Application Note

ATK-MO301 模块使用说明

高性能电容半导体指纹识别模块

使用说明

正点原子

广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.0	2022/06/25	第一次发布
V1.1	2023/03/07	新增阿波罗 F429 与 F767 硬件连接描述

目 录

1，硬件连接

1.1 正点原子 MiniSTM32F103 开发板

ATK-MO301 模块可通过串口直接与正点原子 MiniSTM32F103 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
MiniSTM32F103 开发板	3.3V	-	3.3V	PD2	PC12	GND

表 1.1.1 ATK-MO301 模块与 MiniSTM32F103 开发板连接关系

1.2 正点原子精英 STM32F103 开发板

ATK-MO301 模块可通过串口直接与正点原子精英 STM32F103 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
精英 STM32F103 开发板	3.3V	-	3.3V	PB11	PB10	GND

表 1.2.1 ATK-MO301 模块与精英 STM32F103 开发板连接关系

1.3 正点原子战舰 STM32F103 开发板

ATK-MO301 模块可通过串口直接与正点原子战舰 STM32F103 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
战舰 STM32F103 开发板	3.3V	-	3.3V	PB11	PB10	GND

表 1.3.1 ATK-MO301 模块与战舰 STM32F103 开发板连接关系

1.4 正点原子探索者 STM32F407 开发板

ATK-MO301 模块可通过串口直接与正点原子探索者 STM32F407 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
探索者 STM32F407 开发板	3.3V	-	3.3V	PB11	PB10	GND

表 1.4.1 ATK-MO301 模块与探索者 STM32F407 开发板连接关系

1.5 正点原子 F407 电机控制开发板

ATK-MO301 模块可通过串口直接与正点原子 F407 电机控制开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
F407 电机控制开发板	3.3V	-	3.3V	PC11	PC10	GND

表 1.5.1 ATK-MO301 模块与 F407 电机控制开发板连接关系

1.6 正点原子 MiniSTM32H750 开发板

ATK-MO301 模块可通过串口直接与正点原子 MiniSTM32H750 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
MiniSTM32H750 开发板	3.3V	-	3.3V	PA3	PA2	GND

表 1.6.1 ATK-MO301 模块与 MiniSTM32H750 开发板连接关系

1.7 正点原子阿波罗 STM32F429 开发板

ATK-MO301 模块可通过串口直接与正点原子阿波罗 STM32F429 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
阿波罗 STM32F429 开发板	3.3V	-	3.3V	PA3	PA2	GND

表 1.7.1 ATK-MO301 模块与阿波罗 STM32F429 开发板连接关系

1.8 正点原子阿波罗 STM32F767 开发板

ATK-MO301 模块可通过串口直接与正点原子阿波罗 STM32F767 开发板进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
ATK-MO301 模块	Vtouch	TouchInt	+3.3V	TXD	RXD	GND
阿波罗 STM32F767 开发板	3.3V	-	3.3V	PA3	PA2	GND

表 1.8.1 ATK-MO301 模块与阿波罗 STM32F767 开发板连接关系

2，实验功能

2.1 ATK-MO301 模块测试实验

2.1.1 功能说明

在本实验中，开发板主控芯片通过串口与 ATK-MO301 模块进行通讯，并在上电后自动配置 ATK-MO301 模块，并进行指纹验证和录入等操作。

2.1.2 源码解读

打开本实验的工程文件夹，能够在./Drivers/BSP 目录下看到 ATK_MO301 子文件夹，该文件夹中就包含了 ATK-MO301 模块的驱动文件，如下图所示：

```
./Drivers/BSP/ATK_MO301/
|-- atk_mo301.c
|-- atk_mo301.h
|-- atk_mo301_uart.c
`-- atk_mo301_uart.h
```

图 2.1.2.1 ATK-MO301 模块驱动代码

2.1.2.1 ATK-MO301 模块接口驱动

在图 2.1.2.1 中，atk_mo301_uart.c 和 atk_mo301_uart.h 是开发板与 ATK-MO301 模块通讯而使用的 UART 驱动文件，关于 UART 的驱动介绍，请查看正点原子各个开发板对应的开发指南中 UART 对应的章节。

值得一提的是，由于 ATK-MO301 模块通过 UART 发送给主控芯片的数据的长度是不固定的，因此主控芯片就无法直接通过接收到数据的长度来判断 ATK-MO301 模块传来的一帧数据是否完成。对于这种通过 UART 接收不定长数据的情况，可以通过 UART 总线是否空闲来判断一帧的传输是否完成，恰巧 STM32 的 UART 提供了总线空闲中断功能，因此可以开启 UART 的总线空闲中断，并在中断中做相应的处理，具体的实现过程可以查看 ATK-MO301 模块的模块接口驱动代码，这里不做过多的描述。

2.1.2.2 ATK-MO301 模块驱动

在图 2.1.2.1 中，atk_mo301.c 和 atk_mo301.h 是 ATK-MO301 模块的驱动文件，包含了 ATK-MO301 模块初始化、收发命令和大部分命令的封装函数。函数比较多，下面仅介绍几个重要的 API 函数。

1. 函数 atk_mo301_init()

该函数用于初始化 ATK-MO301 模块，具体的代码，如下所示：

```
/**
 * @brief       ATK-MO301 模块初始化
 * @param       baudrate: ATK-MO301 模块 UART 接口通讯波特率
 * @retval      ATK_MO301_EOK  : ATK-MO301 模块初始化成功
 *              ATK_MO301_ERROR: ATK-MO301 模块初始化失败
 */
uint8_t atk_mo301_init(uint32_t baudrate)
{
```

```
    uint8_t ret;

    atk_mo301_uart_init(baudrate);  /* UART 接口初始化 */
    atk_mo301_cancel();             /* 取消命令 */
    ret = atk_mo301_shake();        /* 握手检测 */

    if (ret != ATK_MO301_EOK)
    {
        return ATK_MO301_ERROR;
    }

    return ATK_MO301_EOK;
}
```

从上面的代码中可以看出，函数 atk_mo301_init()会初始化与 ATK-MO301 模块通讯的 UART 接口，然后通过握手指令检测与 ATK-MO301 模块的通讯是否正常，而“取消命令”测试为了确保 ATK-MO301 模块退出自动录入和验证命令，这样才能够正常响应命令。

2. 函数 atk_mo301_send_cmd_pack()

该函数主要实现了将指令码和参数信息按照 ATK-MO301 模块的通讯协议打包发送，并解析 ATK-MO301 模块的响应包。

```
/**
 * @brief   发送命令包，并接收应答包
 * @param   cmd     : 指令码
 *          arg     : 参数
 *          arg_len : 参数长度
 *          conf    : 应答包的确认码
 *          ack     : 应答包的参数
 *          ack_len : 应答包参数的长度
 *          timeout : 等待相应包的超时时间
 * @retval  ATK_MO301_EOK       : 命令包发送成功
 *          ATK_MO301_ERROR     : 应答包有误
 *          ATK_MO301_ETIMEOUT  : 等待应答包超时
 *          ATK_MO301_EINVAL    : 函数参数有误
 */
static uint8_t atk_mo301_send_cmd_pack(
    uint8_t cmd,
    uint8_t *arg,
    uint16_t arg_len,
    uint8_t *conf,
    uint8_t *ack,
    uint16_t *ack_len,
    uint32_t timeout)
{
    static atk_mo301_pack_t send = {0};
    static atk_mo301_pack_t recv = {0};
```

```
    uint16_t arg_index;
    uint16_t checksum;
    uint8_t *buf;
    uint16_t buf_len;
    uint16_t buf_index;

    if (arg_len > ATK_MO301_PACK_ARG_LEN)
    {
        return ATK_MO301_EINVAL;
    }

    if ((arg_len > 0) && (arg == NULL))
    {
        return ATK_MO301_EINVAL;
    }

    /* 包头 */
    send.head[0] = (uint8_t)(ATK_MO301_PACK_HEAD >> 8) & 0xFF;
    send.head[1] = (uint8_t)ATK_MO301_PACK_HEAD & 0xFF;
    /* 芯片地址 */
    send.addr[0] = (uint8_t)(ATK_MO301_PACK_ADDR >> 24) & 0xFF;
    send.addr[1] = (uint8_t)(ATK_MO301_PACK_ADDR >> 16) & 0xFF;
    send.addr[2] = (uint8_t)(ATK_MO301_PACK_ADDR >> 8) & 0xFF;
    send.addr[3] = (uint8_t)ATK_MO301_PACK_ADDR & 0xFF;
    /* 包标识 */
    send.id[0] = ATK_MO301_PACK_ID_CMD;
    /* 包长度 */
    send.len[0] = (uint8_t)((ATK_MO301_PACK_CODE_LEN + arg_len +
                             ATK_MO301_PACK_CHECKSUM_LEN) >> 8) & 0xFF;
    send.len[1] = (uint8_t)(ATK_MO301_PACK_CODE_LEN + arg_len +
                             ATK_MO301_PACK_CHECKSUM_LEN) & 0xFF;
    /* 指令 */
    send.code[0] = cmd;
    /* 参数 */
    send.arg.len = arg_len;
    if (send.arg.len != 0)
    {
        for (arg_index=0; arg_index<send.arg.len; arg_index++)
        {
            send.arg.arg[arg_index] = arg[arg_index];
        }
    }
    /* 校验和 */
    checksum = atk_mo301_get_checksum(&send);
```

```
    send.checksum[0] = (uint8_t)(checksum >> 8) & 0xFF;
    send.checksum[1] = (uint8_t)checksum & 0xFF;

    /* 发送命令包 */
    atk_mo301_uart_rx_restart();
    atk_mo301_uart_send(send.head, ATK_MO301_PACK_HEAD_LEN);
    atk_mo301_uart_send(send.addr, ATK_MO301_PACK_ADDR_LEN);
    atk_mo301_uart_send(send.id, ATK_MO301_PACK_ID_LEN);
    atk_mo301_uart_send(send.len, ATK_MO301_PACK_LENGTH_LEN);
    atk_mo301_uart_send(send.code, ATK_MO301_PACK_CODE_LEN);
    atk_mo301_uart_send(send.arg.arg, send.arg.len);
    atk_mo301_uart_send(send.checksum, ATK_MO301_PACK_CHECKSUM_LEN);

    /* 接收应答包 */
    while (timeout > 0)
    {
        buf = atk_mo301_uart_rx_get_frame();
        buf_len = atk_mo301_uart_rx_get_frame_len();
        if (buf != NULL)
        {
            /* 查找包头 */
            for (buf_index=0; buf_index<buf_len-1; buf_index++)
            {
                if ((buf[buf_index] ==
                        ((uint8_t)(ATK_MO301_PACK_HEAD >> 8) & 0xFF)) &&
                    (buf[buf_index + 1] ==
                        ((uint8_t)ATK_MO301_PACK_HEAD & 0xFF)))
                {
                    break;
                }
            }

            /* 找到包头 */
            if (buf_index < buf_len - 1)
            {
                /* 包头 */
                recv.head[0] = buf[buf_index + 0];
                recv.head[1] = buf[buf_index + 1];
                buf_index += ATK_MO301_PACK_HEAD_LEN;
                /* 芯片地址 */
                recv.addr[0] = buf[buf_index + 0];
                recv.addr[1] = buf[buf_index + 1];
                recv.addr[2] = buf[buf_index + 2];
                recv.addr[3] = buf[buf_index + 3];
```

```
    buf_index += ATK_MO301_PACK_ADDR_LEN;
    /* 包标识 */
    recv.id[0] = buf[buf_index + 0];
    buf_index += ATK_MO301_PACK_ID_LEN;
    /* 包长度 */
    recv.len[0] = buf[buf_index + 0];
    recv.len[1] = buf[buf_index + 1];
    buf_index += ATK_MO301_PACK_LENGTH_LEN;
    /* 确认码 */
    recv.code[0] = buf[buf_index + 0];
    buf_index += ATK_MO301_PACK_CODE_LEN;
    /* 参数 */
    recv.arg.len = ((uint16_t)(recv.len[0] << 8) | recv.len[1]) -
                   ATK_MO301_PACK_CODE_LEN -
                   ATK_MO301_PACK_CHECKSUM_LEN;
    for (arg_index=0; arg_index<recv.arg.len; arg_index++)
    {
        recv.arg.arg[arg_index] = buf[buf_index + arg_index];
    }
    buf_index += recv.arg.len;
    /* 校验和 */
    recv.checksum[0] = buf[buf_index + 0];
    recv.checksum[1] = buf[buf_index + 1];

    /* 检查校验和 */
    checksum = atk_mo301_get_checksum(&recv);
    if (checksum !=
        ((uint16_t)(recv.checksum[0] << 8) | recv.checksum[1]))
    {
        return ATK_MO301_ERROR;
    }

    /* 确认码 */
    if (conf != NULL)
    {
        *conf = recv.code[0];
    }

    /* 参数 */
    if (ack != NULL)
    {
        for (arg_index=0; arg_index<recv.arg.len; arg_index++)
        {
            ack[arg_index] = recv.arg.arg[arg_index];
```

```
                    }
                }

                /* 参数长度 */
                if (ack_len != NULL)
                {
                    *ack_len = recv.arg.len;
                }

                return ATK_MO301_EOK;
            }

            atk_mo301_uart_rx_restart();
        }
        timeout--;
        delay_ms(1);
    }

    return ATK_MO301_ETIMEOUT;
}
```

从上面的代码中可以看出，函数 atk_mo301_send_cmd_pack()函数会将待发送的命令打包发送给 ATK-MO301 模块，然后等待并解析 ATK-MO301 模块的响应数据包，驱动代码文件中的绝大多数函数都是基于该函数实现的。

2.1.2.3 实验测试代码

实验的测试代码为文件 demo.c，在工程目录下的 User 子目录中。测试代码的入口函数为 demo_run()，具体的代码，如下所示：

```
/**
 * @brief   例程演示入口函数
 * @param   无
 * @retval  无
 */
void demo_run(void)
{
    uint8_t ret;
    uint8_t key;
    uint16_t enroll_id = 0;
    uint16_t detect_id;
    uint16_t score;

    /* 初始化 ATK-MO301 模块 */
    ret = atk_mo301_init(57600);
    if (ret != 0)
    {
        printf("ATK-MO301 init failed!\r\n");
```

```
        while (1)
        {
            LED0_TOGGLE();
            delay_ms(200);
        }
    }

    printf("ATK-MO301 config succedded!\r\n");

    while (1)
    {
        key = key_scan(0);

        switch (key)
        {
            case KEY0_PRES:
            {
                /* 录入指纹 */
                ret = demo_enroll(enroll_id);
                if (ret == 0)
                {
                    printf("Enroll succedded! ID: %d\r\n", enroll_id);
                    enroll_id++;
                }
                else
                {
                    printf("Enroll failed!\r\n");
                }
                    break;
            }
            case KEY1_PRES:
            {
                /* 清空指纹库 */
                ret = atk_mo301_empty();
                if (ret == 0)
                {
                    printf("Empty succedded!\r\n");
                }
                else
                {
                    printf("Empty failed!\r\n");
                }
                break;
            }
```

```
            default:
            {
                break;
            }
        }

        /* 验证指纹 */
        ret = demo_detect(&detect_id, &score);
        if (ret == 0)
        {
            printf("Detect succedded! ID: %d, Score: %d\r\n", detect_id, score);
        }
        else if (ret == 1)
        {
            printf("Detect failed!\r\n");
        }
    }
}
```

从上面的代码中可以看出，整个实验代码的逻辑还是比较简单的，除了不断调用函数 demo_detect()进行指纹验证，就是当按键 0 按下的时候，调用函数 demo_enroll()进行指纹录入，还有当按下按键 1 时，调用驱动文件中的函数 atk_mo301_empty()清空指纹库。

验证指纹的函数 demo_detect()，如下所示：

```
/**
 * @brief   指纹验证
 * @param   id      : 指纹 ID
 *          score   : 分数
 * @retval  0: 指纹验证成功
 *          1: 指纹验证失败
 *          2: 手指未按下
 */
static uint8_t demo_detect(uint16_t *id, uint16_t *score)
{
#if 0
    /* 自动验证指纹
     * 注意：该自动验证指纹 Demo 要求在 300 毫秒内完成指纹验证流程（包括等待手指按下）
     */
    uint8_t ret;

    ret = atk_mo301_autoidentify(0xFFFF, 300, id, score);
    if (ret == ATK_MO301_EOK)
    {
        return 0;
    }
```

```
    return 1;
#else
    /* 分布式验证指纹
     * 注意：该分布式验证指纹 Demo 没有等待手指按下的机制
     */
    uint8_t ret;

    ret = atk_mo301_getimage();
    if (ret == ATK_MO301_EOK)
    {
        delay_ms(100);
        ret = atk_mo301_genchar(ATK_MO301_CHARBUF_1);
        if (ret == ATK_MO301_EOK)
        {
            ret = atk_mo301_searchmb(  ATK_MO301_CHARBUF_1,
                                       0,
                                       ATK_MO301_MAX_PAGE,
                                       id,
                                       score);
            if (ret == ATK_MO301_EOK)
            {
                return 0;
            }
        }

        return 1;
        }

    return 2;
#endif
}
```

从上面的代码中可以看出，该函数通过宏开关分别提供了使用 ATK-MO301 模块自动验证指纹和分布式验证指纹两种指纹验证方式的示例，建议用户在实际使用时，根据实际的业务逻辑稍作修改。

录入指纹的函数 demo_enroll()，如下所示：

```
/**
 * @brief   录入指纹
 * @param   id: 指纹 ID
 * @retval  0: 指纹录入成功
 *          1: 指纹录入失败
 */
static uint8_t demo_enroll(uint16_t id)
{
#if 0
```

```
    /* 自动录入指纹
     * 注意：该自动录入指纹 Demo 要求手指在完成一次特征采集后离开并重新按下（需采集 4 次特征）
     *       整个过程需在 10000 毫秒内完成
     */
    uint8_t ret;

    ret = atk_mo301_autoenroll(id, 4, 10000);
    if (ret == ATK_MO301_EOK)
    {
        return 0;
    }
    atk_mo301_cancel();

    return 1;
#else
    /* 分布式录入指纹
     * 注意：该分布式录入指纹 Demo 没有等待手指按下和录入过程中要求手指离开的机制
     */
    uint8_t ret;
    uint8_t charbuff_id = ATK_MO301_CHARBUF_1;

    while (1)
    {
        ret = atk_mo301_getimage();
        if (ret == ATK_MO301_EOK)
        {
            delay_ms(100);
            ret = atk_mo301_genchar(charbuff_id);
            if (ret == ATK_MO301_EOK)
            {
                printf("Enroll, GenChar[%d/4]\r\n", charbuff_id);
                charbuff_id++;
                if (charbuff_id == 5)
                {
                    ret = atk_mo301_regmb();
                    if (ret == ATK_MO301_EOK)
                    {
                        ret = atk_mo301_stormb(ATK_MO301_CHARBUF_1, id);
                        if (ret == ATK_MO301_EOK)
                        {
                            return 0;
                        }
                    }
                }
```

```
            }
        }

        if (ret != ATK_MO301_EOK)
        {
            return 1;
        }
    }
#endif
}
```

从上面的代码中可以看出，该函数通过宏开关分别提供了使用 ATK-MO301 模块自动录入指纹和分布式录入指纹两种指纹录入方式的示例，建议用户在实际使用时，根据实际的业务逻辑稍作修改。

2.1.3 实验现象

将 ATK-MO301 模块按照第一节“硬件连接”中介绍的连接方式与开发板连接，并将实验代码编译烧录至开发板中，如果此时开发板连接 LCD，那么 LCD 显示的内容，如下图所示：

图 2.1.3.1 LCD 显示内容一

同时，通过串口调试助手输出实验信息，如下图所示：

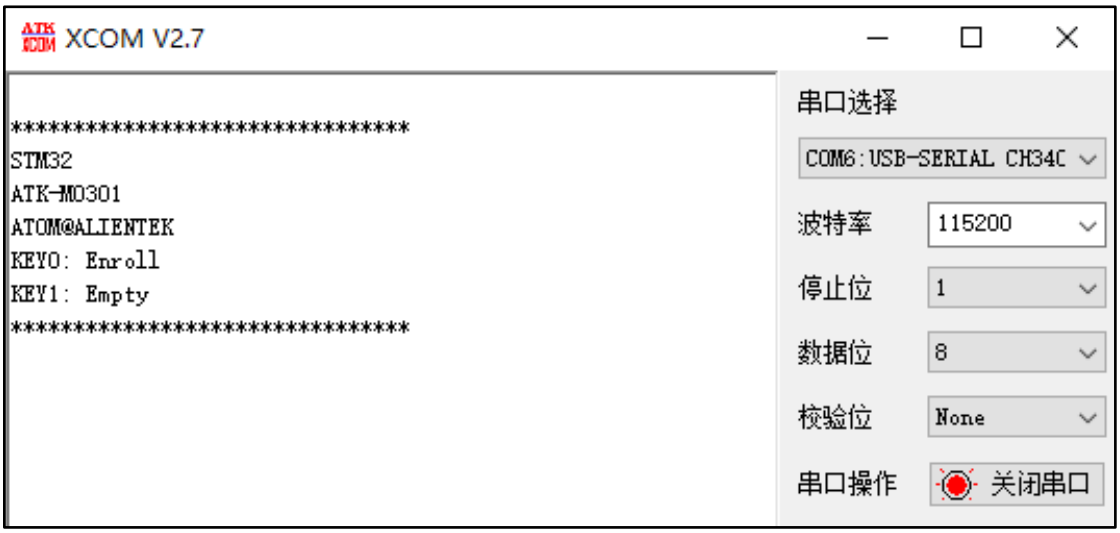

图 2.1.3.2 串口调试助手显示内容一

接下来程序会自动初始化与 ATK-MO301 模块的 UART 接口，并配置 ATK-MO301 模块，配置成功后，如下图所示：

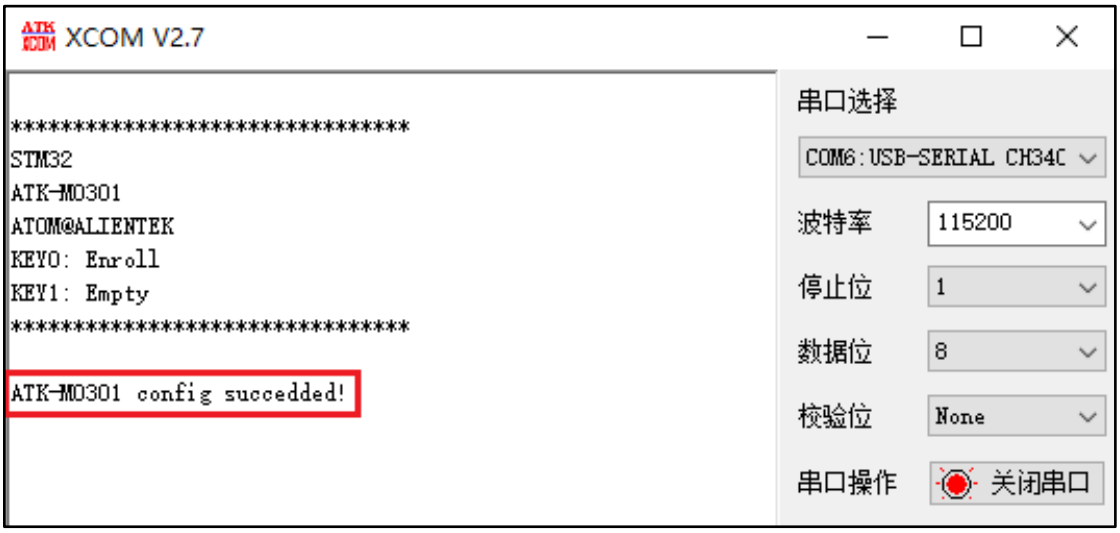

图 2.1.3.3 ATK-MO301 模块配置成功

接下来若将没有录入指纹的手指按在 ATK-MO301 模块上，则会提示验证失败，那么可以按下按键 0，进行指纹录入。在按下按键 0 之前，请先确保手指已经按在 ATK-MO301 模块上，录入成功，如下图所示：

图 2.1.3.4 ATK-MO301 模块指纹录入成功

如上图所示，在指纹录入的过程中，会 4 次采集指纹的特征，指纹录入成功后，会提示当前录入指纹的 ID。

此时，再将已经录入指纹的手指放在 ATK-MO301 模块上，则会提示验证成功，并提示验证匹配的指纹 ID 以及得分，如下图所示：

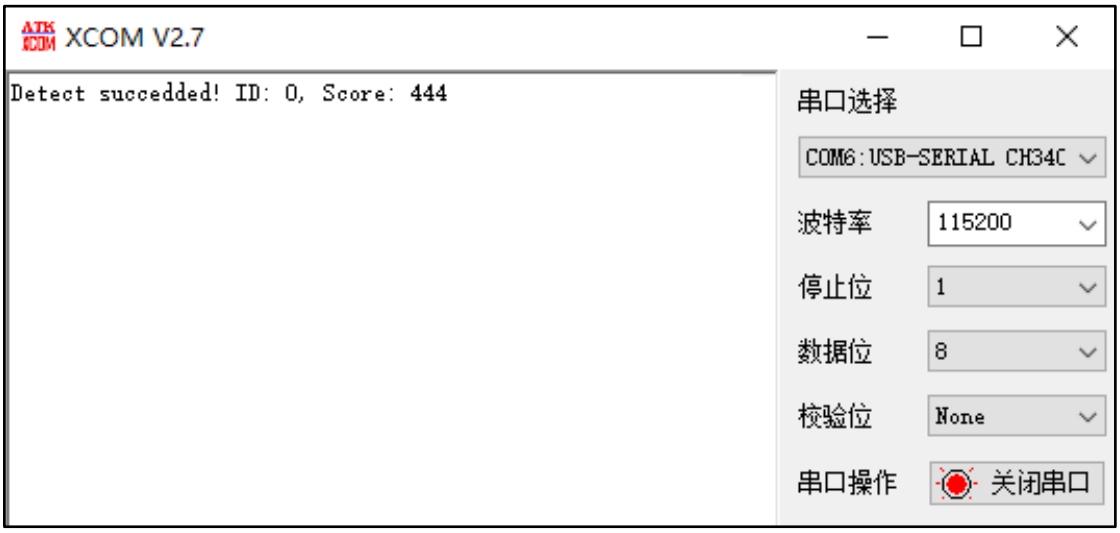

图 2.1.3.5 ATK-MO301 模块指纹验证成功

若按下按键 1，还可清空指纹库，清空指纹库后，之前已录入指纹的手指将验证失败。

3，其他

1、 购买地址：

天猫：https://zhengdianyuanzi.tmall.com

淘宝：https://openedv.taobao.com

2、 资料下载

模块资料下载地址：http://www.openedv.com/docs/modules/other/ATK-301.html

3、 技术支持

公司网址：www.alientek.com

技术论坛：http://www.openedv.com/forum.php

在线教学：www.yuanzige.com

B 站视频：https://space.bilibili.com/394620890

传真：020-36773971

电话：020-38271790

正点原子