

RM56xx 说明书



www.rockmong.com

上海岩獾科技有限公司

V1.1

目录

RM56xx 说明书	1
一、 产品特点.....	3
二、 产品选型.....	3
三、 主要参数.....	3
四、 安装方式.....	4
五、 接线方式.....	4
六、 调试工具使用介绍.....	5
1. 工具总览双击打开后：	5
2. 数字 I/O	6
2.1 引脚模式介绍.....	6
2.2 状态介绍.....	6
3. 脉冲计数器.....	7
3.1 触发方式介绍.....	7
3.2 计数模式介绍.....	8
3.3 使能介绍.....	8
4. 修改上电状态.....	8
5. 修改 SN	9
七、 基础编程一介绍——单个 IO 操作.....	9
1. 获取设备.....	9
2. 读取输入状态.....	9
3. 控制输出状态.....	10
4. 读取输出状态.....	10
八、 基础编程二介绍——多个 IO 同时用 bit 操作	10
1. 读取输入状态.....	11
2. 控制输出状态.....	11
3. 读取输出状态.....	11
九、 其他编程介绍.....	12
1. 多个输入口同时读取.....	12
2. 多个输出口同时写入.....	13
3. 多个输出 IO 同时读取状态.....	14

一、产品特点

- 电源：DC 7-30V；
- 输入输出：多路光耦隔离数字输入；多路晶体管输出；
- 通讯接口：RJ45；
- 通信速率：局域网下一次读写最快只需 1.5ms（具体受电脑、网络环境等因素限制）；
- 通信协议：可以任选下面一种。
 - 1.支持库调用。无需关心底层协议实现，多线程安全，跨平台易移植，使用简单，高效；
 - 2.支持 Modbus TCP 协议。
 - 3.支持 MQTT。
- 跨平台：支持 Windows、Linux、Android、Mac OS；
- 提供各大语言例程：C/C++、C#、Python、Java、LabView 等等；
- 支持修改上电时输出状态；

二、产品选型

型号	输入输出
RM5604	4 入 4 出
RM5608	8 入 8 出
RM5616	16 入 16 出
需要更多通道请联系我们。 需要继电器（干接点）版本请选择 RM57 系列。	

三、主要参数

参数	说明
电源	额定电压 DC 7-30V 额定功率要大于 1W
输入	支持 NPN 和 PNP 电源 24V 时：逻辑高 18~24V，逻辑低 0~18V 电源 12V 时：逻辑高 6~12V，逻辑低 0~6V 电源 9V 时：逻辑高 6~9V，逻辑低 0~6V
输入指示	红色 LED 指示
晶体管输出	最大 30V 3A，NPN 型
输出指示	红色 LED 指示
通讯接口	RJ45 网线
通讯速率	最快 1.5ms
温度范围	工业级，-40℃~85℃

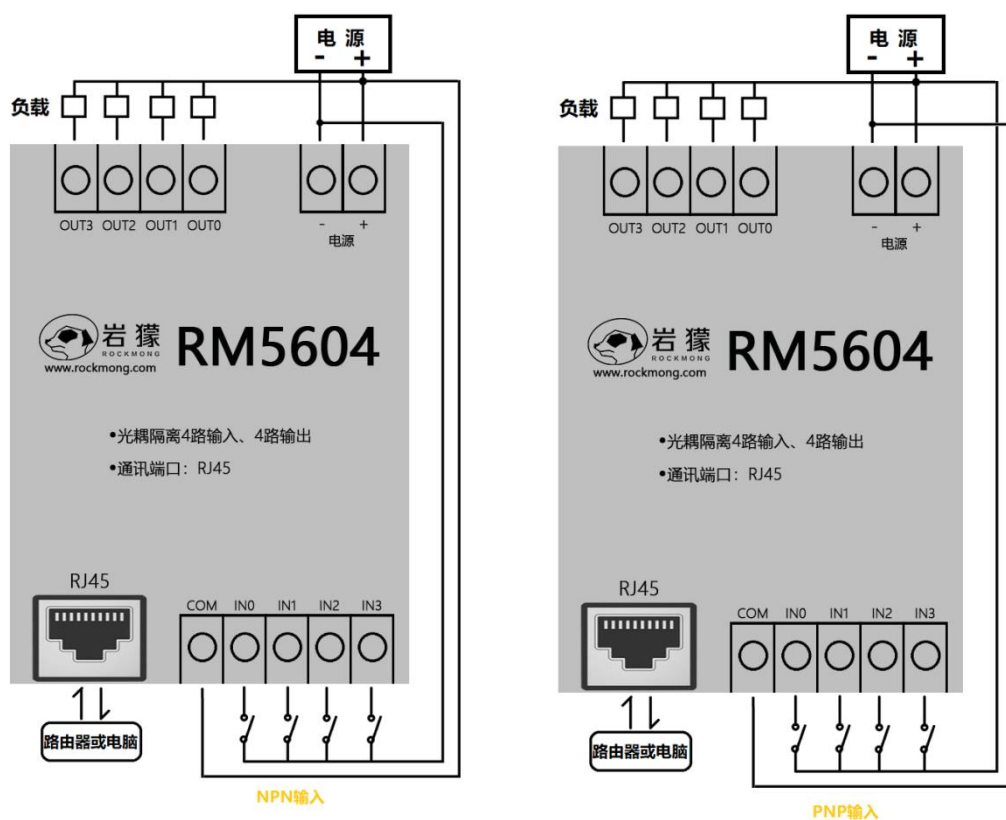
四、 安装方式

工控盒型、导轨、螺纹铜柱型。

五、 接线方式

下图以 RM5604，4 路输入 4 路输出为例。其他 RM56 型号类似。

1. 输入端可以接开关、NPN/PNP 输出传感器、PLC 等等，采用共阴极或共阳极接法。
2. 输出端的负载可以是继电器、PLC、三色灯等等，采用共阳极接法。



六、 调试工具使用介绍

调试工具下载链接见本文后面的“资料下载链接汇总”章节。



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。例如控制灯泡、驱动继电器、三色灯等等。

2.2 状态介绍

1. 输入端：IO 电平状态指示灯。黑色代表低电平，绿色代表高电平。
2. 输出端：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. 脉冲计数器

脉冲计数：



3.1 触发方式介绍

上升沿：低电平到高电平的过程。如下图：



下降沿：高电平到低电平的过程。如下图：



双边沿：上升沿和下降沿。如下图：



3.2 计数模式介绍

加计数：触发发生时，计数值加一。

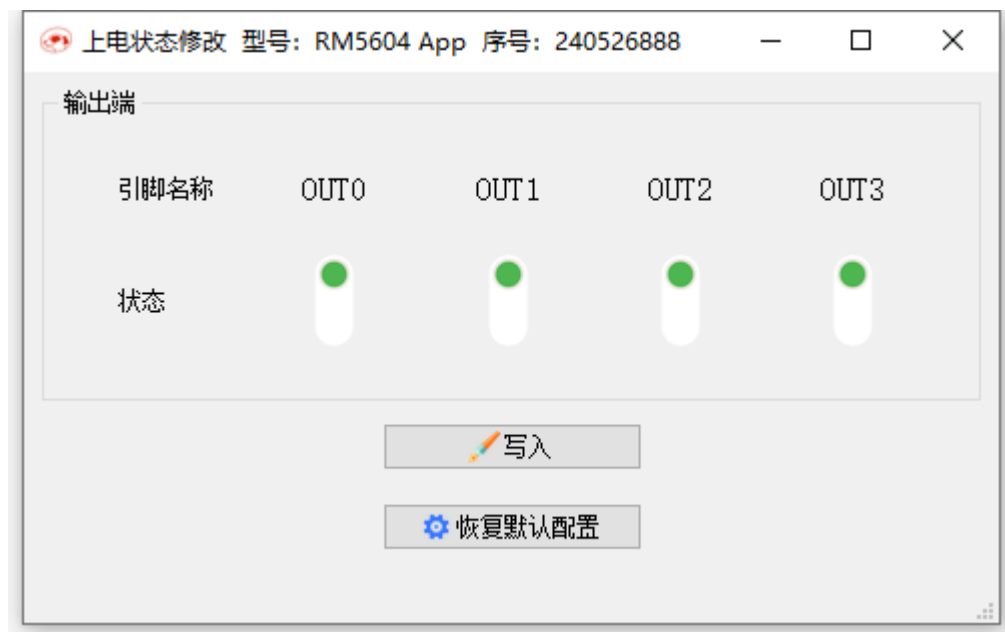
减计数：触发发生时，计数值减一。

3.3 使能介绍

勾选后，使能对应通道计数器开始计数。

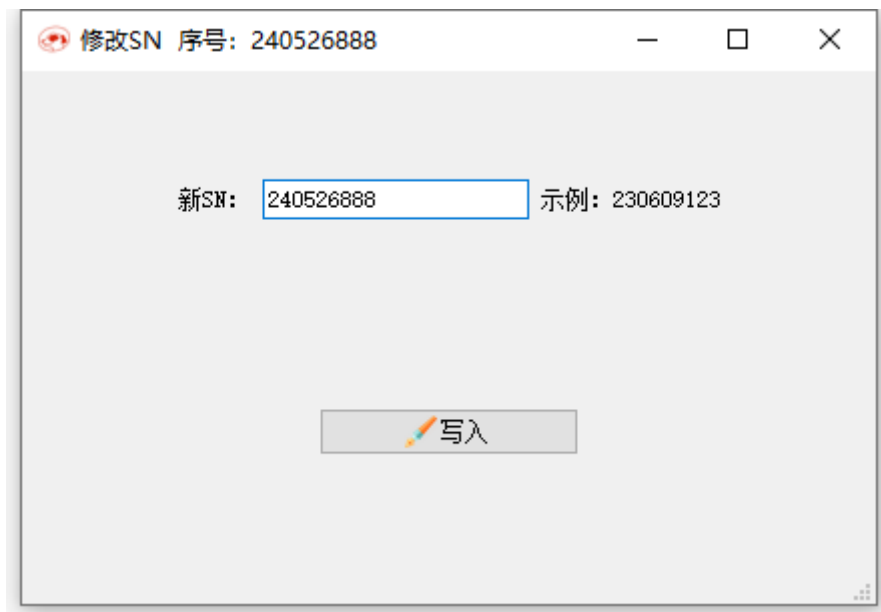
4. 修改上电状态

用来修改上电时，IO 的输出状态。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，电源重新上电后，即可生效。



5. 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。



七、 基础编程一介绍——单个 IO 操作

需要使用到两个库 librockmong

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考各语言下的 Demo_IO 例程。

1. 获取设备

```
1. // 扫描设备，获取设备序列号列表
2. // SerialNumbers: 序列号数组。传递 int 类型数组空间。运行后返回序列号
3. // DeviceType: 设备类型。0x01: USB 设备。0x02: 网络设备。0xFF: 所有类型设备。
4. // 返回值如果大于 0，代表获取到设备的个数。如果等于 0，代表未连接设备。如果小于 0，代表发生错误
5. int Device_Scan(int* SerialNumbers, int DeviceType);
```

2. 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`, `IN0`. `1`, `IN1`...
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);`

3. 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`, `OUT0`. `1`, `OUT1`...
4. `//PinState`: 引脚状态。`0`, 晶体管导通。`1`, 晶体管断开
5. `//函数返回`: `0`, 正常; `<0`, 异常
6. `int IO_WritePin(int SerialNumber, int Pin, int PinState);`
- 7.

4. 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);`

八、 基础编程二介绍——多个 IO 同时用 bit 操作

这里只介绍 C 语言下的库函数, 其他语言类似, 具体请参考各语言下的 Demo_IO_Bit 例程。



1. 读取输入状态

1. //同时读取所有输入引脚状态
2. //SerialNumber: 设备序号
3. //PinState: 返回引脚状态。每一个 bit 代表一个 IO。如 bit0 为 IN0, bit1 为 IN1, 以此类推
4. // 相应 bit 为 0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. `int IO_ReadPin_Bit(int SerialNumber, int* PinState);`

2. 控制输出状态

1. //同时控制所有引脚输出状态
2. //SerialNumber: 设备序号
3. //PinState: 写入引脚状态。每一个 bit 代表一个 IO。如 bit0 为 OUT0, bit1 为 OUT1, 以此类推
4. // 相应 bit 为 0, 继电器断开 (晶体管导通)。1, 继电器吸合 (晶体管断开)
5. //函数返回: 0, 正常; <0, 异常
6. `int IO_WritePin_Bit(int SerialNumber, int PinState);`

3. 读取输出状态

1. //同时读取所有输出引脚状态
2. //SerialNumber: 设备序号
3. //PinState: 返回引脚状态。每一个 bit 代表一个 IO。如 bit0 为 OUT0, bit1 为 OUT1, 以此类推
4. // 相应 bit 为 0, 继电器断开 (晶体管导通)。1, 继电器吸合 (晶体管断开)

5. `//函数返回: 0, 正常; <0, 异常`
6. `int IO_ReadOutputPin_Bit(int SerialNumber, int* PinState);`

九、 其他编程介绍

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考各语言下的 Demo_IO_Multi 例程。

1. 多个输入口同时读取

1. `typedef struct`
2. `{`
3. `uint8_t Pin; //引脚编号`
4. `}IO_Read_Struct_Tx_t;`
- 5.
6. `typedef struct`
7. `{`
8. `uint8_t Ret; //返回: 0, 正常; <0, 异常`
9. `uint8_t PinState; //引脚状态`
10. `}IO_Read_Struct_Rx_t;`
- 11.
12. `//同时读取多个输入口状态`
13. `//SerialNumber: 设备序号`
14. `//TxStruct: 发送数据结构体指针`
15. `//RxStruct: 接收数据结构体指针`



16. `//Number`: 结构体的个数
17. `//函数返回`: 0, 全部正常; <0, 存在异常
18. `int` IO_ReadMultiPin(`int` SerialNumber, IO_Read_Struct_Tx_t* TxStruct, IO_Read_Struct_Rx_t* RxStruct, `int` Number);

2. 多个输出口同时写入

1. `typedef struct`
2. {
3. `uint8_t` Pin; `//引脚编号`
4. `uint8_t` PinState; `//引脚状态`
5. }IO_Write_Struct_Tx_t;
- 6.
7. `typedef struct`
8. {
9. `uint8_t` Ret; `//返回: 0, 正常; <0, 异常`
10. }IO_Write_Struct_Rx_t;
- 11.
12. `//同时写入多个输出口状态`
13. `//SerialNumber`: 设备序号
14. `//TxStruct`: 发送数据结构体指针
15. `//RxStruct`: 接收数据结构体指针
16. `//Number`: 结构体的个量

17. //函数返回: 0, 全部正常; <0, 存在异常

```
18. int IO_WriteMultiPin(int SerialNumber, IO_Write_Struct_Tx_t* TxStruct, IO_Write_
    Struct_Rx_t* RxStruct, int Number);
```

3. 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. 
```

```
int IO_ReadMultiPin(int SerialNumber, IO_ReadStruct_Tx_t* TxStruct, IO_ReadStruct_Rx
```