USB HID 键盘模拟接口输出可配置只读读卡器

通用技术手册

（Revision 3.06）

北京金木雨电子有限公司

2025/3/17

在使用本产品前请详细阅读本说明书，如果有任何疑问，请联系我们，我们会给您详尽的解答

目录

1 简介

USB HID 键盘模拟接口输出可配置只读读卡器的 USB 接口采用键盘接口规范（HID），可以在 Windows、Linux 以及其他支持 USB 键盘的操作系统中模拟 USB 键盘的数据格式输出数据。

USB HID 键盘模拟接口输出可配置只读读卡器基于 NXP 射频芯片设计，可以读取符合 ISO14443A，ISO14443B，ISO15693 标准的卡片，并通过 USB 端口模拟键盘的方式输出，输出内容可由用户自行配置，输出可配置的内容丰富，输出数据支持十进制，十六进制，ASCII 等多种格式，支持数据正序，反序，字符调换，控制字符等多种操作。

产品用途：自动录入。

2 设备型号

MR762
MR762 B
MR7805
QM-ABCM7

3 设备识别

读卡器的 USB 通讯接口使用 HID 通信模式。在 Windows 系统中，连接读卡器到 PC 后，会自动安装驱动，并识别到两个设备：1，HID Keyboard Device；2，USB 输入设备(或者 HID-compliant device)。

4 通信协议

4.1 通信格式

4.1.1 数据包发送格式

- 主机发送：

长度字	通讯地址	命令字	数据域	校验字

- 长度字：2 字节，指明从长度字到数据域最后一字节的字节数，高字节在前，取值范围为：0x0004～0x01FE。
- 通讯地址：**未启用，默认 0x00**。
- 命令字：1 字节，通讯协议的应用层命令，详见 4.2 命令列表
- 数据域：数据长度由命令字决定，长度为 0 至 506 字节，根据所使用的处理器不同，部分型号会小于 506 字节。

- 校验字：1 字节，从长度字到数据域最后一字节的逐字节异或（XOR）值

4.1.2 数据包返回格式

- 成功返回：

长度字	通讯地址	命令字	数据域	校验字

- 失败返回：

长度字	通讯地址	命令字取反	校验字

失败返回的含义是指模块与卡片通讯失败。

通讯地址：**未启用，默认 0x00**。

4.2 命令列表

命令字	说明
0xE0	控制设备
0xE1	读取设备参数
0xE2	设置设备参数
0xE3	擦除脚本
0xE4	写入脚本
0xE5	启用脚本
0xF1	读设备版本信息

4.3 命令详述

4.3.1 控制设备

控制 LED 设备。注：刷卡会导致 LED 灯状态改变。

上位机发送：

帧头	0xE0	DATA	校验字

DATA: 1 字节；

	位	说明
DATA[0]	Bit5~2	RFU
	Bit1	1=ON; 0=OFF; GREEN LED;
	Bit0	1=ON; 0=OFF; RED LED;

模块回应成功：

帧头	0xE0	校验字

模块回应失败：

帧头	0x1F	校验字

4.3.2 读取设备参数

读取设备参数。

上位机发送：

帧头	0xE1	校验字

模块回应成功：

帧头	0xE1	DATA	校验字

DATA: 3 字节；

	位	说明
DATA[0]		按键速率，默认 0x10，单位 100uS(不严格时间)
DATA[1]	Bit7~2	RFU
	Bit1	0:第 2 套脚本已擦除（默认） 1:第 2 套脚本写入完成
	Bit0	0:第 1 套脚本已擦除（默认） 1:第 1 套脚本写入完成
DATA[2]	Bit7~1	RFU
	Bit0	0:第 1 套脚本为当前脚本；（默认） 1:第 2 套脚本为当前脚本；

注：MR762 不支持第 2 套脚本，DATA[1]和 DATA[2]无意义。

模块回应失败：

帧头	0x1E	校验字

4.3.3 设置设备参数

设定设备参数。

上位机发送：

帧头	0xE2	DATA	校验字

DATA: 3 字节；

	位	说明
DATA[0]		按键速率，默认 0x10，单位 1mS(不严格时间)
DATA[1]		脚本状态，只读，忽略当前设定值。
DATA[2]	Bit7~1	RFU
	Bit0	0:第 1 套脚本为当前脚本；（默认） 1:第 2 套脚本为当前脚本；

注：MR762 不支持第 2 套脚本，DATA[1]和 DATA[2]无意义。

模块回应成功：

帧头	0xE2	校验字

模块回应失败：

帧头	0x1D	校验字

4.3.4 擦除脚本

根据设备参数擦除第 1 或 2 套脚本数据。每次重新配置读卡器脚本时，必须擦除。脚本擦除后，在没有写入脚本之前，读卡器不能正常工作。

上位机发送：

帧头	0xE3	0xA5	0x01	校验字

模块回应成功：

帧头	0xE3	校验字

模块回应失败：

帧头	0x1C	校验字

4.3.5 写入脚本

根据设备参数写入第 1 或 2 套脚本命令，写入脚本命令是一个重复命令。每次写入，设备自动保存；如果写入错误，或想重新写，需要擦除后，重新开始写入脚本序列。

上位机发送：

帧头	0xE4	DATA	校验字

DATA: n 字节，详见第 5 章节脚本命令；

模块回应成功：

帧头	0xE4	校验字

模块回应失败：

帧头	0x1B	校验字

4.3.6 启用脚本

写入脚本后可以启用脚本，不需要重新启动设备。

上位机发送：

帧头	0xE5	0xA5	0x01	校验字

模块回应成功：

帧头	0xE5	校验字

模块回应失败：

帧头	0x1A	校验字

4.3.7 读设备版本信息

读取设备版本信息，版本信息按照 ASCII 编码。

上位机发送：

帧头	0xF1	校验字

模块回应成功：

帧头	0xF1	DATA	校验字

DATA: n 字节, 版本信息。

模块回应失败：

帧头	0x0E	校验字

5 脚本命令

5.1 脚本命令表

命令编号	功能
0x1X	SAM 卡操作命令
0x2X	CPU 卡操作命令
0x3X	输出操作命令
0x4X	RAM 操作命令
0x5X	MIFARE 卡操命令
0x6X	Ultralight 卡操作命令
0x7X	SR 系列卡
0x8X	ISO15693 卡操作指令
0x9X	身份证

5.2 SAM 卡操作

DATA[0]	DATA[1]	DATA[2~31]
CMD	卡座编号	APDU

CMD： 0x11 APDU 数据发送到 SAM 卡，命令结果放入 RAM1

0x12 APDU 数据发送到 SAM 卡，命令结果放入 RAM2

0x1A SAM 卡复位操作

DATA[2]表示复位波特率

0：9600bps；1：19200bps；2：38400 bps；3：55800 bps；

4：57600 bps；5：115200 bps；6：230400 bps，其他值：保留

DATA[3~31]无意义

DATA[1]：卡座编号

1，2，3……

对应读写器电路板上的卡座编号

APDU： COS 指令

示例：

说明	脚本	实际输出
SAM1 卡复位，波特率 9600	Send: 1A 01 00 Recv: 00	“将完成对 SAM1 复位操作”
输出复位信息	Send: 34 22 00 00 Recv: 00	SAM1 卡复位信息： 3B6C000243218638075442001 60E5940
对 SAM1 读取随机数	Send: 11 01 00 84 00 00 08 Recv: 00	“结果将存入 RAM1”
输出随机数，即 RAM1 中的前 10 字节数据	Send: 31 22 00 0A Recv: 00	11C057323AD5451E9000
输出回车符	Send: 30 30 58 Recv: 00	“输出回车换行符”
SAM2 卡复位，波特率 9600	Send: 1A 02 00 Recv: 00	“将完成对 SAM2 复位操作”
输出复位信息	Send: 34 22 00 00 Recv: 00	SAM2 卡复位信息： 3B17948065D00175F288
对 SAM2 读取随机数	Send: 11 02 00 84 00 00 08 Recv: 00	“结果将存入 RAM1”
输出随机数，即 RAM1 中的前 10 字节数据	Send: 31 22 00 0A Recv: 00	06602A62707712AD9000
输出回车符	Send: 30 30 58 Recv: 00	“输出回车换行符”

5.3 CPU 卡操作

DATA[0]	DATA[1~31]
CMD	APDU

CMD： 0x21 APDU 数据发送到 CPU 卡，命令结果放入 RAM1
0x22 APDU 数据发送到 CPU 卡，命令结果放入 RAM2
0x2A CPU-A 卡复位操作，无 APDU
0x2B EMV 读卡号操作，无 APDU

APDU： COS 指令

示例 1

说明	脚本	实际输出
CPU A 卡复位	Send: 2A Recv: 00	“将完成对卡片的复位操作”
输出复位信息	Send: 34 22 00 00 Recv: 00	CPU-A 卡复位信息： 1078809002209000000000002100B822
APDU 读取随机数	Send: 21 00 84 00 00 08 Recv: 00	“结果将存入 RAM1”
输出随机数，即 RAM1 中的数据	Send: 31 22 00 0A Recv: 00	45478466732A8B899000

示例 2：读 EMV 卡号

说明	脚本	实际输出
CPU A 卡复位	Send: 2A Recv: 00	“将完成对卡片的复位操作”
EMV 卡号读取	Send: 2B Recv: 00	“完成读取卡号操作”
输出卡号	Send: 33 50 00 00 Recv: 00	“以 ASCII 方式输出卡号”

5.4 输出操作

DATA[0]	DATA[1]	DATA[2]	DATA[3]
CMD	格式	Start/按键值	Length

CMD： 0x30 当前指令中的数据
0x31 RAM1 输出
0x32 RAM2 输出
0x33 卡号输出
0x34 卡复位信息输出

指令表：

<table>
<tr><th>DATA[0]</th><th colspan="3">DATA[1]</th><th>DATA[2]</th><th>DATA[3]</th></tr>
<tr><td rowspan="8">0x30：当前指令中的数据

0x31：RAM1 输出

0x32：RAM2 输出

0x33：卡号输出

0x34：卡复位信息输出</td><td>Bit7- Bit6-Bit5 -Bit4（输出方式）</td><td colspan="2">Bit3- Bit2- Bit1- Bit0</td><td></td><td></td></tr>
<tr><td rowspan="2">0001：十进制输出，默认前边的零输出

1001：十进制输出，前边的零不输出</td><td>Bit3</td><td>数据长度和组合长度不是整数倍，最后一个数据的补零方法
0：默认在后边补零
1：在前边补零</td><td rowspan="4">Start：输出数据的起始字节</td><td rowspan="4">Length：输出字节长度（使用默认长度，此字节为 0x00）</td></tr>
<tr><td>Bit2 ~ Bit0</td><td>转换形式
001：一个字节转化为十进制形式
010：二个字节转化为十进制形式
011：三个字节转化为十进制形式
100：四个字节转化为十进制形式
111：默认长度转化为十进制形式</td></tr>
<tr><td rowspan="4">0010：十六进制输出</td><td>Bit3</td><td>数据正序和反序控制
0: 默认正序
1: 反序</td></tr>
<tr><td>Bit2</td><td>RFU</td></tr>
<tr><td>Bit1</td><td>输出数据中字母的大小写控制
0：字母小写
1：字母大写</td></tr>
<tr><td>Bit0</td><td>RFU</td></tr>
<tr><td>0100：直接输出 RAM1 或是 RAM2 中的数据</td><td>Bit3 ~ Bit0</td><td>RFU</td><td>Start：输出数据的起始字节</td><td>Length：输出字节长度（使用默认长度，此字节为 0x00）</td></tr>
<tr><td rowspan="3">0011：按键值输出</td><td>Bit3</td><td>RFU</td><td rowspan="3"></td><td rowspan="3"></td></tr>
<tr><td>Bit2</td><td>Alt 键控制
0：不输出
1：输出</td></tr>
<tr><td>Bit1</td><td>Shift 键控制
0：不输出</td></tr>
</table>

			1：输出		
		Bit0	Ctrl 键控制 0：不输出 1：输出		
	0101：ASCII 码格式	Bit3	RFU		
		Bit2	RFU		
		Bit1	RFU		
		Bit0	RFU		

示例：

说明	脚本	实际输出
输出 M1 卡号 十六进制 大写	Send: 33 22 00 00 Recv: 00	831194DD
输出 M1 卡号 十进制 单字节变换 前零不输出	Send: 33 91 00 00 Recv: 00	13117148221
输出 M1 卡号 十进制 双字节变换	Send: 33 12 00 00 Recv: 00	3355338109
输出 M1 卡号 十进制 三字节变换	Send: 33 13 00 00 Recv: 00	0858971614483456
输出 M1 卡号 十进制 三字节变换 起始零不输出	Send: 33 1B 00 00 Recv: 00	0858971600000221
输出 M1 卡号 十进制 四字节变换	Send: 33 14 00 00 Recv: 00	2198967517
输出 RAM1 十六进制大写	Send: 3122 00 04 Recv: 00	“RAM1 中，当前的 4 个字节数据
输出 RAM2 十六进制大写	Send: 32 22 00 10 Recv: 00	“RAM2 中，当前的 10 个字节数据
十进制输出 12345678	Send: 30 13 BC614E Recv: 00	12345678
输出回车键	Send: 30 30 58 Recv: 00	“回车按键”
键值输出 ! @ # $ % ^ & * ( ) _ + { } | : " ~ < > ?	Send: 30 32 1E 1F 20 21 22 23 24 25 26 27 2D 2E 2F 30 31 33 34 35 36 37 38 Recv: 00	! @ # $ % ^ & * ( ) _ + { } | : " ~ < > ?
输出当前数据中的字符串“jinmuyudianziyouxiangongsi”刷卡时输出	Send: 30 50 6A 69 6E 6D 75 79 75 64 69 61 6E 7A 69 79 6F 75 78 69 61 6E 67 6F 6E 67 73 69 Recv: 00	jinmuyudianziyouxiangongsi

注：

以 UltraLight 卡号为例：

将读到的卡号（或数据）按每两个字节转化为十进制形式，数据不整齐后端补零。

脚本命令： 33 12 00 00

数据源数据： 04 23 BB E1 ED 25 80

转化数据： 0423 BBE1 ED25 0080
输出数据： 01059 48097 60709 00128

得到数据源数据即卡号为 0x04 23 bb e1 ed 25 80。脚本命令要求数据后端补零，所以转化数据为 0x0423 bbe1 ed25 8000。脚本命令要求按每两个字节转化为十进制，而且两个字节转化为十进制最大数为 65536 为五位数。所以对应的 0x0423 转化十进制数为 01059，所以一次转化得到输出数据 01059 48097 60709 32768。

5.5 RAM 操作

DATA[0]	DATA[1]	DATA[2]	DATA[3]	DATA[4]	DATA[5~n]
数据源	操作方式	数据源起始位置/数据源位置	目标起始位置/操作数值	操作数据长度/无	数据/无

指令表：

<table>
<tr><th>DATA[0]</th><th colspan="3">DATA[1]</th><th>DATA[2]</th><th>DATA[3]</th><th>DATA[4]</th><th>DATA[5~n]</th></tr>
<tr><td>数据源</td><td colspan="3">操作方式</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="6">0x40：当前指令中的数据为数据源

0x41：RAM1 数据为数据源

0x42：RAM2 数据为数据源

0x43：卡号为数据源</td><td colspan="3">0x11：数据源复制到 RAM1
0x12：数据源复制到 RAM2</td><td>数据源
起始位置</td><td>目标
起始位置</td><td>操作数据长度</td><td>数据 DATA[0]=0x40 时，该部分有效，否则为空</td></tr>
<tr><td colspan="3">0x21：加法操作
0x22：减法操作</td><td>指定数据源位置</td><td>操作数值：1 字节，范围 0x00~0xFF</td><td>无</td><td>无</td></tr>
<tr><td colspan="3">0x31：BCD 转 HEX 码
0x32：HEX 转 BCD 码</td><td>指定数据源位置</td><td>无</td><td>无</td><td>无</td></tr>
<tr><td>Bit7-Bit6-Bit5-Bit4</td><td colspan="2">Bit3-Bit2-Bit1-Bit0</td><td colspan="4"></td></tr>
<tr><td rowspan="4">0100：HEX 数据操作指令</td><td>Bit3</td><td>数据正序和反序控制
0: 默认正序
1: 反序</td><td rowspan="4">指定数据源位置</td><td rowspan="4">操作数据长度</td><td rowspan="4">无</td><td rowspan="4">无</td></tr>
<tr><td>Bit2</td><td>字节高位和低位按位交换控制
0: 默认不交换
1: 交换</td></tr>
<tr><td>Bit1</td><td>字节高半字节和低半字节交换控制
0: 默认不交换
1: 交换</td></tr>
<tr><td>Bit0</td><td>字节按位取反控制
0: 默认不取反
1: 取反</td></tr>
</table>

	0101： 十进制转换操作	Bit3	数据长度和组合长度不是整数倍，最后一个数据的补零方法 0：默认在后边补零 1：在前边补零	指定数据源位置	操作数据长度	无	无
		Bit2 ~ Bit0	转换形式 001：一个字节转化为十进制形式 010：二个字节转化为十进制形式 011：三个字节转化为十进制形式 100：四个字节转化为十进制形式 111：默认长度转化为十进制形式				
	0110： 位操作	Bit3	RFU	起始位地址	操作位长度	源数据长度（如果设置为 0，则长度使用默认值，具体值见注 1)	数据 DATA[0]=0x40 时，该部分有效，否则为空
		Bit2	源数据长度不足或操作位长度不是 8 的倍数时末尾补位 0：补 0 1：补 1				
		Bit1 ~ Bit0	操作方式 00：按位截取 Other：RFU				

注 1： 源数据为指令中的数据（DATA[0]=0x40）时，源数据长度默认等于指令总长度减去 5。
源数据为 RAM1/2(DATA[0]=0x41/0x42)时，源数据长度默认等于 128。
源数据为卡号(DATA[0]=0x43)时，源数据长度默认等于读卡器识别到的卡号长度。

示例：

说明	脚本	实际输出
当前指令中的数据为数据源，复制到 RAM1	Send: 40 11 00 00 04 11 22 33 44 Recv: 00	“RAM1 中当前保存的数据为 0x11 22 33 44”
卡号为数据源，复制到 RAM1	Send: 43 11 00 00 04 Recv: 00	“4 个字节卡号复制到 RAM1”
RAM1 中的数据为数据源，复制到 RAM2	Send: 41 12 00 00 10 Recv: 00	“RAM2 中的 16 个字节数据和 RAM1 中的数据一样”
RAM1 中的卡号第 1 个字节加 0x02	Send: 41 21 00 02 Recv: 00	原始卡号：831194DD 将改变为：851194DD
RAM1 中的卡号第 3 个字节减 0x03	Send: 41 22 02 03 Recv: 00	原始卡号：831194DD 将改变为：851191DD
RAM1 中的卡号第 3 个字节 BCD 转换为 HEX 码	Send: 41 31 02 Recv: 00	原始卡号：831194DD 将改变为：830B94DD
RAM1 中的卡号第 2 个字节 HEX 转换为 BCD 码	Send: 41 32 01 Recv: 00	原始卡号：831194DD 将改变为：831794DD
RAM1 中的卡号反序控制	Send: 41 48 00 04 Recv: 00	原始卡号：831194DD 将改变为：DD941183
RAM1 中的卡号高位和低位按位交换	Send: 41 44 00 04 Recv: 00	原始卡号：831194DD 将改变为：C18829BB
RAM1 中的卡号高半字节和低半字节交换	Send: 41 42 00 04 Recv: 00	原始卡号：831194DD 将改变为：381149DD
RAM1 中的卡号按位取反控制	Send: 41 41 00 04 Recv: 00	原始卡号：831194DD 将改变为：7CEE6B22

5.6 MIFARE 卡操作

DATA[0]	DATA[1]	DATA[2]	DATA[3~9]
数据存放处	块地址	密钥类型	6 字节密钥

数据存放处：0x51 MIFARE 卡操作结果存放在 RAM1
0x52 MIFARE 卡操作结果存放在 RAM2
密钥类型： 0x60 密钥 A
0x61 密钥 B

示例：

说明	脚本	实际输出
读 MIFARE 1 卡的块 0，结果存入 RAM1	Send: 51 00 60 FFFFFFFFFFFF Recv: 00	块 0 的 16 个字节将存入 RAM1 中
读 MIFARE 1 卡的块 4，结果存入 RAM2	Send: 52 04 60 FFFFFFFFFFFF Recv: 00	块 4 的 16 个字节将存入 RAM2 中

5.7 UltraLight 卡操作

DATA[0]	DATA[1]
数据存放处	起始块地址

数据存放处：0x61 操作结果存放在 RAM1
0x62 操作结果存放在 RAM2
起始块地址：读取块的起始块地址

示例：

说明	脚本	实际输出
读 UltraLignt 卡的块 4~7，结果存入 RAM1	Send: 61 04 Recv: 00	块 4~7 的 16 个字节将存入 RAM1 中
输出 RAM1 的前 16 个字节	Send: 31 22 00 10 Recv: 00	输出块 4 起始的 16 个字节数据

5.8 SR 系列卡操作

DATA[0]	DATA[1]
数据存放处	块地址

数据存放处：0x71 操作结果存放在 RAM1
0x72 操作结果存放在 RAM2
块地址：待读取数据的块地址

示例：

说明	脚本	实际输出
读 SRIX4K 卡的块 10，结果存入 RAM1	Send: 71 0A Recv: 00	块 10 的 4 个字节将存入 RAM1 中
输出 RAM1 前 4 个字节数据	Send: 31 22 00 04 Recv: 00	块 10 的 4 字节内容

5.9 ISO15693 卡操作

DATA[0]	DATA[1]	DATA[2]
数据存放处	块地址	块数

数据存放处：0x81 操作结果存放在 RAM1
0x82 操作结果存放在 RAM2
块地址：读取块的首地址
块数：读取数据块数量

示例：

说明	脚本	实际输出
读 ISO15693 卡的块 1 为起始块的 8 个数据	Send: 81 01 08	以块 1 为起始块的 8 个块数据，每块 4 字

块，结果存入 RAM1	Recv: 00	节，共计 32 个字节将存入 RAM1 中
输出 RAM1 前 32 字节	Send: 31 22 00 20 Recv: 00	输出块 1~8 的数据

5.10 身份证卡操作

DATA[0]	DATA[1~31]
CMD	APDU

CMD： 0x91 APDU 数据发送到身份证卡，命令结果放入 RAM1

0x92 APDU 数据发送到身份证卡，命令结果放入 RAM2

APDU： COS 指令

示例：

说明	**脚本**	**实际输出**
APDU 读取身份证 SN	Send: 91 00 36 00 00 08 Recv: 00	“结果将存入 RAM1”
输出身份证 SN，即 RAM1 中的数据	Send: 31 22 00 08 Recv: 00	30 F5 A7 A6 00 0F 1E 9F

附录 A 常用键值表

常用附加键值表

Key Name	HID Usage ID
Enter	0x58
Backspace	0x2A
Tab	0x2B
Space	0x2C
Right Arrow	0x4F
Left Arrow	0x50
Down Arrow	0x51
Up Arrow	0x52

其它附件键值请参考“USB HID to PS2 Scan Code Translation Table.pdf”。未能全部测试 HID Usage ID，请谅解。

文档更新记录

版本	日期	改动内容
……	……	……
V2.21	2015 年 3 月 25 日	修改了整体表格样式 增加了脚本例程 调整文章格式
V2.31	2018 年 9 月 28 日	更新文件内容格式 新增示例指令部分内容
V3.00	2022 年 6 月 24 日	添加通信协议的介绍 修改部分描述错误
V3.01	2022 年 7 月 25 日	修改 SAM 卡脚本描述，并添加示例
V3.02	2023 年 9 月 18 日	添加十进制输出的格式种类
V3.03	2023 年 10 月 19 日	添加十六进制输出的格式种类
V3.04	2023 年 11 月 21 日	RAM 操作中添加 hex 转化为 dec 功能 输出操作中添加直接输出 RAM1 或是 RAM2 中的数据
V3.05	2024 年 11 月 18 日	RAM 操作中添加 bit 操作功能
V3.06	2025 年 03 月 17 日	添加读 EMV 卡号功能