Freescale Semiconductor, Inc.
应用笔记

Document Number: AN5218
Rev. 0, 10/2015

用LPUART实现IS07816接口功能

傅志强

内容

1. 简介 1
2. ISO7816 接口简介 2
2.1. 时钟频率、信号波特率和数据格式 2
2.2. 激活、去激活的时序 2
2.3. 复位应答信息的含义和协议参数选择 3
2.4. 传输协议 4
3. 管脚配置 5
4. TPM 和 LPUART 模块初始化 6
4.1. TPM 模块的初始化 6
4.2. LPUART 模块的初始化 6
5. KSDK 中的 LPUART 驱动程序 7
5.1. 需要做的修改 7
6. Demo 程序 8
6.1. LPUART 接收中断服务程序的回调函数 8
6.2. 主程序 9

1. 简介

在物联网中经常会用到各种各样的IC卡。按通信方式分，IC卡可以分为接触式的和非接触式的（无线通信）。按芯片类型分，IC卡可以分为（内部只有存储器的）存储卡和（带有CPU的）智能卡。ISO7816是接触式智能卡的标准。它规定了IC卡的物理特性、触点的尺寸和位置、接口电信号规格和传输协议，以及数据交换的命令等等。

飞思卡尔的 Kinetis L 系列微控制器（后面简称 KL），基于 ARM® Cortex®-M0+内核，具有超低功耗、高集成度和易于使用等特性，并且提供从普通尺寸到超小尺寸的各种封装，非常适合多种物联网的应用。KL 片内的低功耗串口（LPUART）模块支持单线半双工的工作模式，可以在软件的配合下实现 ISO7816 接口的功能。

本文介绍在Kinetis Software Development Kit软件包（简称KSDK）的基础上，用LPUART实现ISO7816接口功能的底层驱动软件的具体设计和注意事项。

2. ISO7816接口简介

ISO7816协议中，跟接口底层驱动程序软件设计密切相关的内容主要有：

- 时钟频率、信号波特率和数据格式；
- 激活、去激活的时序；
- 复位应答信息的含义和协议参数选择。

2.1. 时钟频率、信号波特率和数据格式

根据 ISO/IEC 7816-3 最新的版本（2006 年）的规定，接触式 IC 卡的时钟信号（CLK）的频率允许的最小值为 1 MHz。在冷复位和激活期间，允许的最大值为 5 MHz。在其余的工作期间，允许的最大值由复位应答信息中的 Fi 位段的值确定，当 Fi 缺省时仍然为 5 MHz。

数据信号的波特率在 ISO7816 中以它的倒数来定义，叫做“基本时间单位”，缩写为“etu”。时钟频率和 etu 的关系如下式：

$$1etu = \frac{Fi}{Di} \times \frac{1}{f}$$

式中，f 是时钟频率，Fi 和 Di 是复位应答信息中的两个位段，缺省值为 Fi=372、Di=1，在冷复位和激活期间总是使用缺省值。

数据的格式如*图 1*所示，一个字符帧由 1 个起始位、8 个数据位和 1 个奇偶校验位组成。在奇偶校验位到下一个字符的起始位之间是暂停时间。两个字符之间的间隔（从一个字符的前沿到下一个字符的前沿）时间的最小值称为“警戒时间”，缩写为 GT；最大值称为“等待时间”，缩写为 WT。GT 的缺省值为 12 etu，WT 的缺省值为 9,600 etu。GT 和 WT 都可以由 IC 卡在复位应答信息中指定为其他的值。

图 1 数据字符帧格式

2.2. 激活、去激活的时序

当 IC 卡插入接口设备（读卡器）以后，接口设备需要对卡进行激活和冷复位操作。如果激活和复位成功，那么 IC 卡会发出复位应答信息(简称 ATR)。激活过程的时序如*图 2*所示。

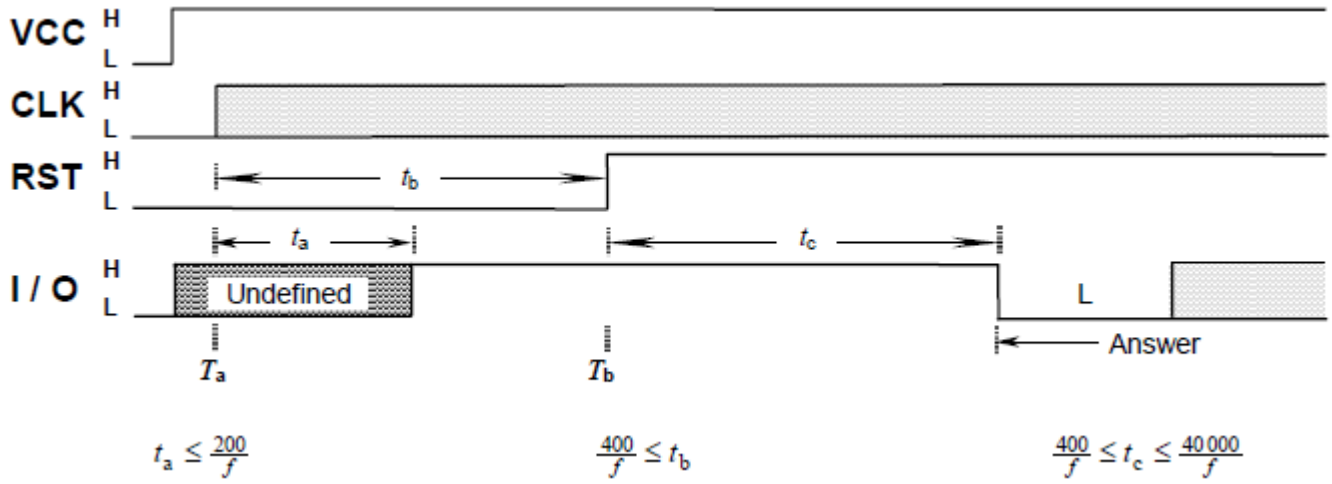

图 2 激活和冷复位时序图

当接口设备完成了与 IC 卡的信息交换，要取出 IC 卡之前，应该对卡进行去激活操作，其时序如*图 3*所示。

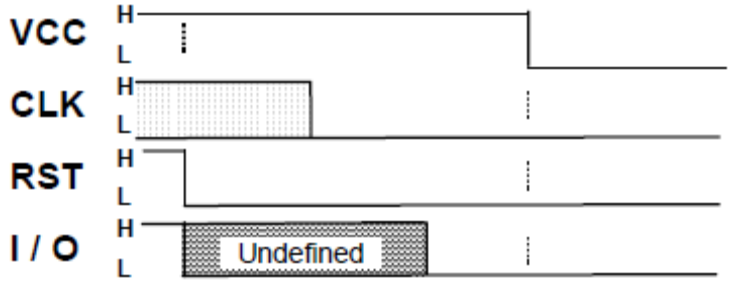

图 3 去激活时序图

2.3. 复位应答信息的含义和协议参数选择

IC 卡在成功复位（包括冷复位和热复位）以后，会主动发出复位应答信息（ATR）。ATR 的第一个字符叫做“初始字符”，记为“TS”；第二个字符叫做“格式字符”，记为“T0”，如*图 4*所示。

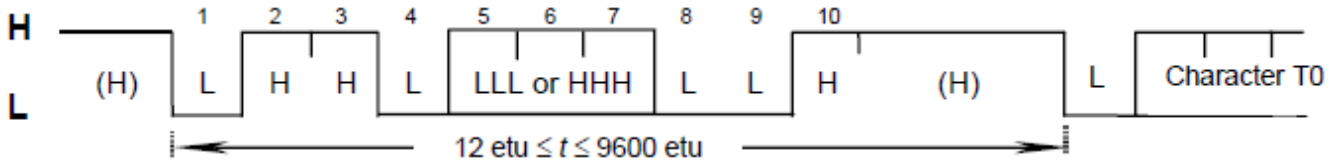

图 4 初始字符 TS

TS 有两种可能的样式，分别是：

- (H) LHHL LLL LLH：表示 IC 卡传输数据时使用反向的方式，即低电平表示“1”，高电平表示“0”，而且发送顺序是高位在前，低位在后。这时，TS 相当于十六进制的“3F”，但是用 UART 串口接收下来会变成“03”。由于 LPUART 硬件不能支持反向的数据传输方式，所以如果使用 KL 时遇到这种 IC 卡，那么需要软件对收发的数据字节进行按位取反和比特交换的处理。
- (H) LHHL HHH LLH：表示 IC 卡传输数据时使用正向的方式，即低电平表示“0”，高电平表示“1”，而且发送顺序是低位在前，高位在后。这时，TS 相当于十六进制的“3B”。

复位应答信息除了 TS 以外，最多可以有 32 个字节，其中包含了 IC 卡支持的传输协议、时钟频率、数据信号波特率和字符间隔时间等等信息。具体的内容可以查阅 ISO/IEC 7816-3 和 ISO/IEC 7816-4 的协议文件。

接口设备在接收完复位应答信息以后，可以根据其内容，选择 IC 卡支持的传输协议之一和相关的时间参数来跟 IC 卡进行后续的通信。当然，如果选择的不是缺省的，那么它就必须先把它的选择告诉 IC 卡。这个过程在 ISO7816 中就叫做协议参数选择，简称“PPS”。由于目前市场上的大部分 IC 卡都可以使用缺省的传输协议和时间参数来进行通信，所以本文对此不做进一步探讨。

2.4. 传输协议

接口设备和 IC 卡之间的通信以“命令-回应对”的方式进行，也就是由接口设备先向 IC 卡发送命令，IC 卡收到命令以后给接口设备发送回应。除了复位应答以外，IC 卡不主动向接口设备发送信息。

在 ISO/IEC 7816-3 中对半双工字符传输（T=0）和半双工块传输（T=1）两种传输协议进行了详细的定义。由于半双工字符传输协议（T=0）用得较多，而且在本文的 demo 程序中用的是 T=0 的协议，所以下面对它进行简单的介绍。

接口设备使用 T=0 协议发送命令时，先发送 5 个字节的命令头；IC 卡收到命令头以后向接口设备发送过程字节；然后接口设备根据过程字节决定下一步动作：可能是等待下一个过程字节，或者继续发送命令的其余数据字节，或者接收 IC 卡发出的数据；最后由 IC 卡发出 2 个字节的状态字（SW1、SW2）结束这条命令的通信过程。一次“命令-回应对”的交互过程如*图 5*所示。

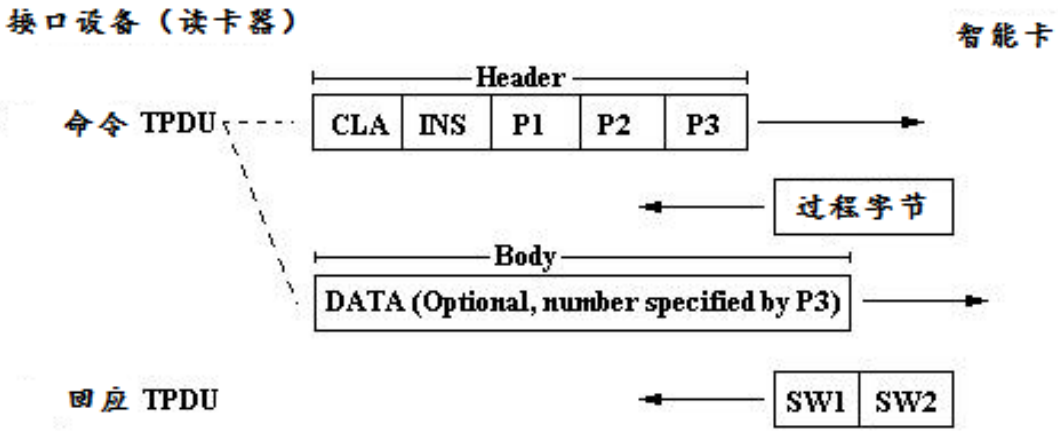

图 5 T=0 传输协议

3. 管脚配置

ISO7816 接口目前有 6 个有用的触点：

- VCC – 电源输入，按它的额定电压可以把 IC 卡分为 3 类：
  - A类：5 V；
  - B类：3 V；
  - C类：1.8 V。
- GND – 地；
- VPP – 编程电压输入，只在 A 类卡上使用，在 B、C 类卡上此触点保留待未来使用；
- RST – 复位信号输入；
- CLK – 时钟信号输入；
- I/O – 串行数据的输入/输出，使用以下两种逻辑状态：
  - 状态Z：当卡和接口设备都处于接收状态，或者数据发送方输出为高阻态。此时应当通过上拉电阻将信号线拉到高电平；
  - 状态A：当数据发送方输出低电平。

当 IC 卡不需要热插拔时，MCU 只需要用 3 个管脚来做 ISO7816 接口：

- 1 个 GPIO 连接 RST；
- 1 个 PWM 通道输出到 CLK；
- 1 个 UART_TX 连接到 I/O。

当 IC 卡要热插拔时，需要另加 GPIO 口：一个 GPIO 连接一个三极管控制电源输出到 VCC；另一个 GPIO 口作插卡到位检测。

4. TPM和LPUART模块初始化

4.1. TPM模块的初始化

对 TPM 模块的初始化主要考虑 IC 卡的 CLK 频率和信号波特率之间的关系。由于在 ISO7816 中规定 CLK 的频率范围是 1~5 MHz，而且 I/O 信号的波特率的缺省值是（f/372），所以我们不妨设置 CLK 的频率为：

$$9600 \times 372 = 3571200\ Hz$$

这样便于在调试的时候通过串口终端来观察 I/O 上传输的信息，因为大部分串口终端软件不能设置波特率为任意值，而 9600 是很常用的波特率。

TPM 初始化的代码如下：

```
const tpm_general_config_t tpmTmr1_InitConfig0 = {
  .isDBGMode = true,          //在调试模式下保持PWM输出
  .isGlobalTimeBase = false,  //使用内部计数器
  .isTriggerMode = false,     //不使用触发模式
  .isStopCountOnOveflow = false,      //计数器溢出后不停止
  .isCountReloadOnTrig = false,
  .triggerSource = kTpmTrigSel0,
};

tpm_pwm_param_t tpmTmr1_ChnConfig0 = {
  .mode = kTpmEdgeAlignedPWM,
  .edgeMode = kTpmHighTrue,
  .uFrequencyHZ = 3571200U,    //输出PWM的频率为3571200Hz
  .uDutyCyclePercent = 50U,    //PWM的占空比为50%
};
……
    PORT_HAL_SetMuxMode(PORTA,12UL,kPortMuxAlt2);    //设置PTA12管脚为TPM1_CH0功能
    SIM_HAL_SetTpmChSrcMode(SIM,TPM1_IDX,0,kSimTpmChSrc0);
    TPM_DRV_Init(FSL_TPMTMR1, &tpmTmr1_InitConfig0);
    TPM_DRV_SetClock(FSL_TPMTMR1, kTpmClockSourceModuleHighFreq, kTpmDividedBy1);
```

4.2. LPUART模块的初始化

对 LPUART 模块的初始化需要在调用 KSDK 的 API 函数以外，再添加一些代码设置相关的寄存器才能实现单线半双工通信。调用 KSDK 的初始化函数时，需要考虑的主要是数据格式。

如前所述，ISO7816 规定每个字符开头有 1 个起始位，然后是 8 个数据位，再接着是 1 个偶校验位。再然后呢？停止位？不，在常用的 T=0 协议中，后面是由接收方发送差错信号（如果出现差错信号，则发送方要重发当前这个字符）。然而，LPUART 模块并不能支持这个功能，而且 UART 的数据格式必须设置停止位。按照 *2.1* **时钟频率、信号波特率和数据格式时钟频率、信号波特率和数据格式**的说明，两个字符之间的间隔时间不小于 12 etu，因此我们应该设置 2 个停止位。具体程序如下：

```
const lpuart_user_config_t lpuartCom1_InitConfig0 = {
  .clockSource = kClockLpuartSrcIrc48M,
  .baudRate = 9600U,          //波特率为9600
  .parityMode = kLpuartParityEven,    //采用偶校验
  .stopBitCount = kLpuartTwoStopBit,  //2个停止位
  .bitCountPerChar = kLpuart9BitsPerChar, //注意：当有奇偶校验位时，数据位数应设为9
};
……
  LPUART_DRV_Init(FSL_LPUARTCOM1,&lpuartCom1_State,&lpuartCom1_InitConfig0);
  LPUART_DRV_InstallRxCallback(FSL_LPUARTCOM1, lpuartCom1_RxCallback, uartRxBuff, NULL, true);
  LPUART_DRV_InstallTxCallback(FSL_LPUARTCOM1, lpuartCom1_TxCallback, NULL, NULL);
```

然后，还需要设置 UART_TX 管脚为开漏输出、使能内部上拉，UART 接收器的输入为 TX 管脚等：

```
PORT_HAL_SetMuxMode(PORTB,1UL,kPortMuxAlt2); //设置PTB1管脚为UART0_TX功能
PORT_HAL_SetPullMode(PORTB, 1, kPortPullUp);
PORT_HAL_SetPullCmd(PORTB, 1, true);         //使能内部上拉
SIM_HAL_SetLpuartOpenDrainCmd(SIM, 0, true); //设置UART0_TX管脚为开漏输出
LPUART_HAL_SetSingleWireCmd(LPUART0, true);      //设置接收器的输入管脚为UART0_TX
LPUART_HAL_SetTxdirInSinglewireMode(LPUART0, kLpuartSinglewireTxdirIn);    //设置工作模式为接收模式
```

5. KSDK中的LPUART驱动程序

5.1. 需要做的修改

KSDK 中的 UART 驱动程序没有考虑半双工模式，而我们在刚才的初始化中把 UART 设置成了接收模式，所以在发送数据时，我们需要修改 KSDK 提供的 API 函数和中断服务程序，才能让 UART 正常工作。

打开*\KSDK\platform\drivers\src\lpuartfsl_lpuart_driver.c* 文件，在 *LPUART_DRV_SendData* 函数的开始部分添加下面的语句把 UART 设置为发送模式：

```
LPUART_HAL_SetTxdirInSinglewireMode(LPUART0, kLpuartSinglewireTxdirOut);   //设置工作模式为发送模式
```

然后在中断服务程序 *LPUART_DRV_IRQHandler* 中，修改代码让它在发送完成时把 UART 恢复到接收模式：

```
    /* Handle transmitter data register empty interrupt */
    if((LPUART_BRD_CTRL_TIE(base)) && (LPUART_BRD_STAT_TDRE(base)))
    {
        /* check to see if there are any more bytes to send */
        if (lpuartState->txSize)
        {
            /* Transmit the data */
            LPUART_HAL_Putchar(base, *(lpuartState->txBuff));
```

```
            /* Invoke callback if there is one */
            if (lpuartState->txCallback != NULL)
            {
                /* The callback MUST set the txSize to 0 if the
                 * transmit is ended.*/
                lpuartState->txCallback(instance, lpuartState);
            }
            else
            {
                ++lpuartState->txBuff;
                --lpuartState->txSize;
            }
        }
        else   //这里以下是修改过的程序
        {   /* If this was the last byte, complete transfer, will disable tx interrupt */
            LPUART_DRV_CompleteSendData(instance);
            LPUART_HAL_ClearStatusFlag(base, kLpuartTxDataRegEmpty);
            //设置工作模式为接收模式
            LPUART_HAL_SetTxdirInSinglewireMode(LPUART0, kLpuartSinglewireTxdirIn);
        }
    }
```

注意

原来的程序是在开始发送最后一个字节的时候就关掉TX中断，而这里我们要改成发送完最后一个字节才关掉TX中断。因为我们不能在刚开始发送最后一个字节时，就把UART的工作模式设置为接收模式，否则最后一个字节就发送不出去了；只能让它在发送完最后一个字节后再产生一次TX中断，然后在中断服务程序中把工作模式设为接收模式。

6. Demo程序

这个 demo 程序的目的是展示如何使用前面写的底层驱动程序，并不是实现 ISO7816 协议的所有功能。它是基于 KSDK 1.2.0 做的，用 IAR EWARM 7.40.3 编译，在 FRDM-KL03Z 评估板上调试通过。

6.1. LPUART接收中断服务程序的回调函数

在前面初始化 LPUART 时，调用了 *LPUART_DRV_InstallRxCallback* 函数给 KSDK 的驱动程序的接收中断服务程序安装了一个回调函数。在这个回调函数里，要调用上层（传输层）软件的函数来进行传输协议（本文使用 T=0）的处理。另外，还需要一个额外的操作 - 录下接收到这个字节的时间。这是为了在主程序中可以根据这个记录下的接收时间，来判断 IC 卡是否已经发送完数据或者没有响应（发生故障）了，因为：

- 两个字符之间的间隔可以大于 GT，但是不应该大于 WT（参见 *2. 1* 节）；
- 在复位应答期间，接口设备并不知道 IC 卡会发出多少个字节的 ATR 信息；
- 在传输“命令-回应对”的通信期间，虽然接口设备知道 IC 卡应该回应多少个字节，但是如果 IC 卡发生故障，那么有可能在只发送了一部分回应字节时停下来。这时，接口设备需要有一个时间记录，来判断是否应该结束等待（剩余的回应字节）。

在 KSDK 中的操作系统适配层（OSA）里提供了一个系统的定时器，我们可以利用它的 API 函数 *OSA_TimeGetMsec* 来获得当前的系统时间。具体的程序如下：

```
void lpuartCom1_RxCallback(uint32_t instance, void * lpuartState)
{
    lpuart_state_t* puartState;
    puartState = (lpuart_state_t*)lpuartState;

    TransT0Proc(*(puartState->rxBuff));   //调用传输协议处理函数
    ++puartState->rxBuff;
    if(++puartState->rxSize >= BUFF_LENGTH)      //防止缓冲区溢出
    {
           UartStartRxNextFrame();
    }
    iUartRxTime = OSA_TimeGetMsec();      //记录下接收到这个字节的时间
}
```

6.2. 主程序

在主程序中定义了 3 个函数来处理传输层的协议：

- *int SendCommand(uint8_t *apduCmd, uint8_t length)* - 供应用层软件调用，用来向 IC 卡发送命令；
- *void ResponseCallback(uint8_t *apduRsp, uint8_t length)* - 回调函数，其内容由应用层软件设计者填写，传输层软件在接收完 IC 卡的回应信息后，通过调用此函数将回应信息交给应用层处理；
- `void TransT0Proc(uint8_t charRx)` - T=0 传输协议的处理函数，在 LPUART 接收中断服务程序的回调函数中被调用。

本 demo 程序中配置下列管脚为 ISO7816 接口管脚：

- PTA8 - RST
- PTA12/TPM1_CH0 - CLK
- PTB1/UART0_TX - I/O

在 FRDM-KL03Z 评估板上通过飞线连接一个普通的手机的 SIM 卡进行验证，如*图 6* 所示。

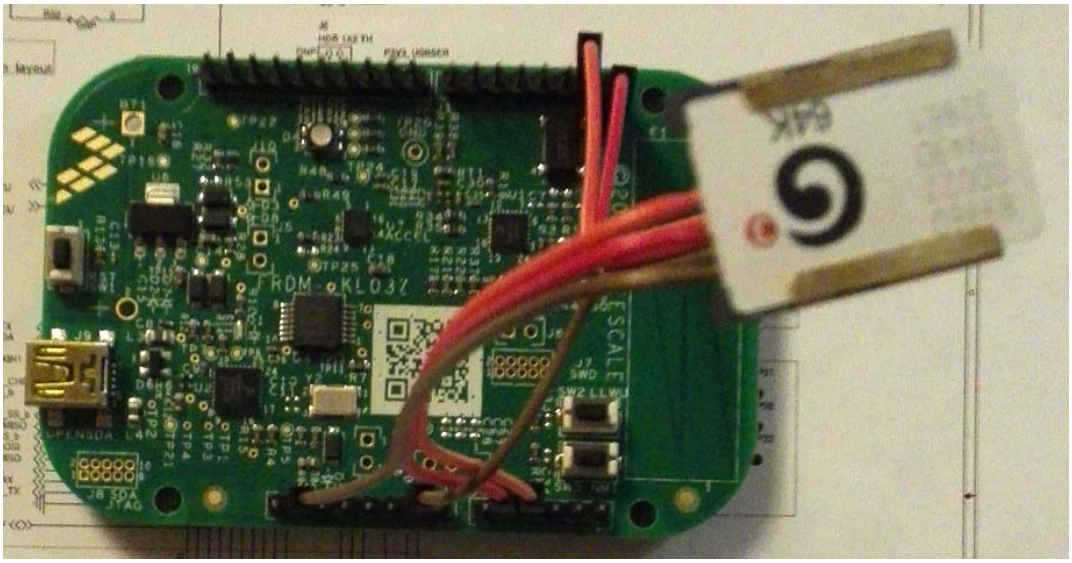

图 6 FRDM-KL03Z 板连接 SIM 卡

使用串口终端观察 MCU 和 SIM 卡之间的通信数据（FRDM-KL03Z 板子的 USB CDC 串口），结果如*图 7*。

图 7 串口终端显示信息

*图 7*中，第一排数据是 SIM 卡在 Activation 过程中发出的 ATR 信息，第二排是 MCU 给 SIM 卡发送“Get Challenge”命令和 SIM 卡回应的信息。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5218
Rev. 0
10/2015