浙江大学 实验报告

专业：电子信息工程
姓名：冯静怡
学号：3220104119
地点：紫金港东三 406
日期：2024 年 6 月 22 日

课程名称：微机原理及应用　　指导老师：吴建德、胡斯登
实验名称：旋转钟

1 实验目的

1. 熟练使用 STM32 的 C 语言编程。
2. 熟悉 STM32 的定时器和中断方式。
3. 学会软硬件协同设计。

2 实验过程

2.1 实验基本完成功能

1. **文字显示**：旋转钟侧面显示“姓名、学号”，并能够进行滚动。
2. **时钟显示**：旋转钟正面显示时间，通过 DS1302 芯片，实时显示时间。
3. **红外控制**：红外遥控对旋转钟的显示进行控制。

2.2 硬件功能分析

2.2.1 旋转灯供电

通过 type-C 接口输入电源，随后通过线圈耦合，使得 type-C 中的电源输入到上面的面板，从而实现 STM32 的供电。

2.2.2 74HC595 功能介绍与分配

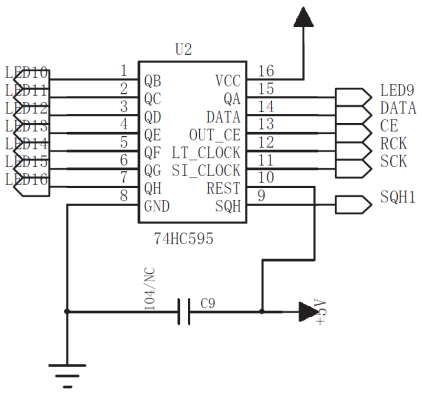


图 1: 74HC595 硬件框图

595 芯片实现了通过 1 个串行引脚、1 个时钟同步控制信号和 1 个使能端，控制 8 个输出，以扩展 STM32 本身芯片的功能。

通过 SQH 引脚，输入到下一个引脚的 DATA 之中，从而实现多片 595 串联，从而可以通过 1 个 DATA 引脚控制（本实验中）32 个引脚的输出。

595 芯片主要由 3 部分组成：移位寄存器，8 位寄存器，输出

RCK（图中 SH_CP）为移位寄存器信号，每一个上升沿将 DATA 中的数据输入到移位寄存器中，并进行移位。

SCK（图中 ST_CP）为 8 位寄存器信号，每一个上升沿将移位寄存器的数据存储到 8 位寄存器，并输出。

$\overline{\text{OE}}$ 控制输出使能，高为禁止输出态，输出高阻态，本实验中直接设置其为低电平，保证能够实时显示 LED 灯亮。

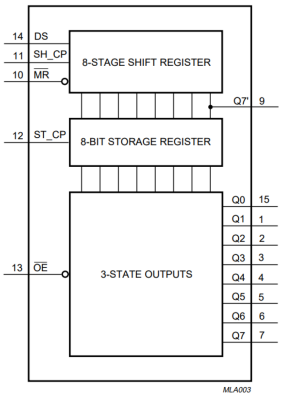


图 2: 595 芯片内部原理

595 芯片控制 32 盏灯的控制代码：

```
shine = LED_Data;
tmp = 0x80000000; //shine的掩码，代表当前处理的位，先处理最高位。
LL_GPIO_ResetOutputPin(SCK_GPIO_Port, SCK_Pin); //shift Register置低，准备迎接上升沿
LL_GPIO_ResetOutputPin(RCK_GPIO_Port, RCK_Pin); //8-bit Register置低，准备迎接上升沿
for(int i=0;i<32;i++)
{
    if(shine & tmp) //如果当前位要亮
        LL_GPIO_ResetOutputPin(DATA_GPIO_Port, DATA_Pin); //DATA置0代表亮灯
    else
        LL_GPIO_SetOutputPin(DATA_GPIO_Port, DATA_Pin);
    Delay;
    LL_GPIO_SetOutputPin(SCK_GPIO_Port, SCK_Pin); //移位寄存器创造一个上升沿，将DATA数据输入
    Delay;
    LL_GPIO_ResetOutputPin(SCK_GPIO_Port, SCK_Pin); //置低，准备迎接上升沿
    tmp = tmp >> 1;
}
LL_GPIO_SetOutputPin(RCK_GPIO_Port, RCK_Pin); //上升沿，存储同时输出移位寄存器中的32位数据
Delay;
LL_GPIO_ResetOutputPin(RCK_GPIO_Port, RCK_Pin);
```

2.2.3 LED 灯显示

LED 灯排布 由于 LED 分布的特殊性，所以在编程过程中，需要注意对 LED 灯进行转换，使其正常输出。

LED17-32 由 STM32 的 GPIO 直接控制。

LED9-16 DATA 数据的 0-7 位；

LED1-8 DATA 数据的 8-15 位；

D1-8 DATA 数据的 23-16 位；

D9-16 DATA 数据的 31-24 位；

数据显示 将旋转钟转一圈分为 240 份。

旋转钟一圈的判定通过硬件中的红外发射/接收设备。当红外停止接收红外信号，产生一个上升沿，从而触发中断，记录两次中断间的时间，重新设置显示中断为中断时间的 1/240，从而能够在每一个位置精确显示特定数值。显示中断每次显示调用 LED 显示函数。

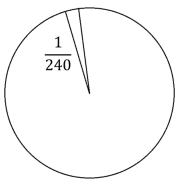
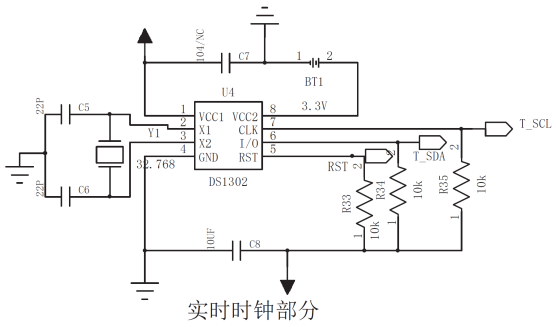


图 3: LED 显示示意图

2.2.4 DS1302 芯片



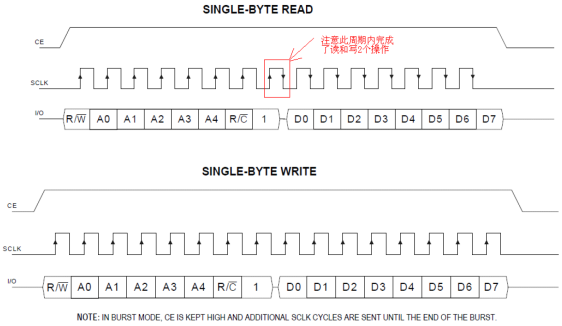
实时时钟部分

图 4: DS1302 硬件框图

DS1302 是 DALLAS 公司推出的涓流充电时钟芯片内含有一个实时时钟/日历和 31 字节静态 RAM，可通过简单的串行接口与单片机进行通信。控制引脚有：

- T_SCL 同步时钟信号，方便 IO 口时序同步。
- T_SDA 串行数据输入/输出。
- RST 为高则启用串行输入/输出。

对 DS1302 进行读写操作 DS1302 有可读写的寄存器，存储了时间信息。



(a) DS1302 读写时序图

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	12/24	0	10 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	—

(b) DS1302 寄存器说明

图 5: DS1302 读写操作

写入数据 写入数据时，命令由两部分组成，1. 寄存器地址；2. 寄存器要存放的 8 位数据

```
1  /*通过GPIO的Output模式模拟串行口输出数据*/
2  void DS1302WriteByte(uint8_t dat)
3  {
4    uint8_t i;
5    LL_GPIO_ResetOutputPin(T_SCL_GPIO_Port,
         T_SCL_Pin);
6    for(int delay=0;delay<2;delay++){}
7    for(i=0;i<8;i++)
8    {
9      if(dat&0x01)
10       LL_GPIO_SetOutputPin(T_SDA_GPIO_Port,
             T_SDA_Pin);
11     else
12       LL_GPIO_ResetOutputPin(T_SDA_GPIO_Port,
             T_SDA_Pin);
13     Delay;
14     LL_GPIO_SetOutputPin(T_SCL_GPIO_Port,
           T_SCL_Pin);
15     Delay;
16     LL_GPIO_ResetOutputPin(T_SCL_GPIO_Port,
           T_SCL_Pin);
17     dat>>=1;
18   }
19 }
20 /* DS1302Write写入命令、数据，主要依靠
       DS1302WriteByte 实现*/
21 void DS1302Write(uint8_t cmd, uint8_t dat)
22 {
23     LL_GPIO_ResetOutputPin(RST_GPIO_Port,
           RST_Pin);
24     LL_GPIO_ResetOutputPin(T_SCL_GPIO_Port,
           T_SCL_Pin);
25     LL_GPIO_SetOutputPin(RST_GPIO_Port,
           RST_Pin);
26     DS1302WriteByte(cmd);
27     DS1302WriteByte(dat);
28     LL_GPIO_SetOutputPin(T_SCL_GPIO_Port,
           T_SCL_Pin);
29     LL_GPIO_ResetOutputPin(RST_GPIO_Port,
           RST_Pin);
30 }
```

读取数据　读入数据时，先要输入要读取数据的地址命令，然后调节 GPIO 模式为 INPUT 从而读入数据

```
/*通过GPIO的Input模式模拟串行口输入数据*/
uint8_t DS1302ReadByte(void)
{
  uint8_t i,dat=0;
  for(int delay=0;delay<1;delay++){}
  for(i=0;i<8;i++)
  {
    dat>>=1;
    if(LL_GPIO_IsInputPinSet(T_SDA_GPIO_Port,
        T_SDA_Pin))
      dat|=0x80;
    LL_GPIO_SetOutputPin(T_SCL_GPIO_Port,
        T_SCL_Pin);
    Delay;
    LL_GPIO_ResetOutputPin(T_SCL_GPIO_Port,
        T_SCL_Pin);
    Dealy;
  }
  return dat;
}
/* DS1302Read写入命令、数据，主要依靠
    DS1302ReadByte/WriteByte 实现*/
uint8_t DS1302Read(uint8_t cmd)
{
  uint8_t dat=0;
  LL_GPIO_ResetOutputPin(RST_GPIO_Port,
      RST_Pin);
  LL_GPIO_ResetOutputPin(T_SCL_GPIO_Port,
      T_SCL_Pin);
  LL_GPIO_SetOutputPin(RST_GPIO_Port, RST_Pin
      );
  DS1302WriteByte(cmd);
  LL_GPIO_SetPinMode(T_SDA_GPIO_Port,
      T_SDA_Pin, LL_GPIO_MODE_INPUT);
  dat=DS1302ReadByte();
  LL_GPIO_SetPinMode(T_SDA_GPIO_Port,
      T_SDA_Pin,LL_GPIO_MODE_OUTPUT);
  LL_GPIO_SetOutputPin(T_SCL_GPIO_Port,
      T_SCL_Pin);
  LL_GPIO_ResetOutputPin(RST_GPIO_Port,
      RST_Pin);
  return dat;
}
```

DS1302 初始化　在第一次使用 DS1302 时，需要对芯片内的数据进行初始化，具体代码如下：

```
void Init_DS1302(void)
{
    DS1302Write(0x8e, 0x00); // 控制寄存器地址 0x8E, 数
        据 0x00 (解除写保护)
    DS1302Write(0x80, 0x20); // 秒寄存器地址 0x80, 数据
        0x20 (20秒)
    DS1302Write(0x82, 0x27); // 分钟寄存器地址 0x82, 数
        据 0x27 (27分钟)
    DS1302Write(0x84, 0x89); // 小时寄存器地址 0x84, 数
        据 0x89 (12小时制, 9点)
    /*年月日星期代码略*/
    DS1302Write(0x90, 0x01); // 充电寄存器地址 0x90, 数
        据 0x01 (启用充电)
    DS1302Write(0xc0, 0xf0); // RAM 地址 0xC0, 数据 0
        xF0 (初始化标志)
    DS1302Write(0x8e, 0x80); // 控制寄存器地址 0x8E, 数
        据 0x80 (启用写保护)
}
```

从 DS1302 芯片中读取实时数据　设置中断，每秒从 DS1302 中读取一次数据，就能够实现时间显示。

```
void getdate()
{
    // 读取年份
    date1 = DS1302Read(0x8D);
    date[0] = (date1 & 0xF0) >> 4; // 年高位
    date[1] = date1 & 0x0F; // 年低位
    // 读取月份
    date1 = DS1302Read(0x89);
    date[2] = (date1 & 0xF0) >> 4; // 月高位
    date[3] = date1 & 0x0F; // 月低位
    /*其余代码略*/
}
```

2.2.5　红外遥控

接收到红外信号终端控制　当接收到红外遥控信号，通过下降沿进行中断

以下代码用于通过外部中断处理程序，以读取红外信号的脉冲宽度，并根据脉冲宽度解码成对应的红外数据。最终将解码得到的数据存储在 command 变量中。

```c
if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_7) != RESET)
{
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_7);

    // 获取当前的定时器计数器值
    currentCapture = LL_TIM_GetCounter(TIM2);

    // 计算脉冲宽度
    pulseWidth = currentCapture - lastCapture;
    lastCapture = currentCapture;
    laaCapture = lastCapture;

    // 判断脉冲宽度，根据不同的条件进行处理
    if (pulseWidth > 13000 && pulseWidth < 14000 && dataReady == 0) // 9000+4500 = 13500
    {
        // 开始读取数据，重置状态
        bitIndex = 0;
        command_pre = 0;
        FinishReading = 0;
    }
    else if (pulseWidth > 10500 && pulseWidth < 12000 && FinishReading != 1) // 9000+2250 = 11250
    {
        // 继续读取数据，重置状态
        bitIndex = 0;
        command_pre = 0;
    }
    else if (FinishReading)
    {
        // 数据已读取完成，退出
        return;
    }
    else if (pulseWidth < 500)
    {
        // 脉冲宽度太短，可能是干扰或者无效信号，忽略
        lastCapture = laaCapture;
        return;
    }
    else if (pulseWidth > 2000 && pulseWidth < 2500) // 560+1690 = 2250
    {
        // 长脉冲表示二进制位为1
        command_pre = (command_pre << 1) | 1;
        bitIndex++;
    }
    else if (pulseWidth > 800 && pulseWidth < 1500) // 560+560 = 1120
    {
        // 短脉冲表示二进制位为0
        command_pre = command_pre << 1;
        bitIndex++;
    }

    // 判断是否已经读取完32位数据
    if (bitIndex == 32)
    {
        // 数据已经完整读取
        bitIndex = 0;
        command = command_pre; // 将解码后的数据存入 command 变量
        dataReady = 1; // 数据标志位，表示数据已准备好
        FinishReading = 1; // 结束读取状态
    }
}
```

代码功能解释

- `currentCapture = LL_TIM_GetCounter(TIM2)`：获取定时器 TIM2 的当前计数值，以计算上一次实验到现在的时间。
- `pulseWidth = currentCapture - lastCapture`：计算当前脉冲的宽度。
- 信号判断：根据红外通信协议，长脉冲代表二进制位为 1，短脉冲代表二进制位为 0。
- **数据存储：**

  `command_pre` 变量存储正在解码的数据。

  `command` 变量最终存储完整的解码数据。

- **状态标志位：**

  `bitIndex` 记录当前位的索引。

  `dataReady` 标志位表示数据已准备好。

  `FinishReading` 标志位表示数据读取完成。

主程序　对读取到的红外数据进行处理，使用查询方法。

```
while (1)
{
    if (dataReady)
    {
        switch (command)
        {
            case IRR_Start:
                LL_GPIO_TogglePin(
                    LED00_GPIO_Port, LED00_Pin);
                break;
            case IRR_Right:
                /*代码略*/
                default:
                    break;
        }
        dataReady = 0;
        /*避免红外信号冲突，延时0.5s*/
        LL_mDelay(500);
        FinishReading = 0;
    }
}
```

设置 `WorkState`，在循环中根据 case 判断结果，在 main.h 中设置 WorkState 掩码，便于操作。

```
#define NEC_Rotate_Right (1<<0)
#define NEC_Rotate_Left (1<<1)
#define NEC_Time_Adjust (1<<2)
#define NEC_Show (1<<3)
```

2.2.6 定时器

TIM1 用于调用 DS1302 读取时间函数，每 500ms 调用一次。

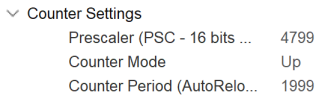


图 6: TIM1 设置

$$t = \frac{1}{48 \cdot 10^6} \cdot (4800 - 1) \cdot (2000 - 1) = 0.5s$$

中断函数代码如下：

```
if(LL_TIM_IsActiveFlag_UPDATE(TIM1)!= RESET)
  {
    LL_TIM_ClearFlag_UPDATE(TIM1);
    getdate(); //读取DS1302时间
    spoint=date[12]*10+date[13]; //秒
    mpoint=date[10]*10+date[11]; //分
    hpoint=(date[8]&1)*10+date[9]; //时
  }
```

TIM2 设置 us 计时器，用于读取红外遥控信号，不设置中断。

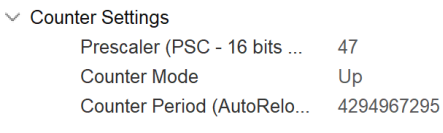


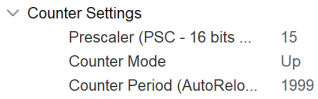
图 7: TIM2 设置

$$t = \frac{1}{48 \cdot 10^6} \cdot (48 - 1) = 1us$$

TIM3 用于 LED 显示，具体时间由红外读取中断内控制。

红外中断控制代码（GPIO 下降沿中断）：

```
if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_6)
    != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_6);
    /* USER CODE BEGIN LL_EXTI_LINE_6 */
    if(timecount<120) return; //防止误触发
    //获取TIM3当前counter值以及autoreload重复次数，以
        便计算240份情况下应该的autoreload值。
    uint16_t timeint = (timecount*(
        LL_TIM_GetAutoReload(TIM3)+1)+
        LL_TIM_GetCounter(TIM3)+1)/240;
    LL_TIM_SetAutoReload(TIM3,timeint-1);
    timecount = 0; //在TIM3中累加，此处清零
  }
```



$$t = \frac{1}{48 \cdot 10^6} \cdot (16 - 1) = 1/3us$$

图 8: TIM3 设置

TIM3 中断控制代码：

```
if(LL_TIM_IsActiveFlag_UPDATE(TIM3)!= RESET)
{
    LL_TIM_ClearFlag_UPDATE(TIM3);
    renew(timecount%240,idx_c); //调用LED显示函数
    timecount++; //计数
}
```

3 程序编写过程

3.1 程序流程设计

程序大致流程图如下所示：

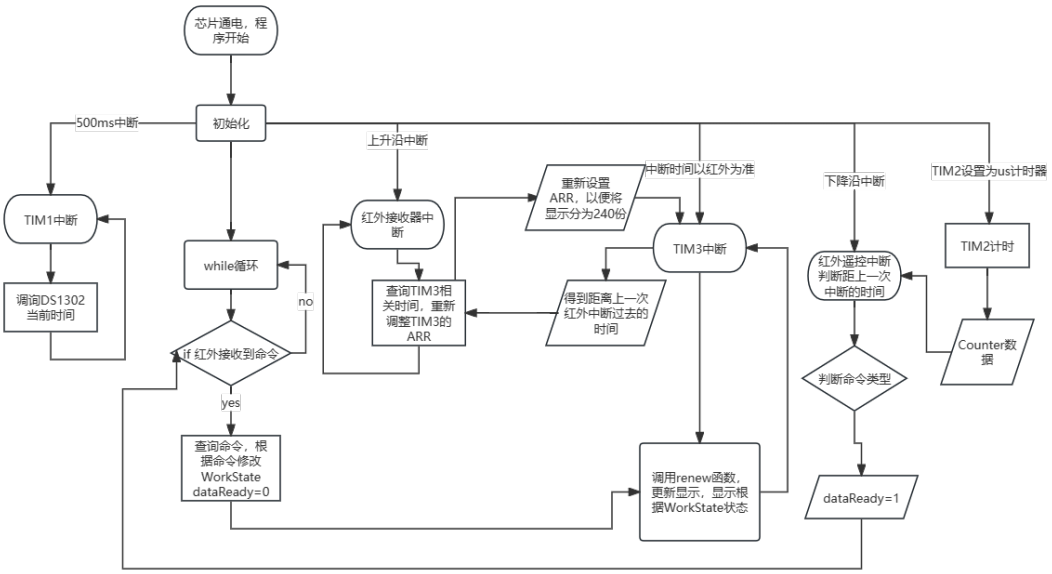


图 9: 程序流程图

3.2 软件功能细节部分实现

3.2.1 坐标转换

64*64xy 轴坐标变换为 240*32 的极坐标，并交给 LED 然后显示。在程序外编写一个坐标转换函数，然后将结果输出，使得能够在 STM32 中使用：

```
void polar_transform(unsigned char polar_bitmap[4][ANGLES][RADIUS]) {
    int cx = WIDTH / 2;
    int cy = HEIGHT / 2;
    for (int w = 0; w < 4; w++)
    {
        for (int angle = 0; angle < ANGLES; angle++) {
            double theta = (2.0 * M_PI * angle) /
```

```
                ANGLES;

            for (int r = 0; r < RADIUS; r++) {
                int x = cx + (int)(r * cos(theta));
                int y = cy - (int)(r * sin(theta));

                if (x >= 0 && x < WIDTH && y >= 0
                    && y < HEIGHT) {
                    int byte_index = (y * WIDTH + x) /
                        8;
                    int bit_index = 7 - ((y * WIDTH + x
                        ) % 8);
                    int pixel_value = (bitmap_bytes[w][
                        byte_index] >> bit_index) & 0
                        x01;

                    polar_bitmap[w][angle][r] =
                        pixel_value;
                } else {
                    polar_bitmap[w][angle][r] = 0;
                }
            }
        }
    }
}
```

3.2.2 LED 显示 renew() 函数

LED 显示数据

```
void renew(uint8_t idx_datab,uint8_t idx_datac)
{
  uint32_t tmp=0x80000000; //595芯片显示掩码
  uint32_t shine=datac_pre[idx_datac]<<16;
  uint8_t spointer,mpointer,hpointer;
  if((WorkState&NEC_Show)==0)
  {
    /*时钟指针的相关代码*/
    shine|=datab_pre[idx_datab];
    /*将s,m,h转换为转盘上240刻度的对应位置*/
    spointer=spoint*4;
    mpointer=spointer/60+mpoint*4;
    hpointer=mpointer/12+hpoint*20;
    shine|=display_date(idx_datab);
    if(idx_datab==(239-spointer)) shine|=0xF0FF;
    else if(idx_datab==(239-mpointer)) shine|=0xFC;
  }
  else
  {
    show_cnt++; //动画切换帧
    if(show_cnt>=480)
    {
      show_cnt=0;
      show_flag=(show_flag+1)%4;
    }
    shine |= x[show_flag][idx_datab];
  }
  /*595芯片显示，代码在上已显示，具体略*/
  if((WorkState&NEC_Show)==0) //相关WorkState状态为0
  {
    /*设置GPIO显示时钟*/
  }
  else //相关WorkState状态为1
  {
    /*显示预设置动画*/
    tmp=y[show_flag][idx_datab]&0xFF;
    LL_GPIO_ResetOutputPin(GPIOA,tmp);
    tmp^=0xFF;
    LL_GPIO_SetOutputPin(GPIOA,tmp);
    tmp=((y[show_flag][idx_datab]>>8)&0x7)|((y[show_flag][idx_datab]>>1)&0x7C00);
    LL_GPIO_ResetOutputPin(GPIOB,tmp);
    tmp^=0x7C07;
    LL_GPIO_SetOutputPin(GPIOB,tmp);
  }
}
```

代码说明

- `idx_datab` 为转盘上当前显示的数据，`idx_datac` 为侧边当前显示的数据的位置，由于侧边会转动，因此两个索引不同，使得两者显示分离。
- 先调用 595 相关显示，再调用 `LL_GPIO` 相关显示，原因为 595 芯片显示速度较慢，需要先显示，否则显示有所扭曲。
- 实验中一开始创建动画数据 x[4][240],y[4][240]，结果显示内存不足。后在命名时将其定义为常量 `const`，内存足量，应为将数据从 RAM 中移动到 ROM 中，减少了 RAM 的使用。

4 最终实验结果

实现功能

- 基础功能
  - 侧边字幕旋转
  - 上面表盘实时转动
- 红外遥控器操纵，接收到红外遥控信号，上下红灯闪烁一次。
  - CH 键，切换显示时钟/动画
  - » 键，侧边字幕右旋，多次点按，旋转速度不同
  - « 键，侧边字幕左旋，多次点按，旋转速度不同
  - 播放键，上下红灯开启/关闭