CH9343
Android 应用程序开发手册

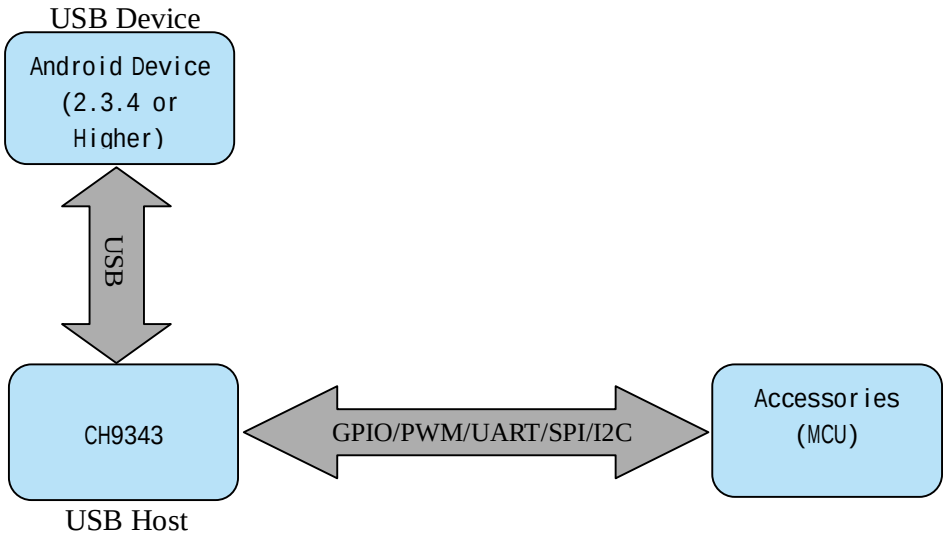
版本：V1.1

http://wch.cn

简介

CH9343 是基于 Android Open Accessory 协议开发的 USB Android Host 芯片。针对 USB Android 设备提供了 GPIO、PWM、UART、SPI Master、SPI Slave 及 I2C 主机，共 6 种接口。用户可以调用相关接口的 API 与 Accessories 通讯。

Android Device、CH9343、Accessories 三者关系如下图。



CH9343 提供的接口需要基于 Android 3.1 及以上版本系统，用户可以选择上述任一接口与 Accessories 通讯。

本文档将会重点说明 Android Device 和 CH9343 的 USB 通讯数据格式。

关于 Android Open Accessory 协议说明，可以参考 Google 官方文档。

1 Android Device

本文档所描述的例子程序皆是在 Android 3.1 及以上版本系统下编写的。Android 应用程序的启动参数是定义在 accessory_filter.xml 文件中的 manufacturer、model 和 version。
基于 CH9343 开发的 Android 应用程序主要分为两个部分：

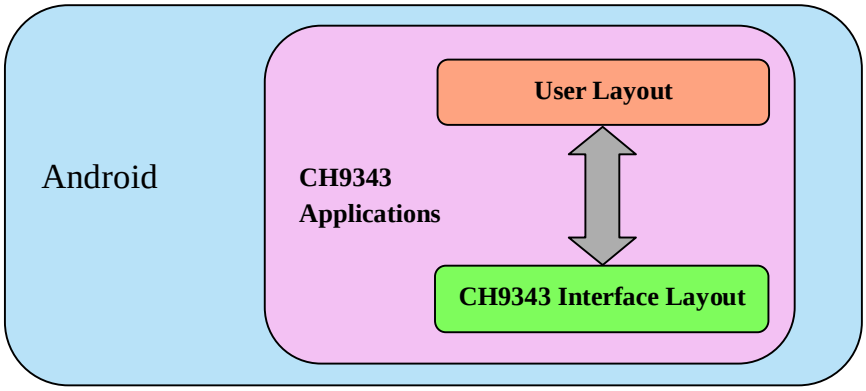


图 1: WCH CH9343 Android Applications

User Layout：主要由用户根据自己的需求完成相关代码，通过调用 CH9343 Interface Layout 提供的 GPIO、PWM、UART、SPI Master、SPI Slave 或 I2C 主机接口函数实现。

CH9343 Interface Layout：实现了 6 种接口模式下的 Interface 类，提供给 User Layout 调用。此外还实现了 Android Device 和 CH9343 芯片之间的 USB 通信，6 种不同的接口具有不同的通信协议，具体内容参考后续说明。

2 Android GPIO Demo

2.1 GPIO

CH9343 提供了 8 个 GPIO 端口，可配置成输入或输出方式。

针对 GPIO 的操作提供了 ReadPort、WritePort、ConfigPort、ResetPort 等方法，并实现与 CH9343 芯片的通讯，同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHGPIOInterface.java 文件添加到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

2.2 GPIO User-Layout

ConfigPort：配置 CH9343 端口方向。

函数原型：public void ConfigPort(int configOutMap, int configINMap)
configOutMap ：bit0～7 依次表示 GPIO0～7 的输出使能位，0：无效、1：输出；
bit8～bit15 依次表示 GPIO8～15 的输出使能位，0：无效、1：输出。
configInMap ：bit0～7 依次表示 GPIO0～7 的输入使能位，0：无效、1：输入；
bit8～15 依次表示 GPIO8～15 的输入使能位，0：无效、1：输入。
注：configOutMap 和 configInMap 对应位必须相反。

ReadPort：读取 CH9343 输入端口的状态。

函数原型：public byte ReadPort()
若读取到的数据第一个字节为 0x11，则第二第三个字节为输入端口的状态；若读取到的数据第一个字节不是 0x11，则此次获取的数据无效。

WritePort：设置 CH9343 输出端口的状态，未被配置成输出的端口将被忽略。portData 从低位

到高位依次表示 GPIO0～15。

函数原型：public void WritePort(int portData)

ResetPort： 设置 CH9343 所有端口为输入模式。

函数原型：public void Reset()

2.3 GPIO Interface-Layout

WCHGPIOInterface 类将 ReadPort、WritePort、ConfigPort 和 ResetPort 操作转换成 5 字节长度的数据包。

ConfigPort：

byte0	byte1	byte2	byte3	byte4
gpioCMD	Port1_dir	0x00	Port2_dir	0x00

gpioCMD ：0x10。
Port1_dir ：bit0～7 依次表示 GPIO0～7 的方向位，0：输入，1：输出。
Port2_dir ：bit8～15 依次表示 GPIO8～15 的方向位，0：输入，1：输出。
byte2、byte4 ：保留字节。

ReadPort：

byte0	byte1	byte2	byte3	byte4
gpioCMD	Port1_sts	Port2_sts	0x00	0x00

gpioCMD ：0x11。
Port1_sts ：读取 GPIO0～7 的输入状态，如果相应 GPIO 口未被配置成输入，则读取状态无效。
Port2_sts ：读取 GPIO8～15 的输入状态，如果相应 GPIO 口未被配置成输入，则读取状态无效。
byte3～4 ：保留字节。

WritePort：

byte0	byte1	byte2	byte3	byte4
gpioCMD	Port1_sts	Port2_sts	0x00	0x00

gpioCMD ：0x12。
Port1_sts ：设置 GPIO0～7 的输出状态，如果相应 GPIO 口未被配置成输出，则设置状态无效。
Port2_sts ：设置 GPIO8～15 的输出状态，如果相应 GPIO 口未被配置成输出，则设置状态无效。
byte3～4 ：保留字节。

ResetPort：

byte0	byte1	byte2	byte3	byte4
gpioCMD	0x00	0x00	0x00	0x00

gpioCMD ：0x13。
byte1～4 ：保留字节。

3 Android PWM Demo

3.1 PWM

CH9343 提供了一个单通道 PWM，应用程序可设置 PWM 周期，动态调节占空比（0%～99%）。

针对 PWM 的的操作提供了 SetPeriod、SetDutyCycle、Reset 等方法，并实现与 CH9343 芯片的通讯。同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHPWMInterface.java 文件包含到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

3.2 PWM User-Layout

SetPeriod： 设置周期。

函数原型：public void SetPeriod (int period)
period ：周期索引：0（5.46ms）、1（0.68ms）、2（85.33us）、3（10.67us）。

SetDutyCycle： 根据单通道设置与双通道设置的区别，使用两个方法实现设置占空比

单通道设置：
函数原型：public void SetDutyCycle(byte pwmChannel, byte dutyCycle)
pwmChannel ：通道号，设置为 0 或 1。
dutyCycle ：占空比，调节范围：0%～99%。
双通道设置：
函数原型：public void SetDutyCycle(boolean bothChannel, byte dutyCycle0, byte dutyCycle1)
bothChannel：通道号，必须设置为 true 方可，若为 false 则无效。
dutyCycle0 ：占空比，调节范围：0%～99%。
dutyCycle1 ：占空比，调节范围：0%～99%。

Reset： 设置默认值，周期：5.46ms、占空比：0%。

函数原型：public void Reset()

3.3 PWM Interface-Layout

WCHPWMInterface 类将 SetPeriod、SetDuty 和 Reset 操作转化成 5 字节的数据包。

SetPeriod：

byte0	byte1	byte2	byte3	byte4
pwmCMD	cyc_idx	0x00	0x00	0x00

pwmCMD ：0x20。
cyc_idx ：周期索引：0（5.46ms），1（0.68ms）、2（85.33us）、3（10.67us）。
byte2～4 ：保留字节。

SetDutyCycle：

byte0	byte1	byte2	byte3	byte4
pwmCMD	duty1	duty2	0x00	0x00

pwmCMD ：0x21。
duty1 ：占空比，调节范围：0%～99%，以 10 进制数据表示。
duty2 ：占空比，调节范围：0%～99%，以 10 进制数据表示。
byte3～4 ：保留字节。
注：设置占空比之前，必须先进行 SetPeriod 设置。

Reset：

byte0	byte1	byte2	byte3	byte4
pwmCMD	0x00	0x00	0x00	0x00

pwmCMD ：0x22。
byte1～4 ：保留字节。

4 Android I2C Master Demo

4.1 I2C Master

针对 I2C 的操作提供了 Reset、SetFrequency、ReadData 和 WriteData 操作方法，并实现与 CH9343 芯片的通讯。同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHI2CInterface.java 文件包含到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

4.2 I2C Master User-Layer

SetFrequency： 设置 I2C 接口的频率。默认频率值为 92KHz。

函数原型：public void SetFrequency(byte freq)
freq：调节范围：92KHz、60KHz、44KHz、23KHz。

ReadData： 读取数据。

```
函数原型：public byte ReadData(byte i2cDeviceAddress,byte transferOptions,
                               byte i2cDataAddress,
                               int numBytes, byte[] readBuffer,
                               byte[] actualNumBytes)
```

i2cDeviceAddress ：设备地址：0x00～0x7F。
transferOptions ：表示传输选项，以位来表示。

bit0：如果置 1，那么在传输开始前发出起始信号。在文件 WCHI2CInterface.java 中的 bOption.START_BIT 定义了相关的位掩码。

bit1：如果置 1，那么在传输结束后发出停止信号。在文件 WCHI2CInterface.java 中的 bOption.STOP_BIT 定义了相关的位掩码。

bit2：由于有些 I2C Slave 请求 I2C Master 在读取最后一字节后产生一个 NAK，所以如果如果 bit2 置 1，将会适用于这种 I2C Slave。在文件 WCHI2CInterface.java 中的 bOption.NAK_LAST_BIT 定义了相关的位掩码。

bit3：如果置1，则参数i2cDeviceAddress将被忽略，该特性将产生一个特殊的I2C总线条件：不发送设备地址。在文件WCHI2CInterface.java中的bOption.**NO_DEVICE_ADDRESS**定义了相关的位掩码。

bit4～7：保留位。

i2cDataAddress ：数据地址：0x00～0xFF。
numBytes ：单次读取字节数，最大为 255。
readBuffer ：接收缓冲区。
actualNumBytes ：实际读取字节数。

WriteData： 发送数据。

```
函数原型：public byte WriteData(byte i2cDeviceAddress,byte transferOptions,
                                byte i2cDataAddress,
                                int numBytes, byte[] buffer,
                                byte[] actualNumBytes)
```

i2cDeviceAddress ：设备地址：0x00～0x7F。
transferOptions ：表示传输选项，以位来表示。

bit0：如果置 1，那么在传输开始前发出起始信号。在文件 WCHI2CInterface.java 中的 bOption.START_BIT 定义了相关的位掩码。

bit1：如果置 1，那么在传输结束后发出停止信号。在文件 WCHI2CInterface.java 中的 bOption.STOP_BIT 定义了相关的位掩码。

bit2：保留位。

bit3：如果置 1，则参数 i2cDeviceAddress 将被忽略，该特性将产生一个特殊的 I2C 总线条件：不发送设备地址。在文件 WCHI2CInterface.java 中的 bOption.**NO_DEVICE_ADDRESS**

定义了相关的位掩码。

bit4～7：保留位。

i2cDataAddress ：数据地址：0x00～0xFF。

numBytes ：单次发送字节数，最大为 255。

readBuffer ：发送缓冲区。

actualNumBytes ：实际发送字节数。

Reset： 复位接口，设置默认频率：92KHz。

函数原型：public void Reset()

4.3 I2C Master Interface-Layout

WCHI2CInterface 类将 SetFrequency、ReadData、WriteData 和 Reset 操作转换为 5 字节长度的数据包。

SetFrequency：

byte0	byte1	byte2	byte3	byte4
i2cCMD	freq_idx	0x00	0x00	0x00

i2cCMD ：0x40。

freq_idx ：频率索引：0（23KHz）、1（44KHz）、2（60KHz）、3（92KHz）。

byte2～4 ：保留字节。

ReadData：

byte0	byte1	byte2	byte3	byte4
i2cCMD	ctrl	dev_addr	dat_addr	len

i2cCMD ：0x42。

ctrl ：由 ReadData 函数的 transferOptions 参数赋值。

dev_addr ：由 ReadData 函数的 i2cDeviceAddress 参数赋值。

dat_addr ：由 ReadData 函数的 i2cDataAddress 参数赋值。

len ：由 ReadData 函数的 numBytes 参数赋值。

当发送完读数据命令包后，CH9343 会返回如下的数据包：

byte0	byte1	byte2	byte3～n
i2cCMD	status	actualNumBytes	data

i2cCMD ：0x42。

status ：状态返回值： 0x03(Nack 错误)、0x05（无效地址错误）、0x00（读操作成功）。

actualNumBytes ：接收字节数。

data ：数据。

WriteData：

byte0	byte1	byte2	byte3	byte4	byte5～n
i2cCMD	ctrl	dev_addr	dat_addr	len	data

i2cCMD ：0x41。

ctrl ：由 WriteData 函数的 transferOptions 参数赋值。

dev_addr ：由 WriteData 函数的 i2cDeviceAddress 参数赋值。

dat_addr ：由 WriteData 函数的 i2cDataAddress 参数赋值。

len ：由 WriteData 函数的 len 参数赋值。

data ：数据。

当发送完写数据包后，CH9343 会返回如下的数据包：

byte0	byte1	byte2	byte3	byte4

i2cCMD	status	actualNumBytes	0x00	0x00

i2cCMD ：0x41。
status ：状态返回值： 0x03(Nack 错误)、0x05（无效地址错误）、0x00（写操作成功）。
actualNumBytes ：实际发送字节数。
byte3～4 ：保留字节。

Reset：

byte0	byte1	byte2	byte3	byte4
i2cCMD	0x00	0x00	0x00	0x00

i2cCMD ：0x43。
byte1～4 ：保留字节。

5 Android UART Demo

5.1 UART

针对 UART 的操作提供了 setConfig、ReadData 和 WriteData 方法，并实现与 CH9343 芯片的通讯。同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHUARTInterface.java 文件添加到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

5.2 UART User-Layout

SetConfig：设置 UART 接口的波特率、数据位、停止位、奇偶校验位以及流控。

函数原型：public void SetConfig(int baud, byte dataBits, byte stopBits,
byte parity, byte flowControl)
baud ：波特率：300，600，1200、2400、4800、9600、19200、38400、57600、115200、230400、460800、921600，默认：9600。
dataBits ：5 个数据位、6 个数据位、7 个数据位、8 个数据位，默认：8 个数据位。
stopBits ：0：1个停止位，1：2个停止位，默认：1个停止位。
parity ：0：none、1：add、2：even、3：mark 和 4：space，默认：none。
flowControl ：0：none，1：cts/rts，默认：none。

SendData：发送数据。

函数原型：public void SendData(byte numBytes,char[] buffer)
numBytes ：发送字节数，最大为 252。
buffer ：发送缓冲区。

ReadData：读取数据。

函数原型：public void ReadData(byte numBytes, byte[] buffer, byte[] actualNumBytes)
numBytes ：读取字节数，最大为 252。
buffer ：接收缓冲区。
actualNumBytes ：实际读取字节数。

5.3 UART Interface-Layout

WCHUARTInterface 类将 SetConfig、SendData、ReadData 操作转换为 5 字节长度的数据包。

SetConfig：

byte0	byte1	byte2	byte3	byte4

uartCMD	baud_idx	cfg_flag	0x00	0x00

uartCMD ：0x30。

baud_idx ：波特率索引，0（300bps）、1(600bps)，2（1200 bps)、3（2400 bps)、4（4800 bps)、5（9600 bps)、6（19200 bps)、7（38400 bps)、8（57600 bps)、9（115200 bps)、10（230400 bps)、11（460800 bps)、12（921600 bps)，默认：9600。

cfg_flag ：

bit0～1 ：数据位计数索引，0（5 个数据位)、1(6 个数据位)、2（7 个数据位)、3（8 个数据位)，默认：8 个数据位。

bit2 ：0：1 个停止位，1：2 个停止位。

bit3 ：校验使能，0：无校验，1：有校验。

bit4～5 ：0：奇校验，1：偶校验，2：标志位（MARK 置 1)，3：空白位（SPACE 清零)。

bit6 ：流控使能，0：无流控，1：流控使能。

注：bit4～5 只有在 bit3 置 1 时才有效。

SendData：

UART Interface 向 CH9343 发送数据时，不附带协议，保持原数据流。

ReadData：

UART Interface 从 CH9343 接收数据时，不附带协议，保持原数据流。

6 Android SPI Slave Demo

6.1 SPI Slave

针对 SPI Slave 的操作提供了 SetConfig、SendData、ReadData 和 Reset 等方法，并实现与 CH9343 芯片的通讯。同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHSPISlaveInterface.java 文件包含到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

6.2 SPI Slave User-Layout

SetConfig：配置 SPI 模式和数据大小端格式。

函数原型：public void SetConfig(byte clockPhase, byte dataOrder)

clockPhase ：0：模式 0（CPOL=0，CPHA=0)，3：模式 3(CPOL=1，CPHA=1)，默认：模式 3。

dataOrder ：数据大小端格式，0（MSB：大端格式)，1（LSB：小端格式)，默认：MSB。

SendData：发送数据。

函数原型：public byte SendData(int numBytes, byte[] buffer, byte[] actualNumBytes)

numBytes ：发送字节数，最大为 255。

buffer ：发送缓冲区。

actualNumBytes ：实际发送字节数。

注：SPI 传输是由主机发起的，一旦 SPI Slave 发起了该指令，数据就会依次被 SPI Master 读取，如果 SPI Master 没有进行读/写操作，那么数据会一直保存，同时 SPI Slave 不能再添加数据。

ReadData：读取数据

函数原型：public byte ReadData(int numBytes, byte[] buffer, byte[] actualNumBytes)

numBytes ：读取字节数。

buffer ：接收缓冲区。

actualNumBytes ：实际读取字节数。

Reset： 复位端口，设置默认值：3MHz、模式 3。

函数原型：public Reset(void)

6.3 SPI Slave Interface-Layout

WCHSPISlaveInterface 类将 SetConfig 和 Reset 操作转换为 5 字节长度的数据包。

SetConfig：

byte0	byte1	byte2	byte3	byte4
spislaveCMD	mode	0x00	0x00	0x00

spislaveCMD ：0x50。
mode ：bit6 表示模式，0：模式 0，1：模式 3。
byte2～4 ：保留字节。

SendData：

byte0	byte1	byte2～n
spislaveCMD	len	data

spislaveCMD ：0x52。
len ：发送长度。
data ：数据。
注：CH9343 发送数据后，无应答包。

ReadData：

CH9343 将从 SPI Master 接收到的数据发送给 Android 应用程序的 SPI Slave 层。SPI Slave-User 层通过 ReadData 函数将数据读取到。读取数据时没有命令码、长度等字节，全部是数据。

Reset：

byte0	byte1	byte2	byte3	byte4
spislaveCMD	0x00	0x00	0x00	0x00

spislaveCMD ：0x53。
byte1～4 ：保留字节。

7 Android SPI Master Demo

7.1 SPI Master

针对 SPI Master 的操作提供了 SetConfig、SendData、ReadData 和 Reset 等方法，并实现与 CH9343 芯片的通讯。同时针对 accessories 的操作，提供了 ResumeAccessory 和 DestroyAccessory 的功能函数，以便于 accessories 的 resume 和 destroy 操作。

在开发相关应用程序时，需要将 WCHSPIMasterInterface.java 文件包含到自己的 Android 工程中，通过调用相应的函数和 CH9343 进行通讯，以及操作 accessories。

7.2 SPI Master User-Layout

SetConfig： 设置模式、数据大小端格式及时钟速度。

函数原型：public void SetConfig(byte clockPhase, byte dataOrder, int clockSpeed)
clockPhase ：0：模式 0（CPOL=0，CPHA=0），3：模式 3(CPOL=1，CPHA=1)，默认：模式 3。
dataOrder ：数据大小端格式，0（MSB：大端格式），1（LSB：小端格式），默认：MSB。
clockSpeed ：时钟索引：0（1MHz）、1（2MHz）、2（3MHz）、3（4MHz）、4（6MHz）、5（12MHz）、

6（24MHz）。

SendData：发送数据。

函数原型：public byte SendData(int numBytes, byte[] buffer, byte[] numBytesSend)
numBytes ：发送字节数，最大为 255。
buffer ：发送缓冲区。
numBytesSend ：实际发送字节数。

ReadData：读取数据。

函数原型：public byte ReadData(int numBytes, byte[] buffer, byte[] numBytesRead)
numBytes ：读取字节数。
buffer ：接收缓冲区。
numBytesRead ：实际读取字节数。

Reset：复位端口，设置默认值：3MHz、模式 3。

函数原型：public Reset(void)

7.3 SPI Master Interface-Layout

WCHSPISlaveInterface 类将 SetConfig 和 Reset 转换为 5 字节长度的数据包。

SetConfig：

byte0	byte1	byte2	byte3	byte4
spimasterCMD	cfg_flag	0x00	0x00	0x00

spimasterCMD ：0x60。
cfg_flag ：
bit0～2 ：时钟索引：0（1MHz)、1（2MHz）、2（3MHz）、3（4MHz）、4（6MHz）、5（12MHz）、6（24MHz）。
bit3～5 ：保留位。
bit6 ：0：模式 0，1：模式 3。
byte2～4 ：保留字节。

SendData：

byte0	byte1	byte2～n
spimasterCMD	numBytes	data

spimasterCMD ：0x61。
numBytes ：发送字节数。
data ：数据。
注：CH9343 会向应用程序回复数据，回复数据长度等于发送数据长度。

ReadData：

byte0	byte1
spimasterCMD	numBytes

spimasterCMD ：0x62。
numBytes ：读取字节数。

当发送完读数据包后，CH9343 会返回如下的数据包：

byte0	byte1	byte2～n
spimasterCMD	numBytes	data

spimasterCMD ：0x62。

numBytes ：读取字节数。
data ：数据。

Reset：

byte0	byte1	byte2	byte3	byte4
spimasterCMD	0x00	0x00	0x00	0x00

spimasterCMD ：0x63。
byte1～4 ：保留字节。