使用德州仪器 (TI) 的NFC技术简化实现的 Bluetooth® 配对

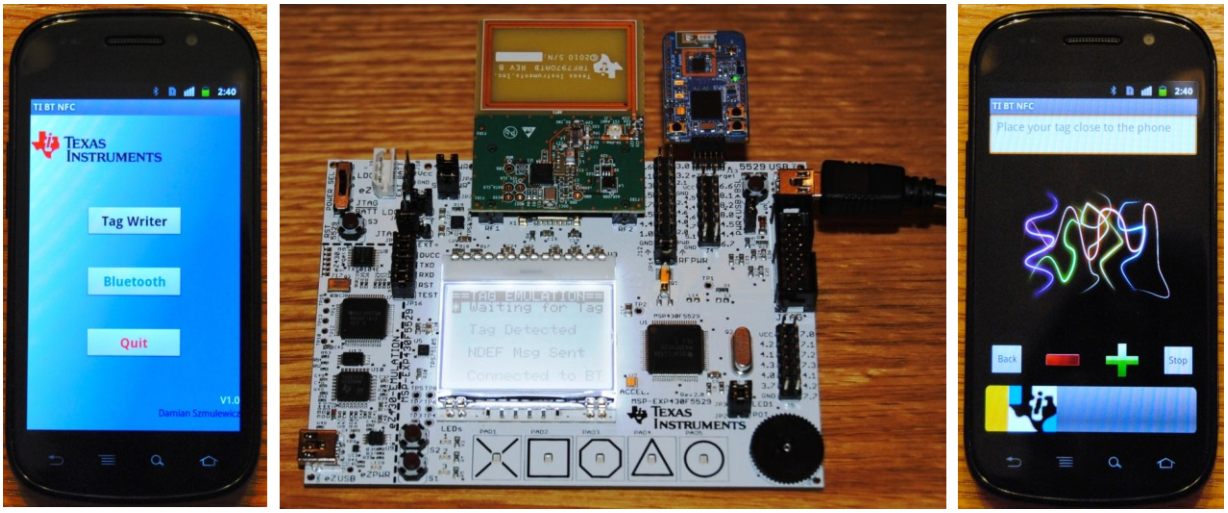
Joshua Wyatt, Eddie LaCost
Erick Macias, Damian Szmulewicz

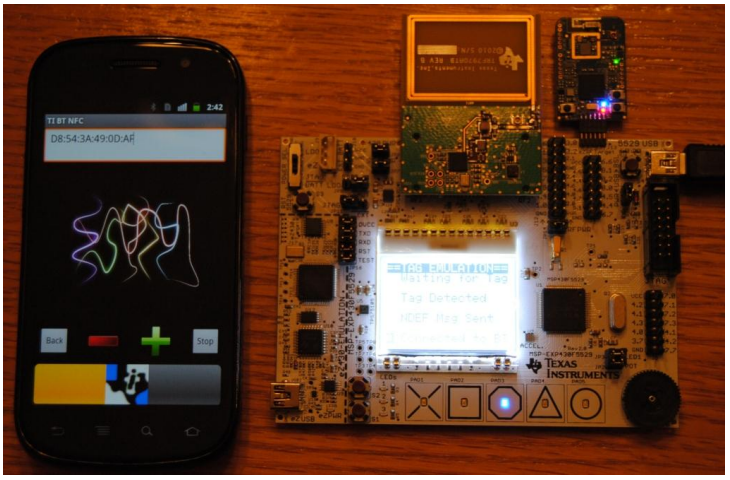
Embedded RF
MSP430

摘要

Bluetooth® 配对通常需要一些层次的用户交互以确认用户身份和/或者器件本身识别号。 在不同的蓝牙 Bluetooth 技术版本（从v2.0到v4.0）中提供多种配对机制。 这个过程通常比较冗长并且有时使用户感到困惑，所以这个应用报告的目的在于使用一个MSP430F5529（1 个 德州仪器 (TI) 超低功率微控制器(MCU)），一个TRF7970A（1 个 德州仪器 (TI) NFC 收发器集成电路 (IC)），和 1 个 CC2560（1个TI蓝牙无线电 IC）为德州仪器 (TI) 技术开发人员演示如何实现 NFC 论坛所提出的简化配对体系方法。

本文档描述的示例代码可从http://www.ti.com/lit/zip/slaa512链接下载。





内容

图片列表

图表列表

1 硬件描述

1.1 MSP-EXP430F5529

MSP430F5529 实验板 (MSP-EXP430F5529)是一个用于MSP430F5529器件的开发平台，来自最新一代具有集成 USB 的 MSP430。 此实验板与很多德州仪器 (TI) 的低功耗射频(RF) 无线评估模块，例如TRF7970ATB模块兼容。 此实验板可帮助设计人员快速研发使用全新的 MSP430F55xx MCU，此 MCU 为诸如能量采集，无线感应，和自动计量设施 (AMI) 的其它应用提供业界最低有源功耗，集成 USB，更多内存和领先的集成技术。

1.2 TRF7970ATB 模块

此TRF7970ATB模块可使软件\固件应用开发人员熟悉TRF7970A NFC 收发器的功能性，同时使他们自由开发他们所选择的德州仪器 (TI) MCU。 该模块用硬件SPI和MCU通信，支持从器件选择和 TRF7970A 直接模式2（默认），直接模式1，和直接模式0运行。 用户还可访问并完全控制此 TRF7970A EN2 和EN 线路，从而设计和开发超低功率高耗 (HF) RFID/NFC 系统。 此模块有一个板载升压转换器 (TPS61222DCKT) ，此转换器可将 3.3VDC 升压至 5VDC，使TRF7970A IC 可以实现 +23dBm（完全发射器功率输出）运行。 1 个 4Ω 至 50Ω 的阻抗匹配电路被组装在此模块上，并且此电路被连接至一个调谐 50Ω 天线电路上，此电路包含一个连接有串行和并行无源元件（电容器和一个电阻器）的板载 4 圈线圈。 借助位于主板背面的 Samtec 电迁移 (EM) 头（连接器 P1/RF1 和 P2/RF2），可实现和TI 微控制器的连接。

1.3 ez430-RF256X 目标板

此eZ430-RF256X目标板是一个用于 MSP430 和 CC2560 的完整的德州仪器 (TI) 蓝牙评估和演示工具，此工具在一个便携式 USB 记忆棒中包括了所需的硬件和软件。 此工具包括 1 个由 USB 供电的仿真器来编辑和调试您的应用程序。 这个基于 CC2650 的蓝牙目标板特有高度集成的MSP430BT5190超低功率 MCU。所需的嵌入式软件在MSP430 器件中是预编译的，开包即用且简便易用。 这个 eZ430-RF256X 软件开发工具包 (SDK)包括 MindTree 的 Ethermind 蓝牙栈，串行端口配置文件 (SPP)，和运行在 FreeRTOS 上的嵌入式示例应用程序。

> **注:** 这个 eZ430-RF256X SDK 许可证要求MSP430BT5190 只能与德州仪器 (TI) 蓝牙解决方案一起组合使用，例如量产的CC2560。

1.4 Android™ 手机 (Nexus S)

本示例中使用的 安卓 (Android)™ 手机的型号是三星 Nexus S(http://www.samsung.com)。 这款智能手机由谷歌和三星共同开发并使用Android “姜饼 (gingerbread)” 操作系统。 它集成了 NFC，蓝牙，和Wi-Fi，使得此款手机成为此应用示例中手机组件的合理选择。

2 运行概况

这份应用报告的目的在于说明用NFC技术是如何简化蓝牙连接和配对过程。 这份报告还深入研究了蓝牙配对过程的一些细节；然而，此报告的目的并不是成为关于蓝牙配对过程的综合性入门读物。 此{NFC 论坛连接移交技术规范(http://www.nfc-forum.org/specs/spec_list/) 中列出了几个执行协商移交的方法，其中包括如何用使用 NFC 技术使2个有源设备的协商来替代载波在 2 个有源器件间进行进一步的数据交换（蓝牙或者 WiFi）。 这一技术规范还列出了一个有源器件和一个无源NFC Forum标签间执行一个静态移交的方法以启用连接的第三方器件（未启用 NFC 论坛的器件）并通过替代载波进行配对的方法。 这份应用报告并不包括后一种方法。图 1显示了本文档中讨论的概况图。

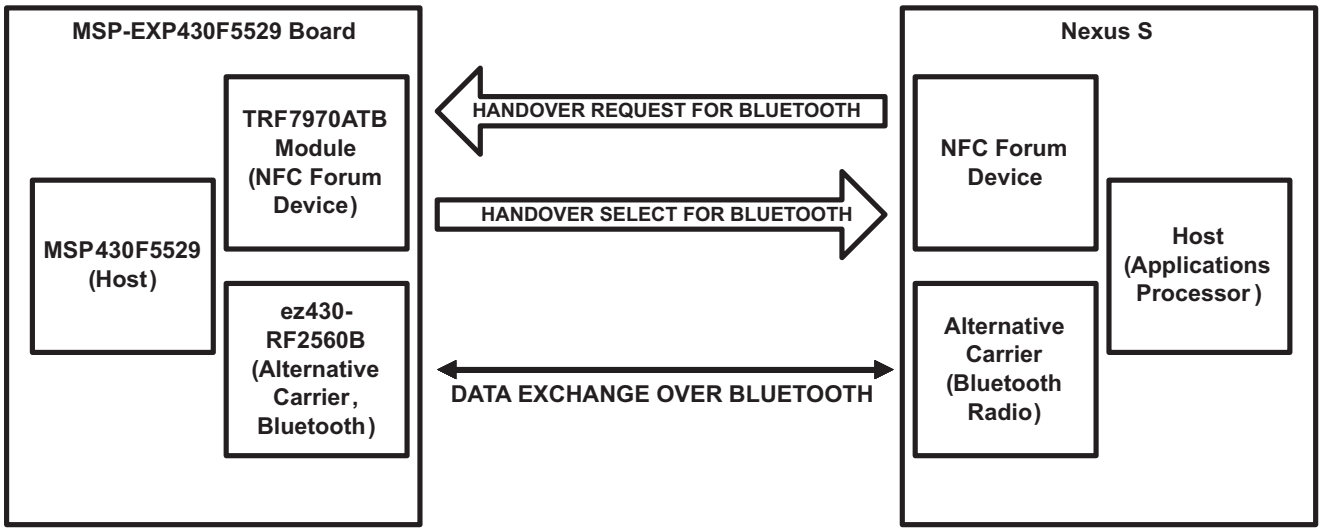


图 **1.** 用单一选择进行协商移交（用 **NFC** 进行配对）

图 1是整个系统的高级视图。3 节描述了需要成功完成此配对过程的细节。4 节和5 节描述了 MSP430F5529 MCU 上的固件/软件代码，此代码用于驱动具有 SPI 的TRF7970A 以及通过 UART 与蓝牙目标方板上的 MSP430BT5190进行通信。6 节描述了此手机上运行的 Android 应用程序，此程序是为了这个示例而编写的。

3 运行细节

图 2显示了手机和德州仪器 (TI) 硬件间的运行细节（命令请求/响应）

注: 此手机以不同的数据速率对多种技术进行轮询，这些技术和数据速率由德州仪器 (TI) 系统检测和处理。图 2显示了此流程，此流程从德州仪器 (TI) 系统检测到一个 ISO14443B 命令已经以 106kbps 的速度被发出开始（这就是本示例中TRF7970A 被配置去管理的事件）。

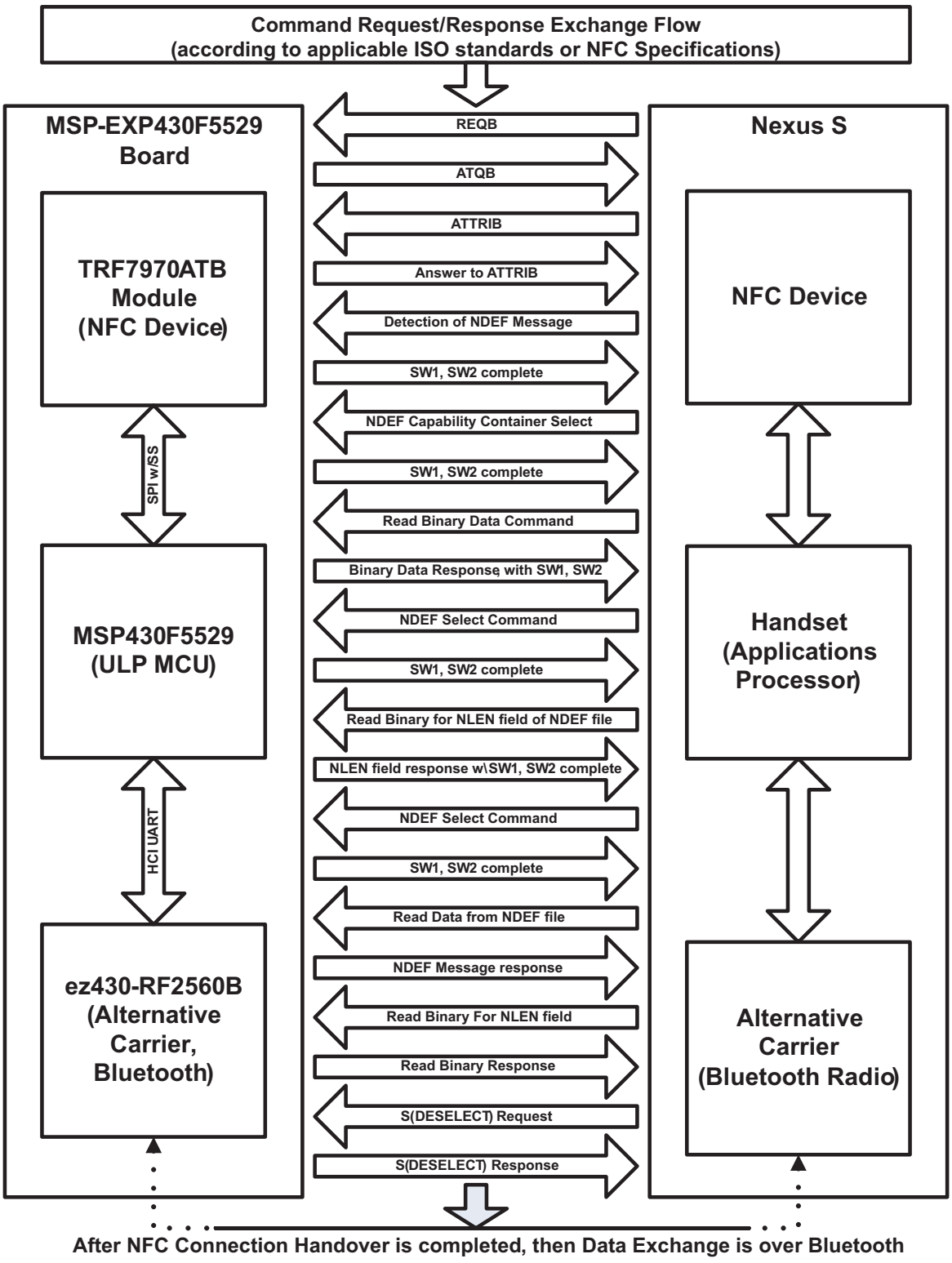


图 **2.** 命令请求/响应交换流程

此命令请求和交换流程显示在图 2中，此流程符合 ISO/IEC14443-3，ISO/IEC14443-4/ISO/IEC7816-4，和 NFC 论坛 4 类标签操作规范。

为了汇总本示例中的全部操作流程，TRF7970A首先被初始化并被微控制器配置为 NFC Type 4 Tag（正如运行在 106kbps 的 ISO14443B 器件）。 与此同时，蓝牙无线电设备被初始化并且手机上的它的蓝牙 ID 和 TI NFC BT 应用被启动。 当手机被放置在德州仪器 (TI) 的 NFC/BT 演示系统附近的时候，如图 2所示，此手机与的德州仪器 (TI) 的NFC/BT系统相互作用，然后系统中的蓝牙无线电设备完成配对并且在手机通过MSP430F5529 上的 GPIO 具有串行端口控制后，用户就可以通过使用手机上的触摸屏来驱动LED。 这些动作在4 节，5 节和6 节中进行了描述。

4 MSP430F5529 固件

4.1 概览

MSP430F5529 固件使用户能够通过蓝牙在Nexus S 和 MSP430BT5190 之间建立通信。 当通过 NFC 进行通信时，有一个发起设备（生成一个具有特定波形（称为命令）的调制的磁场来启动通信）和一个目标方（使用负载调制来接收初始方的命令并作出应答）。 在这个示例中，TRF7970A 被配置成卡仿真模式，正如一个符合 ISO14443B协议的 卡片。 Nexus S NFC 读取器硬件，此硬件处于一个轮询环路中，在这个环路中，它可处于顺序功能状态，这意味着它可以在主动模式，读取器/写入器模式，卡仿真模式，目标方模式或者暂停模式中进行轮询。 在这个示例中，我们正在等待手机进入读写器模式并且等待它发出一个106kbps的 ISO14443-3 ReqB 命令。 在TRF7970A 检测到这个 ISO14443B 读写器命令之后，图 2中描述的握手就开始了。 随后传递的 NDEF 消息包含 MSP430BT5190 MAC 地址，此消息包含与 Nexus S 建立连接所必须的数据包。当 Nexus S 停止发送TRF7970A ISO14443B/NFC 命令时，此程序通过 MSP430BT5190 的UART 读取正在进入的命令并驱动位于 MSP-EXP430F5529 实验板上的 LED，当使用 TI NFC BT app 时，此过程基于用户在 Nexus S 触摸屏上的输入。当此蓝牙通信终止时，MSP-EXP430F5529 主板上的固件重新开始此过程。

4.2 详细信息

这个部分描述了固件流程的细节和 MSP430 的要求。图 4包括此程序的流程图。

此程序从初始化 MSP430F5529 开始：

1. 将 MCLK 频率设定为 25MHz
2. 配置 GPIO 端口
3. 设定 V_{CORE} 为 3 级
4. 启用全局中断

下一步，LCD 被初始化：

1. 将亮度设定为最大值 (11)
2. 对比度设定为 11（0 至 31 级，31 级时为最黑）

下一步，TRF7970A 无线电设备被初始化：

1. 配置和启用 GPIO
2. 配置 IRQ 引脚
3. 配置 SPI 模块
   (a) 将波特率设定为 2MHz
   (b) 设定时钟相位以捕捉下降沿
   (c) 设定时钟极性为低
   (d) 设定字符长度 = 8 位，3 个引脚，主模式

UART 模块用于与 MSP430BT5190 通信。 波特率被初始化为 9600，并且 MSP430 进入低功率模式 3 秒钟，这允许MSP430BT5190 对 CC2560 进行配置。 随后，MSP430F5529 使用命令 0x39 请求蓝牙 MAC 地址，并持续等待直到它接收到 6 字节地址。 然后，此 MAC 地址被转换为 ASCII 字符并被存储在 g_ndef_message[] 阵列中。 此程序继续到一个状态机。 初始状态是断电状态 (POWER_OFF_STATE)，节 4.2.1中对此状态进行了描述。 节 4.2.2到节 4.2.5中对另外的状态进行了描述。

4.2.1 POWER_OFF_STATE

这是加电复位后或者 Nexus S 与 CC2560 通信终止后的默认状态。 TRF7970A 的初始状态为卡仿真目标方模式（参见Appendix A）。 对 LCD 屏幕进行了升级（参见图 3）。 下一个状态为感应状态 (SENSE_STATE)。

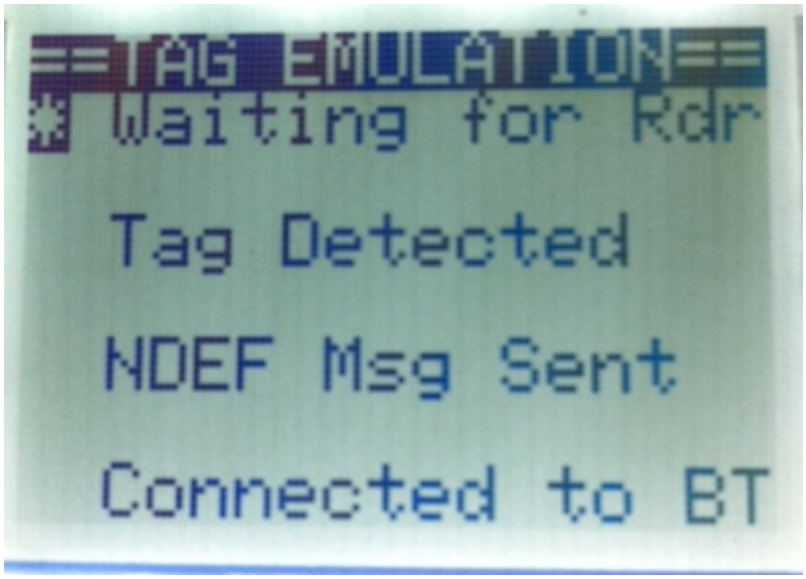


图 3. LCD 状态屏

4.2.2 SENSE_STATE

此程序对 IRQ 引脚进行轮询直到此引脚在 RF 域变化的影响下变为 0x01。 对 IRQ 状态 (0x0C)，碰撞位置 (0x0D)，和NFC 目标协议 (0x19) 寄存器进行读取。 如果初始方，Nexus S，发送一个 ISO14443B 命令到 TRF7970A，此程序的状态变为仿真状态 (EMULATION_STATE)。 当 NFC 目标协议寄存器的值为 0xC5 时，这种情况会发生。 否则，TRF7970A 被复位并且初始化为卡仿真目标模式（参见Appendix A）。

4.2.3 EMULATION_STATE

LCD 状态从 “Waiting for Rdr” 升级为 “Tag Detected”。 在这个状态下，图 2中描述的TRF7970A 和Nexus S 握手开始。 当 TRF7970A 发送包含了蓝牙 MAC 地址的NDEF 消息到 Nexus 时，此程序的状态变成NDEF 消息状态(NDEF_MESSAGE_STATE)。 如果在与 Nexus S 通信时发生错误并引起NFC目标方协议寄存器 (0x19) 产生 0xC5 之外的值，此程序返回 SENSE_STATE。

4.2.4 NDEF_MESSAGE_STATE

LCD 状态从 “Tag Detected” 升级为 “NDEF Msg Sent”。 在 NFC 目标方协议寄存器的值不在为0xC5之前，此程序将在这一点上对来自 Nexus 的命令进行应答。 随后，此程序状态变为不选择状态(DESELECTED_STATE)。

4.2.5 DESELECTED_STATE

LCD 状态从 ”NDEF Msg Sent“ 升级为 ”Connected to BT“并且此代码返回到 main() 中。

Nexus S 中的应用程序读取蓝牙 MAC 地址并连接至 MSP430BT5190。 这个手机可以发送 7 种不同的命令到MSP430F5529（参见表 1）。 当 Nexus S 与 MSP430BT5190 间的通信终止时，此程序返回断电状态(POWER_OFF_STATE)。

表 **1. MSP430BT5190** 命令

命令 **ID**	说明
0x00	关闭所有 LED
0x01	打开 LED 1.1
0x02	打开 LED 1.2
0x03	打开 LED 1.3
0x04	打开 LED 1.4
0x05	打开 LED 1.5
0x0F	断开蓝牙

4.2.6 MSP430 要求条件

- RAM：大约 2kB
- 闪存：大约 20kB
- SMCLK = 25MHz
- MCLK = 25 MHz
- ACLK ≈ 32 kHz

注: MCLK 频率可低至接近 8MHz。 此固件以足够快的速度升级此 LCD 并通过SPI与 TRF7970A 以 2MHz 的频率进行通信。

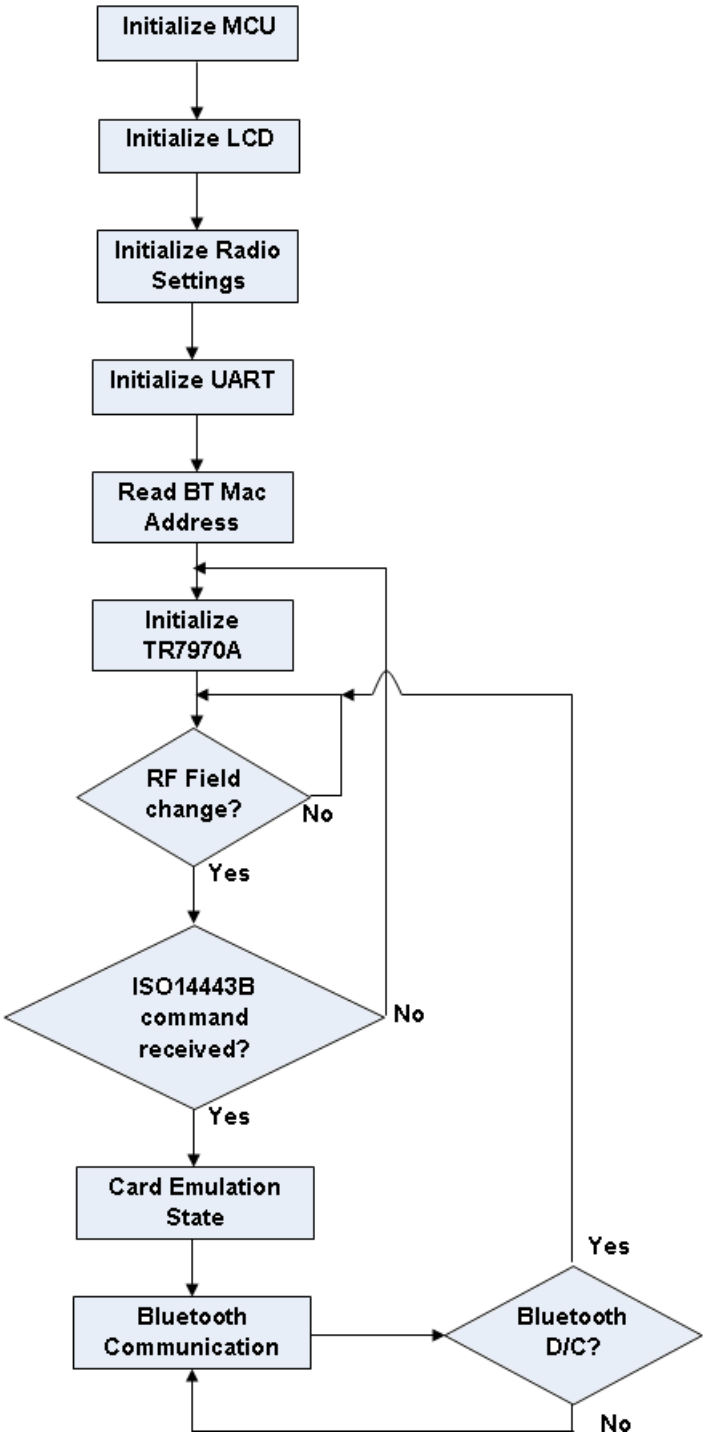


图 **4.** 固件流程图

5 MSP430BT5190 固件改动

5.1 概览

MSP430BT5190 微控制器是为了与德州仪器 (TI) 的基于 CC2560 的蓝牙解决方案的商业应用而设计，此解决方案与MindTree 的 Ethermind 蓝牙栈和串口协议配置文件 (SPP)协同工作。 这个 MSP430BT5190+CC2560 蓝牙平台是无线连接替代线缆连接应用的理想选择，此类应用诸如遥控器，恒温器，智能仪表，血糖仪，脉动血氧计，以及其它多方面的应用。

MindTree 的 EtherMind 蓝牙 SDK 为最终系统设计人员提供一个平台以使他们能够对EtherMind 蓝牙软件协议栈和配置文件进行快速评估，并使用它来实现应用。 该 SDK 是在 TI MSP430BT5190 处理器。 套件中包含的参考应用和开发工具可帮助实施人员在 MSP430BT5190+CC2560 平台上快速开发定制的蓝牙原型应用。 这个 SDK 支持串行端口配置文件 (SPP)。 FreeRTOS 用于支持蓝牙读写任务。 用户可以开发他们自己的应用并与蓝牙 Ethermind 堆栈 API 进行交互，以在他们的终端产品上启用蓝牙连接。

这个用于EZ430-RF256X 工具包的完整平台可以从http://www.ti.com/tool/mt-bt-sdk中获得。 对于这个报告中描述的应用，所需的软件包为EZ430-RF256X-SDK-GA。 这个软件包包含一个示例应用，可对此示例应用进行修改以通过 NFC 实现简化的蓝牙配对。 本报告在这个部分对所需的改动进行了描述。

5.2 详细信息

EZ430-RF256X-SDK-GA包含一个企鹅赛车 (Penguin Racer) 应用程序，此应用程序在点到点网络中使用 MindTree 的 EtherMind 蓝牙栈和串行端口配置文件 (SPP) 提供了一个 EZ430-RF256X 演示。 在这个演示应用程序中，需要 2 个EZ430-RF256X 器件。 器件 A 从板载加速计内搜集数据并通过 SPP 连接将搜集到的数据无线发送到器件 B。在另一端，器件 B 从 SPP 连接接收此数据并将其发出至 USB 连接器（在 EZ430U 上使用 UART 到USB 的桥接）。 请参见EZ430-RF2560 的维基网页获得此演示应用的更多信息。

第一组改动被要求用来创建 UART 到蓝牙的并使用提供给 CC256x 的 Mindtree 栈的桥接应用。 这些变化使得 CC256x 系统能够将数据从 UART 端口传递到其它开启蓝牙的器件。图 5为此系统解决方案的概览。 EZ430-RF256X 能够通过蓝牙向任何开启蓝牙的器件发送数据以及接受数据。 接收到的数据通过 UART 传送到 MSP430 微控制器。 相似的，通过 UART 总线接受到的数据由EZ430-RF256X 模块通过蓝牙链路进行传送。 因此，这个解决方案为任一含 USCI 模块的MSP430 提供蓝牙功能。 对于这个特别应用，MSP430F5529 实验板被选择作为 MSP430 MCU。

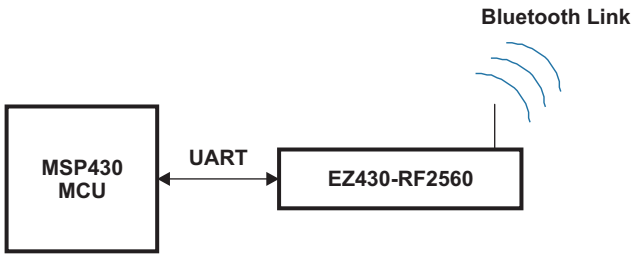


图 5. 系统概览

为了实现 UART 到蓝牙桥的连接而对 Penguin Racer 应用程序所进行的所有改动可从CC256x MT UART BRIDGE 的维基网站中获得。

SLAA512 — http://www-s.ti.com/sc/techlit/SLAA512

每个蓝牙器件具有特定的蓝牙地址。 在建立蓝牙连接时需要这个地址。 例如，为了使用一个开启蓝牙的手机连接到一个EZ430-RF256X，这个用户必须从手机的设置中扫描蓝牙器件，然后从可获得的器件列表中选择正确的器件。 每个可用器件由蓝牙地址进行标识。 这个手机必须具有蓝牙地址以建立连接。
在MSP430F5529 实验板的请求下，对Penguin Racer 应用程序进行第二组所需改动，改动后，EZ430-RF256X 可通过 UART 通道发射它自己的蓝牙地址。 然后，MSP430F5529 可以以 NDEF 信息格式来格式化这个蓝牙地址并将其存储在 TRF7970A NFC 收发器模块中。 当这一步完成时，一部具有 NFC 功能的电话，例如 Nexus S，能够从 TRF7970A NFC 收发器中读取 EZ430-RF256X 蓝牙地址并建立一个连接。

必须对表 2中列出的文件进行修改以实现从 EZ430-RF256X 到 MSP430F5529 的按要求蓝牙地址传输：User_task.c，sdk_common.c， BT_task.h。表 2显示了所需的改动。 通过在 UART 链路上发送一个0x39，MSP430F5529 可完成一个蓝牙地址请求。

表 2. 软件变化

文件	改变	说明
sdk_common.c	添加到文件顶部： UCHAR bluetooth_address[BT_BD_ADDR_SIZE]；	定义一个尺寸阵列 BT_BD_ADDR_SIZE 以保持本地蓝牙地址

sdk_common.c	添加到功能内部
	void sdk_set_config_local_name_suffix(UINT32 * name_length_ptr):
	for(int i = BT_BD_ADDR_SIZE; i>0 ; i--)
	bluetooth_address [i-1] = hci_local_bd_addr[i-1];
		复制蓝牙地址到蓝牙地址。 当有请求时，这个阵列确保传输地址有效

BT_task.h	添加到文件顶部： 外部UCHAR bluetooth_address[BT_BD_ADDR_SIZE]；	定义一个全局阵列以保持蓝牙地址

user_task.c	添加到中断空内部 void USB_UART_ISR(void):
	char temp_data;
	temp_data = USB_RXBUF;
	if(temp_data == 0x39)
	{
	for(int i = BT_BD_ADDR_SIZE; i>0 ; i--)
	halUsbSendChar(bluetooth_address [i-1]);
	}
	else
	halUsbReceiveBuffer[bufferSize++] = temp_data；
		如果在 USB_RXBUF 上接收到 0x39，则发送蓝牙地址。 或者，将接收到的数据存储在接收缓冲器(halUsbReceiveBuffer) 中进行处理

6 TI BT NFC Android™ 应用

6.1 概览

这个 Android 应用程序的目的在于通过 NFC 链路从 TRF7970A 获得 EZ430-RF256X 蓝牙地址并且如图表 1 中显示的那样处理蓝牙连接。为了达到此目的，这个手机必须支持 NFC 和SPP 蓝牙功能。 最小的 NFC API 级是 API 9 级；因此这个应用中所需的 Android 最低版本为Gingerbread（Android 2.3 及以上版本）。 这个应用中选择的是 Nexus S。 此报告的这一部分提供了对于实现所描述的任务所需的 Android 程序设计的描述。 认为已经具有 Android 开发的基本知识。 所开发的代码只支持 NDEF 数据格式。

注: 此手机蓝牙设置中的器件名必须设定为 ”BlueMSP-Demo“。 这是因为 EZ430-RF256X 只允许到使用此名称器件的非安全连接。

6.2 详细信息

6.2.1 应用程序和用户接口 (UI)

图 6显示了这个应用程序的用户接口。 在文件顶部，一个文本框显示的文本为 ”将您的标签靠近手机“。 当接收到一个 NFC 信息时，NDEF 文本信息显示在这个文本框中。 用户可使用 6 种方法输入数据：5 个按钮和 1 个滚动条。 按下后退按钮来终止蓝牙连接并返回到之前的活动。 按下 - 号按钮发送一个介于 0 到 5 的值，这个值比之前发送的值小 1。 例如，如果之前发送的值为 4，当按下 - 号键时，通过蓝牙发出的值为 3。 如果之前的值是 0，则按下按钮后，发出的值为5。 相似的，按下 + 号按钮发送一个介于 0 到 5 的值，这个值比之前发送的值大 1。 例如，如果之前发送的值为 4，当按下+号键时，通过蓝牙发出的值为 5。 如果之前的值是 5，则按下 + 按钮后，发出的值为0。 停止按钮发出的值一直为 0。轻敲魔术背景发送一组数（所有的数在 0 到 5 之间）。 滚动条的工作方式与 - 号和 + 号键一样，只不过它能以更快的方式发送数值。

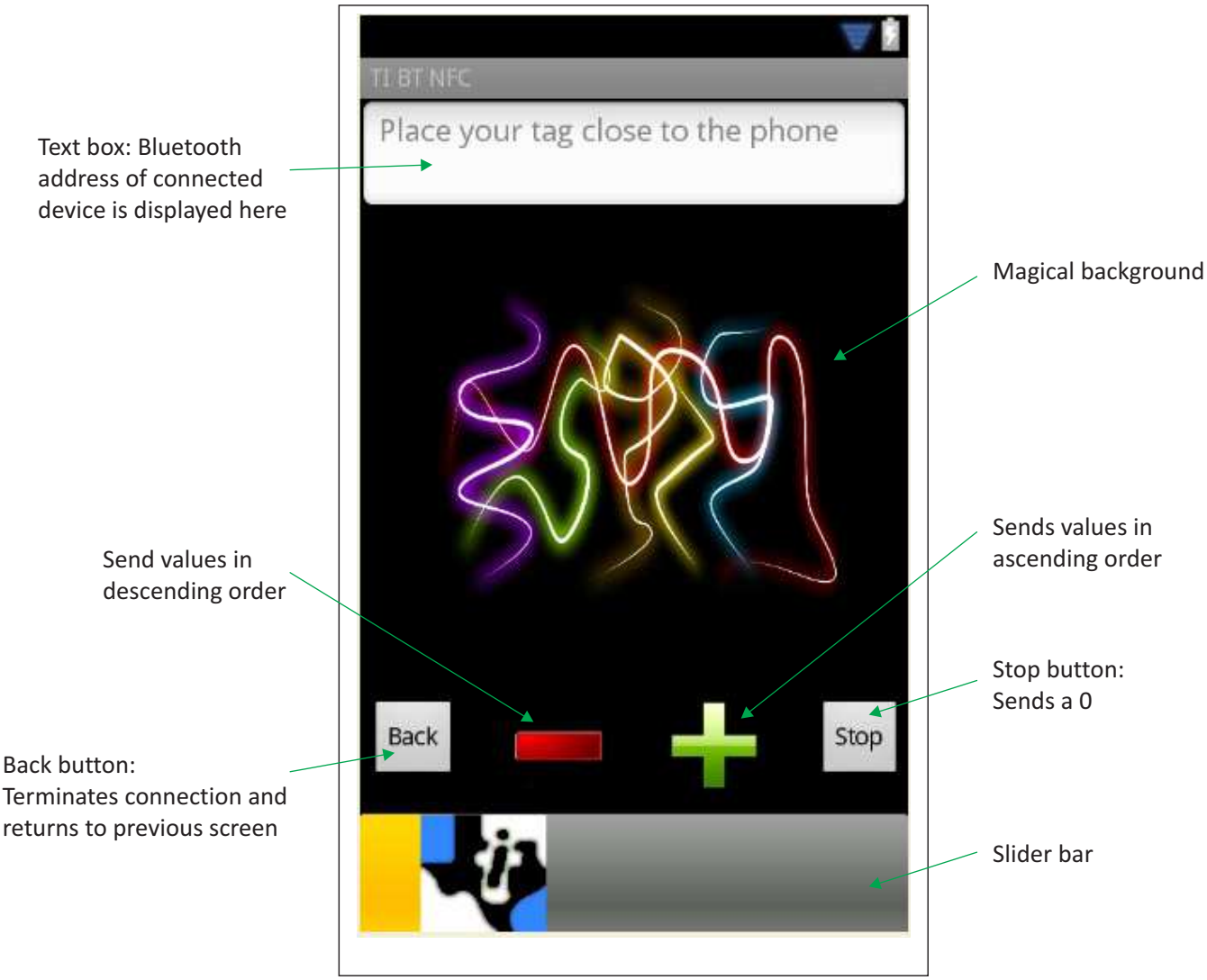


图 6. Android 应用程序 UI

6.2.2 NFC 初始化和处理

为了在蓝牙对话活动中可声明具有 NFC 适配器，第一步要将 NFC 功能性添加到 Android 应用中：

```
NfcAdapter mNfcAdapterM
```

这行代码声明 mNfcAdapter为 NfcAdapter类型；因此，这个应用程序可访问这个手机的内置 NFC适配器。下一步，当有 NFC 事件发生时，必须声明意图和意图过滤器以定义动作：

```
PendingIntent     mNfcPendingIntent;
IntentFilter[]    mNdefExchangeFilters;
```

当 Android 接收到一个与创建的意图过滤器相匹配时，需要此意图过滤器，调度前台的活动。

当 NFC 参数被声明时，需要对这些参数进行定义。 在 onCreate 方法内，必须从手机获得的 NFC适配器并连接到 mNfc 适配器上，而且必须为应用的意图过滤器定义意图：

```
// Get adapter from phone
mNfcAdapter = NfcAdapter.getDefaultAdapter(this);

// Handle all of our received NFC intents in this activity.
mNfcPendingIntent = PendingIntent.getActivity(this PersonName, 0 PersonName, new Intent(this
PersonName, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP) PersonName, 0);

// Intent filters for reading a note from a tag or exchanging over P2P.
IntentFilter ndefDetected = new IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED);
mNdefExchangeFilters = new IntentFilter[] ndefDetected ;
```

下一步就是启用 NDEF 消息的发送和接收。 此步骤通过 onResume 方法中的功能调用来完成：

```
// Enable pushing and receiving NDEF messages
enableNdefExchangeMode();
```

这一步看起来像：

```
private void enableNdefExchangeMode() {
                    mNfcAdapter.enableForegroundDispatch(this,
                    mNfcPendingIntent,
                    mNdefExchangeFilters, null);
                    }
```

enableForegroundDispatch 为经滤波的意图部署监听器。 当一个与意图过滤器匹配的意图被识别后，enableForegroundDispatch 调用活动的onNewIntent 方法：

```
public void onNewIntent(Intent intent) {
// If NFC intent occurred
if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction())) {

// Get NDEF message
NdefMessage[] msgs = getNdefMessages(intent);

// Store message in a string named "body"
String body = new String(msgs[0].getRecords()[0].getPayload());

// Make BT address value read by NFC. Body has the BT address of the
// connecting device (From the NFC adapter)
String temp_string = new String(body.substring(1, 18));
bluetoothAddress = temp_string;
setNoteBody(bluetoothAddress);
```

setNoteBody 是一个将NDEF信息发送到屏幕的方法：

```
private void setNoteBody(String bluetoothAddress) {
            Editable text = mNote.getText();
            text.clear();
            text.append(bluetoothAddress);
            }
```

下方显示了 getNdefMessages。 这一方法与 Android 开发指南中便利贴 (Sticky Notes) 演示程序的一样。

```
NdefMessage[] getNdefMessages(Intent intent) {
    // Parse the intent
    NdefMessage[] msgs = null;
    String action = intent.getAction();
    if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action) ||
NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
    Parcelable[] rawMsgs = intent
    .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if (rawMsgs != null) {
            msgs = new NdefMessage[rawMsgs.length];
            for (int i = 0; i < rawMsgs.length; i++) {
            msgs[i] = (NdefMessage) rawMsgs[i];
            }
            } else {
            // Unknown tag type
            byte[] empty = new byte[] {};
            NdefRecord record =
            newNdefNdefRecord(NdefRecord.TNF_UNKNOWN,
            empty, empty, empty);
            NdefMessage msg = new NdefMessage(new NdefRecord[] { record });
            msgs = new NdefMessage[] { msg };
            }
            } else {
            Log.d(TAG, "Unknown intent.");
            finish();
            }
            return msgs;
            }
```

最终，必须在清单中定义 NFC 允许和意图过滤器：

```
<uses-permission android:name="android.permission.NFC"> </uses-permission>

   <intent-filters>
       <action android:name="android.nfc.action.TAG_DISCOVERED" />
       <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
```

6.2.3 蓝牙初始化和处理

将蓝牙功能性添加到 Android 应用程序的第一步是为了声明蓝牙对话活动，一个用于连接的UUID，一个蓝牙适配器，一个蓝牙插槽和 2 个通信缓冲器：输出流 (OutputStream) 和输入流 (InputStream)。

```
// well known SPP UUID

private static final UUID MY_UUID = UUID
   .fromString("00001101-0000-1000-8000-00805F9B34FB");

// Bluetooth state

   private BluetoothAdapter mBluetoothAdapter = null;
   private BluetoothSocket mBluetoothSocket = null;

   private OutputStream mOutputStream = null; // Output data buffer
   private InputStream mInputStream = null; // Input data buffer
```

下一步在 'onNewIntent' 内调用一个启动蓝牙连接的方法只要 NFC 事件发生就调用 'onNewIntent'；因此，只有在 NFC 适配器接收到一个信息后，一个蓝牙连接才会发生。 这步已经完成，因此只有当一个有效地蓝牙地址可用时，连接才会发生：

```
public void onNewIntent(Intent intent) {
       ...
```

```
        connection_state = BTConnect();
        ...
        }
```

看起来像：

```
private boolean BTConnect() {
// Get adapter, open socket and connect
   mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
   // Check if adapter is available on phone
   if (mBluetoothAdapter == null) {
       return false;
       }
   // Check if Bluetooth is turned on
   if (!mBluetoothAdapter.isEnabled()) {
       Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
       Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
       discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
       startActivityForResult(enableBtIntent, RESULT_OK);
       }
       try {
   BluetoothDevice device = mBluetoothAdapter
      .getRemoteDevice(bluetoothAddress);
   mBluetoothSocket = device
   .createInsecureRfcommSocketToServiceRecord(MY_UUID);
   // Discovery is a heavy weight process: disable while making a connection
   mBluetoothAdapter.cancelDiscovery();
   mBluetoothSocket.connect();
   // Handle incoming and outgoing BT data
   mOutputStream = mBluetoothSocket.getOutputStream();
   mInputStream = mBluetoothSocket.getInputStream();
   } catch (Exception e) {
      Context context = getApplicationContext();
      CharSequence text = "Failed to Connect";
      int duration = Toast.LENGTH_SHORT;
      Toast toast = Toast.makeText(context, text, duration);
      toast.setGravity(Gravity.TOP, 0, 0);
      toast.show();
      Log.e(TAG, "BTConnect + " + bluetoothAddress, e);
      return false;
      }
   // If connection has been established, return true
   return true;
   }
```

当连接被建立时，就可以在蓝牙通道上开始通信了。 一个方法调用被用于通过 mOutputStream 将字节从这个手机发送到连接的蓝牙器件：

```
private void BTWrite(int b) {
    try {
       // Attempt to write
       mOutputStream.write(b);
       } catch (IOException e) {
       // Disconnect if write fails
          Log.e(TAG, "BTWrite" + e);
          BTDisconnect();
          }
        }
```

这个 Android 应用中蓝牙部分所实现的最后一步是连接终止。 用户可以使用 EZ430-RF2560 模块将蓝牙连接终止：

```
private void BTClose() {
    try {
```

```
        // Close mOutputStream
        mOutputStream.close();
        } catch (Exception e) {
        Log.e(TAG, "BTClose" + e);
    }
    try {
        // Close mInputStream
        mInputStream.close();
        } catch (Exception e) {
        Log.e(TAG, "BTClose" + e);
    }
    try {
         // Close mBluetoothSocket
         mBluetoothSocket.close();
         } catch (Exception e) {
         Log.e(TAG, "BTClose" + e);
         }
    mBluetoothSocket = null;
    mBluetoothAdapter = null;
}
```

最后，此蓝牙必须在清单中定义：

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

7 参考

1. TRF7970A 数据表(SLOS743)
2. ISO/IEC14443-3:2009(E) (http://www.ansi.org)
3. ISO/IEC14443-4:2008(E) (http://www.ansi.org)
4. ISO/IEC7816-4:2005(E) (http://www.ansi.org)
5. ISO/IEC18092 / ECMA-340 (NFCIP-1) (http://www.ansi.org)
6. ISO/IEC21481/ECMA-352 (NFCIP-2) (http://www.ansi.org)
7. NFCForum-TS-Type-4-Tag_2.0（4 类标签操作规范） (http://www.nfc-forum.org)
8. NFCForum-TS-NDEF_1.0 (NFC 数据交换格式 (NDEF) 规范) (http://www.nfc-forum.org)
9. NFCForum-TS-ConnectionHandover_1_2 （连接移交技术规范）(http://www.nfc-forum.org)
10. CCS/IAR MSP430F5529 固件项目（网络连接）
11. CCS/IAR MSP430BT5190 代码项目(http://www.ti.com/tool/mt-bt-sdk)
12. Android TI BT NFC APK （网络连接）

Appendix A 用TRF7970A实现ISO14443B的卡仿真

1. 使用直接命令 0x03 → 0x83（软件初始化）对具有 SS 的SPI进行初始化（参见 TRF7970A 数据表的表 5-18）。
2. 发送 0x00 → 0x80（idle）将 TRF7970A 置（参见 TRF7970A 数据表的表 5-18）。
3. 写入寄存器将 TRF7970A 配置成所需的模式（本示例中为 ISO14443B）。
4. 表 3显示了寄存器和按顺序的写入值。

表 3. ISO14443B 106kbps 卡仿真的寄存器值

寄存器	值	注释
0x09	0x01	SYS_CLK 和 MOD
0x01	0x25	NFC 卡仿真，Type B
0x0B	0x01	稳压器，可被设定为 0x87（自动）
0x0A	0x3C	ISO14443B 的 RX 专门设置
0x18	0x07	NFC 目标方检测水平
0x17	0x80123456	NFCID/PUPI 示例
0x16	0x03	NFC 低场强检测级
0x02	0x02	ISO14443B TX 选项
0x00	0x21	芯片状态控制

5. 用直接命令 0x0F → 0x8F 来将 FIFO 复位
6. 用直接命令 0x16 → 0x96 和 0x17 → 0x97来关闭/启用接收器
7. TRF7970A 现在被配置为一个 NFCID/PUPI 值为 0x8012345616 的 ISO14443B 发射机应答器，等待场出现和命令发布。

Appendix B 与 Android 手机交互作用

此信息是针对运行 Gingerbread 2.3.4 或者更高版本的三星 Nexus S 手机。

正如之前解释的那样，当使用含 TRF7970A 的Android 手机上的 NFC 的时候，可以看到轮询正在发生。MSP430 代码中使用的实例陈述用来处理这些IRQ：当场改变并且生成 IRQ时，此IRQ被处理，然后寄存器 0x19 状态被读取。

对于此文档，值 0xC5 被返回到寄存器 0x19 这一事件是启动点或者触发点，这些是图 2之后的细节。

B.1 ISO14443B 命令

被返回到寄存器 0x19 的值 0xC5 (110001012) 说明此命令（来自手机）曾经是106kbps 的ISO14443B 类命令（完全寄存器定义请参见 TRF7970A 数据表）。 然后 FIFO 状态寄存器 (0x1C) 被读取，返回的值说明在 TRF7970A FIFO 中共有 4 个字节在等待。 从 FIFO 将这些字节检索出来，这是被称为 REQB 的 ISO14443-3 B类命令。

（值为：**APF = 0x05**，**AFI = 0x00** （所有系列和子系列），**PARAM = 0x00** （不被 **PCD** 和 **REQB** 所支持的扩展 **ATQB** 被发布，单槽被发布)，**CRC_B = 0x00)**

然后 FIFO 被复位。

然后 TRF7970A 必须按照 ISO14443-3 规范对这条命令做出响应。 这称为对 B 或者 ATQB 的应答。

由于扩展 ATQB 不被来自手机的 REQB 命令的 PARAM 字节所请求，所以 MSP430 中控制TRF7970A 的固件代码（在这个示例中）被设计为使用基本 ATQB 格式进行响应（参见ISO/IEC14443-3 技术规范中的图表 24）。

图 7显示了这一响应格式。

1st byte	2nd, 3rd, 4th, 5th bytes	6th, 7th, 8th, 9th bytes	10th, 11th, 12th bytes	13th, 14th bytes
'50' (1 byte)	PUPI (4 bytes)	Application Data (4 bytes)	Protocol Info (3 bytes)	CRC_B (2 bytes)

图 7. 响应格式，基本 ATQB

根据之前的陈述，真正的响应是：

- 第 1 字节：0x5
- 第 2 字节到第 5 字节：0x80，0x12，0x34，0x56 = PUPI
- 第 6 字节到第 9 字节：0x40，0xE2，0xAF，0x11 = 应用数据
- 第 10 字节到第 12 字节：0x80，0x71, 0x85 = 协议信息
- 由于读取器自动生成这些字节并且将它们无线发回手机，第 13 和第 14 字节 (CRC_B) 并不由MSP430 代码进行处理或者在数字域中可见（通过逻辑分析器）。

正如任何其它通过无线向外发射的命令一样，MSP430 中控制 TRF7970A 的代码必须作为这个响应的开头。 这意味着 FIFO 必须被复位，发送带有 CRC的 直接命令字节，从 FIFO 持续写入，并且长度字节是此前缀的一部分。 参见图 8中的示例，此示例直接显示了这个意思，但是这里最多只能显示到 ATQB 的第 7 个字节(0xE2)。

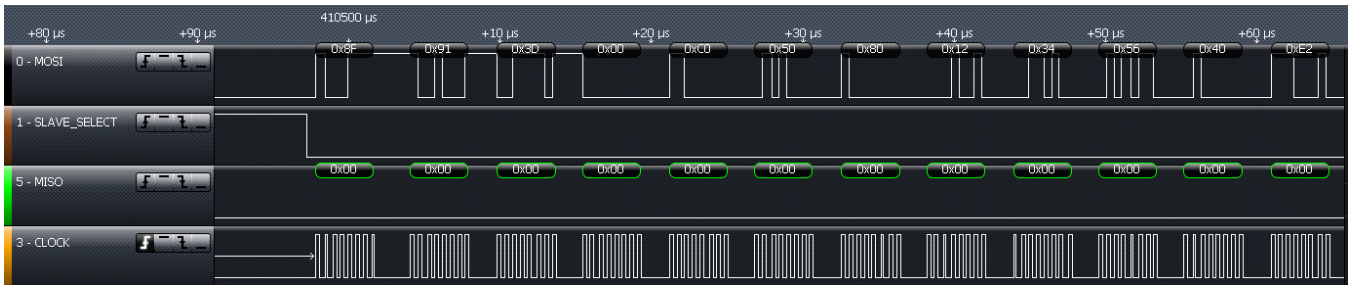


图 **8. UART** 通信，基本 **ATQB**

针对应用数据和协议信息字节（依据ISO/IEC14443-3规范）的所发送的值（在此示例中）定义如下：

应用数据（**4** 字节）：

- 0x40 = AFI 对于公共电话/GSM
- 0xE2，0xAF = CRC_B (AID)
- 0x11 = 目前应用的 # = 1 应用

协议信息（**3**字节）：

- 0x80 = 数据速率能力（支持106kbps）
- 0x71 = 协议信息（最大帧/协议类型）（128字节/兼容 PICC 到 -4）
- 0x85 = (FWI/ADC/FO) (FWT = 77.3 ms，ADC = 根据AFI进行编码，支持CID

TX 完整 IRQ 被接收并被处理并且 FIFO 被复位。

RX 完整 IRQ 被接收并被处理，FIFO 状态寄存器被读取。 返回的值标示 FIFO 中有 10 个字节正在等待。

这 10 个字节（来自手机）从 FIFO 时钟输出并且它们是被称为 ATTRIB 的 ISO14443B 命令。 字节值和其定义（根据 ISO 标准）如下：

- 0x1D → ATTRIB 命令开始字节（通过TRF7970A 由手机到 MCU）
- 0x80 → NFC ID / PUPI（标识符）
- 0x12 → NFC ID（标识符）
- 0x34 → NFC ID（标识符）
- 0x56 → NFC ID / PUPI（标识符）
- 0x00 →参数 1（0x00 = TR0，TR1，和所需SOF/EOF 的默认值）
- 0x08 →参数 2（读取器告诉 PICC 最大的帧尺寸为 256，双向比特率为 106kbps）
- 0x01 → 参数 3（PICC -4 兼容的 ACK）
- 0x00 → 参数 4 (CID = 0)
- 0x00 → 更高层 - INF 字节（可选场）

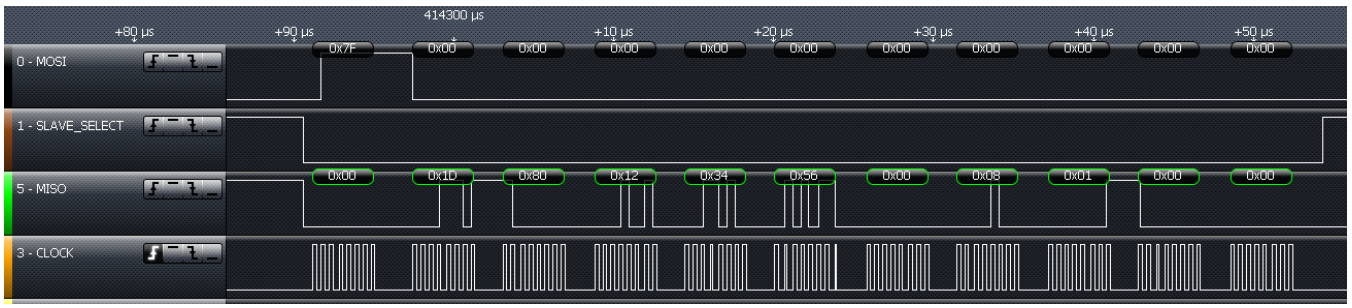


图 **9. UART** 通信，**FIFO** 状态寄存器读取

FIFO 被复位，然后对 ATTRIB 的应答被发回手机。 由于 ATTRIB 命令中没有更高层的请求 (-INF)，此应答，正如规定的那样，为 1 个字节，加上 2 个字节的 CRC_B（由读取器生成并且在数字域中不可见）。

此响应（无更高层）的规定格式显示在图 10中：

1st byte		2nd, 3rd bytes
MBLI	CID	CRC_B
(1 byte)		(2 bytes)

图 **10.** 对 **ATTRIB** 格式的应答

字节值和其定义（符合ISO标准）显示如下：

- 0x00 → MBLI + CID，在这里 MBLI = 0 意味着 TRF7970A（工作为 PICC）在它的内部缓冲器尺寸上没有提供任何信息并且 CID = 0，这与手机发送的数据项匹配，所以这是手机 PCD 验证 TRF7970A（工作为PICC）是否已经被成功选择的方式。

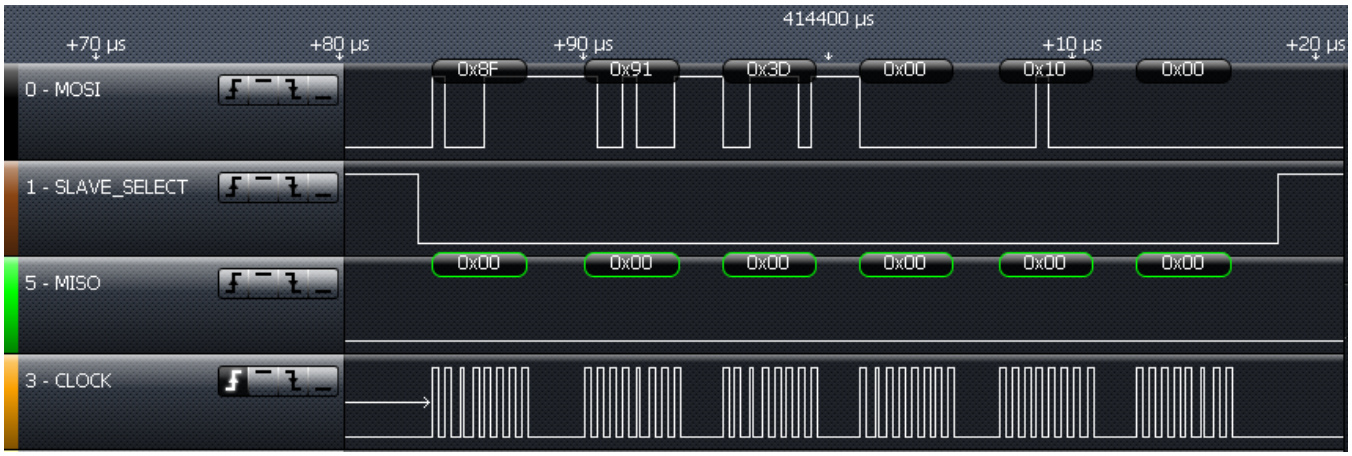


图 **11. UART** 通信，对 **ATTRIB** 应答

B.2 检测 NDEF 信息

1. TX 完成 IRQ 被接收和处理
2. 寄存器 0x19 被重新读取并且 FIFO 被复位。
3. RX 完成 IRQ 被接收和处理。
4. 对 FIFO 状态寄存器的读取值表明 FIFO 中有 2 个字节正在等待。
5. 这些字节被读出，它们是：**0xB2，0x80 (ISO14443-4 R (NAK)**消息，用于检查标签是否有效 **-** 这来自**ISO14443-4** 的规则**12**（**PICC** 规则）。
6. FIFO 被复位
7. 对这个进行的响应是：**0xA2** （**ISO14443-4 R(ACK)，'Ping'** 响应）
8. TX 完整 IRQ 被接收并处理，寄存器 0x19 仍为0xC5。 FIFO 被复位。
9. RX 完成 IRQ 被接收和处理。
10. 对 FIFO 状态寄存器的读取值表明 FIFO 中有 2 个字节正在等待。
11. 这些字节被读出，它们是：**0xC2, 0x80** （**ISO 14443-4 S** （取消选择**(DESELECT)**） 请求）。
12. FIFO 被复位。
13. 对此的响应为：**0x02** （对 **ISO14443-4 S (DESELECT)** 请求的确认**(ACK)**）。

14. TX 完成 IRQ 被接收，处理并且寄存器 0x19 仍为 0xC5。 FIFO 被复位。
15. RX 完成 IRQ 被接收和处理。
16. 对 FIFO 状态寄存器的读取值表明 FIFO 中有 4 个字节正在等待。
17. 这些字节被读出，它们是：**0x05，0x00，0x08，0x00 (WupB)**。
18. FIFO 被复位。
19. 对此命令的响应正在重新发送回 ATQB。
20. 从手机发出 ATTRIB。
21. 从TRF7970A 发出对 ATTRIB 的应答。
22. 来自手机的 RX 为 15 个字节： **0x02, 0x00, 0xA4, 0x04, 0x00, 0x07, 0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01, 0x00, 0x00**（检测 **NDEF** 消息，部分**5.4.2**，**4** 类标签操作规范，表**10**，**C-APDU**）。
23. FIFO 被复位。
24. 响应发出：0x02, 0x90, 0x00（表**12, SW1**，**SW2** 命令完整）。
25. TX 完成 IRQ 被接收，处理，FIFO 被复位。
26. 来自手机的 RX 为 9 个字节： **0x03, 0x00, 0xA4, 0x00, 0x0C, 0x02, 0xE1, 0x03, 0x00**（**NDEF** 功能容器选择过程，表**13**，**C-APDU**，来自 **4** 类标签操作规范 **5.4.3**）。
27. FIFO 被复位。
28. 响应发出： **0x03, 0x90, 0x00**（表**15**，**SW1**，**SW2**命令完整）。
29. TX 完成 IRQ 接收并处理。 FIFO 被复位。
30. 来自手机的 RX 为 7 个字节： **0x02, 0x00, 0xB0, 0x00, 0x00, 0x0F, 0x00 (**从功能容器文件读取二进制数据命令（表**16**，**C-APDU**，部分**5.4.4))**。
31. FIFO 被复位。
32. 响应发出： **0x02, 0x00, 0x0F, 0x20, 0x00, 0x40, 0x00, 0x40, 0x04, 0x06, 0xE1, 0x01, 0x08, 0x00, 0x00, 0xFF, 0x90, 0x00**（预期响应，**SW1** 和 **SW2** 完整）
33. TX 完成 IRQ 接收并处理。 FIFO 被复位。
34. 来自手机的 RX 为 9 字节： **0x03, 0x00, 0xA4, 0x00, 0x0C, 0x02, 0xE1, 0x01, 0x00** （**NDEF** 选择命令（表**19**，**C-APDU**，部分**5.4.5**））。
35. FIFO 被复位。
36. 响应发出：**0x03, 0x90, 0x00** （**SW1**， **SW2** 命令完整）
37. TX 完成 IRQ 接收和处理。 FIFO 被复位。
38. 来自手机的 RX 为 7 个字节： **0x02, 0x00, 0xB0, 0x00, 0x00, 0x02, 0x00**（从**NDEF** 文件的 **NLEN** 域中读取二进制数）
39. FIFO 被复位。
40. 响应发出： **0x02，0x00，0x1B，0x90，0x00**（在这里 **0x1B** 是 **NDEF** 消息的长度，**SW1** 和 **SW2** 完整）
41. TX 完成 IRQ 接收和处理。 FIFO 被复位。
42. 来自手机的 9 个字节 RX： **0x03, 0x00, 0xA4, 0x00, 0x0C, 0x02, 0xE1, 0x01, 0x00**（**NDEF** 选择命令（表**19**，**C-APDU**，部分**5.4.5**））。
43. FIFO 被复位。
44. 响应发出：**0x03, 0x90, 0x00** （**SW1**，**SW2** 命令完整）。
45. TX 完成 IRQ 接收和处理。 FIFO 被复位。
46. 来自手机的 RX 为 7 个字节： **0x02, 0x00, 0xB0, 0x00, 0x00, 0x1B, 0x00**（从**NDEF** 文件中读取数据，部分**5.4.6**）。
47. FIFO 被复位
48. 响应输出：0x02, 0x00, 0x1B, 0xD1, 0x01, 0x17, 0x55, 0x00, **0x44, 0x38, 0x3A, 0x35, 0x34, 0x3A, 0x33, 0x41, 0x3A, 0x34, 0x39, 0x3A, 0x30, 0x44, 0x3A, 0x41, 0x46**, 0x20, 0x00, 0x00, 0x90, 0x00 → 这是十六进制的 NDEF 消息（参见5.4.6部分），在ASCII中：

    (a) 0x02 = STX（文本开始）

    (b) 0x00 = NULL

(c) 0x1B = NDEF 消息长度
(d) 0xD1 = NDEF 消息域
(e) 0x01 = SOH（头开始）
(f) 0x17 = ETB（传输块终点）
(g) 0x55 = U
(h) 0x00 = NULL

注: 项目 (i) 到 (y) 是蓝牙无线设备的 MAC 地址（此例中为 D8:54:3A:49:0D:AF）

(i) 0x44 = D
(j) 0x38 = 8
(k) 0x3A = :
(l) 0x35 = 5
(m) 0x34 = 4
(n) 0x3A = :
(o) 0x33 = 3
(p) 0x41 = A
(q) 0x3A = :
(r) 0x34 = 4
(s) 0x39 = 9
(t) 0x3A = :
(u) 0x30 = 0
(v) 0x44 = D
(w) 0x3A = :
(x) 0x41 = A
(y) 0x46 = F
(z) 0x20 = 空格 (SPACE)
(za) 0x00 = 空 (NULL)
(zb) 0x00 = NULL
(zc) 0x90 = SW1 完成
(zd) 0x00 = SW2 完成

49. TX 完成 IRQ 被接收并处理。 FIFO 被复位。
50. 来自手机的 7 个字节 RX：**0x03, 0x00, 0xB0, 0x00, 0x1B, 0x02, 0x00** （为**NLEN** 域读取二进制数）。
51. FIFO 被复位。
52. 响应输出： **0x03，0x2F，0x20，0x90，0x00**。
53. TX 完成 IRQ 接收和处理。 FIFO 被复位。
54. 来自手机的 2 个字节 RX： **0xC2, 0x2F**（**ISO 14443-4 S-DESELECT** 请求）。
55. 来自手机的 1 个字节 RX：**0xC2**（**ISO 14443-4 S-DESELECT** 响应）。
56. 手机场复位（从 TRF7970A 接收到的 IRQ 将此标示为状态 = 0x04）。
57. 寄存器 0x19 读取值0x05 = 00000101 = 106kbps 的 B 类命令。
58. TRF7970A 重新初始化序列开始。
59. 此时，如果 TI BT NFC 应用程序或者ISO14443B 标签 + NDEF 消息被处理，此手机将被连接并与 BT 无线电设备配对，如果在此手机上运行其它标签读取应用程序（例如，NFC TagInfo）的话，则可随时查看此手机的连接状态。

重要声明

德州仪器(TI) 及其下属子公司有权在不事先通知的情况下, 随时对所提供的产品和服务进行更正、修改、增强、改进或其它更改，并有权随时中止提供任何产品和服务。客户在下订单前应获取最新的相关信息，并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的硬件产品的性能符合TI 标准保修的适用规范。仅在TI 保证的范围内，且TI 认为有必要时才会使用测试或其它质量控制技术。除非政府做出了硬性规定，否则没有必要对每种产品的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用TI 组件的产品和应用自行负责。为尽量减小与客户产品和应用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何TI 专利权、版权、屏蔽作品权或其它与使用了TI 产品或服务的组合设备、机器、流程相关的TI 知识产权中授予的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从TI 获得使用这些产品或服务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是TI 的专利权或其它知识产权方面的许可。

对于TI 的产品手册或数据表，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况下才允许进行复制。在复制信息的过程中对内容的篡改属于非法的、欺诈性商业行为。TI 对此类篡改过的文件不承担任何责任。

在转售TI 产品或服务时，如果存在对产品或服务参数的虚假陈述，则会失去相关TI 产品或服务的明示或暗示授权，且这是非法的、欺诈性商业行为。TI 对此类虚假陈述不承担任何责任。

TI 产品未获得用于关键的安全应用中的授权，例如生命支持应用（在该类应用中一旦TI 产品故障将预计造成重大的人员伤亡），除非各方官员已经达成了专门管控此类使用的协议。购买者的购买行为即表示，他们具备有关其应用安全以及规章衍生所需的所有专业技术和知识，并且认可和同意，尽管任何应用相关信息或支持仍可能由TI 提供，但他们将独力负责满足在关键安全应用中使用其产 品及TI 产品所需的所有法律、法规和安全相关要求。此外，购买者必须全额赔偿因在此类关键安全应用中使用TI 产品而对TI 及其 代表造成的损失。

TI 产品并非设计或专门用于军事/航空应用，以及环境方面的产品，除非TI 特别注明该产品属于“军用”或“增强型塑料”产品。只 有TI 指定的军用产品才满足军用规格。购买者认可并同意，对TI 未指定军用的产品进行军事方面的应用，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

TI 产品并非设计或专门用于汽车应用以及环境方面的产品，除非TI 特别注明该产品符合ISO/TS 16949 要求。购买者认可并同意，如果他们在汽车应用中使用任何未被指定的产品，TI 对未能满足应用所需要求不承担任何责任。

可访问以下URL 地址以获取有关其它TI 产品和应用解决方案的信息：

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP 机动性处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity		
	德州仪器在线技术支持社区	www.deyisupport.com	

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122