2020 年第四届全国大学生 FPGA 创新设计竞赛

参赛作品设计报告

基于 FPGA 的自动书写时钟机器人

参赛队伍

3bug

参赛人员

赵持恒

毛梓宇

李晨铭

目录

第一部分　设计概述

1.1 设计目的

随着计算机技术的发展，智能家居领域的进步也突飞猛进。受到现在各种新颖的家具产品的启示，我们将 FPGA、机械臂控制技术与时钟模块结合，开发了一款自动书写时钟机器人。

与传统的时钟相比，该产品没有采用指针行走或单纯的显示屏显示时间功能，而是靠机械臂的自动手写功能来实现。通过这样不寻常的设计，我们希望可以挖掘 FPGA 技术的更大潜力，为开发一些新颖的观赏性与实用性具备的产品提供良好的范例，注入新的活力。

1.2 应用领域

本产品应用领域广泛，核心技术是通过通信获取时间信息并通过机械臂替代人手完成手写功能，适用于需要定时更新并显示信息的场景，比如商场、车站等地。由于它具有较好的可擦除性，纠错能力较强，且机械臂能绘制不同的图形，因此趣味性也是该产品的一大特点，因此可以考虑批量生产作为一种新型的家具时钟产品。FPGA 的探索永无止境，在娱乐领域、教育领域、智能家居领域都有着广泛的应用前景。

1.3 主要技术特点

通过 SPI 协议与时钟芯片进行通信，经时间校准后可以实现北京时间的 实时获取。对舵机的灵活控制是另一大核心特点，结合 scara 运动模型和舵机的控制原理，针对舵机系统自动控制的精度低，响应速度低的问题，提出了一种基于 FPGA 的直线运动插补算法的数字系统设计方案，通过采用具有可综合，灵活度高，可并行执行等特点的 Verilog HDL 语言，再加上内部具有丰富逻辑资源的 FPGA，从而快速精准地实现舵机控制，加上舵机角度算法可以进行精准的坐标定位。

1.4 关键性能指标

(1)时钟模块 DS1302 性能：0-40℃范围内，精度 2ppm，年误差约 1 分钟；时间获取、校准的误差低至 5ns；IIC 总线接口，限高传输速度 400kHz（工作电压为 5V 时）；工作时功耗低，保持数据和时钟信息时功率小于 1mW。

(2)舵机性能：空载速度 0.09s/60°，死区设定为 8us。

(3)舵机角度算法将舵机角度控制误差在 1.5°以内。

(4)书写精准度高，机器人运动过程出错率低。

1.5 主要创新点

(1)使用机械臂控制画笔书写完成信息的显示，同时用机械臂控制移动，完成内容的擦除，具有较强的趣味性和观赏性。

(2)结合 scara 运动学原理对机器人模型的设计。

(3)使用 FPGA 实现舵机的精准、灵活控制。

第二部分　系统组成及功能说明

2.1 整体介绍

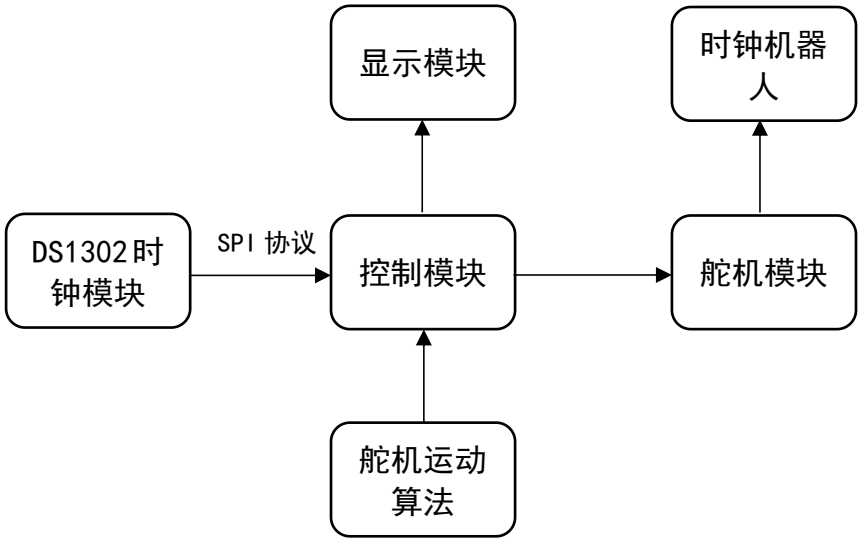

图 1 系统整体框图

本作品实现实时自动写字时钟，整体介绍如图所示。核心是 FPGA 信息处理模块、舵机运动算法和时钟模块。工作时，FPGA 从 DS1302 时钟芯片获取当前时间，通过 SPI 协议将时间信息传递到处理器模块。处理器通过舵机运动算法，将时间信息对应到不同占空比的 PWM 波输出，来控制舵机运动。舵机模块操纵设计

好的写字机械臂模型，在白板上完成时间的自动书写和擦除。

2.2 各模块介绍

2.2.1 算法模块

算法模块实现将白板上的坐标经过计算，对应成舵机的控制信号 PWM 波，流程如下图。

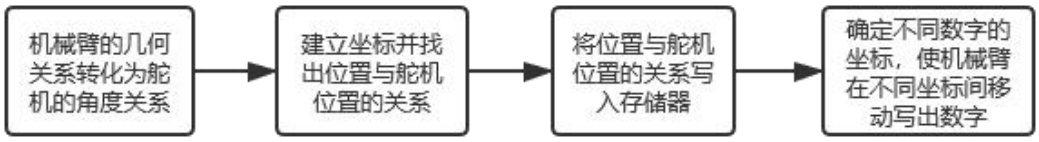

图 2 舵机运动算法流程

图 3 为时钟机架模型，进行坐标参数定义。

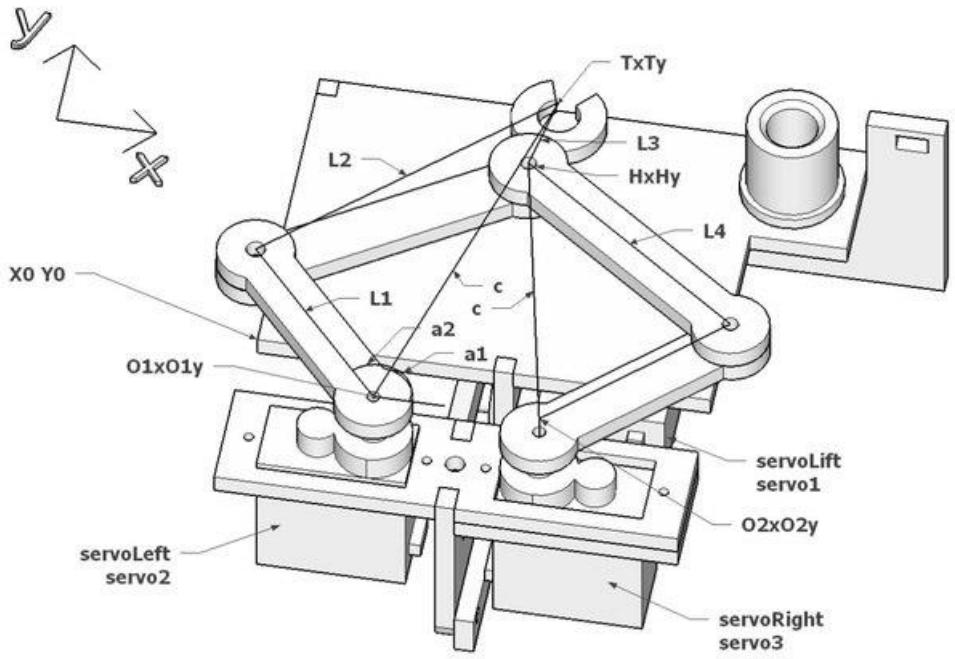

图 3 坐标参数定义

先以白板左下角为原点，向上为 y 轴，向右为 x 轴建立直角坐标系。先求左舵机转动角度与坐标位置的关系，设笔的位置为（x,y），左右舵机的位置分别为（X01，Y01），（X02，Y02）。

对于左舵机：

$$\alpha_1 = \arccos\frac{L_1 + x^2 + y^2 - L_2^2}{2 \times L_1 \times \sqrt{x^2 + y^2}}$$

$$\alpha_2 = \arctan\frac{y}{x}$$

对于右舵机，笔的位置与右臂连接点的距离不确定，所以我们就对于右臂假设了一个类似于左臂的关系，这样我们找到一个辅助点，这个点与连接点的距离是不变的为（L2 - L3）。

$$x' = x + L_3 \times \cos((\alpha_1 - \alpha_2 + 36.5) + \pi)$$

$$y' = y + L_3 \times \sin((\alpha_1 - \alpha_2 + 36.5) + \pi)$$

$$\alpha_1' = \arctan\frac{y'}{x'}$$

$$c = \sqrt{x'^2 + y'^2}$$

$$\alpha_2' = \frac{L_1^2 + c^2 - (L_2 - L_3)^2}{2 \times L_1 \times c}$$

2.2.2 SPI 协议

SPI 是一种高速，全双工，同步的通信总线，在芯片上只占用四根线(CS、MOSI、MISO、SCK)，极大节约了芯片的引脚。

Master 会根据将要交换的数据产生相应的时钟脉冲，组成时钟信号，时钟信号通过时钟极性(CPOL)和时钟相位(CPHA)控制两个 SPI 设备何时交换数据以及何时对接收数据进行采样，保证数据在两个设备之间是同步传输的。

SPI 协议规定一个 SPI 设备不能在数据通信过程中仅仅充当一个发送者或者接受者。在片选信号 CS 为 0 的情况下，每个 clock 周期内，SPI 设备都会发送并接收 1 bit 数据，相当于有 1 bit 数据被交换了。数据传输高位在前，低位在后。SPI 主从结构内部数据传输示意图如下图所示。

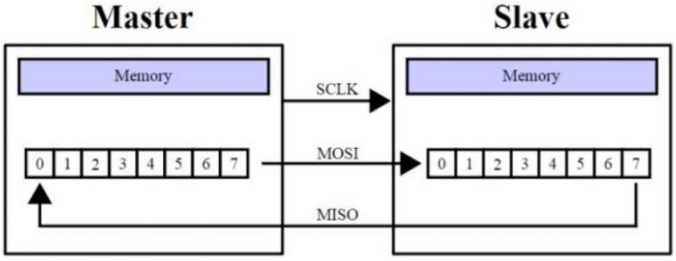

图 4 SPI 主从传输示意图

SPI 总线传输一共有 4 中模式，这 4 种模式分别由时钟极性 CPOL 和时钟相位 CPHA 来定义，其中 CPOL 参数规定了 SCK 时钟信号空闲状态的电平，CPHA 规定了数据是在 SCK 时钟的上升沿被采样还是下降沿被采样。这四种模式的时序图如下图所示：

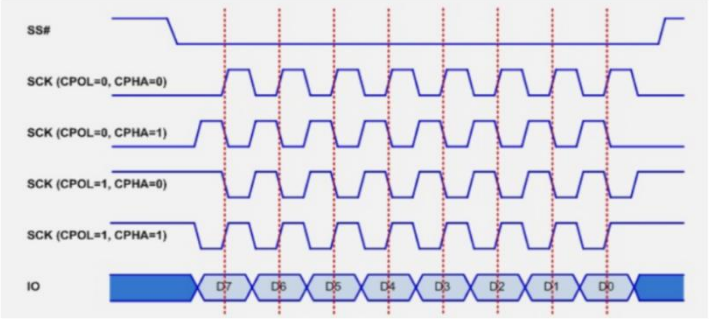

图 5　SPI 四种模式时序图

SPI 协议的 Verilog 实现方法：

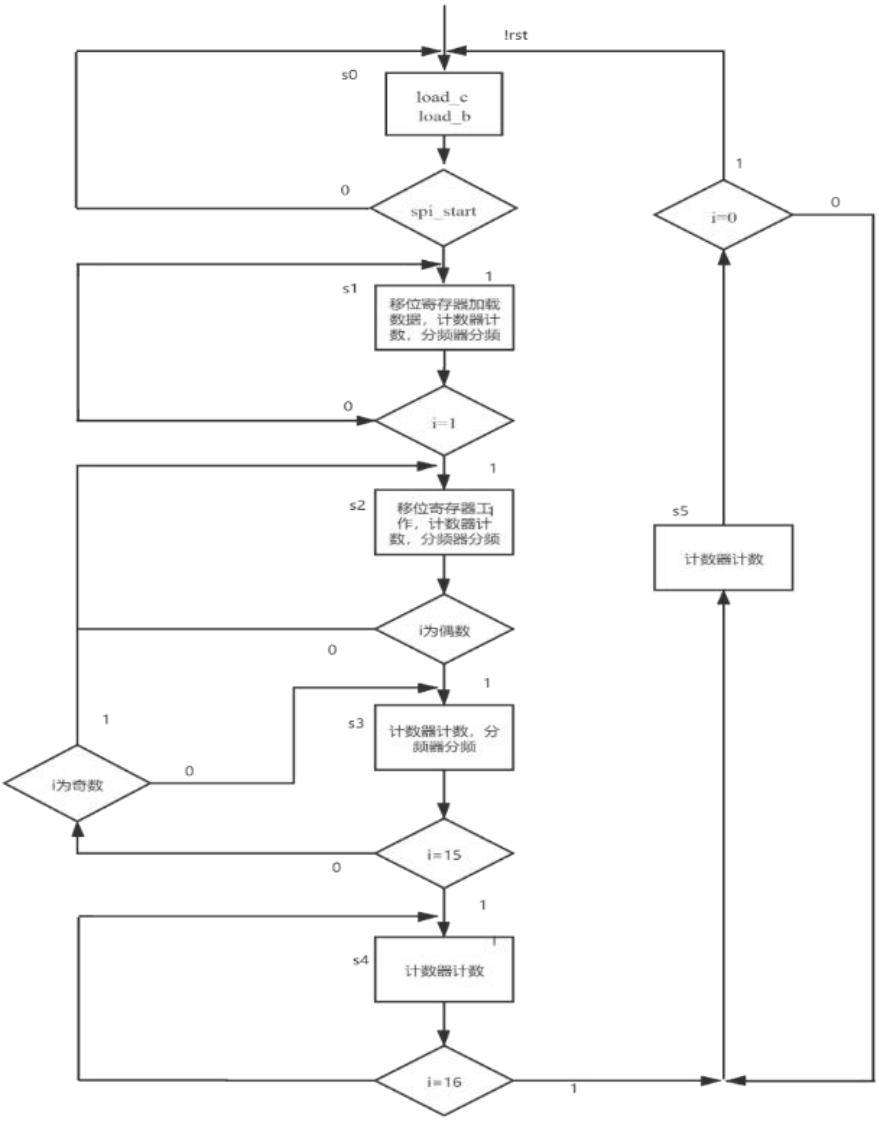

图 6 SPI 实现流程

以 SPI 模式 3 为例，首先在复位信号到来时，进入 s0 状态，在 s0 状态计数器和分频器模块加载初始值，如果发送数据开始信号 spi_start 有效进入 s1 状态，s1 状态加载待发送的数据，同时计数器计数计数，分频器开始工作，如果 i=1，进入 s2 状态，s2 状态主要用来发送数据，如果 i 为偶数，进入 s3 状态，该状态是用来采集数据，由于只考虑发送，因此此模块不进行数据采集工作，如果 i=15，进入 s4 状态，否则如果 i 为奇数，则进入 s2 状态。在 s4 状态，发送最后一位数据，如果 i=16，进入 s5 状态，此时整个 SPI 时序模拟完成。

2.2.3 DS1302 时钟模块

- 实时时钟计算年、月、日、时、分、秒、 星期，直到2100 年，并有闰年调节功能
- 31 x 8 位 通用暂存RAM
- 串行输入输出使管脚数最少
- 2.0V 至 5.5V 宽电压范围操作
- 读写时钟或RAM 数据时有单字节或多字节（脉冲串模式）数据传送方式。

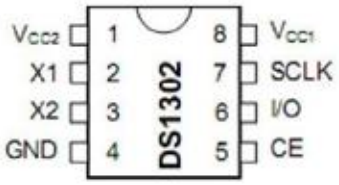

图 7 DS1302 时钟

工作电路：

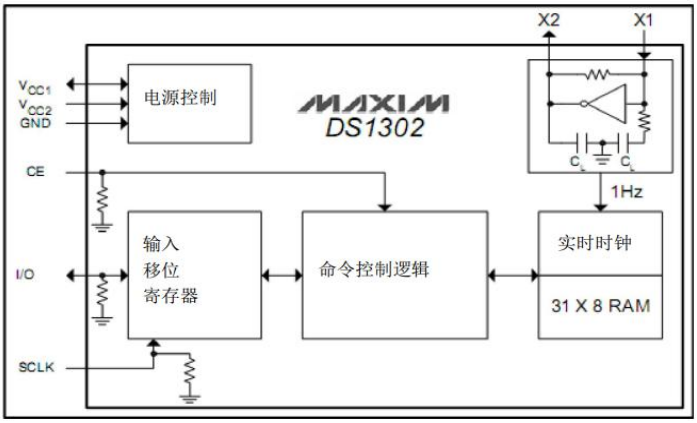

图 8 DS1302 工作电路

下图显示的是命令字。命令字启动每一次数据传输。MSB (位 7)必须是逻辑 1。如果是 0，则禁止对 DS1302 写入。位 6 在逻辑 0 时规定为时钟/日历

数据，逻辑 1 时为 RAM 数据。位 1 至 位 5 表示了输入输出的指定寄存器。LSB （位 0） 在逻辑 0 时为写操作(输出)，逻辑 1 时为读操作(输入)。命令字以 LSB （位 0)开始总是输入。

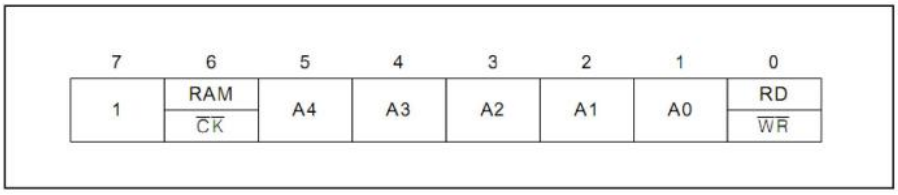

图 9 DS1302 命令字

数据输入输出

输入写命令字的 8 个 SCLK 周期后 ，接下来的 8 个 SCLK 周期的上升沿数据字节被输入，输入读命令字的 8 个 SCLK 周期后，随后的 8 个 SCLK 周期的下降沿，一个数据字节被输出。SCLK 的每个上升沿被置为三态。数据输出从位 0 开始。

时钟/日历脉冲串模式

时钟/日历命令字节指定脉冲串模式操作。此模式下，前八个时钟/日历寄存器必须从地址 0 的位 0 开始连续读写(见下表)。

寄存器地址/定义表

RTC

READ	WRITE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	RANGE
81h	80h	CH	10 Seconds			Seconds				00–59
83h	82h		10 Minutes			Minutes				00–59
85h	84h	12/$\overline{24}$	0	10 $\overline{\text{AM/PM}}$	Hour	Hour				1–12/0–23
87h	86h	0	0	10 Date		Date				1–31
89h	88h	0	0	0	10 Month	Month				1–12
8Bh	8Ah	0	0	0	0	0	Day			1–7
8Dh	8Ch	10 Year				Year				00–99
8Fh	8Eh	WP	0	0	0	0	0	0	0	—
91h	90h	TCS	TCS	TCS	TCS	DS	DS	RS	RS	—

时钟脉冲串

BFh	BEh

图 10 寄存器地址/定义

读写信号时序图

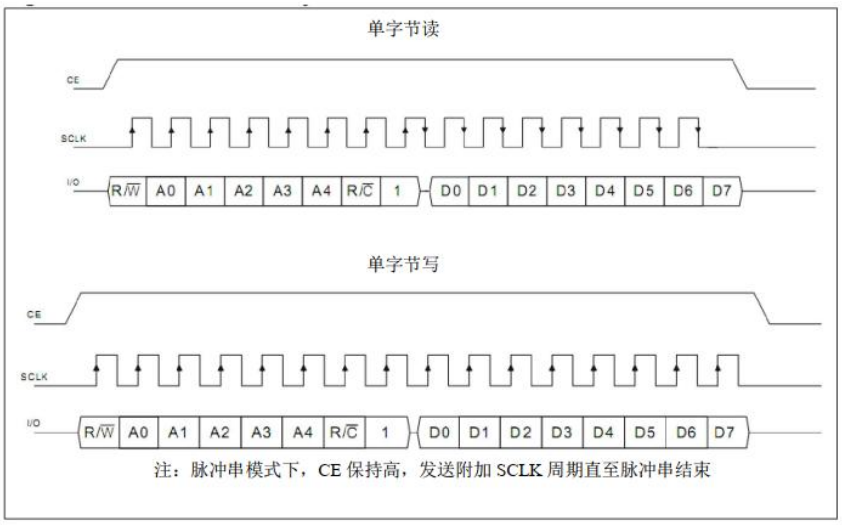

图 11 读写信号时序图

设置初始时间并与北京时间校准。

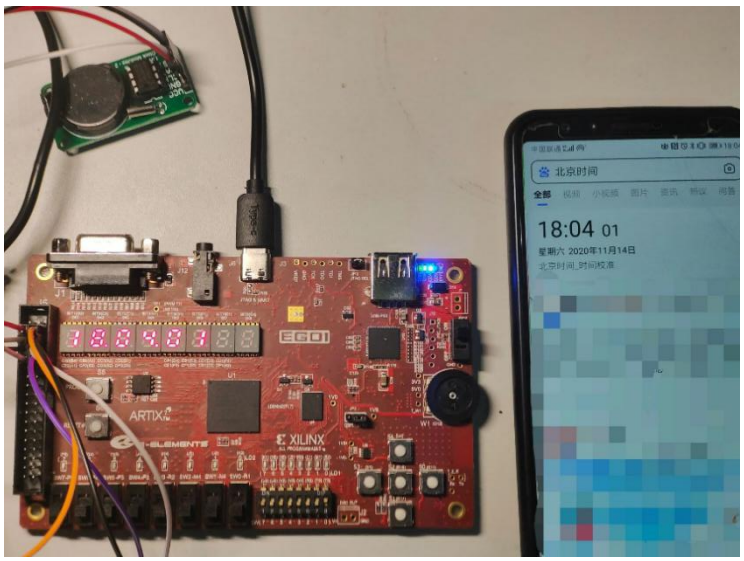

图 12 北京时间校准

2.2.4 控制模块

FPGA 经过时间信息的接收、算法将时间信息转换成舵机的角度后，控制模块经过直线运动插补算法让舵机控制的机械臂模型完成写下不同数字、擦除等功能，进而完成时钟机器人的整个运动过程。

控制逻辑：

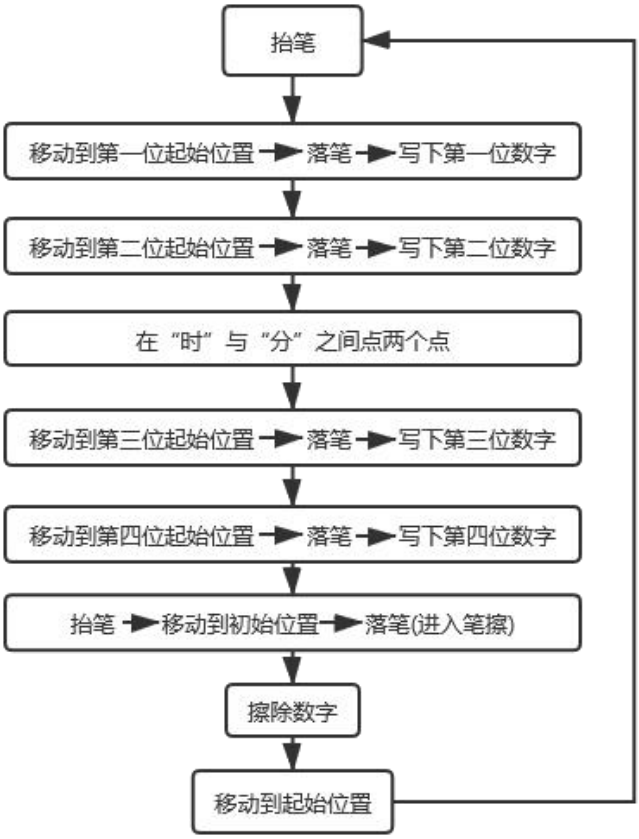

图 13 控制流程

图 14 为白板模型，虚线表示书写时可能经过的所有运动轨迹，图 15 为每个数字的初始位置编号。

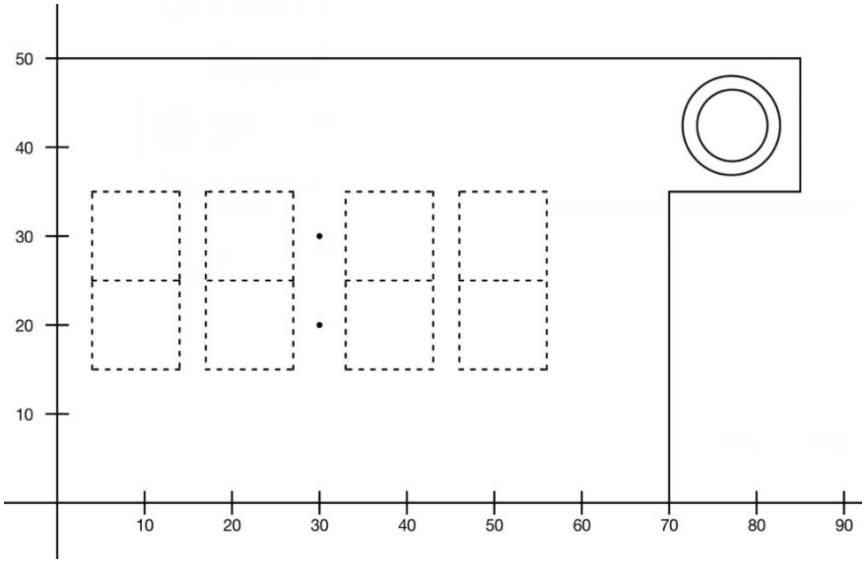

图 14 白板模型

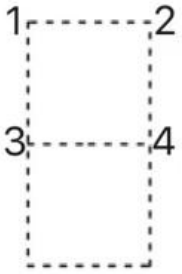

图 15 初始位置

0~9 每个数字对应的初始位置如下表：

数字	1	2	3	4	5	6	7	8	9
初始位置	2 号位	1 号位	1 号位	1 号位	2 号位	2 号位	1 号位	3 号位	4 号位

表 1 初始位置表

每个数字在确定初始位置后，完成书写的运动轨迹如下表：

	第1步	第2步	第3步	第4步	第5步	第6步	第7步
0	x=x+1	y=y-1	y=y-1	x=x-1	y=y+1	y=y+1	0
1	y=y-1	y=y-1	0	0	0	0	0
2	x=x+1	y=y-1	x=x-1	y=y-1	x=x+1	0	0
3	x=x+1	y=y-1	x=x-1	x=x+1	y=y-1	x=x-1	0
4	y=y-1	x=x+1	y=y+1	y=y-1	y=y-1	0	0
5	x=x-1	y=y-1	x=x+1	y=y-1	x=x-1	0	0
6	x=x-1	y=y-1	y=y-1	x=x+1	y=y+1	x=x-1	0
7	x=x+1	y=y-1	y=y-1	0	0	0	0
8	y=y+1	x=x+1	y=y-1	y=y-1	x=x-1	y=y+1	x=x+1
9	x=x-1	y=y+1	x=x+1	y=y-1	y=y-1	x=x-1	0

表 2 运动轨迹表

以画“2”为例，2 的起始坐标在整个“8”形虚线框的左上角，坐标依次需要经过 x=x+1、y=y−1、x=x−1、y=y−1、x=x+1，若要使每笔都是直线段，需要用到直线运动插补算法，既在每两点之间插入多个等间隔的点，使其逼近为直线段。以 19：23 为例，最终结果如下图所示。

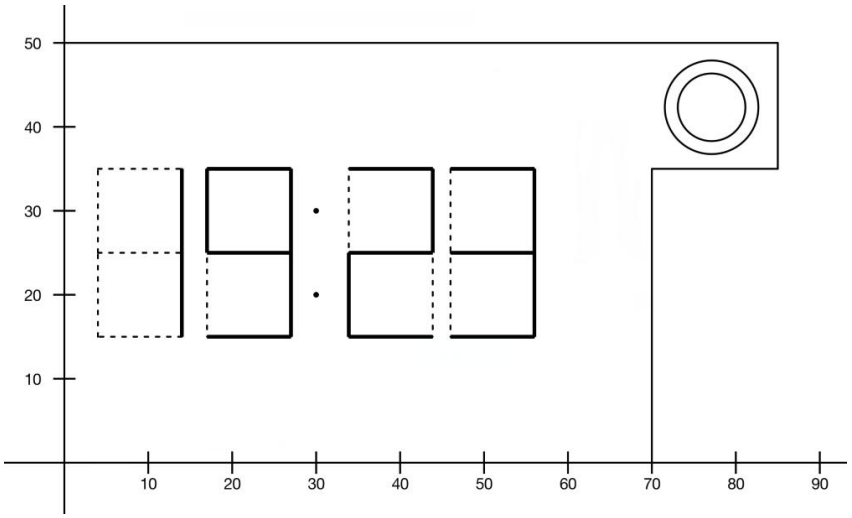

图 16 白板模型-2

2.2.5 显示模块

显示模块采用数码管扫描显示，将获取的时钟模块的时间动态显示在数码管上。EGO-1 上有共阴极七段数码管，真值表如下：

字形	h	g	f	e	d	c	b	a	16进制
.	1	0	0	0	0	0	0	0	0x80
0	0	0	1	1	1	1	1	1	0x3f
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5b
3	0	1	0	0	1	1	1	1	0x4f
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7f
9	0	1	1	0	1	1	1	1	0x6f

表 3 数码管真值表

2.2.6 舵机模块

这里采用 TS90A 舵机，舵机的控制信号是周期为 20ms 的 PWM 信号，其中脉冲宽度为从 0.5ms-2.5ms，PWM 信号由处理器经过运算后给出。

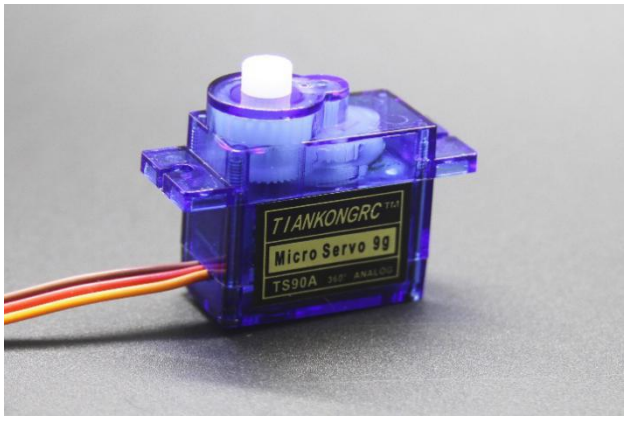

图 17 TS90A 舵机

控制舵机转动的 PWM 波，其占空比与舵机对应转动角度如下图：

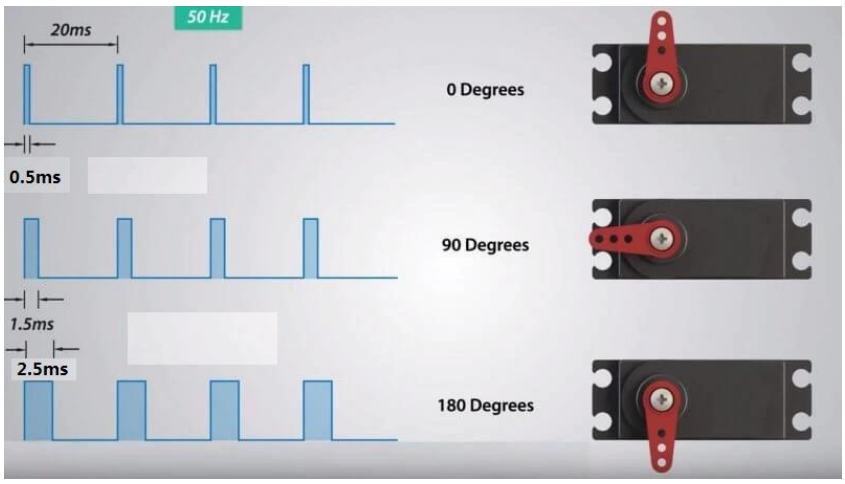

图 18 舵机控制原理

2.2.7 时钟机器人

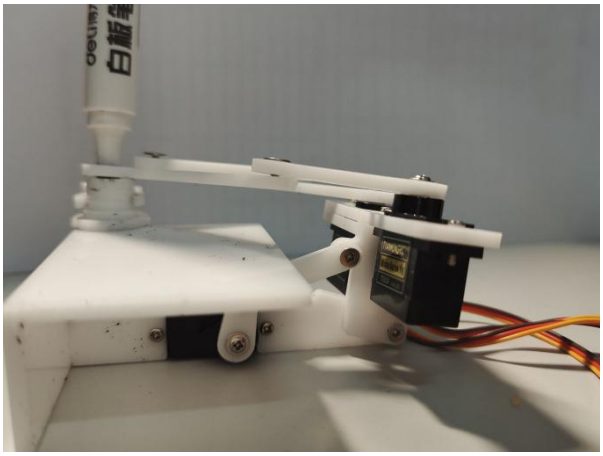

图 19 时钟机器人(侧面)

图 20 时钟机器人(正面)

机械臂控制模块由三个舵机和设计好的亚克力切割板组接而成，前面两个舵机控制二维白板上的书写功能，后面一个舵机实现抬笔，落笔功能。

第三部分　完成情况及性能参数

根据自动书写始终机器人的书写工整度、擦除完整度、运动过程（笔擦复位情况和落笔回槽情况）进行性能测试。

测试编号	测试时间	书写工整度	擦除完整度	笔擦是否成功复位	落笔是否成功回槽
1	09：21	较好	较好	是	是
2	10：28	较好，8 有一点角度偏移	较好	是	是
3	11：35	较好	较好	是	是
4	16：04	较好，6 因物理原因有一点抖	左上角有一点残余	是	是
5	17：50	较好	较好	是	是
6	18：07	较好	左上角有一点残余	是	是
7	20：27	较好	较好	是	是

表 4 性能测试表

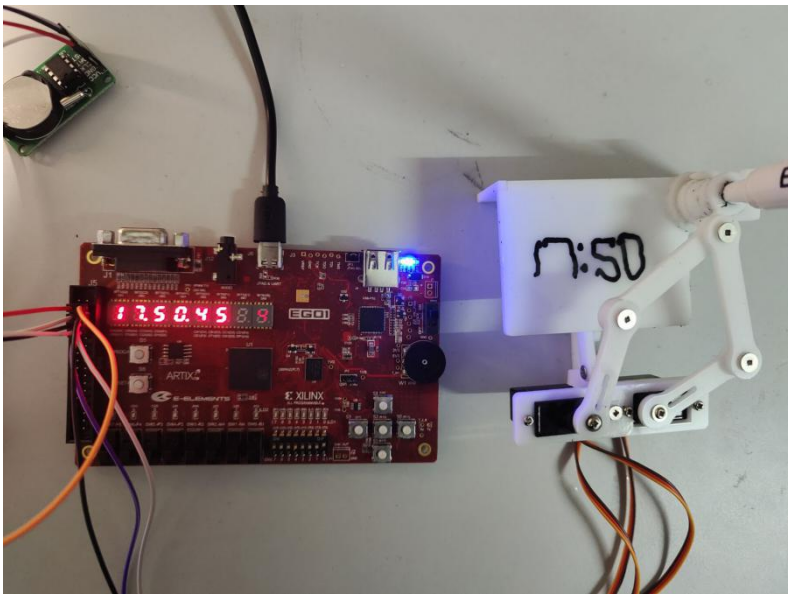

图 21 测试实例-1

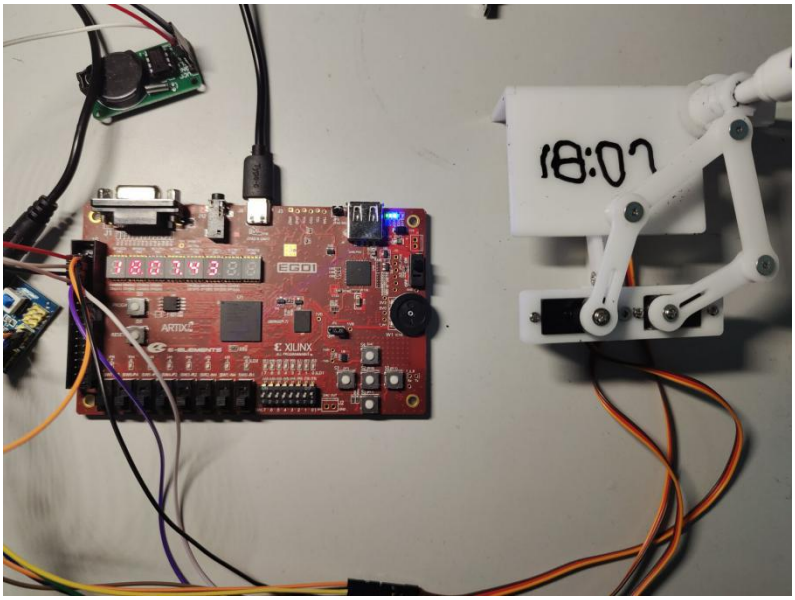

图 22 测试实例-2

第四部分 总结

4.1 可扩展之处

(1)可增加闹钟功能、可增加读取天气功能并在白板上另外显示天气状况使“智能时钟”功能更加完善。

(2)由于物理因素（测量误差机架和摆臂不能完全固定等）的影响，书写流畅度和工整度还可进一步提升，可通过优化算法提高机械臂控制的流畅度、精准度和速度。

4.2 心得体会

通过这次竞赛的作品设计，我们深入学习到了 FPGA 的性能以及使用方法，甚至对机械臂控制理论、机械结构设计也有了些许了解。在作品设计开始时期走了很多弯路，对一个项目没有“自顶向下”的规划导致我们初期起步困难；在算法构建一步我们最终选择了查表法，利用 Matlab 及 java 等工具协助完成存储模块；坐标校正是耗费时间最长的一步，舵机的起始角度，算法，机架的物理条件都对起始坐标有着很大的影响，而这一部分也使得我们收获很多。在 DS1302 时钟模块，要正确传输时钟信息，我们需要对照着时序资料不断调整代码；在机械臂控制书写模块，我们不断尝试优化代码并校正参数以便能提高书写的准确度；

在 FPGA 核心处理模块，我们发挥 FPGA 并行运算的计算优势，不断提高信号处理速度……从各个模块之间的连接，各个模块之间的时序安排以及数据处理，无一不需要我们进行调整。虽然调整的过程是漫长的，但我们从中收获甚多。

此外我们知道还有许多方面等着我们去深入探索，我们将通过不断努力，进一步优化我们的产品，使得它的实用性以及潜力得到进一步挖掘。总体来说，这对我们是一次有价值的体验，感谢组委会为我们提供这样一个机会，让我们将课堂所学、课下所想结合在一起，实现这个有趣的产品。

第五部分　参考文献

[1]张普行. 基于 FPGA 的机械臂多路舵机控制器设计[J]. 陕西：西北工业大学，驱动控制，2011，4：73-75.

[2]黄华兴, 蒋潇. 基于 FPGA 的逐点比较插补算法的舵机系统设计[J]. 自动化与仪器仪表, 2020(05):71-74.

[3]姚德法, 张洪林. 串行时钟芯片 DS1302 的原理与使用[J]. 信息技术与信息化, 2006(01):92-94.

第六部分 附录

1. top.v

```
module top(
    input           clk,
    input           rst_n,
    output          rtc_sclk,
    output          rtc_ce,
    inout           rtc_data,
    output [7:0] seg_sel,
    output [7:0] seg_data,
    output [7:0] seg_datatwo,
    output pwm_out1,
    output pwm_out2,
```

```
        output pwm_out3
        );

wire[7:0] read_second;
wire[7:0] read_minute;
wire[7:0] read_hour;
wire[7:0] read_date;
wire[7:0] read_month;
wire[7:0] read_week;
wire[7:0] read_year;
wire[7:0] second;
assign second = 10 * read_second[7:4] + read_second[3:0];

seg_bcd seg_bcd_m0(
        .clk                (clk),
        .rst_n              (rst_n),
        .seg_sel            (seg_sel),
        .seg_data           (seg_data),
        .seg_datatwo        (seg_datatwo),
        .seg_bcd            ({read_hour,read_minute,read_second})
);

control control_m0(
        .clk            (clk),
        .rst_n          (rst_n),
        .time_h1        (read_hour[7:4]),
        .time_h2        (read_hour[3:0]),
        .time_m1        (read_minute[7:4]),
        .time_m2        (read_minute[3:0]),
        .sec            (second),
        .pwm_out1       (pwm_out1),
        .pwm_out2       (pwm_out2),
        .pwm_out3       (pwm_out3)
);


ds1302_test ds1302_test_m0(
        .rst            (~rst_n),
        .clk            (clk),
        .ds1302_ce      (rtc_ce),
        .ds1302_sclk    (rtc_sclk),
        .ds1302_io      (rtc_data),
        .read_second    (read_second),
        .read_minute    (read_minute),
```

```
        .read_hour      (read_hour),
        .read_date      (read_date),
        .read_month     (read_month),
        .read_week      (read_week),
        .read_year      (read_year)
);
endmodule
```

2. ds1302_test.v

```
module ds1302_test(
        input           rst,
        input           clk,
        output          ds1302_ce,
        output          ds1302_sclk,
        inout           ds1302_io,
        output[7:0] read_second,
        output[7:0] read_minute,
        output[7:0] read_hour,
        output[7:0] read_date,
        output[7:0] read_month,
        output[7:0] read_week,
        output[7:0] read_year
);

localparam S_IDLE       =       0;
localparam S_READ       =       1;
localparam S_WRITE      =       2;
localparam S_READ_CH =      3;
localparam S_WRITE_CH =     4;
localparam S_WAIT       =       5;
reg[2:0] state,next_state;

reg write_time_req;

reg write_time_req_latch;
wire write_time_ack;
reg read_time_req;
wire read_time_ack;
reg[7:0] write_second_reg;
reg[7:0] write_minute_reg;
reg[7:0] write_hour_reg;
reg[7:0] write_date_reg;
reg[7:0] write_month_reg;
```

```
reg[7:0] write_week_reg;
reg[7:0] write_year_reg;
wire CH;

assign CH = read_second[7];
ds1302 ds1302_m0(
        .rst(rst),
        .clk(clk),
        .ds1302_ce(ds1302_ce),
        .ds1302_sclk(ds1302_sclk),
        .ds1302_io(ds1302_io),
        .write_time_req(write_time_req),
        .write_time_ack(write_time_ack),
        .write_second(write_second_reg),
        .write_minute(write_minute_reg),
        .write_hour(write_hour_reg),
        .write_date(write_date_reg),
        .write_month(write_month_reg),
        .write_week(write_week_reg),
        .write_year(write_year_reg),
        .read_time_req(read_time_req),
        .read_time_ack(read_time_ack),
        .read_second(read_second),
        .read_minute(read_minute),
        .read_hour(read_hour),
        .read_date(read_date),
        .read_month(read_month),
        .read_week(read_week),
        .read_year(read_year)

);

always@(posedge clk)
begin
        if(write_time_ack)
                write_time_req <= 1'b0;
        else if(state == S_WRITE_CH)
                write_time_req <= 1'b1;
end

always@(posedge clk)
begin
        if(read_time_ack)
                read_time_req <= 1'b0;
```

```
    else if(state == S_READ || state == S_READ_CH)
        read_time_req <= 1'b1;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        state <= S_IDLE;
    else
        state <= next_state;
end


always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        write_second_reg <= 8'h00;
        write_minute_reg <= 8'h00;
        write_hour_reg <= 8'h00;
        write_date_reg <= 8'h00;
        write_month_reg <= 8'h00;
        write_week_reg <= 8'h00;
        write_year_reg <= 8'h00;
    end
    else if(state == S_WRITE_CH)
    begin

        write_second_reg <= 8'h00;
        write_minute_reg <= 8'h11;
        write_hour_reg <= 8'h18;
        write_date_reg <= 8'h11;
        write_month_reg <= 8'h11;
        write_week_reg <= 8'h03;
        write_year_reg <= 8'h20;
    end
end

always@(*)
begin
    case(state)
        S_IDLE:
                next_state <= S_READ_CH;
        S_READ_CH:
            if(read_time_ack)
```

```
                next_state <= CH ? S_WRITE_CH : S_READ;
            else
                next_state <= S_READ_CH;
        S_WRITE_CH:
            if(write_time_ack)
                next_state <= S_WAIT;
            else
                next_state <= S_WRITE_CH;
        S_WAIT:
            next_state <= S_READ;
        S_READ:
            if(read_time_ack)
                next_state <= S_IDLE;
            else
                next_state <= S_READ;
        default:
            next_state <= S_IDLE;
    endcase
end

endmodule
```

3. ds1302.v

```
module ds1302(
    input               rst,
    input               clk,
    output              ds1302_ce,
    output              ds1302_sclk,
    inout               ds1302_io,
    input               write_time_req,
    output              write_time_ack,
    input[7:0]          write_second,
    input[7:0]          write_minute,
    input[7:0]          write_hour,
    input[7:0]          write_date,
    input[7:0]          write_month,
    input[7:0]          write_week,
    input[7:0]          write_year,
    input               read_time_req,
    output              read_time_ack,
    output reg[7:0] read_second,
    output reg[7:0] read_minute,
    output reg[7:0] read_hour,
```

```
        output reg[7:0] read_date,
        output reg[7:0] read_month,
        output reg[7:0] read_week,
        output reg[7:0] read_year

);
localparam S_IDLE              =    0;
localparam S_WR_WP             =    1;
localparam S_WR_SEC            =    2;
localparam S_WR_MIN            =    3;
localparam S_WR_HOUR           =    4;
localparam S_WR_MON            =    5;
localparam S_WR_WEEK           =    6;
localparam S_WR_YEAR           =    7;
localparam S_RD_SEC            =    8;
localparam S_RD_MIN            =    9;
localparam S_RD_HOUR           = 10;
localparam S_RD_MON            = 11;
localparam S_RD_WEEK           = 12;
localparam S_RD_YEAR           = 13;
localparam S_RD_DATE           = 15;
localparam S_ACK               = 14;
localparam S_WR_DATE           = 16;

reg[4:0] state, next_state;
reg[7:0] read_addr;
reg[7:0] write_addr;
reg[7:0] write_data;
wire[7:0] read_data;
reg cmd_write;
reg cmd_read;
wire cmd_read_ack;
wire cmd_write_ack;
assign write_time_ack = (state == S_ACK);
assign read_time_ack = (state == S_ACK);


always@(posedge clk or posedge rst)
begin
        if(rst)
                cmd_write <= 1'b0;
        else if(cmd_write_ack)
                cmd_write <= 1'b0;
        else
```

```
            case(state)
                S_WR_WP,
                S_WR_SEC,
                S_WR_MIN,
                S_WR_HOUR,
                S_WR_DATE,
                S_WR_MON,
                S_WR_WEEK,
                S_WR_YEAR:
                cmd_write <= 1'b1;
            endcase
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        cmd_read <= 1'b0;
    else if(cmd_read_ack)
        cmd_read <= 1'b0;
    else
        case(state)
            S_RD_SEC,
            S_RD_MIN,
            S_RD_HOUR,
            S_RD_DATE,
            S_RD_MON,
            S_RD_WEEK,
            S_RD_YEAR:
                cmd_read <= 1'b1;
        endcase
end


always@(posedge clk or posedge rst)
begin
    if(rst)
        read_second <= 8'h00;
    else if(state == S_RD_SEC && cmd_read_ack)
        read_second <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_minute <= 8'h00;
    else if(state == S_RD_MIN && cmd_read_ack)
```

```
            read_minute <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_hour <= 8'h00;
    else if(state == S_RD_HOUR && cmd_read_ack)
        read_hour <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_date <= 8'h00;
    else if(state == S_RD_DATE && cmd_read_ack)
        read_date <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_month <= 8'h00;
    else if(state == S_RD_MON && cmd_read_ack)
        read_month <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_week <= 8'h00;
    else if(state == S_RD_WEEK && cmd_read_ack)
        read_week <= read_data;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        read_year <= 8'h00;
    else if(state == S_RD_YEAR && cmd_read_ack)
        read_year <= read_data;
end


always@(posedge clk or posedge rst)
begin
    if(rst)
        read_addr <= 8'h00;
    else
```

```
            case(state)
                    S_RD_SEC:
                        read_addr <= 8'h81;
                    S_RD_MIN:
                        read_addr <= 8'h83;
                    S_RD_HOUR:
                        read_addr <= 8'h85;
                    S_RD_DATE:
                        read_addr <= 8'h87;
                    S_RD_MON:
                        read_addr <= 8'h89;
                    S_RD_WEEK:
                        read_addr <= 8'h8b;
                    S_RD_YEAR:
                        read_addr <= 8'h8d;
                default:
                    read_addr <= read_addr;
            endcase
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        begin
            write_addr <= 8'h00;
            write_data <= 8'h00;
        end
    else
        case(state)
            S_WR_WP:
                begin
                    write_addr <= 8'h8e;
                    write_data <= 8'h00;
                end
            S_WR_SEC:
                begin
                    write_addr <= 8'h80;
                    write_data <= write_second;
                end
            S_WR_MIN:
                begin
                    write_addr <= 8'h82;
                    write_data <= write_minute;
                end
            S_WR_HOUR:
```

```
                begin
                    write_addr <= 8'h84;
                    write_data <= write_hour;
                end
            S_WR_DATE:
                begin
                    write_addr <= 8'h86;
                    write_data <= write_date;
                end
            S_WR_MON:
                begin
                    write_addr <= 8'h88;
                    write_data <= write_month;
                end
            S_WR_WEEK:
                begin
                    write_addr <= 8'h8a;
                    write_data <= write_week;
                end
            S_WR_YEAR:
                begin
                    write_addr <= 8'h8c;
                    write_data <= write_year;
                end
            default:
                begin
                    write_addr <= 8'h00;
                    write_data <= 8'h00;
                end
        endcase
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        state <= S_IDLE;
    else
        state <= next_state;
end
always@(*)
begin
    case(state)
        S_IDLE:
            if(write_time_req)
```

```
                next_state <= S_WR_WP;
        else if(read_time_req)
                next_state <= S_RD_SEC;
        else
                next_state <= S_IDLE;
S_WR_WP:
        if(cmd_write_ack)
                next_state <= S_WR_SEC;
        else
                next_state <= S_WR_WP;
S_WR_SEC:
        if(cmd_write_ack)
                next_state <= S_WR_MIN;
        else
                next_state <= S_WR_SEC;
S_WR_MIN:
        if(cmd_write_ack)
                next_state <= S_WR_HOUR;
        else
                next_state <= S_WR_MIN;
S_WR_HOUR:
        if(cmd_write_ack)
                next_state <= S_WR_DATE;
        else
                next_state <= S_WR_HOUR;
S_WR_DATE:
        if(cmd_write_ack)
                next_state <= S_WR_MON;
        else
                next_state <= S_WR_DATE;
S_WR_MON:
        if(cmd_write_ack)
                next_state <= S_WR_WEEK;
        else
                next_state <= S_WR_MON;
S_WR_WEEK:
        if(cmd_write_ack)
                next_state <= S_WR_YEAR;
        else
                next_state <= S_WR_WEEK;
S_WR_YEAR:
        if(cmd_write_ack)
                next_state <= S_ACK;
        else
```

```
                next_state <= S_WR_YEAR;
            S_RD_SEC:
                if(cmd_read_ack)
                    next_state <= S_RD_MIN;
                else
                    next_state <= S_RD_SEC;
            S_RD_MIN:
                if(cmd_read_ack)
                    next_state <= S_RD_HOUR;
                else
                    next_state <= S_RD_MIN;
            S_RD_HOUR:
                if(cmd_read_ack)
                    next_state <= S_RD_DATE;
                else
                    next_state <= S_RD_HOUR;
            S_RD_DATE:
                if(cmd_read_ack)
                    next_state <= S_RD_MON;
                else
                    next_state <= S_RD_DATE;
            S_RD_MON:
                if(cmd_read_ack)
                    next_state <= S_RD_WEEK;
                else
                    next_state <= S_RD_MON;
            S_RD_WEEK:
                if(cmd_read_ack)
                    next_state <= S_RD_YEAR;
                else
                    next_state <= S_RD_WEEK;
            S_RD_YEAR:
                if(cmd_read_ack)
                    next_state <= S_ACK;
                else
                    next_state <= S_RD_YEAR;
            S_ACK:
                next_state <= S_IDLE;
            default:
                next_state <= S_IDLE;
        endcase
    end
    ds1302_io ds1302_io_m0(
        .clk(clk),
```

```
        .rst(rst),
        .ds1302_ce(ds1302_ce),
        .ds1302_sclk(ds1302_sclk),
        .ds1302_io(ds1302_io),
        .cmd_read(cmd_read),
        .cmd_write(cmd_write),
        .cmd_read_ack(cmd_read_ack),
        .cmd_write_ack(cmd_write_ack),
        .read_addr(read_addr),
        .write_addr(write_addr),
        .read_data(read_data),
        .write_data(write_data)
);
endmodule
```

4. ds1302_io.v

```
module ds1302_io(
        input                clk,
        input                rst,
        output               ds1302_ce,
        output               ds1302_sclk,
        inout                ds1302_io,
        input                cmd_read,
        input                cmd_write,
        output               cmd_read_ack,
        output               cmd_write_ack,
        input[7:0]           read_addr,
        input[7:0]           write_addr,
        output reg[7:0] read_data,
        input[7:0]           write_data
);
localparam S_IDLE              =  0;
localparam S_CE_HIGH           =  1;
localparam S_READ              =  2;
localparam S_READ_ADDR         =  3;
localparam S_READ_DATA         =  4;
localparam S_WRITE             =  5;
localparam S_WRITE_ADDR        =  6;
localparam S_WRITE_DATA        =  7;
localparam S_CE_LOW            =  8;
localparam S_ACK               =  9;

reg[3:0] state, next_state;
```

```
reg[19:0] delay_cnt;
reg wr_req;
wire wr_ack;
reg wr_ack_d0;
reg CS_reg;
wire DCLK;
wire MOSI;
wire MISO;
reg[7:0] send_data;
wire[7:0] data_rec;
reg ds1302_io_dir;
assign ds1302_io = ~ds1302_io_dir ? MOSI : 1'bz;
assign MISO = ds1302_io;
assign ds1302_sclk = DCLK;
assign cmd_read_ack = (state == S_ACK);
assign cmd_write_ack = (state == S_ACK);
always@(posedge clk or posedge rst)
begin
    if(rst)
        state <= S_IDLE;
    else
        state <= next_state;
end

always@(*)
begin
    case(state)
        S_IDLE:
            if(cmd_read || cmd_write)
                next_state <= S_CE_HIGH;
            else
                next_state <= S_IDLE;
        S_CE_HIGH:
            if(delay_cnt == 20'd255)
                next_state <= cmd_read ? S_READ : S_WRITE;
            else
                next_state <= S_CE_HIGH;
        S_READ:
            next_state <= S_READ_ADDR;
        S_READ_ADDR:
            if(wr_ack)
                next_state <= S_READ_DATA;
            else
                next_state <= S_READ_ADDR;
```

```
            S_READ_DATA:
                if(wr_ack)
                    next_state <= S_ACK;
                else
                    next_state <= S_READ_DATA;
            S_WRITE:
                next_state <= S_WRITE_ADDR;
            S_WRITE_ADDR:
                if(wr_ack)
                    next_state <= S_WRITE_DATA;
                else
                    next_state <= S_WRITE_ADDR;
            S_WRITE_DATA:
                if(wr_ack)
                    next_state <= S_ACK;
                else
                    next_state <= S_WRITE_DATA;
            S_ACK:
                next_state <= S_CE_LOW;
            S_CE_LOW:
                if(delay_cnt == 20'd255)
                    next_state <= S_IDLE;
                else
                    next_state <= S_CE_LOW;
            default:next_state <= S_IDLE;
        endcase
    end

    always@(posedge clk or posedge rst)
    begin
        if(rst)
            delay_cnt <= 20'd0;
        else if(state == S_CE_HIGH || state == S_CE_LOW)
            delay_cnt <= delay_cnt + 20'd1;
        else
            delay_cnt <= 20'd0;
    end
    always@(posedge clk or posedge rst)
    begin
        if(rst)
            wr_req <= 1'b0;
        else if(wr_ack)
            wr_req <= 1'b0;
        else if(state == S_READ_ADDR || state == S_READ_DATA || state == S_WRITE_ADDR ||
```

```
state == S_WRITE_DATA)
            wr_req <= 1'b1;
end


always@(posedge clk or posedge rst)
begin
      if(rst)
            ds1302_io_dir <= 1'b0;
      else
            ds1302_io_dir <= (state == S_READ_DATA);
end
always@(posedge clk or posedge rst)
begin
      if(rst)
            CS_reg <= 1'b0;
      else if(state == S_CE_HIGH)
            CS_reg <= 1'b1;
      else if(state == S_CE_LOW)
            CS_reg <= 1'b0;
end

always@(posedge clk or posedge rst)
begin
      if(rst)
            read_data <= 8'h00;
      else if(state == S_READ_DATA && wr_ack)
            read_data                                                                    <=
{data_rec[0],data_rec[1],data_rec[2],data_rec[3],data_rec[4],data_rec[5],data_rec[6],data_rec[7]};
end

always@(posedge clk or posedge rst)
begin
      if(rst)
            send_data <= 8'h00;
      else if(state == S_READ_ADDR)
            send_data                                                                    <=
{1'b1,read_addr[1],read_addr[2],read_addr[3],read_addr[4],read_addr[5],read_addr[6],1'b1};
      else if(state == S_WRITE_ADDR)
            send_data                                                                    <=
{1'b0,write_addr[1],write_addr[2],write_addr[3],write_addr[4],write_addr[5],write_addr[6],1'b1};
      else if(state == S_WRITE_DATA)
            send_data                                                                    <=
{write_data[0],write_data[1],write_data[2],write_data[3],write_data[4],write_data[5],write_data[6]
```

```
,write_data[7]};
end

spi_master spi_master_m0(
        .sys_clk(clk),
        .rst(rst),
        .nCS(ds1302_ce),
        .DCLK(DCLK),
        .MOSI(MOSI),
        .MISO(MISO),
        .CPOL(1'b0),
        .CPHA(1'b0),
        .nCS_ctrl(CS_reg),
        .clk_div(16'd50),
        .wr_req(wr_req),
        .wr_ack(wr_ack),
        .data_in(send_data),
        .data_out(data_rec)
);
endmodule
```

5. spi_master.v

```
module spi_master
(
        input                   sys_clk,
        input                   rst,
        output                  nCS,
        output                  DCLK,
        output                  MOSI,
        input                   MISO,
        input                   CPOL,
        input                   CPHA,
        input                   nCS_ctrl,
        input[15:0]             clk_div,
        input                   wr_req,
        output                  wr_ack,
        input[7:0]              data_in,
        output[7:0]             data_out
);
localparam              IDLE            = 0;
localparam              DCLK_EDGE       = 1;
localparam              DCLK_IDLE       = 2;
localparam              ACK             = 3;
```

```
localparam                          LAST_HALF_CYCLE = 4;
localparam                          ACK_WAIT              = 5;
reg                                  DCLK_reg;
reg[7:0]                            MOSI_shift;
reg[7:0]                            MISO_shift;
reg[2:0]                            state;
reg[2:0]                            next_state;
reg [15:0]                          clk_cnt;
reg[4:0]                            clk_edge_cnt;
assign MOSI = MOSI_shift[7];
assign DCLK = DCLK_reg;
assign data_out = MISO_shift;
assign wr_ack = (state == ACK);
assign nCS = nCS_ctrl;
always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        state <= IDLE;
    else
        state <= next_state;
end

always@(*)
begin
    case(state)
        IDLE:
            if(wr_req == 1'b1)
                next_state <= DCLK_IDLE;
            else
                next_state <= IDLE;
        DCLK_IDLE:
            if(clk_cnt == clk_div)
                next_state <= DCLK_EDGE;
            else
                next_state <= DCLK_IDLE;
        DCLK_EDGE:
            if(clk_edge_cnt == 5'd15)
                next_state <= LAST_HALF_CYCLE;
            else
                next_state <= DCLK_IDLE;
        LAST_HALF_CYCLE:
            if(clk_cnt == clk_div)
                next_state <= ACK;
            else
```

```
                    next_state <= LAST_HALF_CYCLE;
            ACK:
                next_state <= ACK_WAIT;
            ACK_WAIT:
                next_state <= IDLE;
            default:
                next_state <= IDLE;
        endcase
end

always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        DCLK_reg <= 1'b0;
    else if(state == IDLE)
        DCLK_reg <= CPOL;
    else if(state == DCLK_EDGE)
        DCLK_reg <= ~DCLK_reg;
end
always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        clk_cnt <= 16'd0;
    else if(state == DCLK_IDLE || state == LAST_HALF_CYCLE)
        clk_cnt <= clk_cnt + 16'd1;
    else
        clk_cnt <= 16'd0;
end
always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        clk_edge_cnt <= 5'd0;
    else if(state == DCLK_EDGE)
        clk_edge_cnt <= clk_edge_cnt + 5'd1;
    else if(state == IDLE)
        clk_edge_cnt <= 5'd0;
end
always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        MOSI_shift <= 8'd0;
    else if(state == IDLE && wr_req)
        MOSI_shift <= data_in;
    else if(state == DCLK_EDGE)
```

```
            if(CPHA == 1'b0 && clk_edge_cnt[0] == 1'b1)
                MOSI_shift <= {MOSI_shift[6:0],MOSI_shift[7]};
            else if(CPHA == 1'b1 && (clk_edge_cnt != 5'd0 && clk_edge_cnt[0] == 1'b0))
                MOSI_shift <= {MOSI_shift[6:0],MOSI_shift[7]};
end
always@(posedge sys_clk or posedge rst)
begin
    if(rst)
        MISO_shift <= 8'd0;
    else if(state == IDLE && wr_req)
        MISO_shift <= 8'h00;
    else if(state == DCLK_EDGE)
        if(CPHA == 1'b0 && clk_edge_cnt[0] == 1'b0)
            MISO_shift <= {MISO_shift[6:0],MISO};
        else if(CPHA == 1'b1 && (clk_edge_cnt[0] == 1'b1))
            MISO_shift <= {MISO_shift[6:0],MISO};
end
endmodule
```

6. seg_bcd.v

```
module seg_bcd(
    input clk,
    input rst_n,
    output[7:0] seg_sel,
    output[7:0] seg_data,
    output[7:0] seg_datatwo,
    input [23:0]seg_bcd
);

wire[6:0] seg7_data_0;
seg_decoder seg_decoder_m0(
    .bin_data(seg_bcd[23:20]),
    .seg_data(seg7_data_0)
);
wire[6:0] seg7_data_1;
seg_decoder seg_decoder_m1(
    .bin_data(seg_bcd[19:16]),
    .seg_data(seg7_data_1)
);
wire[6:0] seg7_data_2;
seg_decoder seg_decoder_m2(
    .bin_data(seg_bcd[15:12]),
    .seg_data(seg7_data_2)
```

```
);
wire[6:0] seg7_data_3;
seg_decoder seg_decoder_m3(
        .bin_data(seg_bcd[11:8]),
        .seg_data(seg7_data_3)
);
wire[6:0] seg7_data_4;
seg_decoder seg_decoder_m4(
        .bin_data(seg_bcd[7:4]),
        .seg_data(seg7_data_4)
);

wire[6:0] seg7_data_5;
seg_decoder seg_decoder_m5(
        .bin_data(seg_bcd[3:0]),
        .seg_data(seg7_data_5)
);
wire[7:0] seg_data_0;
wire[7:0] seg_data_1;
wire[7:0] seg_data_2;
wire[7:0] seg_data_3;
wire[7:0] seg_data_4;
wire[7:0] seg_data_5;
assign seg_data_0 = {1'b0,seg7_data_0};
assign seg_data_1 = {1'b1,seg7_data_1};
assign seg_data_2 = {1'b0,seg7_data_2};
assign seg_data_3 = {1'b1,seg7_data_3};
assign seg_data_4 = {1'b0,seg7_data_4};
assign seg_data_5 = {1'b0,seg7_data_5};
seg_scan seg_scan_m0(
        .clk(clk),
        .rst_n(rst_n),
        .seg_sel(seg_sel),
        .seg_data(seg_data),
        .seg_datatwo(seg_datatwo),
        .seg_data_0(seg_data_0),
        .seg_data_1(seg_data_1),
        .seg_data_2(seg_data_2),
        .seg_data_3(seg_data_3),
        .seg_data_4(seg_data_4),
        .seg_data_5(seg_data_5)
);
endmodule
```

7.seg_decoder.v

```
module seg_decoder
(
    input[3:0]          bin_data,
    output reg[6:0] seg_data
);

always@(*)
begin
    case(bin_data)
            4'd0:seg_data <= 7'b011_1111;
            4'd1:seg_data <= 7'b000_0110;
            4'd2:seg_data <= 7'b101_1011;
            4'd3:seg_data <= 7'b100_1111;
            4'd4:seg_data <= 7'b110_0110;
            4'd5:seg_data <= 7'b110_1101;
            4'd6:seg_data <= 7'b111_1101;
            4'd7:seg_data <= 7'b000_0111;
            4'd8:seg_data <= 7'b111_1111;
            4'd9:seg_data <= 7'b110_1111;

        default:seg_data <= 7'b000_0000;
    endcase
end
endmodule
```

8.seg_scan.v

```
module seg_scan(
    input                   clk,
    input                   rst_n,
    output reg[7:0] seg_sel,
    output reg[7:0] seg_data,
    output reg[7:0] seg_datatwo,
    input[7:0]          seg_data_0,
    input[7:0]          seg_data_1,
    input[7:0]          seg_data_2,
    input[7:0]          seg_data_3,
    input[7:0]          seg_data_4,
    input[7:0]          seg_data_5
);
parameter SCAN_FREQ = 200;
```

```
parameter CLK_FREQ = 100000000;

parameter SCAN_COUNT = CLK_FREQ /(SCAN_FREQ * 6) - 1;

reg[31:0] scan_timer;
reg[31:0] scan_timertwo;
reg[3:0] scan_sel;
reg[3:0] scan_seltwo;
always@(posedge clk or negedge rst_n)
begin
    if(rst_n == 1'b0)
    begin
        scan_timer <= 32'd0;
        scan_sel <= 4'd0;
    end
    else if(scan_timer >= SCAN_COUNT)
    begin
        scan_timer <= 32'd0;
        if(scan_sel == 4'd3)
            scan_sel <= 4'd0;
        else
            scan_sel <= scan_sel + 4'd1;
    end
    else
        begin
            scan_timer <= scan_timer + 32'd1;
        end
end
always@(posedge clk or negedge rst_n)
begin
    if(rst_n == 1'b0)
    begin
        seg_sel <= 6'b000000;
        seg_data <= 8'hff;
    end
    else
    begin
        case(scan_sel)
            4'd0:
            begin
                seg_sel[3:0] <= 4'b0001;
                seg_data <= seg_data_0;
            end
            4'd1:
```

```
        begin
            seg_sel[3:0] <= 4'b0010;
            seg_data <= seg_data_1;
        end
        4'd2:
        begin
            seg_sel[3:0] <= 4'b0100;
            seg_data <= seg_data_2;
        end
        4'd3:
        begin
            seg_sel[3:0] <= 4'b1000;
            seg_data <= seg_data_3;
        end

        default:
        begin
            seg_sel[3:0] <= 4'b0000;
            seg_data <= 8'hff;
        end
    endcase

    case(scan_seltwo)
        4'd0:
        begin
            seg_sel[7:4] <= 4'b0001;
            seg_datatwo <= seg_data_4;
        end
        4'd1:
        begin
            seg_sel[7:4] <= 4'b0010;
            seg_datatwo <= seg_data_5;
        end
        4'd2:
        begin
            seg_sel[7:4] <= 4'b0100;
            seg_datatwo <= 0;
        end
        4'd3:
        begin
            seg_sel[7:4] <= 4'b1000;
            seg_datatwo <= 0;
        end
        default:
```

```
                begin
                    seg_sel[7:4] <= 4'b0000;
                    seg_datatwo <= 8'hff;
                end
                endcase
        end
end

always@(posedge clk or negedge rst_n)
begin
        if(rst_n == 1'b0)
        begin
                scan_timertwo <= 32'd0;
                scan_seltwo <= 4'd0;
        end
        else if(scan_timertwo >= SCAN_COUNT)
        begin
                scan_timertwo <= 32'd0;
                if(scan_seltwo == 4'd3)
                        scan_seltwo <= 4'd0;
                else
                        scan_seltwo <= scan_seltwo + 4'd1;
        end
        else
                begin
                        scan_timertwo <= scan_timertwo + 32'd1;
                end
end

endmodule
```

9.calculate.v（部分）

```
module calculate(
        input [7:0] x,
        input [7:0] y,
        output    [31:0] result_1,
        output    [31:0] result_2
);

wire [15:0] xy;
reg [31:0] result1;
reg [31:0] result2;
assign xy = {x[7:0],y[7:0]};
```

```
assign result_1 = result1;
assign result_2 = result2;

always@(*) begin
case(xy)
16'd0 : result1 = 322341;
16'd1 : result1 = 318810;
16'd2 : result1 = 315412;
16'd3 : result1 = 312138;
16'd4 : result1 = 308980;
16'd5 : result1 = 305930;
16'd6 : result1 = 302980;
16'd7 : result1 = 300124;
16'd8 : result1 = 297355;
16'd9 : result1 = 294668;
16'd10 : result1 = 292057;
//..... 3000+ lines
16'd17962 : result1 = 144686;
16'd17963 : result1 = 143319;
16'd17964 : result1 = 141847;
16'd17965 : result1 = 140254;
16'd17966 : result1 = 138514;
16'd17967 : result1 = 136592;
16'd17968 : result1 = 134434;
16'd17969 : result1 = 131951;
16'd17970 : result1 = 128978;
16'd19758 : result1 = 118000;
endcase;

case(xy)

16'd0 : result2 = 150955;
16'd1 : result2 = 150776;
16'd2 : result2 = 150567;
16'd3 : result2 = 150330;
16'd4 : result2 = 150073;
16'd5 : result2 = 149798;
16'd6 : result2 = 149511;
16'd7 : result2 = 149216;
16'd8 : result2 = 148915;
16'd9 : result2 = 148612;
16'd10 : result2 = 148310;
//..... 3000+ lines
16'd17961 : result2 = 56456;
```

```
16'd17962 : result2 = 57499;
16'd17963 : result2 = 58549;
16'd17964 : result2 = 59605;
16'd17965 : result2 = 60669;
16'd17966 : result2 = 61741;
16'd17967 : result2 = 62822;
16'd17968 : result2 = 63913;
16'd17969 : result2 = 65015;
16'd17970 : result2 = 66127;
16'd19758 : result2 = 55500;
endcase;
end

endmodule
```

10. control.v

```
module    control(
                        clk,
                        rst_n,
                        time_h1,
                        time_h2,
                        time_m1,
                        time_m2,
                        sec,
                        pwm_out1,
                        pwm_out2,
                        pwm_out3
);
input      clk;
input      rst_n;
input    [4:0]       time_h1;
input    [4:0]       time_h2;
input    [4:0]       time_m1;
input    [4:0]       time_m2;
input    [7:0]       sec;
output reg pwm_out1;
output reg pwm_out2;
output reg pwm_out3;

reg clk_20ms;
reg [7:0] x;
reg [7:0] y;
reg [31:0] cnt;
```

```
reg [31:0] cnt_clk;
reg [15:0] cnt_20ms;
wire [31:0] cnt_r1;
wire [31:0] cnt_r2;
reg [31:0] cnt_r3;

calculate calculate0(
    .x    (x),
    .y    (y),
    .result_1     (cnt_r1),
    .result_2     (cnt_r2)
);

always@(posedge clk_20ms or negedge rst_n)begin
    if(!rst_n) begin
        cnt_r3 <= 100000;
    end

    else if(sec>=0 && sec<2) begin                    //抬动降 1
        if(cnt_20ms <= 10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 10 && cnt_20ms <= 90) begin
            case(time_h1)
                4'd0,4'd2,4'd3,4'd4,4'd7,4'd8: x <= 13 +7;
                4'd1,4'd5,4'd6,4'd9: x <= 21 +7;
                default;
            endcase
            case(time_h1)
                4'd0,4'd1,4'd2,4'd3,4'd4,4'd5,4'd6,4'd7: y <= 36;
                4'd8,4'd9: y <= 28;
                default;
            endcase
        end
        else if(cnt_20ms > 90 && cnt_20ms <= 100) begin
            cnt_r3 <= 156000;
        end
        else;
    end

    else if(sec>=2 && sec<4) begin              //画第一位
        if(cnt_20ms > 100+0 && cnt_20ms <= 100+8 ) begin                    //1
            case(time_h1)
                0,2,3,7: x <= x+1;
```

```
            5,6,9: x <= x-1;
            8: y <= y+1;
            1,4: y <= y-1;
            default;
        endcase
    end
    else if(cnt_20ms > 100+8 && cnt_20ms <= 100+16 ) begin                    //2
        case(time_h1)
            4,8: x <= x+1;
            9: y <= y+1;
            0,1,2,3,5,6,7: y <= y-1;
        default;
        endcase
    end
    else if(cnt_20ms > 100+16 && cnt_20ms <= 100+24 ) begin                   //3
        case(time_h1)
            5,9: x <= x+1;
            2,3: x <= x-1;
            4: y <= y+1;
            0,6,7,8: y <= y-1;
            default;
        endcase
    end
    else if(cnt_20ms > 100+24 && cnt_20ms <= 100+32 ) begin                   //4
        case(time_h1)
            3,6: x <= x+1;
            0: x <= x-1;
            2,4,5,8,9: y <= y-1;
            default;
        endcase
    end
    else if(cnt_20ms > 100+32 && cnt_20ms <= 100+40 ) begin                   //5
        case(time_h1)
            2: x <= x+1;
            5,8: x <= x-1;
            0,6: y <= y+1;
            3,4,9: y <= y-1;
            default;
        endcase
    end
    else if(cnt_20ms > 100+40 && cnt_20ms <= 100+48 ) begin                   //6
        case(time_h1)
            3,6,9: x <= x-1;
            0,8: y <= y+1;
```

```
                default;
            endcase
        end
        else if(cnt_20ms > 100+48 && cnt_20ms <= 100+56 ) begin                    //7
            case(time_h1)
                8: x <= x+1;
                default;
            endcase
        end
        else;
end

else if(sec>=4 && sec<5) begin                    //抬动降 2
        if(cnt_20ms <= 200+10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 200+10 && cnt_20ms <= 200+40) begin
            case(time_h2)
                4'd0,4'd2,4'd3,4'd4,4'd7,4'd8: x <= 13 + 11 +7;
                4'd1,4'd5,4'd6,4'd9: x <= 21 + 11 +7;
                default;
            endcase
            case(time_h2)
                4'd0,4'd1,4'd2,4'd3,4'd4,4'd5,4'd6,4'd7: y <= 36;
                4'd8,4'd9: y <= 28;
                default;
            endcase
        end
        else if(cnt_20ms > 200+40 && cnt_20ms <= 200+50) begin
            cnt_r3 <= 157500;
        end
        else;
end

else if(sec>=5 && sec<7) begin                //画第二位
        if(cnt_20ms > 250+0 && cnt_20ms <= 250+8 ) begin                    //1
            case(time_h2)
                0,2,3,7: x <= x+1;
                5,6,9: x <= x-1;
                8: y <= y+1;
                1,4: y <= y-1;
                default;
            endcase
        end
```

```
else if(cnt_20ms > 250+8 && cnt_20ms <= 250+16 ) begin                    //2
    case(time_h2)
        4,8: x <= x+1;
        9: y <= y+1;
        0,1,2,3,5,6,7: y <= y-1;
        default;
    endcase
end
else if(cnt_20ms > 250+16 && cnt_20ms <= 250+24 ) begin                   //3
    case(time_h2)
        5,9: x <= x+1;
        2,3: x <= x-1;
        4: y <= y+1;
        0,6,7,8: y <= y-1;
        default;
    endcase
end
else if(cnt_20ms > 250+24 && cnt_20ms <= 250+32 ) begin                   //4
    case(time_h2)
        3,6: x <= x+1;
        0: x <= x-1;
        2,4,5,8,9: y <= y-1;
        default;
    endcase
end
else if(cnt_20ms > 250+32 && cnt_20ms <= 250+40 ) begin                   //5
    case(time_h2)
        2: x <= x+1;
        5,8: x <= x-1;
        0,6: y <= y+1;
        3,4,9: y <= y-1;
        default;
    endcase
end
else if(cnt_20ms > 250+40 && cnt_20ms <= 250+48 ) begin                   //6
    case(time_h2)
        3,6,9: x <= x-1;
        0,8: y <= y+1;
        default;
    endcase
end
else if(cnt_20ms > 250+48 && cnt_20ms <= 250+56 ) begin                   //7
    case(time_h2)
        8: x <= x+1;
```

```
                default;
            endcase
        end
        else;
end

else if(sec>=7 && sec<8) begin                    //抬动降 3
        if(cnt_20ms <= 350+10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 350+10 && cnt_20ms <= 350+40) begin
            x <= 35 +7;
            y <= 35;
        end
        else if(cnt_20ms > 350+40 && cnt_20ms <= 350+50) begin
            cnt_r3 <= 154000;
        end
        else;
end

else if(sec>=8 && sec<9) begin                    //抬动降 4
        if(cnt_20ms <= 400+10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 400+10 && cnt_20ms <= 400+40) begin
            x <= 35 +7;
            y <= 25;
        end
        else if(cnt_20ms > 400+40 && cnt_20ms <= 400+50) begin
            cnt_r3 <= 154000;
        end
        else;
end

else if(sec>=9 && sec<10) begin                    //抬动降 5
        if(cnt_20ms <= 450+10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 450+10 && cnt_20ms <= 450+40) begin
            case(time_m1)
                4'd0,4'd2,4'd3,4'd4,4'd7,4'd8: x <= 13 + 25 +7;
                4'd1,4'd5,4'd6,4'd9: x <= 21 + 25 +7;
                default;
            endcase
```

```
            case(time_m1)
                4'd0,4'd1,4'd2,4'd3,4'd4,4'd5,4'd6,4'd7: y <= 36;
                4'd8,4'd9: y <= 28;
                default;
            endcase
        end
        else if(cnt_20ms > 450+40 && cnt_20ms <= 450+50) begin
            cnt_r3 <= 154000;
        end
        else;
end

else if(sec>=10 && sec<12) begin                //画第三位
        if(cnt_20ms > 500+0 && cnt_20ms <= 500+7 ) begin                    //1
            case(time_m1)
                0,2,3,7: x <= x+1;
                5,6,9: x <= x-1;
                8: y <= y+1;
                1,4: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 500+7 && cnt_20ms <= 500+14 ) begin                  //2
            case(time_m1)
                4,8: x <= x+1;
                9: y <= y+1;
                0,1,2,3,5,6,7: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 500+14 && cnt_20ms <= 500+21 ) begin                 //3
            case(time_m1)
                5,9: x <= x+1;
                2,3: x <= x-1;
                4: y <= y+1;
                0,6,7,8: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 500+21 && cnt_20ms <= 500+28 ) begin                 //4
            case(time_m1)
                3,6: x <= x+1;
                0: x <= x-1;
                2,4,5,8,9: y <= y-1;
```

```
                default;
            endcase
        end
        else if(cnt_20ms > 500+28 && cnt_20ms <= 500+35 ) begin                    //5
            case(time_m1)
                2: x <= x+1;
                5,8: x <= x-1;
                0,6: y <= y+1;
                3,4,9: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 500+35 && cnt_20ms <= 500+42 ) begin                    //6
            case(time_m1)
                3,6,9: x <= x-1;
                0,8: y <= y+1;
                default;
            endcase
        end
        else if(cnt_20ms > 500+42 && cnt_20ms <= 500+49 ) begin                    //7
            case(time_m1)
                8: x <= x+1;
                default;
            endcase
        end
        else;
end

else if(sec>=12 && sec<13) begin                    //抬动降 6
    if(cnt_20ms <= 600+10) begin
        cnt_r3 <= 100000;
    end
    else if(cnt_20ms > 600+10 && cnt_20ms <= 600+40) begin
        case(time_m2)
            4'd0,4'd2,4'd3,4'd4,4'd7,4'd8: x <= 13 + 36 +7;
            4'd1,4'd5,4'd6,4'd9: x <= 21 + 36 +7;
            default;
        endcase
        case(time_m2)
            4'd0,4'd1,4'd2,4'd3,4'd4,4'd5,4'd6,4'd7: y <= 36;
            4'd8,4'd9: y <= 28;
            default;
        endcase
    end
```

```
        else if(cnt_20ms > 600+40 && cnt_20ms <= 600+50) begin
            cnt_r3 <= 154000;
        end
        else;
    end

    else if(sec>=13 && sec<15) begin                    //画第四位
        if(cnt_20ms > 650+0 && cnt_20ms <= 650+6 ) begin                    //1
            case(time_m2)
                0,2,3,7: x <= x+1;
                5,6,9: x <= x-1;
                8: y <= y+1;
                1,4: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+6 && cnt_20ms <= 650+12 ) begin                    //2
            case(time_m2)
                4,8: x <= x+1;
                9: y <= y+1;
                0,1,2,3,5,6,7: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+12 && cnt_20ms <= 650+18 ) begin                    //3
            case(time_m2)
                5,9: x <= x+1;
                2,3: x <= x-1;
                4: y <= y+1;
                0,6,7,8: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+18 && cnt_20ms <= 650+24 ) begin                    //4
            case(time_m2)
                3,6: x <= x+1;
                0: x <= x-1;
                2,4,5,8,9: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+24 && cnt_20ms <= 650+30 ) begin                    //5
            case(time_m2)
                2: x <= x+1;
```

```
                5,8: x <= x-1;
                0,6: y <= y+1;
                3,4,9: y <= y-1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+30 && cnt_20ms <= 650+36 ) begin                    //6
            case(time_m2)
                3,6,9: x <= x-1;
                0,8: y <= y+1;
                default;
            endcase
        end
        else if(cnt_20ms > 650+36 && cnt_20ms <= 650+42 ) begin                    //7
            case(time_m2)
                8: x <= x+1;
                default;
            endcase
        end
        else;
end

else if(sec>=15 && sec<16) begin                    //抬动降 7 到笔擦
        if(cnt_20ms <= 750+10) begin
            cnt_r3 <= 100000;
        end
        else if(cnt_20ms > 750+10 && cnt_20ms <= 750+40) begin
            x <= 77;
            y <= 46;
        end
        else if(cnt_20ms > 750+40 && cnt_20ms <= 750+50) begin
            cnt_r3 <= 154000;
        end
        else;
end

else if(sec>=51 && sec<57) begin                    //清屏
        cnt_r3 <= 158500;
        if(cnt_20ms > 2550 && cnt_20ms <= 2550+50) begin
            x <= 55 + 7;
            y <= 40;
        end
        else if(cnt_20ms > 2550+50 && cnt_20ms <= 2550+90) begin
            cnt_r3 <= 160000;
```

```
            x <= x-1;
        end
        else if(cnt_20ms > 2550+90 && cnt_20ms <= 2550+99) begin
            y <= y-1;
        end
        else if(cnt_20ms > 2550+99 && cnt_20ms <= 2550+139) begin
            x <= x+1;
        end
        else if(cnt_20ms > 2550+139 && cnt_20ms <= 2550+148) begin
            cnt_r3 <= 158500;
            y <= y-1;
        end
        else if(cnt_20ms > 2550+148 && cnt_20ms <= 2550+186) begin
            x <= x-1;
        end
        else if(cnt_20ms > 2550+186 && cnt_20ms <= 2550+195) begin
            y <= y-1;
        end
        else if(cnt_20ms > 2550+195 && cnt_20ms <= 2550+235) begin
            x <= x+1;
        end
        else if(cnt_20ms > 2550+235 && cnt_20ms <= 2550+244) begin
            y <= y-1;
        end
        else if(cnt_20ms > 2550+244 && cnt_20ms <= 2550+282) begin
            x <= x-1;
        end
        else;
    end

        else if(sec>=57 && sec<59) begin                    //最后动到笔擦
            x <= 77;
            y <= 46;

        end

    else if(sec >=59 && sec <60) begin
        cnt_r3 <= 100000;
    end

    else begin
       cnt_r3 <= 154000;
    end
```

```
end                    //结束


always@(posedge clk or negedge rst_n)begin                    //20ms 计数
    if(!rst_n)
        cnt_20ms <= 16'd0;
    else if(sec==59)
        cnt_20ms <= 16'd0;
    else if(cnt_clk == 2000000)
        cnt_20ms <= cnt_20ms + 1;
    else cnt_20ms <= cnt_20ms;
end

always@(posedge clk or negedge rst_n)begin                    //普通 cnt
    if(!rst_n)
        cnt <= 31'd0;
    else if(cnt >= 2000000)
        cnt <= 31'd0;
    else
        cnt <= cnt + 1'b1;
end
always@(posedge clk or negedge rst_n)begin                    //为了便于 20ms 计数的 cnt_clk
    if(!rst_n)
        cnt_clk <= 31'd0;
    else if(sec == 59)
        cnt_clk <= 31'd0;
    else if(cnt_clk >= 2000000)
        cnt_clk <= 31'd0;
    else
        cnt_clk <= cnt_clk + 1'b1;
end

always@(posedge clk or negedge rst_n)begin                    //分频
    if(!rst_n)
        clk_20ms <= 31'd0;
    else if(sec == 59)
        clk_20ms <= 31'd0;
    else if(cnt_clk <= 1000000)
        clk_20ms <= 1;
    else
        clk_20ms <= 0;
end

always@(posedge clk or negedge rst_n)begin
```

```
        if(!rst_n)
            pwm_out1 <= 1'b0;
        else if(cnt_clk <= cnt_r1)
            pwm_out1 <= 1'b1;
        else
            pwm_out1 <= 1'b0;
end
always@(posedge clk or negedge rst_n)begin
        if(!rst_n)
            pwm_out2 <= 1'b0;
        else if(cnt_clk <= cnt_r2)
            pwm_out2 <= 1'b1;
        else
            pwm_out2 <= 1'b0;
end
always@(posedge clk or negedge rst_n)begin
        if(!rst_n)
            pwm_out3 <= 1'b0;
        else if(cnt_clk <= cnt_r3)
            pwm_out3 <= 1'b1;
        else
            pwm_out3 <= 1'b0;
end

endmodule
```