# 微电路3组程序接口使用说明

# 仓库地址：[PyConqueror-16/Sweeping](https://github.com/PyConqueror-16/Sweeping)

## LED

项目中共有四颗可供使用的空闲LED（LED0 ~ 1），为方便使用、扩展进行了统一的封装。目前四颗均为上拉连接低电平触发。

### 1. 硬件描述

首先我们需要对LED的硬件状态进行描述，定义结构体如下：

typedef struct {

GPIO\_TypeDef \*GPIOx; // GPIO 端口

uint16\_t GPIO\_Pin; // GPIO 引脚

LED\_ActiveMode ActiveMode; // LED 工作模式（主动低或主动高）

} LED\_HandleTypeDef;

其中工作模式如下：

typedef enum {

LED\_ACTIVE\_LOW = 0, // LED 在低电平时点亮（即连接到 VCC）

LED\_ACTIVE\_HIGH // LED 在高电平时点亮（即连接到 GND）

} LED\_ActiveMode;

#### 示例：LED0连接在PD端口0号引脚，低电平触发，则如下：

**LED\_HandleTypeDef LED0 = {**

**.GPIOx = GPIOD,**

**.GPIO\_Pin = GPIO\_PIN\_0,**

**.ActiveMode = LED\_ACTIVE\_LOW**

**};**

### LED初始化

**效果为根据描述的硬件连接匹配对应的电平输出。**

/\* LED 初始化函数 \*/

void LED\_Init(LED\_HandleTypeDef \*led) {

// 根据LED的工作模式（主动低或主动高）初始化LED

if (led->ActiveMode == LED\_ACTIVE\_LOW) {

// 如果是主动低模式，初始化时将LED设置为关闭（对应的GPIO引脚输出高电平）

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_SET);

} else {

// 如果是主动高模式，初始化时将LED设置为关闭（对应的GPIO引脚输出低电平）

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_RESET);

}

}

#### 示例：初始化LED0，则为：

**LED\_Init (&LED0);**

### 点亮LED

**效果为点亮LED。**

/\* 点亮LED \*/

void LED\_On(LED\_HandleTypeDef \*led) {

if (led->ActiveMode == LED\_ACTIVE\_LOW) {

// 如果是主动低模式，点亮LED需要将GPIO引脚输出低电平

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_RESET);

} else {

// 如果是主动高模式，点亮LED需要将GPIO引脚输出高电平

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_SET);

}

}

#### 示例：点亮LED0，则为：

**LED\_On (&LED0)**

### 熄灭LED

**效果为熄灭LED。**

/\* 熄灭LED \*/

void LED\_Off(LED\_HandleTypeDef \*led) {

if (led->ActiveMode == LED\_ACTIVE\_LOW) {

// 如果是主动低模式，熄灭LED需要将GPIO引脚输出高电平

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_SET);

} else {

// 如果是主动高模式，熄灭LED需要将GPIO引脚输出低电平

HAL\_GPIO\_WritePin(led->GPIOx, led->GPIO\_Pin, GPIO\_PIN\_RESET);

}

}

#### 示例：初始化LED0，则为：

**LED\_Off (&LED0);**

### 5.翻转LED

**效果为转换当前LED的亮灭状态。**

/\* 切换LED状态（开/关） \*/

void LED\_Toggle(LED\_HandleTypeDef \*led) {

// 切换LED状态（如果是开就关，反之亦然）

HAL\_GPIO\_TogglePin(led->GPIOx, led->GPIO\_Pin);

}

#### 示例：初始化LED0，则为：

**LED\_Toggle (&LED0);**

## 按键

项目有两个可供使用的微动按键（KEY0、KEY1），均为上拉连接低电平触发，为GPIO的Input模式未使用中断。

### 1.硬件描述

硬件描述结构体如下：

typedef struct {

GPIO\_TypeDef \*GPIOx; // GPIO端口（如GPIOD）

uint16\_t GPIO\_Pin; // GPIO引脚（如GPIO\_PIN\_11）

Key\_ActiveMode ActiveMode; // 有效电平模式

Key\_State CurrentState; // 当前状态

uint32\_t LastTick; // 最后一次状态变化时的时间戳

} Key\_HandleTypeDef;

其中，按键状态又有如下枚举：

/\* 按键工作电平模式 \*/

typedef enum {

KEY\_ACTIVE\_LOW = 0, // 低电平有效（按下接GND）

KEY\_ACTIVE\_HIGH // 高电平有效（按下接VCC）

} Key\_ActiveMode;

/\* 按键状态 \*/

typedef enum {

KEY\_STATE\_RELEASED = 0, // 松开状态

KEY\_STATE\_PRESSED, // 按下状态

KEY\_STATE\_JUST\_PRESSED, // 刚刚按下（边沿）

KEY\_STATE\_JUST\_RELEASED // 刚刚松开（边沿）

} Key\_State;

#### 示例：直接描述两个按钮如下，均为低电平触发：

**Key\_HandleTypeDef key[] = {**

**{ // KEY0(PD11)**

**.GPIOx = GPIOD,**

**.GPIO\_Pin = GPIO\_PIN\_11,**

**.ActiveMode = KEY\_ACTIVE\_LOW**

**},**

**{ // KEY1(PE15)**

**.GPIOx = GPIOE,**

**.GPIO\_Pin = GPIO\_PIN\_15,**

**.ActiveMode = KEY\_ACTIVE\_LOW**

**}**

**};**

### 按键初始化

**效果为重置按键状态，并且更新时间戳。**

/\* 按键初始化 \*/

void Key\_Init(Key\_HandleTypeDef \*key) {

key->CurrentState = KEY\_STATE\_RELEASED;

key->LastTick = HAL\_GetTick();

}

**示例：初始化key0：**

**Key\_Init (&key[0]);**

### 按键更新

**效果为检测并更新按键的按下状态。**

/\* 按键更新（需要定期调用） \*/

void Key\_Update(Key\_HandleTypeDef \*key) {

uint32\_t current\_tick = HAL\_GetTick();

// 读取当前按键引脚状态

GPIO\_PinState pin\_state = HAL\_GPIO\_ReadPin(key->GPIOx, key->GPIO\_Pin);

uint8\_t is\_pressed = (key->ActiveMode == KEY\_ACTIVE\_LOW) ?

(pin\_state == GPIO\_PIN\_RESET) :

(pin\_state == GPIO\_PIN\_SET);

// 按键状态机

switch (key->CurrentState) {

case KEY\_STATE\_RELEASED:

if (is\_pressed) {

if (current\_tick - key->LastTick > DEBOUNCE\_DELAY\_MS) {

key->CurrentState = KEY\_STATE\_JUST\_PRESSED;

key->LastTick = current\_tick;

}

}

break;

case KEY\_STATE\_JUST\_PRESSED:

key->CurrentState = KEY\_STATE\_PRESSED;

break;

case KEY\_STATE\_PRESSED:

if (!is\_pressed) {

if (current\_tick - key->LastTick > DEBOUNCE\_DELAY\_MS) {

key->CurrentState = KEY\_STATE\_JUST\_RELEASED;

key->LastTick = current\_tick;

}

}

break;

case KEY\_STATE\_JUST\_RELEASED:

key->CurrentState = KEY\_STATE\_RELEASED;

break;

}

}

#### 示例：检测并更新key0：

**Key\_Update (key[0]);**

### 获取按键状态

**效果为配合按键状态更新函数，会返回Key\_State中的状态，进行逻辑编写。**

/\* 获取当前按键状态 \*/

Key\_State Key\_GetState(Key\_HandleTypeDef \*key) {

return key->CurrentState;

}

#### 示例：读取key1：

**Key\_State state = Key\_GetState(&key[1]);**

## IMU

模块连接在UART2上，采用中断的方式接收数据。共需要接收加速度、角速度、欧拉角三组共九个数据。

### IMU初始化

**功能为配置缓冲区并开启首次接收中断。**

void JY901S\_Init(UART\_HandleTypeDef \*huart)

{

jy\_uart = huart;

static uint8\_t rx\_buf[FRAME\_LEN];

HAL\_UART\_Receive\_IT(jy\_uart, rx\_buf, FRAME\_LEN); // 开启首次接收中断

}

#### 示例：初始化IMU：

**JY901S\_Init(&huart2);**

### IMU数据解算

**功能为根据官方文档解算IMU回传的数据。**

void JY901S\_UART\_RxHandler(uint8\_t \*data)

{

if (data[0] == 0x55) {

switch (data[1]) {

case 0x51: // 加速度

imu\_data.ax = (short)(data[3] << 8 | data[2]) / 32768.0f \* 16;

imu\_data.ay = (short)(data[5] << 8 | data[4]) / 32768.0f \* 16;

imu\_data.az = (short)(data[7] << 8 | data[6]) / 32768.0f \* 16;

break;

case 0x52: // 角速度

imu\_data.gx = (short)(data[3] << 8 | data[2]) / 32768.0f \* 2000;

imu\_data.gy = (short)(data[5] << 8 | data[4]) / 32768.0f \* 2000;

imu\_data.gz = (short)(data[7] << 8 | data[6]) / 32768.0f \* 2000;

break;

case 0x53: // 欧拉角

imu\_data.roll = (short)(data[3] << 8 | data[2]) / 32768.0f \* 180;

imu\_data.pitch = (short)(data[5] << 8 | data[4]) / 32768.0f \* 180;

imu\_data.yaw = (short)(data[7] << 8 | data[6]) / 32768.0f \* 180;

break;

default:

break;

}

}

// 重新开启下一次接收中断

HAL\_UART\_Receive\_IT(jy\_uart, data, FRAME\_LEN);

}

#### 示例：定义中断回调

**void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)**

**{**

**if (huart->Instance == USART2) {**

**JY901S\_UART\_RxHandler(jy\_rx\_buf);**

**HAL\_UART\_Receive\_IT(&huart2, jy\_rx\_buf, 11);**

**}**

**}**

### 获取IMU数据

IMU\_Data\_t\* JY901S\_GetData(void)

{

return &imu\_data;

}

#### 示例：在主循环中获取数据：

**imu = JY901S\_GetData();**

此处imu作全局变量定义（IMU\_Data\_t \*imu;），便于debug监视。

## 电机

采用两路PWM差值控制。使用方法直接，不给出示例。

### 电机初始化

void A4950\_Init(void)

{

// 启动左电机 PWM（TIM1\_CH1=PE9, TIM1\_CH2=PE11）

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_2);

// 启动右电机 PWM（TIM9\_CH1=PE5, TIM9\_CH2=PE6）

HAL\_TIM\_PWM\_Start(&htim9, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim9, TIM\_CHANNEL\_2);

}

### 左（右）轮驱动

根据CubeMX的设置，速度值绝对值上限为99

#### 左轮

void A4950\_SetLeft(int16\_t speed)

{

uint16\_t duty;

if (speed >= 0) {

duty = (speed > A4950\_PWM\_MAX ? A4950\_PWM\_MAX : speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_2, duty);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_1, 0);

} else {

duty = ((-speed) > A4950\_PWM\_MAX ? A4950\_PWM\_MAX : -speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_2, 0);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_1, duty);

}

}

#### 右轮

void A4950\_SetRight(int16\_t speed)

{

uint16\_t duty;

if (speed >= 0) {

duty = (speed > A4950\_PWM\_MAX ? A4950\_PWM\_MAX : speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_1, duty);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_2, 0);

} else {

duty = ((-speed) > A4950\_PWM\_MAX ? A4950\_PWM\_MAX : -speed);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_1, 0);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_2, duty);

}

}

### 制动

void A4950\_Brake(void)

{

// 两输入同时高 → 主动制动

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_1, A4950\_PWM\_MAX);

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_2, A4950\_PWM\_MAX);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_1, A4950\_PWM\_MAX);

\_\_HAL\_TIM\_SET\_COMPARE(&htim9, TIM\_CHANNEL\_2, A4950\_PWM\_MAX);

}

## 编码器

### 结构体关联句柄

typedef struct {

TIM\_HandleTypeDef \*htim; ///< 关联的定时器句柄（如 &htim2 或 &htim5）

int32\_t count; ///< 累计脉冲计数（带符号，可正可负）

} Encoder\_HandleTypeDef;

#### 示例：在主程序中

**Encoder\_HandleTypeDef encL, encR;**

### 编码器初始化

效果为关联定时器的编码器接口。

void Encoder\_Init(Encoder\_HandleTypeDef \*enc, TIM\_HandleTypeDef \*htim)

{

enc->htim = htim;

enc->count = 0;

// 启动 TIM 的 TI1 和 TI2 通道以进入编码器模式

HAL\_TIM\_Encoder\_Start(enc->htim, TIM\_CHANNEL\_1 | TIM\_CHANNEL\_2);

}

#### 示例：绑定左右编码器并启动对应的接口

#### Encoder\_Init(&encL, &htim2);

#### Encoder\_Init(&encR, &htim5);

### 获取计数

int32\_t Encoder\_GetCount(Encoder\_HandleTypeDef \*enc)

{

int16\_t raw = \_\_HAL\_TIM\_GET\_COUNTER(enc->htim);

// 计算与上次读取的差值（考虑 16 位上下溢出）

int32\_t delta = (int32\_t)raw - (int32\_t)(enc->count & 0xFFFF);

if (delta > 32767) delta -= 65536;

if (delta < -32768) delta += 65536;

enc->count += delta;

return enc->count;

}

#### 示例：

**leftCountGlobal = Encoder\_GetCount(&encL);**

**rightCountGlobal = Encoder\_GetCount(&encR);**

### 编码器清零

void Encoder\_Reset(Encoder\_HandleTypeDef \*enc)

{

\_\_HAL\_TIM\_SET\_COUNTER(enc->htim, 0);

enc->count = 0;

}

#### 示例：

**Encoder\_Reset(&encL);**

**Encoder\_Reset(&encR);**

## 悬崖传感器

共有四个悬崖传感器（0~3），使用ADC和DMA进行读取并TIM3定时触发（CubeMX中配置频率）。采用按位操作来记录是否为悬崖状态。

### 初始化

启动定时器3的定时触发以及DMA

void CliffSensor\_Init(void)

{

HAL\_TIM\_Base\_Start(&htim3); // 启动 TIM3 基本定时器

HAL\_ADC\_Start\_DMA(&hadc1, (uint32\_t\*)adc\_values, CLIFF\_SENSOR\_COUNT); //启动 ADC + DMA，读取 CLIFF\_SENSOR\_COUNT 个通道

}

### 读取四个传感器值

void CliffSensor\_GetValues(uint16\_t \*out)

{

for (int i = 0; i < CLIFF\_SENSOR\_COUNT; i++)

out[i] = adc\_values[i];

}

#### 示例：

**CliffSensor\_GetValues(sensor\_vals);**

### 判断单个传感器的状态

bool CliffSensor\_IsCliff(uint8\_t idx)

{

if (idx >= CLIFF\_SENSOR\_COUNT) return false; // 越界保护

return (adc\_values[idx] < cliff\_thresholds[idx]);

}

传入对应传感器编号即可。

### 用掩码返回所有传感器状态

uint8\_t CliffSensor\_GetMask(void)

{

uint8\_t mask = 0;

if (CliffSensor\_IsCliff(0)) mask |= CLIFF\_1;

if (CliffSensor\_IsCliff(1)) mask |= CLIFF\_2;

if (CliffSensor\_IsCliff(2)) mask |= CLIFF\_3;

if (CliffSensor\_IsCliff(3)) mask |= CLIFF\_4;

return mask;

}

#### 示例：

**if (mask == 0) {**

**LED\_Toggle(&LED0);**

**LED\_Toggle(&LED1);**

**LED\_Toggle(&LED2);**

**LED\_Toggle(&LED3);**

**HAL\_Delay(200);**

**} else {**

**if (mask & CLIFF\_1) LED\_On(&LED0);**

**if (mask & CLIFF\_2) LED\_On(&LED1);**

**if (mask & CLIFF\_3) LED\_On(&LED2);**

**if (mask & CLIFF\_4) LED\_On(&LED3);**

**}**

## 碰撞传感器

无特别说明