

RM10x0 说明书



www.rockmong.com

上海岩獾科技有限公司
V1.5

目录

RM10x0 说明书	1
一、 产品特点.....	3
二、 产品选型.....	3
三、 主要参数.....	3
四、 接线说明.....	4
五、 驱动.....	4
六、 运行指示灯	4
七、 调试工具使用介绍.....	5
1. 工具总览.....	5
2. 数字 I/O	6
2.1 引脚模式介绍.....	6
2.2 内部电阻介绍.....	6
2.3 状态介绍.....	7
3. 修改上电状态.....	7
4. 修改 SN	8
八、 基础编程库函数介绍.....	8
1. 获取设备.....	8
2. 配置 IO 口输入输出方向	8
3. 读取输入状态.....	9
4. 控制输出状态.....	9
5. 读取输出状态.....	9
九、 高级编程——同时操作多个 IO.....	9
1. 多个 IO 同时初始化	9
2. 多个 IO 同时读	10
3. 多个 IO 同时写	10
4. 多个输出 IO 同时读取状态.....	11

一、产品特点

- USB 转 16 通道数字 IO，每路均可软件单独配置输入输出方向
- USB 通信速率：一次指令读写只需 0.5ms（具体受电脑环境等因素影响）
- 支持二次开发：无需关心底层实现，调用库函数即可，多线程、易移植、使用简单、高效
- 跨平台：支持 Windows、Linux、Android、Mac OS
- 提供主流语言示例：C/C++、C#、Python、Java、LabView 等
- 支持修改上电时输入输出状态
- USB 过流保护、接口 IO 瞬态抑制保护

二、产品选型

型号	IO 口高电平电压
RM1000	3.3V
RM1010	5V

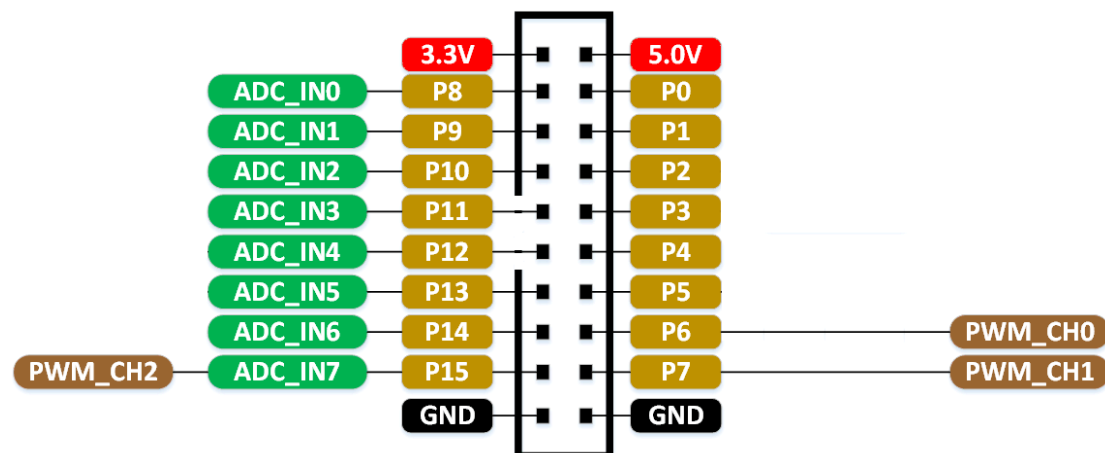
三、主要参数

参数	说明
电源额定电压	USB 供电（MAX 500mA）或外部 DC5V
输入输出	16 路数字 IO，每路可单独软件配置输入输出方向
输出驱动能力	最大 10mA； 一般用作信号传输控制，如需大电流驱动控制器请选择 RM16 系列。
通讯接口	USB
通讯速率	最大 2KHz
温度范围	工业级，-40℃~85℃

1. 5V 可输入也可输出。但 USB 供电最大 500mA，所以对外输出电流会小于 500mA。
2. 电源 3.3V 为输出。

四、 接线说明

4. 电源 5V 可输入也可输出。但 USB 供电最大 500mA，所以对外输出电流会小于 500mA。
5. 电源 3.3V 为输出。
6. 电源 GND，地线。外接传感器或设备用于共地。
7. P0~P15，16 路输入输出。每路可单独软件配置输入输出方向。输入模式下，可软件配置使能上拉、下拉电阻，用于某些用途（如按键/开关输入识别等）。

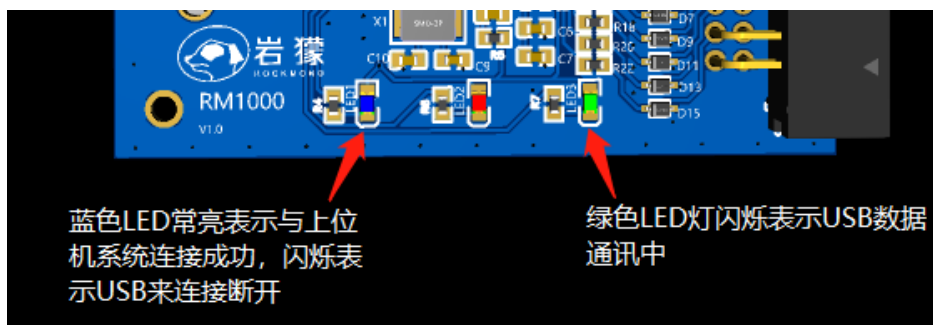


五、 驱动

Windows、Linux、Android、Mac OS 免安装驱动。（Win7 或更低版本需要安装）

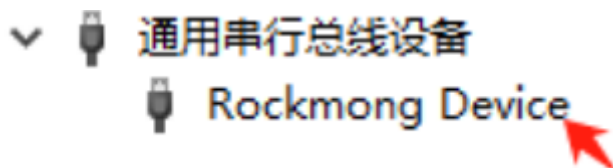
六、 运行指示灯

如下图运行指示灯。



如果蓝色 LED 是熄灭状态，表示 USB 未连接成功，请检查以下事项：

1. USB 线缆连接是否正常。
2. Windows 下需要检查 USB 驱动是否安装正常。其他系统是无驱设计，无需检查此项。



3. 工作电压是否正常。

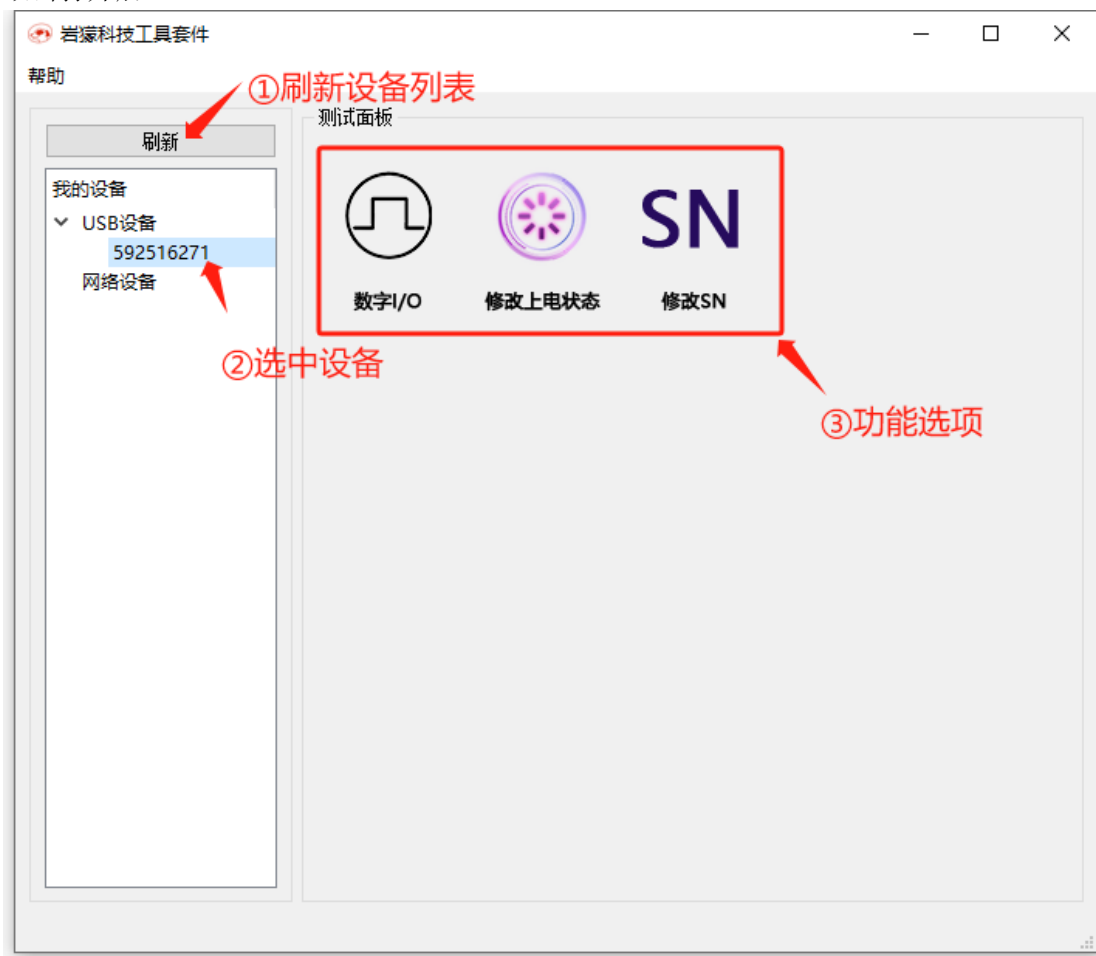
七、 调试工具使用介绍

调试工具图标：



1. 工具总览

双击打开后：



2. 数字 I/O

用来实时读写 IO。点击“数字 I/O”打开后：



2.1 引脚模式介绍

1. 输入模式：用来读取 IO 的电平状态。例如用来识别开关状态、读取传感器输出等等。
2. 输出模式：用来控制输出 IO 电平状态。例如控制三极管驱动继电器、灯泡等等。

2.2 内部电阻介绍

在输入模式下使用。除了一些接线用途，也能起到抗干扰的作用。

1. 上拉：该引脚上使能了一个内部上拉电阻。当此引脚未接负载时，就会一直高电平。例如下图一个开关，一端接 P0，一端接 GND。开关断开时，读取 P0 为高电平。开关导通时，读取 P0 为低电平。



2. 下拉：该引脚上使能了一个内部下拉电阻。当此引脚未接负载时，就会一直低电平。例如下图一个开关，一端接 P0，一端接 VCC。开关断开时，读取 P0 为低电平。开关导通时，读取 P0 为高电平。



2.3 状态介绍

1. 输入模式下：就是一个 IO 电平状态指示灯。黑色代表低电平，绿色代表高电平。
2. 输出模式下：就是一个 IO 电平控制开关按钮。黑色代表低电平，绿色代表高电平。点一下就会翻转电平。

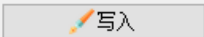
3. 修改上电状态

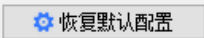
用来修改上电时，IO 的输入输出状态。模式、内部电阻、状态选项参照“数字 I/O”章节中的描述。修改好后，点击写入按钮。弹出写入成功弹框后，重新插拔 USB 线，即可生效。

上电状态修改 型号: RM1000 序号: 592516271

引脚名称	P0	P1	P2	P3	P4	P5	P6	P7
模式	输入	输入	输入	输入	输入	输入	输入	输入
内部电阻	上拉	上拉	上拉	上拉	上拉	上拉	上拉	上拉
状态	●	●	●	●	●	●	●	●

引脚名称	P8	P9	P10	P11	P12	P13	P14	P15
模式	输入	输入	输入	输入	输入	输入	输入	输入
内部电阻	上拉	上拉	上拉	上拉	上拉	上拉	上拉	上拉
状态	●	●	●	●	●	●	●	●

 写入

 恢复默认配置

4. 修改 SN

先填写新的 SN，长度范围 1~9 个数字。写好后点击写入按钮。提示成功后，回到主程序页面，点击刷新按钮刷新设备列表。



八、基础编程库函数介绍

需要使用到两个库 librockmong 和 libusb-1.0.

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考相关例程。

1. 获取设备

1. //扫描 USB 设备，获取设备序列号列表。
2. //返回值如果大于 0，代表获取到设备的个数。如果等于 0，代表未插入设备。如果小于 0，代表发生错误
3. `int UsbDevice_Scan(int* SerialNumbers);`

2. 配置 IO 口输入输出方向

1. //初始化引脚工作模式
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, P0. 1, P1...
4. //Mode: 输入输出模式。0, 输入。1, 输出。2, 开漏
5. //Pull: 上拉下拉电阻。0, 无。1, 使能内部上拉。2, 使能内部下拉
6. `int IO_InitPin(int SerialNumber, int Pin, int Mode, int Pull);`

3. 读取输入状态

```
1. //读取引脚状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, P0. 1, P1...
4. //PinState: 返回引脚状态。0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. int IO_ReadPin(int SerialNumber, int Pin, int *PinState);
```

4. 控制输出状态

```
1. //控制引脚输出状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, P0. 1, P1...
4. //PinState: 引脚状态。0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. int IO_WritePin(int SerialNumber, int Pin, int PinState);
```

5. 读取输出状态

```
1. //读取输出引脚状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, P0. 1, P1...
4. //PinState: 返回引脚状态。0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. int IO_ReadOutputPin(int SerialNumber, int Pin, int *PinState);
```

九、 高级编程——同时操作多个 IO

这里介绍 C 语言下的高级应用，其他语言类似，具体请参考相关例程。

1. 多个 IO 同时初始化

```
1. struct IO_InitStruct_Tx
2. {
3.     uint8_t Pin;
4.     uint8_t Mode;
5.     uint8_t Pull;
6. };
7. typedef struct IO_InitStruct_Tx IO_InitStruct_Tx_t;
8.
```

```
9. struct IO_InitStruct_Rx
10. {
11.     uint8_t Ret;
12. };
13. typedef struct IO_InitStruct_Rx IO_InitStruct_Rx_t;
14.
15. int IO_InitMultiPin(int SerialNumber, IO_InitStruct_Tx_t* TxStruct, IO_InitStruct_Rx_t* RxStruct, int Number);
```

2. 多个 IO 同时读

```
1. struct IO_ReadStruct_Tx
2. {
3.     uint8_t Pin;
4. };
5. typedef struct IO_ReadStruct_Tx IO_ReadStruct_Tx_t;
6.
7. struct IO_ReadStruct_Rx
8. {
9.     uint8_t Ret;
10.    uint8_t PinState;
11. };
12. typedef struct IO_ReadStruct_Rx IO_ReadStruct_Rx_t;
13.
14. int IO_ReadMultiPin(int SerialNumber, IO_ReadStruct_Tx_t* TxStruct, IO_ReadStruct_Rx_t* RxStruct, int Number);
```

3. 多个 IO 同时写

```
1. struct IO_WriteStruct_Tx
2. {
3.     uint8_t Pin;
4.     uint8_t PinState;
5. };
6. typedef struct IO_WriteStruct_Tx IO_WriteStruct_Tx_t;
7.
8. struct IO_WriteStruct_Rx
9. {
10.    uint8_t Ret;
11. };
12. typedef struct IO_WriteStruct_Rx IO_WriteStruct_Rx_t;
13.
```

```
14. int IO_WriteMultiPin(int SerialNumber, IO_WriteStruct_Tx_t* TxStruct, IO_WriteStruct_Rx_t* RxStruct, int Number);
```

4. 多个输出 IO 同时读取状态

```
1. struct IO_ReadOutput_TxStruct
2. {
3.     uint8_t Pin;
4. };
5. typedef struct IO_ReadOutput_TxStruct IO_ReadOutput_TxStruct_t;
6.
7. struct IO_ReadOutput_RxStruct
8. {
9.     uint8_t Ret;
10.    uint8_t PinState;
11. };
12. typedef struct IO_ReadOutput_RxStruct IO_ReadOutput_RxStruct_t;
13.
14. int IO_ReadMultiPin(int SerialNumber, IO_ReadStruct_Tx_t* TxStruct, IO_ReadStruct_Rx_t* RxStruct, int Number);
```