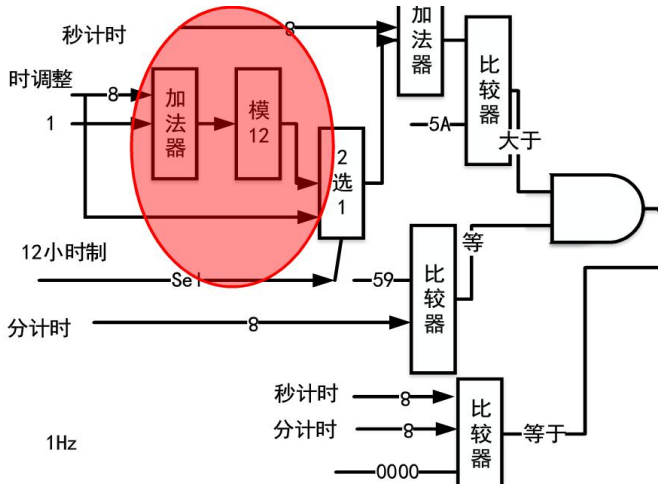


实现代码：

```
module day24_12(
    input [3:0]bcd_ht,
    input [3:0]bcd_hu,
    input day_night,
    output day,
    output [3:0]bcd_hto,
    output [3:0]bcd_huo
);

    wire modify,borrowu,mid_night_zero,noon;
    wire [3:0]bcd_temp_hu,bcd_temp_ht,bcd_temp_huo,bcd_temp_hto;
    assign noon=(bcd_ht==1)&& (bcd_hu==2);//7中午12点
    assign modify = (bcd_ht>1)||( (bcd_hu>2)&&(bcd_ht==1));//12点以后
    assign borrowu= (bcd_hu<2) ?1:0;//BCD减法调整
    assign bcd_temp_hu=borrowu? (bcd_hu+4'b1010-4'b0010):(bcd_hu-4'b0010);
    assign bcd_temp_ht=borrowu?(bcd_ht-4'b0010): (bcd_ht-4'b0001);
    assign mid_night_zero = (bcd_ht==0)&&(bcd_hu==0);//零点
    assign bcd_temp_huo=(modify&day_night) ?bcd_temp_hu:bcd_hu;
    assign bcd_temp_hto=(modify&day_night) ?bcd_temp_ht:bcd_ht;
    assign {bcd_hto,bcd_huo} =(mid_night_zero&day_night)?8'h12:{bcd_temp_hto,bcd_temp_huo};//午夜12点
    assign day=(modify|noon)&day_night;//712点以后标志下午
endmodule
```

e） 整点报时模块：

模块框图：



实现代码：

```
module radio(
    input [3:0] bcd_su,
    input [3:0] bcd_st,
    input [3:0] bcd_mu,
    input [3:0] bcd_mt,
    input [3:0] bcd_hu,
    input [3:0] bcd_ht,
    input clk_1hz,
    input clk_2hz,
    input en,
    input cr,
    input day_night,
    output radio_alarm
    );
wire ld,min_equ;
wire equ,grt,less,noon;
wire [3:0] bcd_temp_hu,bcd_temp_ht;
wire led_shine_en;
assign noon = ({bcd_ht,bcd_hu}==8'h12);
assign {bcd_temp_ht,bcd_temp_hu} = (noon&day_night)?8'h00:{bcd_ht,bcd_hu};
assign min_equ = ((bcd_mt==0)&&(bcd_mu==0))?1'b1:1'b0;
assign equ = ({bcd_st,bcd_su,bcd_mt,bcd_mu}==16'h0000)?1'b1:1'b0;
//assign grt = (({bcd_st,bcd_su}+{bcd_temp_ht,bcd_temp_hu}+8'h1)>8'h5a)?1'b1:1'b0;
assign led_shine_en = ({bcd_st,bcd_su}<=({bcd_temp_ht,bcd_temp_hu}))?1'b1:1'b0;
assign radio_alarm = (({en,min_equ,led_shine_en}==3'b111)?1'b1:1'b0)&clk_1hz;
endmodule
```

f） 模块总结：

① 计时模块：

采用自底向上方法，先实现 counter6 与 counter10，再实现 counter60 与 counter24，分秒调用 counter60，小时调用 counter24。通过 adjust 开关位调节时间以及 min_hour 开关位来选择调节分钟位还是小时位。

② 分频器模块：

分频模块是将原始的时钟信号进行分频操作，得到我们想要的时钟信号，其思路就是设置一个计数器，当捕获到原始时钟周期的跳变次数达到计数器的阈值，我们的时钟信号才进行一次反向

③ 显示模块：

显示模块需要使用 6 进制计数器来循环刷新数码管，6 选 1 复用器选择合适的输入信号，以及数码管的段码与位码输入总共四个文件，顶层模块分别调用上述四个模块。

④ 24/12 小时制切换模块：

bcd_ht,bcd_hu 指示 24 小时制的时间，day_night 控制进制模式，day 是当开启 12 小时制的情况下，指示上午或者下午的一位信号，bcd_hto、bcd_huo 是显示输出。

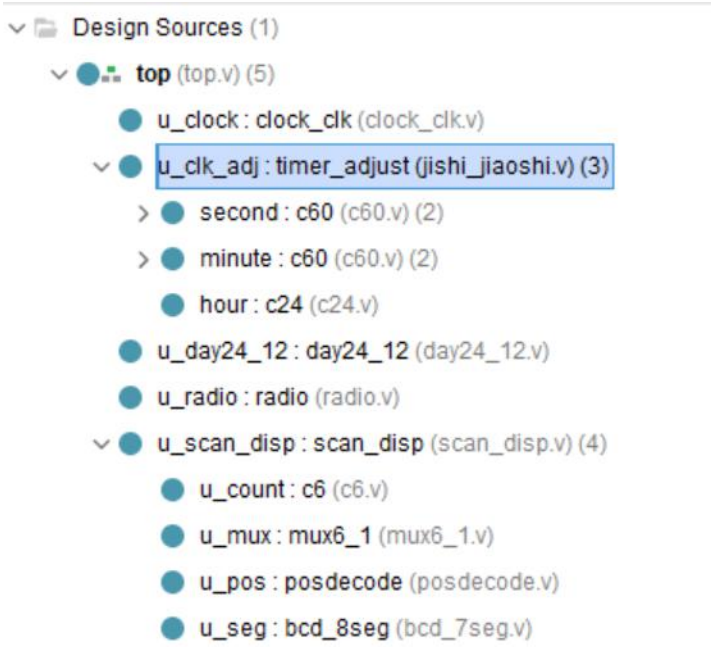
⑤ 整点报时模块：

报时是根据数码管上显示的小时数来决定闪的次数，我们需要 day_night

开关来控制数码管上显示的小时数并将结果传送给 radio_alarm 得到闪的秒数，再和 clk_2hz 相与得到半秒闪烁一次的 LED 信号。

g） 顶层模块：

模块结构：



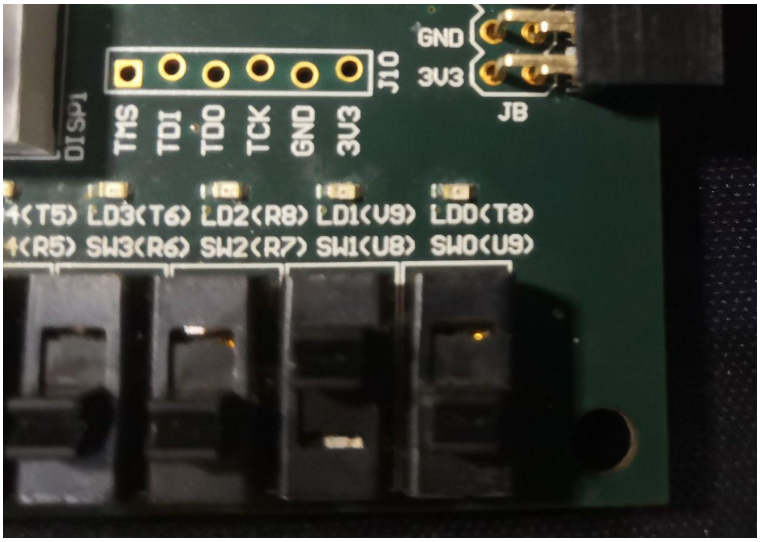
实现代码：

```
20  //////////////////////////////////////////////////////////////////////////////////
21  module top(
22      input clk_100m,
23      input cr,
24      input en_clock,
25      input clock_adjust,
26      //input clock_alarm_en,
27      //input clock_alarm_set,
28      input min_hour_set,
29      //input alarm_adjust_disp,
30      input day_night,
31      output [7:0] pos,
32      output [7:0] seg,
33      output alarm,
34      output day
35      );
36      wire clk_1hz, clk_1k, clk_5h, clock_alarm, radio_alarm, mux_sel_set, day_radio, clk_2hz;
37      wire [3:0] bcd_day_hu, bcd_day_ht, bcd_day_ht_radio, bcd_day_hu_radio;
38      wire [3:0] bcd_smu, bcd_smt, bcd_shu, bcd_sht, bcd_mu, bcd_mt, bcd_su, bcd_st;
39      wire [3:0] bcd_hu, bcd_ht, bcd_temp_mu, bcd_temp_mt, bcd_temp_hu, bcd_temp_ht;
40      //assign alarm=radio_alarm|clock_alarm&clock_alarm_en;     //整点报时或闹铃
41      //assign mux_sel_set=clock_alarm_set|alarm_adjust_disp;    //设置闹铃或显示闹铃
42      //assign {bcd_temp_mu, bcd_temp_mt, bcd_temp_hu, bcd_temp_ht}=mux_sel_set?{bcd_smu, bcd_smt, bcd_shu, bcd_sht}:{bcd_mu, bcd_mt, bcd_hu, bcd_ht}; //显示2选1
43      clock_clk u_clock(clk_100m, clk_1k, clk_5h, clk_1hz, clk_2hz, cr);      //分频
44      timer_adjust u_clk_adj(clk_1hz, clock_adjust, cr, min_hour_set, bcd_su, bcd_st, bcd_mu, bcd_mt, bcd_hu, bcd_ht); //计数、校时
45      //clock_alarm u_clk_alarm(clk_1hz, clk_1k, clk_5h, clock_alarm, en, clock_alarm_set, cr, min_hour_set, bcd_mu, bcd_mt,
46      //bcd_hu, bcd_ht, bcd_smu, bcd_smt, bcd_shu, bcd_sht, clock_alarm);
47      day24_12 u_day24_12(bcd_ht, bcd_hu, day_night, day, bcd_day_ht, bcd_day_hu);    //显示24/12小时调整
48      //day24_12 radio_day24_12(bcd_ht, bcd_temp_hu, day_night, day, bcd_day_ht, bcd_day_hu);    //仿电台24/12调整
49      radio u_radio(bcd_su, bcd_st, bcd_mu, bcd_mt, bcd_day_hu, bcd_day_ht, clk_1hz, clk_2hz, en_clock, cr, day_night, alarm);
50      scan_disp u_scan_disp(clk_1k, cr, en_clock, bcd_day_ht, bcd_day_hu, bcd_mt, bcd_mu, bcd_st, bcd_su, seg, pos);
51  endmodule
```

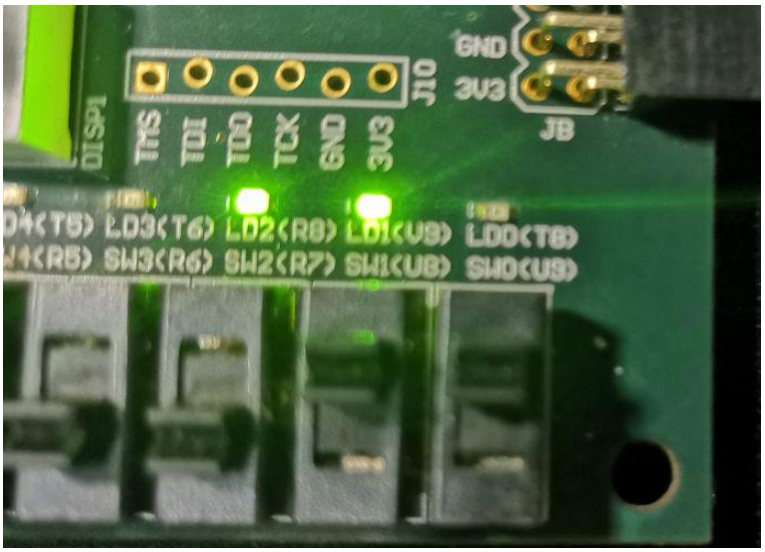
五. 实验结果

（1） 步进电机脉冲分配器：

nCR = 0 时，状态置为 000；

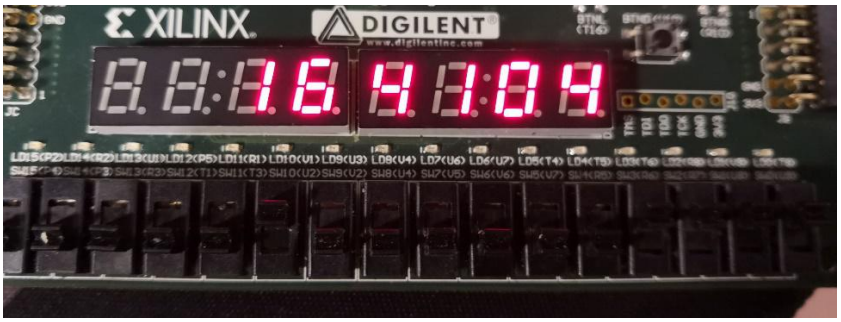


nCR = 1 时，步进电机正常运转



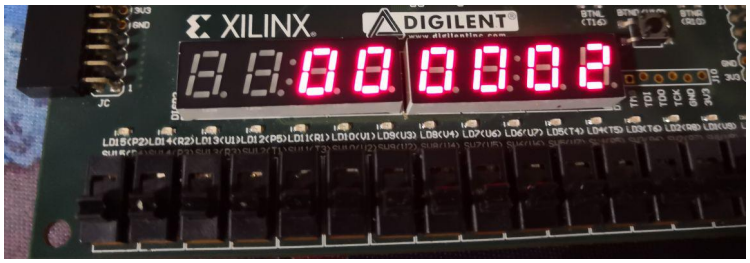
（1） 多功能数字钟：

24 小时制显示：



12 小时制显示，LED1 区分上下午：

清零：



整点报时，LED 灯闪烁：

六. 实验问题及分析

（1） 开始时使用一个 8 位 2 进制数来表示小时、分钟和秒数但使用两位数码管来显示的时候，无法分离该数字的十进制的十位与个位

解：开始是尝试使用循环语句来实现，但在生成 Implementation 是总是出现错误，后来使用移位加 3 法（①. 将二进制码左移一位（或者乘 2）。 ②. 找到左移后的码所对应的个、十、百位。 ③. 判断在个位和百位的码是否大于 5，如果是则该段码加 3。 ④. 继续重复以上三步直到移位 8 次后停止。）但会使数码管显示错误，仍未实现理想效果。

后来修改计时的底层模块，从一开始就使用两个四位 2 进制来分别表示个位与十位，成功实现。

（2） 模块调用时语句不规范，使得虽然没有报错，但是仍然没有实现理想效果。

解：模块调用方式不正确，应熟悉模块调用的三种方式：调用方式一:位置对应调用方式；调用方式二:端口名对应调用方式；调用方式三:存在不连接端口的调用方式。

（3） 开始时，使用 Nexys4 上面的按键作为调整时间的输入端口，但是调整时经常会出现波动与不识别的情况。

解：按键会有一定抖动，如果有竞争与冒险现象，则抖动会带来一些逻辑错误，使得电路出现错误，需要特定的按键消抖模块，后来将输入端口改为使用 Nexys4 上面的开关来实现，则功能正常。

（4） 在步进电机实验中，因为状态较多，使得书写 case 语句块的时候经常会写错状态，且修改起来较为复杂。

解：描述电路前，对各个状态进行人为的标号，在程序中使用宏定义的方式写在开头，使用时直接写出各个状态的标号，大大简化的语句，使代码的可读性与可移植性大大增强。

七.实验感想与总结

在本次使用 Vivado 开发软件与 Nexys4 开发板的实验过程,从安装软件、设计电路、进行仿真、引脚约束到运行结果都自己实际操作完成。是对我们利用现代化可编程逻辑电路与相关主流开发仿真软件进行现代化电路设计与功能开发的一次综合训练。通过本次实验，我们也对较复杂 FPGA 电路的开发流程有了一个全面的了解与实践。

在 FPGA 开发板上做实验，大大减少了搭建电路的时间，同时由于可编程，我们能实现相对来说更加复杂的电路。通过本次实验，也简单的了解了使用 Nexys4 开发板开发的 基本流程：首先在计算机上设计算法、编写代码，然后进行仿真分析，通过代码仿真 保证设计方案符合实际要求，最后进行板机调试，利用配置电路将相关文件下载至 FPGA 芯片中，验证实际运行效果。

通过本次实验，我对数电的相关知识有了更深入的了解，将其运用到了实际中来，明白了学习电子技术基础的意义，也达到了其培养的目的。也明白了一个道理：成功就是在不断摸索中前进实现的，遇到问题我们不能灰心、烦躁，甚至放弃，而要静下心来仔细思考，分部检查，找出最终的原因进行改正，这样才会有进步，才会一步步向自己的目标靠近，才会取得自己所要追求的成功。

通过近来对 FPGA 的学习，我对 Vivado 软件总体操作步骤已比较熟悉。接下来的时间是要提高对 FPGA 的掌握能力，将 FPGA 运用到数字信号处理和通信原理上，另外，对 Verilog 语言的学习不仅是能读懂别人的程序，更重要的是掌握数字电路开发背后的底层逻辑，能写出条理清晰的程序。

当然,通过本次实验,我也发现自己的仿真技术和应用能力还是有很大的提升空间，虽然完成了基本的设计要求和一部分扩展功能,但是很多自己想要的扩展功能还未能实现。而且很多时候会走过很多弯路，浪费了很多不必要的时间。不过，这次设计经历必将使我受益终身，让我明白如何更好的获取知识，如何更好的理论联系实际。今后的学习更需要不断努力，在获得知识的同时获得快乐，真正的主动探索，主动学习，形成自己的思维方式，不断应用，不断进取。