计控实验代码复习

姓名：杨溴天 学号：20221689 成绩：_____

目录

1 实验一：屏幕驱动

1.1 软件 SPI 通信

```
void WriteBytes(unsigned char Data)
{
    for (unsigned char i = 0; i < 8; i++)
    {
        // 根据最高位决定写入高电平还是低电平
        if (Data & 0x80)
        {
            HAL_GPIO_WritePin(LCD12864_SDA_GPIO_Port, LCD12864_SDA_Pin, GPIO_PIN_SET);
        }
        else
        {
            HAL_GPIO_WritePin(LCD12864_SDA_GPIO_Port, LCD12864_SDA_Pin, GPIO_PIN_RESET);
        }

        // 模拟时钟下降沿（保持低电平一段时间）
        SCK_L;
        SCK_L;
        SCK_L;
        SCK_L; // 经测试，4条SCK_L指令时间刚好能正常显示

        SCK_H; // 上升沿触发数据写入

        Data <<= 1; // 准备下一位
    }
}
```

1.2 软件 SPI 写入数据

```
// 写3个字节数据到液晶显示器
void WriteData(unsigned char *p)
{
    for (unsigned char j = 0; j < 3; j++)
    {
        WriteBytes(p[j]);
    }
}

// 重组一个字节数据并写入液晶显示器
void SerialWriteData(unsigned char send)
{
    unsigned char temp[3];

    temp[0] = 0xFA;             // 向液晶写数据命令
    temp[1] = send & 0xF0;      // 取高4位
    temp[2] = (send << 4) & 0xF0; // 取低4位

    CS_H_SW; // 片选置高

    WriteData(temp);

    CS_L_SW; // 片选置低
    // 不可检测忙状态，使用延时代替
}

```

```
// 串行写入一个"数据"到LCD12864
void SerialWriteData2(unsigned char dat)
{
    CS_H_SW; // 片选置高

    WriteBytes(0xFA);           // 向液晶写“数据”命令
    WriteBytes(dat & 0xF0);    // 高4位
    WriteBytes((dat << 4) & 0xF0); // 低4位

    CS_L_SW; // 片选置低
}
```

1.3 串行驱动八位数码管

```
/**
 * 发送一个字节到74HC595芯片
 * @param byte 要发送的8位数据，高位先发
 */
void HC595_Send_Byte(unsigned char byte)
{
    for (unsigned char i = 0; i < 8; i++)
    {
        // 步骤1：将当前bit写入DS引脚（高位先发）
        if (byte & 0x80)
            HC595_Data_High();  // 数据为1，DS高电平
        else
            HC595_Data_Low();   // 数据为0，DS低电平

        // 步骤2：SHCP产生上升沿，移位寄存器接收当前bit
        HC595_SHCP_Low();
        HC595_SHCP_High();

        // 左移一位，准备下一位数据
        byte <<= 1;
    }
}

/**
 * 输出锁存使能（STCP上升沿触发，将移位寄存器数据加载到存储寄存器）
 */
void HC595_CS(void)
{
    // 步骤3：STCP产生一个上升沿，更新输出
    HC595_STCP_Low();
    HAL_Delay(1);               // 稳定延时
    HC595_STCP_High();
    HAL_Delay(1);
}

/**
 * 发送多个字节到74HC595（支持级联）
 * @param data 指向要发送的数据数组
 * @param len 数据长度（字节数）
 */
void HC595_Send_Multi_Byte(unsigned char *data, unsigned short int len)
{
    for (unsigned short int i = 0; i < len; i++)
    {
```

```
45          HC595_Send_Byte(data[i]);
46      }
47
48      HC595_CS();  // 所有字节发送完成后，触发输出更新
49  }
50
51  /**
52   * 使用595控制数码管显示（双字节发送模式）
53   * 第一个字节：位选（DIG_CS_CA_INVERTED）
54   * 第二个字节：段选（digitMapCC[count[i]]）
55   */
56  void LAB_595_display(void)
57  {
58      char cmd[2];
59
60      for (int i = 0; i < 8; i++)
61      {
62          cmd[0] = DIG_CS_CA_INVERTED[7 - i];  // 位选
63          cmd[1] = digitMapCC[count[i]];       // 段选
64
65          HC595_Send_Multi_Byte((unsigned char *)cmd, 2);
66          // HAL_Delay(1);  // 可根据需要取消注释以增加刷新间隔
67      }
68  }
```

1.4 AI 给出的软件模拟 SPI、IIC

```
    #define SPI_SCK_PORT   GPIOB
#define SPI_SCK_PIN    GPIO_PIN_5

#define SPI_MOSI_PORT  GPIOB
#define SPI_MOSI_PIN   GPIO_PIN_6

#define SPI_MISO_PORT  GPIOB
#define SPI_MISO_PIN   GPIO_PIN_7

#define SPI_CS_PORT    GPIOB
#define SPI_CS_PIN     GPIO_PIN_8

// 延时函数（根据系统频率调整）
void Delay_us(uint32_t us) {
    HAL_Delay(us / 1000 + (us % 1000 ? 1 : 0)); // 简单替代，可用 SysTick 更精确
}
void Soft_SPI_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    __HAL_RCC_GPIOB_CLK_ENABLE();

    // SCK, MOSI, CS 设置为输出
    GPIO_InitStruct.Pin = SPI_SCK_PIN | SPI_MOSI_PIN | SPI_CS_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    // MISO 设置为输入
    GPIO_InitStruct.Pin = SPI_MISO_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    // 默认状态
    HAL_GPIO_WritePin(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET);
}void Soft_SPI_WriteByte(uint8_t data)
{
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_RESET); // 选中设备

    for (int i = 0; i < 8; i++)
    {
        if (data & 0x80)
            HAL_GPIO_WritePin(SPI_MOSI_PORT, SPI_MOSI_PIN, GPIO_PIN_SET);
        else
            HAL_GPIO_WritePin(SPI_MOSI_PORT, SPI_MOSI_PIN, GPIO_PIN_RESET);

        data <<= 1;

        HAL_GPIO_WritePin(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_PIN_SET); // 上升沿发送数据
        Delay_us(1);
        HAL_GPIO_WritePin(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_PIN_RESET);
        Delay_us(1);
    }

    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET); // 取消选中
}uint8_t Soft_SPI_ReadByte(void)
```

```
{
    uint8_t data = 0;

    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_RESET);

    for (int i = 0; i < 8; i++)
    {
        data <<= 1;

        HAL_GPIO_WritePin(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_PIN_SET);
        Delay_us(1);

        if (HAL_GPIO_ReadPin(SPI_MISO_PORT, SPI_MISO_PIN))
            data |= 0x01;

        HAL_GPIO_WritePin(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_PIN_RESET);
        Delay_us(1);
    }

    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET);

    return data;
}

#define I2C_SCL_PORT   GPIOB
#define I2C_SCL_PIN    GPIO_PIN_9

#define I2C_SDA_PORT   GPIOB
#define I2C_SDA_PIN    GPIO_PIN_10

void I2C_Delay(void)
{
    for (volatile int i = 0; i < 100; i++); // 根据主频调整
}void Set_SDA_Output(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = I2C_SDA_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(I2C_SDA_PORT, &GPIO_InitStruct);
}

void Set_SDA_Input(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = I2C_SDA_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(I2C_SDA_PORT, &GPIO_InitStruct);
}void I2C_Start(void)
{
    Set_SDA_Output();
    HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_SET);
    I2C_Delay();

    HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_RESET);
    I2C_Delay();

```

```
118     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
119 }
120 
121 void I2C_Stop(void)
122 {
123     Set_SDA_Output();
124     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
125     HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_RESET);
126     I2C_Delay();
127 
128     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_SET);
129     I2C_Delay();
130 
131     HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_SET);
132 }void I2C_WriteByte(uint8_t byte)
133 {
134     for (int i = 0; i < 8; i++)
135     {
136         HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
137         I2C_Delay();
138 
139         if (byte & 0x80)
140             HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_SET);
141         else
142             HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, GPIO_PIN_RESET);
143 
144         byte <<= 1;
145         I2C_Delay();
146 
147         HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_SET);
148         I2C_Delay();
149     }
150 
151     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
152 }uint8_t I2C_ReadByte(uint8_t ack)
153 {
154     uint8_t byte = 0;
155 
156     Set_SDA_Input();
157 
158     for (int i = 0; i < 8; i++)
159     {
160         byte <<= 1;
161 
162         HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_SET);
163         I2C_Delay();
164 
165         if (HAL_GPIO_ReadPin(I2C_SDA_PORT, I2C_SDA_PIN))
166             byte |= 0x01;
167 
168         HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
169         I2C_Delay();
170     }
171 
172     Set_SDA_Output();
173     HAL_GPIO_WritePin(I2C_SDA_PORT, I2C_SDA_PIN, ack ? GPIO_PIN_RESET : GPIO_PIN_SET);
174     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_SET);
175     I2C_Delay();
176     HAL_GPIO_WritePin(I2C_SCL_PORT, I2C_SCL_PIN, GPIO_PIN_RESET);
```

```
177
178     return byte;
179 }
```

2 实验二：按键扫描

2.1 按键扫描

```
1  int key_scan(void)
2  {
3      // 读取当前按键状态
4      int key1 = HAL_GPIO_ReadPin(S1_GPIO_Port, S1_Pin);
5      int key2 = HAL_GPIO_ReadPin(S3_GPIO_Port, S3_Pin);
6      int key3 = HAL_GPIO_ReadPin(S4_GPIO_Port, S4_Pin);
7      int key4 = HAL_GPIO_ReadPin(S5_GPIO_Port, S5_Pin);
8
9      // 检查按键1是否按下（低电平有效）
10     if (key1 != 1)
11     {
12         HAL_Delay(20); // 延时去抖动
13         if (key1 == HAL_GPIO_ReadPin(S1_GPIO_Port, S1_Pin))
14         {
15             keysta[0] = key1; // 更新按键状态
16             return 1;
17         }
18     }
19
20     // 检查按键2是否按下
21     if (key2 != 1)
22     {
23         HAL_Delay(20);
24         if (key2 == HAL_GPIO_ReadPin(S3_GPIO_Port, S3_Pin))
25         {
26             keysta[1] = key2;
27             return 2;
28         }
29     }
30
31     // 检查按键3是否按下
32     if (key3 != 1)
33     {
34         HAL_Delay(20);
35         if (key3 == HAL_GPIO_ReadPin(S4_GPIO_Port, S4_Pin))
36         {
37             keysta[2] = key3;
38             return 3;
39         }
40     }
41
42     // 检查按键4是否按下
43     if (key4 != 1)
44     {
45         HAL_Delay(20);
46         if (key4 == HAL_GPIO_ReadPin(S5_GPIO_Port, S5_Pin))
47         {
48             keysta[3] = key4;
49             return 4;
```

```
        }
    }

    return 0; // 无按键按下
}
```

2.2 16 键键盘

```
void scan_KeyBoard(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    uint8_t row = 0, col = 0;

    // ----------------------------
    // 第一步：检测按键所在的行（Row）
    // 配置列为输出，并拉低电平
    // ----------------------------

    GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull  = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

    GPIO_InitStruct.Pin = C1_Pin;
    HAL_GPIO_Init(C1_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = C2_Pin | C3_Pin | C4_Pin;
    HAL_GPIO_Init(C2_GPIO_Port, &GPIO_InitStruct);

    // 所有列输出低电平
    HAL_GPIO_WritePin(C1_GPIO_Port, C1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C2_GPIO_Port, C2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C3_GPIO_Port, C3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C4_GPIO_Port, C4_Pin, GPIO_PIN_RESET);

    // 配置行为输入模式，带上拉电阻
    GPIO_InitStruct.Mode  = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull  = GPIO_PULLUP;

    GPIO_InitStruct.Pin = R1_Pin | R2_Pin | R3_Pin | R4_Pin;
    HAL_GPIO_Init(R1_GPIO_Port, &GPIO_InitStruct);

    // 延时稳定信号
    HAL_Delay(10);

    // 检测哪一行被拉低
    if (HAL_GPIO_ReadPin(R1_GPIO_Port, R1_Pin) == GPIO_PIN_RESET) row = 1;
    else if (HAL_GPIO_ReadPin(R2_GPIO_Port, R2_Pin) == GPIO_PIN_RESET) row = 2;
    else if (HAL_GPIO_ReadPin(R3_GPIO_Port, R3_Pin) == GPIO_PIN_RESET) row = 3;
    else if (HAL_GPIO_ReadPin(R4_GPIO_Port, R4_Pin) == GPIO_PIN_RESET) row = 4;

    if (row == 0) return; // 无按键按下

    // ----------------------------
    // 第二步：检测按键所在的列（Column）
    // 配置行为输出，并拉低电平
    // ----------------------------

    GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
```

```
    GPIO_InitStruct.Pull  = GPIO_NOPULL;

    GPIO_InitStruct.Pin = R1_Pin;
    HAL_GPIO_Init(R1_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = R2_Pin;
    HAL_GPIO_Init(R2_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = R3_Pin;
    HAL_GPIO_Init(R3_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = R4_Pin;
    HAL_GPIO_Init(R4_GPIO_Port, &GPIO_InitStruct);

    // 所有行输出低电平
    HAL_GPIO_WritePin(R1_GPIO_Port, R1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(R2_GPIO_Port, R2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(R3_GPIO_Port, R3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(R4_GPIO_Port, R4_Pin, GPIO_PIN_RESET);

    // 配置列为输入模式，带上拉电阻
    GPIO_InitStruct.Mode  = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull  = GPIO_PULLUP;

    GPIO_InitStruct.Pin = C1_Pin;
    HAL_GPIO_Init(C1_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = C2_Pin;
    HAL_GPIO_Init(C2_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = C3_Pin;
    HAL_GPIO_Init(C3_GPIO_Port, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = C4_Pin;
    HAL_GPIO_Init(C4_GPIO_Port, &GPIO_InitStruct);

    // 延时稳定信号
    HAL_Delay(10);

    // 检测哪一列被拉低
    if (HAL_GPIO_ReadPin(C1_GPIO_Port, C1_Pin) == GPIO_PIN_RESET) col = 1;
    else if (HAL_GPIO_ReadPin(C2_GPIO_Port, C2_Pin) == GPIO_PIN_RESET) col = 2;
    else if (HAL_GPIO_ReadPin(C3_GPIO_Port, C3_Pin) == GPIO_PIN_RESET) col = 3;
    else if (HAL_GPIO_ReadPin(C4_GPIO_Port, C4_Pin) == GPIO_PIN_RESET) col = 4;

    if (col == 0) return; // 无按键按下

    // 设置按键标志
    lcd_flag = 1;

    // 计算键值：(col - 1) * 4 + row
    int ret = (col - 1) * 4 + row;

    if (ret <= 10)
    {
        curr_key = (ret - 1) + '0';
    }
    else
    {
```

```
        curr_key = (ret - 11) + 'A';
    }

    // 存入历史按键缓冲区
    history_key[index_key] = curr_key;
    index_key = (index_key + 1) % history_key_size;

    // 可选：刷新LCD显示
    // LCD_Display_Words(1, 0, (unsigned char *)&curr_key);
}
```

3 实验三：D/A 转换

3.1 DAC0832 生成正弦波

```
    #include "dac0832.h"
#include "math.h"

// DAC0832 引脚定义（请根据实际电路调整）
#define DAC_D0_Pin      GPIO_PIN_12
#define DAC_D0_GPIO_Port GPIOB
#define DAC_D1_Pin      GPIO_PIN_13
#define DAC_D1_GPIO_Port GPIOB
#define DAC_D2_Pin      GPIO_PIN_14
#define DAC_D2_GPIO_Port GPIOB
#define DAC_D3_Pin      GPIO_PIN_15
#define DAC_D3_GPIO_Port GPIOB
#define DAC_D4_Pin      GPIO_PIN_6
#define DAC_D4_GPIO_Port GPIOC
#define DAC_D5_Pin      GPIO_PIN_7
#define DAC_D5_GPIO_Port GPIOC
#define DAC_D6_Pin      GPIO_PIN_8
#define DAC_D6_GPIO_Port GPIOC
#define DAC_D7_Pin      GPIO_PIN_9
#define DAC_D7_GPIO_Port GPIOC

#define DAC_CS_Pin      GPIO_PIN_8
#define DAC_CS_GPIO_Port GPIOA
#define DAC_WR_Pin      GPIO_PIN_9   // 写使能引脚
#define DAC_WR_GPIO_Port GPIOA

// 正弦波查找表配置
#define SINE_POINTS     256
uint8_t sine_table[SINE_POINTS];
uint16_t sine_index = 0;

/**
 * @brief  初始化DAC0832
 */
void DAC0832_Init(void)
{
    // 确保GPIO已在MX_GPIO_Init()中初始化
    HAL_GPIO_WritePin(DAC_CS_GPIO_Port, DAC_CS_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(DAC_WR_GPIO_Port, DAC_WR_Pin, GPIO_PIN_SET);
}

/**
```

```
 * @brief  向DAC0832写入一个字节的数据
 * @param  data 要写入的8位数据
 */
void DAC0832_Write(uint8_t data)
{
    // 拉低CS信号，选中芯片
    HAL_GPIO_WritePin(DAC_CS_GPIO_Port, DAC_CS_Pin, GPIO_PIN_RESET);

    // 拉低WR信号，开始写入
    HAL_GPIO_WritePin(DAC_WR_GPIO_Port, DAC_WR_Pin, GPIO_PIN_RESET);

    // 设置数据线 D0 ~ D7
    HAL_GPIO_WritePin(DAC_D0_GPIO_Port, DAC_D0_Pin, (data & 0x01) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D1_GPIO_Port, DAC_D1_Pin, (data & 0x02) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D2_GPIO_Port, DAC_D2_Pin, (data & 0x04) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D3_GPIO_Port, DAC_D3_Pin, (data & 0x08) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D4_GPIO_Port, DAC_D4_Pin, (data & 0x10) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D5_GPIO_Port, DAC_D5_Pin, (data & 0x20) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D6_GPIO_Port, DAC_D6_Pin, (data & 0x40) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DAC_D7_GPIO_Port, DAC_D7_Pin, (data & 0x80) ? GPIO_PIN_SET : GPIO_PIN_RESET);

    // 稳定等待一小段时间
    for (uint8_t i = 0; i < 10; i++);

    // 拉高WR信号，锁存数据
    HAL_GPIO_WritePin(DAC_WR_GPIO_Port, DAC_WR_Pin, GPIO_PIN_SET);

    // 拉高CS信号，结束写入
    HAL_GPIO_WritePin(DAC_CS_GPIO_Port, DAC_CS_Pin, GPIO_PIN_SET);
}

/**
 * @brief  初始化正弦波查找表
 */
void DAC0832_SineWave_Init(void)
{
    float pi = 3.14159265359f;

    for (int i = 0; i < SINE_POINTS; i++)
    {
        float angle = (2.0f * pi * i) / SINE_POINTS;
        sine_table[i] = (uint8_t)((sinf(angle) + 1.0f) * 127.5f);  // 映射到 0~255
    }

    sine_index = 0;
}

/**
 * @brief  更新正弦波输出
 */
void DAC0832_SineWave_Update(void)
{
    DAC0832_Write(sine_table[sine_index]);
    sine_index = (sine_index + 1) % SINE_POINTS;
}
```

3.2 PWM 输出方波

```c
    #include "main.h"

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_TIM3_Init();  // 假设使用 TIM3

    // 启动 PWM 输出（CH1）
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);

    while (1)
    {
        // 主循环可添加其他逻辑
    }
}

/* 定时器 TIM3 初始化函数（由 STM32CubeMX 自动生成） */
void MX_TIM3_Init(void)
{
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 83;          // 84MHz / (83+1) = 1 MHz
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 999;            // 1 MHz / (999 + 1) = 1 kHz
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);

    // PWM 通道配置
    TIM_OC_InitTypeDef sConfig = {0};
    sConfig.OCMode = TIM_OCMODE_PWM1;
    sConfig.Pulse = 500;                // 占空比 50% (500 / 1000)
    sConfig.Comparable = TIM_COMPAREOP_EQ;
    sConfig.OCPolarity = TIM_OCPOLARITY_HIGH;
    sConfig.OCFastMode = TIM_OCFAST_DISABLE;
    HAL_TIM_PWM_ConfigChannel(&htim3, &sConfig, TIM_CHANNEL_1);
}
```

3.3 DAC122S085 生成锯齿波

```c
    #include "DAC122S085.h"
extern SPI_HandleTypeDef hspi1;

/**
 * @brief  初始化 DAC122S085
 * @param  hspi: SPI句柄指针
 * @retval None
 */
void DAC122S085_Init(SPI_HandleTypeDef *hspi)
{
    // 初始化CS引脚为高电平
    CS_H;
    HAL_Delay(1);
}

/**
 * @brief  设置DAC输出电压
```

```
 * @param  hspi: SPI句柄指针
 * @param  channel: DAC通道 (0 = A通道, 1 = B通道)
 * @param  value: 12位DAC值 (0 - 4095)
 * @retval None
 */
void DAC122S085_SetVoltage(SPI_HandleTypeDef *hspi, uint8_t channel, uint16_t value)
{
    uint16_t data = 0;

    // 限制输入值在12位范围内 (0 ~ 4095)
    value &= 0x0FFF;

    // 构建发送数据: 前4位为控制位
    if (channel == DAC_CHANNEL_A)
    {
        data = 0x1000 | value;  // A通道控制字: 0x1000
    }
    else
    {
        data = 0x9000 | value;  // B通道控制字: 0x9000
    }

    // 开始SPI传输
    CS_L;
    HAL_SPI_Transmit(hspi, (uint8_t *)&data, 2, HAL_MAX_DELAY);
    CS_H;
}

/**
 * @brief  向DAC122S085写入数据并更新DA输出
 * @param  ch: 通道选择
 *         - 00: A通道
 *         - 01: B通道
 *         - 10/11: 输出0
 * @param  md: 模式选择
 *         - 00: 写入数据, 不刷新输出
 *         - 01: 写入数据并刷新输出
 *         - 10: 同时写入两个通道并刷新输出
 *         - 11: 进入休眠模式, 输出0
 * @param  daout: 要写入的12位DAC值 (0 ~ 4095)
 * @retval None
 */
void DAC122S085(unsigned char ch, unsigned char md, unsigned int daout)
{
    uint16_t data;

    // 只保留12位有效数据
    daout &= 0xFFF;

    // 构造16位数据帧: [ch(2位)][md(2位)][daout(12位)]
    data = ((uint16_t)ch << 14) | ((uint16_t)md << 12) | (uint16_t)daout;

    // 片选使能
    CS_H;
    CS_L;

    // 发送数据
    if (HAL_SPI_Transmit(&hspi1, (uint8_t *)&data, 2, HAL_MAX_DELAY) != HAL_OK)
    {
```

```
        // TODO: 错误处理
        // Error_Handler();
    }
}
```

3.4 片内 DAC 输出三角波

```
    #include "main.h"

#define WAVE_SAMPLES  64     // 波形点数
uint16_t triangle_table[WAVE_SAMPLES];

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DAC_Init();

    // 生成三角波查找表
    for (int i = 0; i < WAVE_SAMPLES; i++)
    {
        if (i < WAVE_SAMPLES / 2)
            triangle_table[i] = (i * 2 * 4095) / WAVE_SAMPLES;   // 上升段
        else
            triangle_table[i] = ((WAVE_SAMPLES - i) * 2 * 4095) / WAVE_SAMPLES; // 下降段
    }

    // 开启 DAC 通道 1
    HAL_DAC_Start(&hdac, DAC_CHANNEL_1);

    while (1)
    {
        // 循环输出三角波
        for (int i = 0; i < WAVE_SAMPLES; i++)
        {
            HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, triangle_table[i]);
            HAL_Delay(1);  // 控制频率，1ms/点 => 约 15Hz（可改为更精确延时或用定时器）
        }
    }
}
```

4 实验四：A/D 转换

4.1 AD1674 进行输入电压采样

```
    // AD1674 控制引脚定义
#define AD1674_CS_Pin          GPIO_PIN_0
#define AD1674_CS_GPIO_Port    GPIOB
#define AD1674_RD_Pin          GPIO_PIN_1
#define AD1674_RD_GPIO_Port    GPIOB
#define AD1674_WR_Pin          GPIO_PIN_2
#define AD1674_WR_GPIO_Port    GPIOB
#define AD1674_MODE_Pin        GPIO_PIN_3
#define AD1674_MODE_GPIO_Port  GPIOB

```

```
// 数据总线 D0-D15（假设接在GPIOC）
#define AD1674_DATA_PORT      GPIOC
void AD1674_Init(void);
void AD1674_StartConversion(void);
uint16_t AD1674_ReadResult(void);
float AD1674_GetVoltage(float vref);
void AD1674_Init(void)
{
    // 设置数据端口为输入
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    // 数据总线：PC0 ~ PC15 输入
    GPIO_InitStruct.Pin = GPIO_PIN_All;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(AD1674_DATA_PORT, &GPIO_InitStruct);

    // 控制引脚：CS, RD, WR, MODE 输出
    GPIO_InitStruct.Pin = AD1674_CS_Pin | AD1674_RD_Pin | AD1674_WR_Pin | AD1674_MODE_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(AD1674_CS_GPIO_Port, &GPIO_InitStruct);

    // 默认状态设置
    HAL_GPIO_WritePin(AD1674_CS_GPIO_Port, AD1674_CS_Pin, GPIO_PIN_SET);     // CS 高（非选中）
    HAL_GPIO_WritePin(AD1674_RD_GPIO_Port, AD1674_RD_Pin, GPIO_PIN_SET);     // RD 高（无效）
    HAL_GPIO_WritePin(AD1674_WR_GPIO_Port, AD1674_WR_Pin, GPIO_PIN_SET);     // WR 高（无效）
    HAL_GPIO_WritePin(AD1674_MODE_GPIO_Port, AD1674_MODE_Pin, GPIO_PIN_SET); // MODE 高：12位模式
}
void AD1674_StartConversion(void)
{
    // 选中芯片
    HAL_GPIO_WritePin(AD1674_CS_GPIO_Port, AD1674_CS_Pin, GPIO_PIN_RESET);

    // 下降沿触发转换（WR 上升沿触发）
    HAL_GPIO_WritePin(AD1674_WR_GPIO_Port, AD1674_WR_Pin, GPIO_PIN_RESET);
    HAL_Delay(1);  // 短暂延时确保稳定
    HAL_GPIO_WritePin(AD1674_WR_GPIO_Port, AD1674_WR_Pin, GPIO_PIN_SET);

    // 取消选中（可选，取决于时序要求）
    HAL_GPIO_WritePin(AD1674_CS_GPIO_Port, AD1674_CS_Pin, GPIO_PIN_SET);
}
uint16_t AD1674_ReadResult(void)
{
    uint16_t result = 0;

    // 再次选中芯片
    HAL_GPIO_WritePin(AD1674_CS_GPIO_Port, AD1674_CS_Pin, GPIO_PIN_RESET);

    // 拉低RD，允许读取数据
    HAL_GPIO_WritePin(AD1674_RD_GPIO_Port, AD1674_RD_Pin, GPIO_PIN_RESET);

    // 读取16位数据总线（只用12位有效）
    result = (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_0) << 0) |
             (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_1) << 1) |
             (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_2) << 2) |
             (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_3) << 3) |
             (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_4) << 4) |
             (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_5) << 5) |
```

```
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_6) << 6) |
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_7) << 7) |
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_8) << 8) |
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_9) << 9) |
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_10) << 10) |
            (HAL_GPIO_ReadPin(AD1674_DATA_PORT, GPIO_PIN_11) << 11);

    // 恢复RD高电平
    HAL_GPIO_WritePin(AD1674_RD_GPIO_Port, AD1674_RD_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(AD1674_CS_GPIO_Port, AD1674_CS_Pin, GPIO_PIN_SET);

    return result & 0x0FFF;  // 返回12位结果
}
float AD1674_GetVoltage(float vref)
{
    uint16_t adc_val = AD1674_ReadResult();
    return ((float)adc_val / 4095.0f) * vref;
}
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();

    AD1674_Init();

    while (1)
    {
        AD1674_StartConversion();
        HAL_Delay(1);  // 等待转换完成（视具体时钟频率调整）

        float voltage = AD1674_GetVoltage(5.0);  // 假设参考电压为5.0V
        // 打印电压值或用于其他处理
    }
}
```

4.2 内置 ADC 采样

```
    #include "main.h"

ADC_HandleTypeDef hadc1;

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_ADC1_Init();

    // 启动 ADC
    HAL_ADC_Start(&hadc1);

    while (1)
    {
        // 触发单次转换
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

        // 获取12位ADC值（0~4095）
```

```
        uint16_t adcValue = HAL_ADC_GetValue(&hadc1);

        // 将ADC值转换为电压（假设参考电压为3.3V）
        float voltage = (adcValue / 4095.0f) * 3.3f;

        // 可以在这里添加串口打印电压值
        // printf("ADC Value: %d, Voltage: %.3f V\r\n", adcValue, voltage);
    }
}
```

5 实验五：串口通信

5.1 根据按键进行串口发送

```
    if(scan_KeyBoard())
    {
      if (curr_key == '3')
      {
        HAL_UART_Transmit(&huart2, (uint8_t *)name_key, sizeof(name_key) * 30, 1000); //发送名字
      }
      if(curr_key == '6')
      {
        HAL_UART_Transmit(&huart2, (uint8_t *)number_key, sizeof(number_key) * 30, 1000); //发送学号
      }

    }
    void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if(htim == &htim2){
    snprintf(ADC_str, sizeof(ADC_str), "ADC value: %.2fV\nADC value(12bit): %d\n", ADC_Value_float,
     ADC_value);
    HAL_UART_Transmit(&huart2, (uint8_t *)ADC_str, sizeof(ADC_str), 1000); //发送AD值
                    //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);

  }
```

5.2 根据接收信息进行操作

```
初始化接收中断：  HAL_UART_Receive_IT(&huart2, uartRxBuffer, sizeof(uartRxBuffer)); //开启串口接收中
    断
初始化接收中断：  HAL_UART_Receive_IT(&huart2, uartRxBuffer, sizeof(uartRxBuffer)); //开启串口接收中
    断
初始化接收中断：  HAL_UART_Receive_IT(&huart2, uartRxBuffer, sizeof(uartRxBuffer)); //开启串口接收中
    断

uint8_t uartRxBuffer[12];
uint8_t uartTxBuffer[128];
/**
 * @brief UART 接收完成回调函数
 * @param huart 指向UART句柄
 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART2) // 判断是否为 USART2 的接收中断
    {
        // 处理接收到的数据
        if (strstr((char*)uartRxBuffer, "open02LED"))
```

```
        {
            // 打开 LED（低电平点亮）
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
            ledStatus = 1;

            // 发送确认信息
            sprintf((char*)uartTxBuffer, "LED D1已打开\r\n");
            HAL_UART_Transmit(&huart2, uartTxBuffer, strlen((char*)uartTxBuffer), 100);
        }
        else if (strstr((char*)uartRxBuffer, "close02LED"))
        {
            // 关闭 LED（高电平熄灭）
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
            ledStatus = 0;

            // 发送确认信息
            sprintf((char*)uartTxBuffer, "LED D1已关闭\r\n");
            HAL_UART_Transmit(&huart2, uartTxBuffer, strlen((char*)uartTxBuffer), 100);
        }
        else if (strstr((char*)uartRxBuffer, "flicker02"))
        {
            // 解析闪烁频率
            char *freqStr = strstr((char*)uartRxBuffer, "flicker02") + 9;
            int freq = freqStr[0] - '0';

            if (freq >= 1 && freq <= 9)
            {
                flashFrequency = freq;
                ledStatus = 2;

                // 发送确认信息
                sprintf((char*)uartTxBuffer, "LED D1正以%dHz频率闪烁\r\n", flashFrequency);
                HAL_UART_Transmit(&huart2, uartTxBuffer, strlen((char*)uartTxBuffer), 100);
            }
            else
            {
                // 频率无效
                sprintf((char*)uartTxBuffer, "无效闪烁频率\r\n");
                HAL_UART_Transmit(&huart2, uartTxBuffer, strlen((char*)uartTxBuffer), 100);
            }
        }
        else
        {
            // 默认回传收到的信息
            sprintf((char*)uartTxBuffer, "收到: %s\r\n", uartRxBuffer);
            HAL_UART_Transmit(&huart2, uartTxBuffer, strlen((char*)uartTxBuffer), 100);
        }

        // 清空接收缓冲区并重新开启接收中断
        memset(uartRxBuffer, 0, sizeof(uartRxBuffer));
        HAL_UART_Receive_IT(&huart2, uartRxBuffer, sizeof(uartRxBuffer));
    }
    else
    {
        // 错误处理: 非预期的 UART 实例
        HAL_UART_Transmit(&huart2, (uint8_t *)"Error: Unknown UART instance\n", 30, 1000);
    }
}
```

6 实验 7：电机驱动

6.1 电机速度测量

```
int PWM_pulse = 100; // PWM占空比，初始值10%
int Moter_direction = 0; // 电机方向，0-正转，1-反转，2-停止

volatile uint32_t IC_Val1 = 0;          // 保存第一次捕获值
volatile uint32_t IC_Val2 = 0;          // 保存第二次捕获值
volatile uint32_t Difference = 0;       // 保存两次捕获的差值（计数值）
volatile uint8_t Is_First_Captured = 0; // 状态标志，0表示等待第一次捕获，1表示等待第二次捕获
volatile float Frequency = 0.0f;        // 计算得到的频率（Hz）
volatile float RPM = 0.0f;             // 计算得到的转速（RPM）
const uint32_t Timer_ARR = 65535;      // TIM2的ARR值（根据配置修改）
const float Timer_Clock_Frequency = 1000000.0f; // 定时器计数频率（Hz），根据PSC和时钟源计算得到
                                                // 例如 72MHz / (71+1) = 1MHz
const float Pulses_Per_Revolution = 10.0f; // 每转脉冲数（根据实际传感器修改）
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    // 确保是TIM2的中断源且是通道1触发的（虽然回调函数是特定的，但检查htim->Instance是好习惯）
    if (htim->Instance == TIM2 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
    {
      if (Is_First_Captured == 0) // 等待第一次捕获
      {
        // 读取第一次捕获值
        IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // 读取CCR1寄存器值
        Is_First_Captured = 1; // 标记已捕获第一次
      }
      else // 已捕获第一次，现在是第二次捕获
      {
        // 读取第二次捕获值
        IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

        // 计算差值，处理定时器溢出
        if (IC_Val2 >= IC_Val1)
        {
          Difference = IC_Val2 - IC_Val1;
        }
        else // 定时器溢出
        {
          // ARR是自动重载值（例如 65535）
          // 定时器周期是 ARR+1 个计数
          Difference = (Timer_ARR - IC_Val1) + IC_Val2 + 1;
          // 或者写成 Difference = ((htim->Instance->ARR) - IC_Val1) + IC_Val2 + 1;
        }

        // 计算频率（Hz）
        if (Difference != 0) // 避免除以零
        {
          // 注意：Timer_Clock_Frequency 需要根据你的实际配置计算
          // Timer_Clock_Frequency = HAL_RCC_GetPCLK1Freq() / (htim->Init.Prescaler + 1);
          // **重要**：如果APB1预分频器不是1分频（RCC_ClkInitStruct.APB1CLKDivider），
          // APB1上的定时器时钟通常是PCLK1的2倍（除非PCLK1等于AHB时钟）。
          // 请务必通过CubeMX的时钟配置界面或参考手册确认TIM2的精确时钟源频率。
          // 假设 Timer_Clock_Frequency 已正确计算并定义为常量或变量

          Frequency = Timer_Clock_Frequency / (float)Difference;

          // 计算RPM
```

```
56            RPM = (Frequency / Pulses_Per_Revolution) * 60.0f;
57          }
58          else
59          {
60            // 差值为0，可能速度过快或错误，置零处理
61            Frequency = 0.0f;
62            RPM = 0.0f;
63          }
64
65          // 将当前的捕获值作为下一次计算的第一个值，准备下一次测量
66          IC_Val1 = IC_Val2;
67          // Is_First_Captured 保持为 1，持续测量
68        }
69      }
70  }
```

6.2 电机正反转控制

表 1: L298 的逻辑功能

IN1	IN2	ENA	电机状态
X	X	0	停止
1	0	1	顺时针
0	1	1	逆时针
0	0	0	停止
1	1	0	停止

```
void Change_motor_direction(int direction)
{
    if(direction == 0)
    {
      //正转
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_SET);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    }
    if(direction == 1)
    {
      //反转
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_SET);
    }
    if(direction == 2)
    {
      //停止
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, GPIO_PIN_RESET);
    }
}
```

6.3 电机 PWM 控制

```
void Moteor_Speed_change(int falg)
{
    if(falg == 1)
    {
        //加速
        PWM_pulse += 100;
        if(PWM_pulse > 1000) PWM_pulse = 1000;
    }
    if(falg == 2)
    {
        //减速
        PWM_pulse -= 100;
        if(PWM_pulse <= 200) PWM_pulse = 200;
    }
    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, PWM_pulse);
}
```

7 实验八：PID 控制电机

7.1 PID 控制算法

```
int Set_RPM = 0;                                    // 设置的转速 (RPM) 使用PID来维持在这个转速
// PID Controler Variables
float Kp = 2.0;  // 比例增益 (需要根据实际情况调整)
float Ki = 0.2f; // 积分增益 (需要根据实际情况调整)
float Kd = 0.1f; // 微分增益 (需要根据实际情况调整)

float pid_error = 0.0f;
float pid_previous_error = 0.0f;
float pid_integral = 0.0f;
float pid_derivative = 0.0f;
float pid_output = 0.0f;
float pid_prev_prev_error = 0.0f; // 上上次误差 e(k-2)  <--- 新增变量

// PID 输出限制 (例如PWM占空比范围, 假设htim3的ARR为999, 则最大值为999)
// 您需要根据htim3的实际ARR值来设置PID_OUTPUT_MAX
const float PID_OUTPUT_MIN = 0.0f;
const float PID_OUTPUT_MAX = 999.0f; // 重要: 请根据htim3的ARR值修改
const float INTEGRAL_MIN = -500.0f;  // 积分项抗饱和下限 (需要调整)
const float INTEGRAL_MAX = 500.0f;   // 积分项抗饱和上限 (需要调整)

char rpm_input_buffer[5];     // 用于存储最多4位数字 + 空终止符
uint8_t rpm_digit_count = 0;  // 当前输入的RPM数字位数
uint8_t rpm_input_active = 0; // 0: 非RPM输入模式, 1: RPM输入模式
在定时器中断调用该函数
在定时器中断调用该函数
在定时器中断调用该函数
在定时器中断调用该函数

void Update_PID_Controller(void)
{
  // 1. 计算当前误差 e(k)
  // RPM 是由 HAL_TIM_IC_CaptureCallback 中断更新的当前电机转速
  pid_error = (float)Set_RPM - RPM; // pid_error 充当 e(k)

  // 2. 计算增量 delta_pid_output = Δu(k)
  // Δu(k) = Kp * [e(k) - e(k-1)] + Ki * e(k) + Kd * [e(k) - 2*e(k-1) + e(k-2)]
  float delta_pid_output = Kp * (pid_error - pid_previous_error) + \
                           Ki * pid_error + \
                           Kd * (pid_error - 2.0f * pid_previous_error + pid_prev_prev_error);

  // 3. 更新总输出 U(k) = U(k-1) + Δu(k)
  // pid_output 在这里代表 U(k-1), 然后更新为 U(k)
  pid_output += delta_pid_output;

  // 4. 限制PID总输出在有效范围内
  if (pid_output > PID_OUTPUT_MAX)
  {
    pid_output = PID_OUTPUT_MAX;
  }
  else if (pid_output < PID_OUTPUT_MIN)
  {
    pid_output = PID_OUTPUT_MIN;
  }

  // 5. 更新历史误差, 为下一次计算做准备
```

```
56    // e(k-2) = e(k-1)
57    pid_prev_prev_error = pid_previous_error;
58    // e(k-1) = e(k)
59    pid_previous_error = pid_error;
60
61    // 6. 将PID输出应用到电机PWM控制
62    if (Set_RPM > 0) // 只有在目标转速大于0时才更新PWM
63    {
64      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, (uint16_t)pid_output);
65    }
66    else // 如果目标转速为0，则停止电机 (PWM设为0)
67    {
68      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_1, 0);
69      pid_output = 0.0f; // 重置PID累积输出
70      // 重置历史误差，防止下次启动时因旧的误差值导致delta_pid_output过大
71      pid_previous_error = 0.0f;
72      pid_prev_prev_error = 0.0f;
73      // pid_error 此时也应接近0 (如果RPM也为0)
74    }
75  }
```