Z18 读写器开发说明

V1.1

目录:

1. 系统介绍

Z18 读写器高速安全控制器，内置加密算法。可读写符合 ISO14443 协议的非接触射频卡（双界面卡）；支持两个 SAM 卡；带背光高分辨率 LCD 蓝色显示屏，四行中文显示；红、绿两种颜色的指示灯；语音提示，音量适中、音质清晰；内置高精度时钟；内置大容量存储；带红外接口。

在 Z18 硬件基础上，我们提供了 SDK 开发平台，可以使得客户自行开发底层应用程序，来使用 Z18 丰富的硬件资源。可以方便的根据自己的应用来修改底层程序。程序加密保存，无法读出。数据存储也采用加密方式。

采用 SDK 自行开发底层应用程序，可以自行控制整个密钥安全体系，而且可以及时、快速的根据实际应用情况做程序修改。对以后做系统的拓展应用也提供了可利用的代码和开发经验，保证了项目可扩展性和延续性。

2. 开发说明

函数开发语言采用 C 语言，对于复杂的底层硬件控制，已经采用了库的形式，并做了详细的函数说明，并提供有例程。

为了方便开发我们建立了标准的工程项目模板，开发的时候可以在模板的基础上添加自己的代码。

3. 函数说明

类型说明：

INT8U 表示 unsigned char

INT16U 表示 unsigned int

INT32U 表示 unsigned long

3.1 串口函数

3.1.1 bp_SerialCtl

原型:	**void bp_SerialCtl(INT8U Num,INT32U iBand)**
描述:	打开串口开关
入口参数:	Num = 0 下载程序串口 = 1 GPRS 模块串口 iBaud --- 设置的拨特率
返回参数:	无
说明:	
范例:	

3.1.2 set_char

原型:	**INT8U set_char(INT8U bDat)**
描述:	向串口(下载程序串口/GPRS 模块串口)发送数据

入口参数:	bDat : 向串口发送的数据
返回参数:	= 0 发送数据出错 = 1 发送数据成功
说明:	
范例:	

3.1.3 get_char

原型:	**INT8U get_char(INT8U *bDat)**
描述:	向串口(下载程序串口/GPRS 模块串口)接收数据
入口参数:	bDat : 向串口接收的数据指针
返回参数:	= 0 发送数据出错 = 1 发送数据成功
说明:	
范例:	

3.1.4 SerialCtl_1

原型:	**void SerialCtl_1(INT8U Num,INT32U iBand)**
描述:	打开串口开关
入口参数:	Num = 0 串口采集数据口 = 1 红外采集口 iBaud --- 设置的拨特率

返回参数:	无
说明:	
范例:	

3.1.5 set_char_1

原型:	**INT8U set_char_1(INT8U bDat)**
描述:	向串口(外部采集串口/红外模块串口)发送数据
入口参数:	bDat : 向串口发送的数据
返回参数:	= 0 发送数据出错 = 1 发送数据成功
说明:	
范例:	

3.1.6 get_char_1

原型:	**INT8U get_char_1(INT8U *bDat)**
描述:	向串口(外部采集串口/红外模块串口)接收数据
入口参数:	bDat : 向串口接收的数据指针
返回参数:	= 0 发送数据出错 = 1 发送数据成功
说明:	

范例:	

3.1.7 DebugString

原型:	`void DebugString (INT8U *cString)`
描述:	向某个串口发送字符串
入口参数:	cString – 所要发送的数据所要存在的指针
返回参数:	无
说明:	改函数是针对下载程序端口
范例:	

3.1.8 DebugHex

原型:	`void DebugHex (INT8U cDat)`
描述:	将一个字节转换成 2 个字符向某个串口发送
入口参数:	cDat – 所要发送的数据
返回参数:	无
说明:	改函数是针对下载程序端口
范例:	

3.1.9 DebugWord

原型:	`void DebugWord (INT16U cDat)`

描述:	将一个字转换成 4 个字符向某个串口发送
入口参数:	cDat – 所要发送的整形数据
返回参数:	无
说明:	改函数是针对下载程序端口
范例:	//向串口发送一个字节 0x1234 DebugWord(0,0x1234); //在 PC 终端上将显示”1234”

3.1.10 DebugLong

原型:	void DebugLong (INT32U cDat)
描述:	将一个长字转换成 8 个字符向某个串口发送
入口参数:	cDat – 所要发送的长形数据
返回参数:	无
说明:	改函数是针对下载程序端口
范例:	//向串口发送一个字节 0x123456 DebugLong(0,0x123456); //在 PC 终端上将显示”123456”

3,1.11 DebugData

原型:	void DebugData (INT32U lLen, INT16U x, INT8U *cDat)
描述:	将一串数组转换成字符串发送到串口上
入口参数:	lLen – 该数组数据的长度. X -- 在屏幕上每行显示的数据个数 cDat – 所要发送的数组

返回参数:	无
说明:	改函数是针对下载程序端口
范例:	串口发送数组 SendBuff[10] ={0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09, 0x0a}; DebugData(0,10,5,SendBuff); //在屏幕上显示如下: 01 02 03 04 05 //每行显示 5 个字节的字符串 06 07 08 09 0A

3.2 显示函数

3.2.1 DispInitialize

原型:	void DispInitialize (void)
描述:	初始化显示 LCD 硬件
入口参数:	无
返回参数:	无
说明:	在调用显示函数之前,必须调用该函数进行初始化.
范例:	

3.2.2 Display

原型:	void Display （INT8U x,INT8U y,INT8U *pStr,INT8U mode）
描述:	显示 8X16 点阵字符和 16X16 点阵的汉字,总共能显示 4 行,一行总共能显示 16 个字符或者 8 个汉字
入口参数:	X -- 行的定位,从第几行开始显示 Y --- 列的定位.从第几列开始显示 pStr -- 所要显示字符字符串的指针 mode – 显示的模式选择 = 0 正向显示 = 1 反向显示
返回参数:	无
说明:	显示字符跟汉字的混合显示.

范例:	//显示”Z18 车载机设备”
	Display(0,0,” Z18 车载机设备”,0);

3.2.3 cls

原型:	void cls （void）
描述:	清全屏
入口参数:	无
返回参数:	无
说明:	
范例:	

3.2.4 Clear_line

原型:	void Clear_line （INT8U Line）
描述:	清除某行
入口参数:	Line – 从第几行开始清除(0 – 3)
返回参数:	无
说明:	
范例:	

3.2.5 DisplayStrSmall

原型:	void DisplayStrSmall （INT8U x,INT8U y,INT8S *pStr,INT8U mode）
描述:	显示点阵 8X8 的字符(数字和英文)总共显示 8 行每行 16 个字符
入口参数:	X -- 行的定位,从第几行开始显示 Y --- 列的定位.从第几列开始显示 pStr -- 所要显示字符字符串的指针 mode – 显示的模式选择 = 0 正向显示 = 1 反向显示
返回参数:	无
说明:	

范例:	

3.2.6 DisplayBcd

原型:	void DisplayBcd （INT8U x, INT8U y, INT8U *uBcd, INT8U Len）
描述:	显示点阵 8X16 点的一串数据的 BCD 码
入口参数:	X -- 行的定位,从第几行开始显示 Y --- 列的定位.从第几列开始显示,取值范围 0 -- 3 uBcd -- 所要显示一组数据的指针 Len – 显示的数据长度
返回参数:	无
说明:	该函数主要用在调试一串数据在 LCD 上显示. 一行能显示 8 个 HEX 数据,总共能显示 4 行
范例:	

3.2.7 DisSmallBcd

原型:	void DisSmallBcd （INT8U x, INT8U y, INT8U *uBcd, INT8U Len）
描述:	显示点阵 8X8 点的一串数据的 BCD 码
入口参数:	X -- 行的定位,从第几行开始显示 Y --- 列的定位.从第几列开始显示,取值范围 0 -- 7 uBcd -- 所要显示一组数据的指针

	Len – 显示的数据长度
返回参数:	无
说明:	该函数主要用在调试一串数据在 LCD 上显示.,一行能显示 8 个 HEX 数据,总共能显示 8 行
范例:	

3.3 按键语音函数

3.3.1 GetkeyVal

原型:	INT8U GetKey (INT8U mode)
描述:	判断按键是否有按键按下,并读出按键值
入口参数:	Mode = 1 按键过程中启动蜂鸣器, Mode = 0 按键过程中不启动蜂鸣器
返回参数:	= 0xff 无按键按下 <>=0xff 键值,具体含义请参考说明
说明:	键值表: 0x08 回车键 0x05 取消键 0x01 向下键 0x02 向上键
范例:	

3.3.2 audio

原型:	void audio (INT8U Index, INT8U bDelay)
描述:	报语音
入口参数:	Index 语音端口号 bDelay 语音延时(ms)
返回参数:	无
说明:	无
范例:	

3.4 存储函数

3.4.1 FeSysWrite

原型:	INT8U FeSysWrite(INT16U startadd, INT8U len, INT8U *buff)
描述:	对 FeRom 存储器进行写操作。
入口参数:	Startadd ---写入地址。注意地址范围(0x64 --- 0x790 Len --- 写入长度 注意长度 <= 64 Buff --- 所写数据指针
返回参数:	= 1 成功 =0 不成功
说明:	在这里一定要注意地址范围

范例:	

3.4.2 FeSysRead

原型:	INT8U FeSysRead(INT16U startadd,INT8U len,INT8U *buff)
描述:	对 FeRom 存储器进行读操作。
入口参数:	Startadd ---写入地址。注意地址范围(0x64 --- 0x790 Len --- 写入长度 注意长度 <= 64 Buff --- 所读数据指针
返回参数:	= 1 成功 =0 不成功
说明:	在这里一定要注意地址范围
范例:	

3.4.3 EeSysRead

原型:	INT8U EeSysRead(INT32U lAddr,INT8U length,INT8U *Dat)
描述:	对 E2Rom 存储器进行读操作。
入口参数:	lAddr: 读 E2PROM 的地址

	地址必须是 16 的倍数 注意地址范围需根据实际设备安装了 E2 IC 的数量来决定。 1 片 E2　　laddr -> (0x00000000　--　0x0001FFFF) 2 片 E2　　laddr -> (0x00000000　--　0x0003FFFF) 3 片 E2　　laddr -> (0x00000000　--　0x0005FFFF) 4 片 E2　　laddr -> (0x00000000　--　0x0007FFFF) 5 片 E2　　laddr -> (0x00000000　--　0x0009FFFF) 6 片 E2　　laddr -> (0x00000000　--　0x000BFFFF) 7 片 E2　　laddr -> (0x00000000　--　0x000DFFFF) 8 片 E2　　laddr -> (0x00000000　--　0x000FFFFF) Length　读长度　　每次读不能超过 80 个字节 Dat　　读数据的缓冲区指针
返回参数:	= 1　成功 =0　不成功
说明:	在这里一定要注意地址范围
范例:	

3.4.4 EeSysWrite

原型:	**INT8U EeSysWrite(INT32U EeAddr,INT8U length,INT8U *Dat);**
描述:	对 EeRom 存储器进行写操作。
入口参数:	lAddr: 写 E2PROM 的地址

	地址必须是 16 的倍数 注意地址范围需根据实际设备安装了 E2 IC 的数量来决定。 1 片 E2　　laddr -> (0x00000000　--　0x0001FFFF) 2 片 E2　　laddr -> (0x00000000　--　0x0003FFFF) 3 片 E2　　laddr -> (0x00000000　--　0x0005FFFF) 4 片 E2　　laddr -> (0x00000000　--　0x0007FFFF) 5 片 E2　　laddr -> (0x00000000　--　0x0009FFFF) 6 片 E2　　laddr -> (0x00000000　--　0x000BFFFF) 7 片 E2　　laddr -> (0x00000000　--　0x000DFFFF) 8 片 E2　　laddr -> (0x00000000　--　0x000FFFFF) Length　写长度　　每次写不能超过 80 个字节 Dat　　写数据的缓冲区指针
返回参数:	= 1　成功 =0　　不成功
说明:	在这里一定要注意地址范围
范例:	

3.5 非接触卡函数

3.5.1 TypeA 卡函数

3.5.1.1 loadconfig

原型:	INT8U loadconfig (void)

描述:	对射频模块卡进行初始化
入口参数:	无
返回参数:	= 0　初始化失败 =1　初始化成功
说明:	
范例:	

3.5.1.2　mif_requeset

原型:	INT8U mif_request (INT8U mode)
描述:	对 TypeA 卡进行进行寻卡
入口参数:	Mode　　寻卡模式 = 0 IDLE 模式 = 1 ALL 模式
返回参数:	= 0　寻卡失败 =1　寻卡成功
说明:	注：关于寻卡模式的说明： 如果以IDLE模式寻卡，对卡读写操作完成时，调用rf_halt()函数中止卡操作，那么读写器再次寻卡将不会选择同一张卡；除非该卡离开并又重新进入操作区。 如果以ALL模式寻卡，对卡读写操作完成时，调用rf_halt()中止对该卡的操作，那么读写器再次寻卡时可以选择同一张卡（如果该卡不离开操作区）。
范例:	

3.5.1.2 mif_anticoll

原型:	INT8U mif_anticoll (INT8U mode, INT8U *cSerial)
描述:	对 TypeA 卡进行防止卡冲突
入口参数:	Mode 防冲突级别 = 0 第一级防冲突 = 1 第二级防冲突 cSerial --- 返回卡的序列号
返回参数:	= 0 防冲突失败 =1 防冲突成功,所返回的卡序列号有效
说明:	对于S50卡/S70卡只有一级防冲突.而对于DesFire卡有二级防冲突.具体的卡防冲突时请查阅卡的资料说明
范例:	

3.5.1.3 mif_select

原型:	INT8U mif_select (INT8U mode, INT8U *cSerial, INT8U *SNK)
描述:	选一个给定序列号的卡
入口参数:	Mode 选卡级别 = 0 第一级选卡 = 1 第二级选卡 cSerial --- 所给定的序列号 SNK --- 返回的卡容量状态码 = 0x04 Mifare ultralight 第一级选卡

	= 0x00 Mifare ultralight 第二级选卡 = 0x08 Mifare S50 选卡(该卡只有一级) = 0x18 Mifare S70 选卡(该卡只有一级) = 0x24 Mifare DesFire 第一级选卡 = 0x20 Mifare DesFire 第二级选卡 = 0x28 Mifare Pro 选卡(该卡只有一级)
返回参数:	= 0 选卡失败 =1 选卡成功,所返回的卡状态码有效
说明:	对于S50卡/S70卡只有一级选卡.而对于DesFire卡有二级选卡.具体的卡选卡时请查阅卡的资料说明 在进行操作卡过程中,
范例:	

3.5.1.4 mif_loadkey

原型:	INT8U mif_loadkey (INT8U *cKey)
描述:	对 TypeA 卡进行装载密钥
入口参数:	cKey --- 装载密钥的指针
返回参数:	= 0 装载密钥失败 =1 装载密钥成功
说明:	该密钥6个字节
范例:	

3.5.1.5 mif_auten

原型:	INT8U mif_auten (INT8U cKeyAB, INT8U cSec, INT8U *cSerial)
描述:	对某扇区密钥进行认证
入口参数:	cKeyAB – 认证密钥类型 = 1 B 密钥 =0 A 密钥 cSec -- 认证的扇区号 cSerial ---- 认证所需的卡序列号
返回参数:	= 0 认证密钥失败 =1 认证密钥成功
说明:	
范例:	

3.5.1.6 mif_Read

原型:	INT8U mif_Read (INT8U block, INT8U *cData)
描述:	对某块进行读操作
入口参数:	block – 块号码 cData ---- 读出某块数据所存放的指针,总共 16 个字节
返回参数:	= 0 读数据失败 =1 读数据成功,cData 指针数据有效

说明:	块号说明: 对于S50卡,每扇区4个块,分别是4x扇区号，4x扇区号+1，4x扇区号+2，4x扇区号+3
范例:	

3.5.1.7 mif_Write

原型:	INT8U mif_Write (INT8U block,INT8U *cData)
描述:	对某块进行写操作
入口参数:	block – 块号码 cData ---- 某块数据所写数据存放的指针，总共 16 个字节
返回参数:	= 0 写数据失败 =1 写数据成功.
说明:	块号说明: 对于S50卡,每扇区4个块,分别是4x扇区号，4x扇区号+1，4x扇区号+2，4x扇区号+3
范例:	

3.5.1.8 mif_change

原型:	INT8U mif_change (INT8U mode,INT8U block,INT8U *cValue)
描述:	对某块进行减值加值操作
入口参数:	Mode -- 操作模式 = 1 减值操作 = 0 加值操作

	block – 块号码 cValue ---- 所要操作的 4 个字节的数据
返回参数:	= 0 操作数据失败 =1 操作数据成功.
说明:	块号说明: 对于S50卡, 每扇区4个块, 分别是4x扇区号, 4x扇区号+1, 4x扇区号+2, 4x扇区号+3
范例:	

3.5.1.9 mif_transfer

原型:	INT8U mif_transfer (INT8U block)
描述:	对某块进行传输操作
入口参数:	block – 块号码
返回参数:	= 0 操作数据失败 =1 操作数据成功.
说明:	块号说明: 对于S50卡, 每扇区4个块, 分别是4x扇区号, 4x扇区号+1, 4x扇区号+2, 4x扇区号+3
范例:	

3.6 PSAM/CPU 卡函数

3.6.1 IccPowOpen

原型:	INT8U IccPowOpen (INT8U cLot, INT32U lBaud, INT8U ucVolt, INT8U *cDat)

描述:	对 PSAM/CPU 卡进行冷复位,读取复位应答.
入口参数:	cLot – 卡槽号. = 1,3,4 为 PSAM 卡. = 2 为 CPU 卡槽 lBaud – 复位的拨特率.(9600,19200,38400,115200) ucVolt – 复位电压. = 1 1.8V 2 3V 3 5V cDat – 接收复位应答数据的指针.
返回参数:	= 0 复位应答出错 <>0 复位应答正确,返回复位应答的数据长度.
说明:	

范例:	//对 PSAM 1 进行复位应答
	INT8U cRe,Buff[20];
	cRe =IccPowOpen(1,9600,3,Buff);
	if(cRe == 0)
	DebugString(0,“读复位失败”);
	Else{ DebugString(0,”读复位应答成功”);
	DebugData(0,cRe,10,Buff);}
	//将在屏幕上显示如下: 3B

3.6.2 IccIsoCommand

原型:	void IccIsoCommand (INT8U cLot,APDU_SEND *ApduSend,APDU_RESP *ApduRecv)
描述:	对 PSAM/CPU 卡进行命令操作
入口参数:	cLot – 卡槽号. = 1,3,4 为 PSAM 卡. = 2 为 CPU 卡槽 ApduSend -- 发送命令 APDU_SEND 结构指针 ApduRecv --- 接收命令 APDU_RESP 结构指针 APDU_SEND 结构如下: typedef struct { unsigned char Command[4]; unsigned char Lc; unsigned char DataIn[256]; unsigned char Le; }APDU_SEND; 其中 Command[] = {CLA，INS，P1，P2}。 Lc = DataIn 的长度。 DataIn = 要发送到 IC 卡的数据指针. Le = 期望返回数据的长度 当 Lc=0,Le=0 时,无数据发送也无数据返回 当 Lc=0;Le>0 时,无数据发送但有期望值返回, 如果在实际应用中终端期望返回的数据个数未知，应置 Le=256；否则为确定的数值。 当 Lc>0;Le=0 时,有数据发送但无期望值返回. 当 Lc>0;Le>0 时,有数据发送也有期望值返回, 如果在实际应用中终端期望返回的数据个数未知，应置 Le=256；否则为确定的数值。 这里的数据不包括命令数据和状态码. APDU_RESP 结构如下: typedef struct { unsigned char LenOut; unsigned char DataOut[256];

	unsigned int SW; }APDU_RESP; LenOut --- 实际返回的数据长度（不包括 SW1，SW2） DataOut -- 存放返回的数据 SW --- 返回的状态码
返回参数:	
说明:	

范例:	APDU_SEND Scmd;
	APDU_RESP Rcmd;
	bp_IccIsoCommand(1,3,buffer);
	//取随机数据
	Memcpy(Scmd.command,”\x00\x84\x00\x00”,4);
	Scmd.Lc=0;
	Scmd.Le=4;
	If(bp_IccIsoCommand(1,&Scmd,&Rcmd))
	{
	DebugString(0,“OK”);
	//Rcmd.SW=0x9000
	//Rcmd.DataOut={0xfc,0x78,0x12,0xa5}
	}
	Else
	DebugString(0,“ERROR”);

3.7 时钟函数

3.7.1 InitRtcTime

原型:	void InitRtcTime (void)
描述:	时钟初始化
入口参数:	无
返回参数:	无

说明:	在调用读写时钟函数之前,必须调用该函数
范例:	

3.7.2 WriteRtcTime

原型:	INT16U WriteRtcTime (INT8U *pDate)
描述:	设置时钟函数
入口参数:	pDate -- 设置日期和定时器(YYYYMMDDHHMMSS)指针
返回参数:	= 0 写日期出错 = 1 写日期正确
说明:	
范例:	

3.7.3 ReadRtcTime

原型:	INT16U ReadRtcTime (INT8U *pDate)
描述:	读时钟函数
入口参数:	pDate -- 读日期和定时器(YYYYMMDDHHMMSS)指针
返回参数:	= 0 写日期出错 = 1 写日期正确,说明 pDate 参数值有效
说明:	在读参数中,已经对日期和时间的合法性已做判断
范例:	

3.8 看门狗函数

3.8.1 bp_Odog

原型:	void bp_Odog (void)
描述:	打开看门狗功能
入口参数:	无
返回参数:	无
说明:	
范例:	

3.8.2 bp_Cdog

原型:	INT16U bp_Cdog (void)
描述:	关看门狗
入口参数:	
返回参数:	
说明:	
范例:	

3.8.3 clr_dog

原型:	INT16U clr_dog (void)

描述:	清看门狗
入口参数:	
返回参数:	
说明:	
范例:	

3.9 辅助函数

3.9.1 DESEncode

原型:	**DESEncode(INT8U *src,INT8U *dec,INT8U *key,INT8U mode)**
描述:	DES 加密运算
入口参数:	Src ---- 源数据指针 Dec ---- 目标数据指针 Key ---- 密码指针 Mode =0 单 DES 运算 = 1 3DES 运算
返回参数:	无
说明:	
范例:	

3.9.2 DESDecode

原型:	**DESDecode(INT8U *src,INT8U *dec,INT8U *key,INT8U mode)**
描述:	DES 解密运算
入口参数:	Src ---- 源数据指针 Dec ---- 目标数据指针 Key ---- 密码指针 Mode =0 单 DES 运算 = 1 3DES 运算
返回参数:	无
说明:	
范例:	

3.9.3 bp_beep

原型:	void bp_beep (INT16U iTimes, INT16U iDelay)
描述:	启动蜂鸣器, 按照一定的频率和一定音调响
入口参数:	iTimes -- 蜂鸣器频率 iDelay -- 蜂鸣器音调的毫秒数
返回参数:	无
说明:	
范例:	

3.9.4 bp_TimerSet

原型:	void bp_TimerSet （INT8U iLot,INT16U n100ms）
描述:	设置 5 个时间定时器的时间
入口参数:	iLot --- 5 个时间定时器的位号取值范围(0 --- 4) n100Ms --- 定时 n100ms *100 的毫秒数
返回参数:	无
说明:	
范例:	用定时器 0 定义 1 秒 TimerSet(0,10);

3.9.5 bp_TimerCheck

原型:	INT8U bp_TimerCheck （INT8U iLot）
描述:	读取定时器的计数器的值
入口参数:	iLot --- 5 个时间定时器的位号取值范围(0 --- 4)
返回参数:	返回定时器计数器的值,范围从 0 到 TimerSet 所设置定时值 n100Ms = 0 定时器定时 TimerSet 所设置定时值 n100Ms*100 毫秒数已完成 <>0 定时器计数器的值
说明:	iLot 取值范围在(0 – 4)
范例:	定时 1 秒后打印 Timer is OK 字符串 TimerSet(0,10); While(TimerCheck(0) != 0); //判断是否定时完成

	DebugString(0,”Timer is OK”); //通过串口打印出 Timer is OK

3.9.6 Led_onOff

原型:	void Led_onOff （INT8U iLot, INT8U Mode）
描述:	控制 LED 灯亮灭
入口参数:	iLot --- 1，2 Mode --- 0 灭 1 亮
返回参数:	
说明:	
范例:	

3.9.7 Lcd_onoff

原型:	void Lcd_onoff （INT8U mode）
描述:	控制 LCD 的背光
入口参数:	Mode --- 0 灭 1 亮
返回参数:	
说明:	
范例:	

3.10 红外函数

3.9.1 Init_Infrared_comm

原型:	**Void Init_Infrared_comm(INT32U lBaud)**
描述:	对红外模块进行初始化
入口参数:	lBaud: 红外模块初始化拨特率(9600,19200,38400,57600,115200)
返回参数:	无
说明:	在对红外模块进行初始化之后，就可以用 set_char_1/get_char_1 函数进行发送接收数据.
范例:	