Tiny4412 Android 硬件开发指南

2014-05-15

(本手册适用于 Tiny4412 开发板)

Copyright © 2007-2014 FriendlyARM

All rights reserved.

简介

本手册由广州友善之臂计算机科技有限公司(简称“友善之臂”)创建和维护，并作为标准用户手册的一个补充，仅供用户学习参考使用，友善之臂目前并不对本手册的内容提供任何解释和解答服务，用户可以在论坛中反馈你所遇到的问题和疑问，我们将在以后的更新中修正或者采纳您的建议，本手册主要以首页日期为版本标志。

本手册由友善之臂软件开发工程师编写制作，以 Tiny4412 作为开发平台，讲解如何建立 Android 开发环境，以及如何**使用 Tiny4412 开发板作为真机调试程序**，以及如何在 Android 程序中访问硬件资源,包括 I2C, GPIO, SPI, 串口等资源, 为你的项目开发提供参考。

为了方便用户，本手册的所用到的软件包放在光盘 B 的 tools 目录下，手册中的示例代码，完整的工程都放在光盘 A 的 Android/examples 目录的，供参考。

我们欢迎各位网友复制传播本手册，但不得擅自摘抄部分或全部内容用作商业用途，违者必究，友善之臂保留本手册的解释和修改权。

友善之臂公司网址：http://www.arm9.net

本手册由ARM9 之家论坛(http://www.arm9home.net)发布，转载请注明出处，手册内难免有遗漏和不足之处，欢迎大家提出宝贵意见，请发邮件至：dev_friendlyarm@163.com。

FriendlyARM

更新说明

2014-05-15	加上看门狗示例
2014-04-12	本手册第一次发布

FriendlyARM

目录

FriendlyARM

FriendlyARM

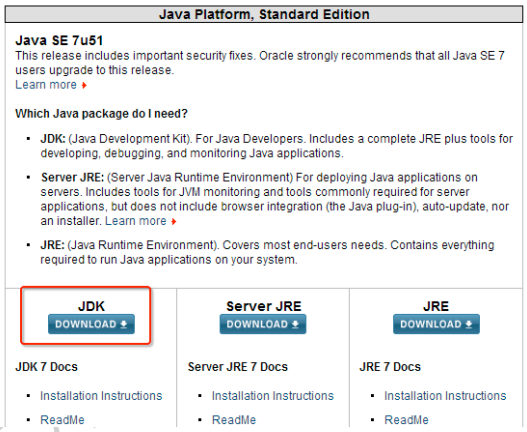
第一章　建立Android应用开发环境

本章节将介绍如何在 Windows7 系统中搭建 Android 开发环境,在文中所用到的所有软件包都可以在光盘 B 中的 tools 目录上找到。

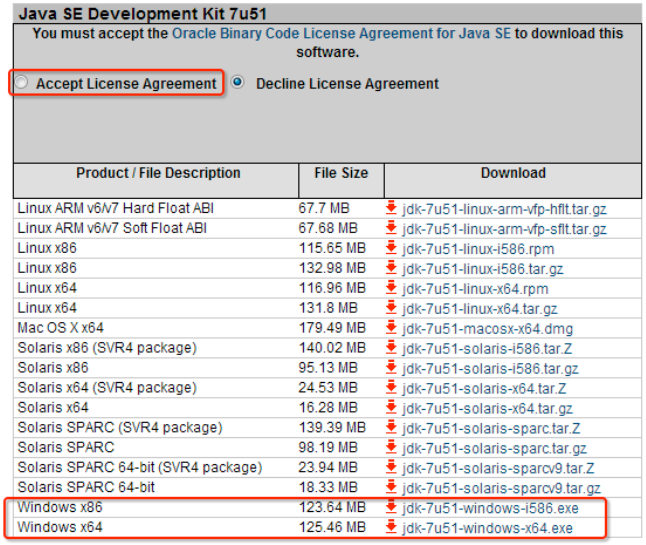
1.1 步骤一：下载并安装JDK (Java SE Development Kit)

由于 Android SDK 和 ADT 集成开发环境都是用 Java 编写的，因此需要先在 Windows 7 上安装 JDK，JDK 可按以下方法下载：

访问网站http://www.oracle.com/technetwork/java/javase/downloads/index.html，在页面中点击JDK：



在打开的页面中，选中 Accept License Agreement：



Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement　Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

接着根据平台选择 jdk 下载链接，Windows7 64bit 版本是点击 jdk-7u51-windows-x64.exe 下载 JDK 的安装程序，下载完成后，双击安装程序,根据向导的提示完成安装即可。

安装完成后，**需要将 JDK 命令添加到 Path 环境变量中**，通过下面的方法将 JDK 命令所在的路径添加到 Path 环境变量中：

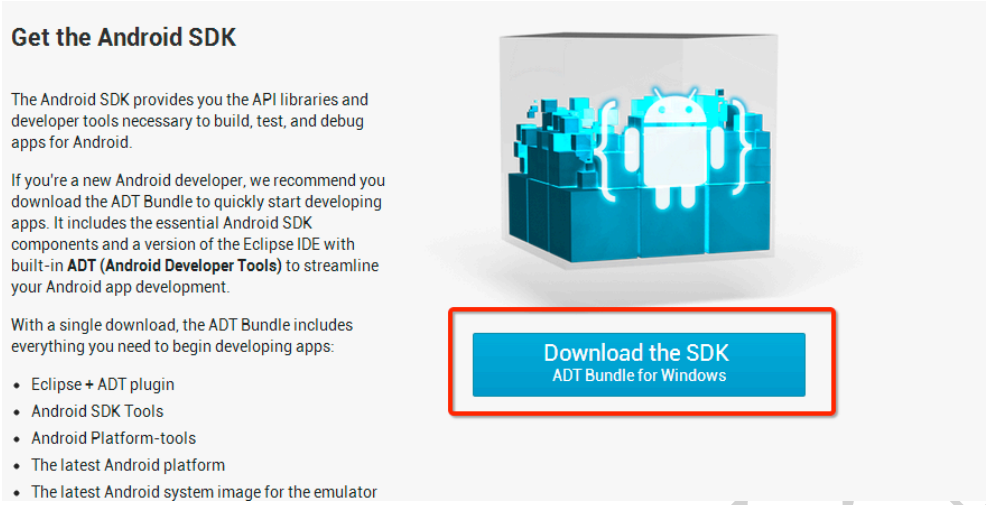
1) 右击“我的电脑”->属性，再选择左边导航的“高级系统设置”选项。
2) 点击右下角的“环境变量”选项。
3) 在"系统变量"中，找到 Path　环境变量，双击它，在变量值前面追加以下内容：“C:\Program Files\Java\jdk1.7.0_51\;”，注意后面有一个分号
4) 点击“确定”完成环境变量设置。

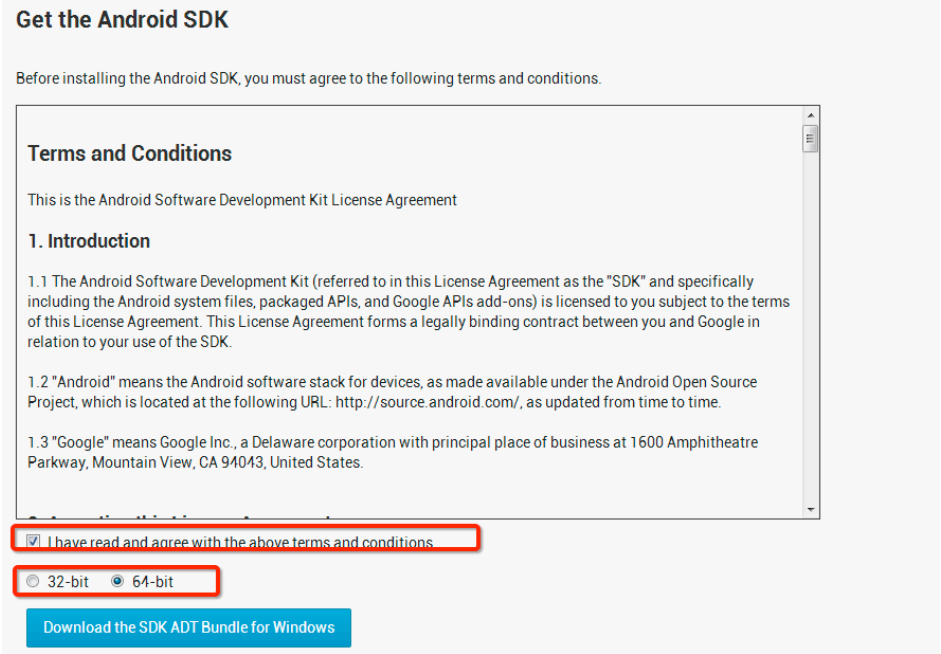
环境变量设置成功后,在 DOS 命令行下执行 java –version 应该能得到如下输出:

```
C:\>java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)

C:\>
```

1.2 步骤二：下载并安装ADT集成开发环境和Android SDK

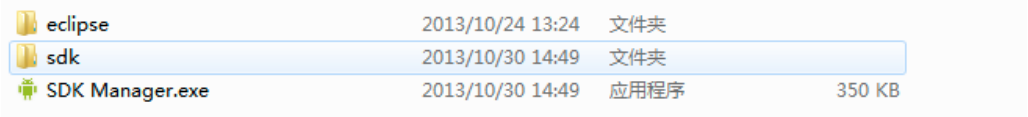
首先，前往从网站http://developer.android.com/sdk/获取Windows平台的最新的Android SDK 套件，下载完成会得到一个压缩包，下载页面如下图所示，请下载红色方框框出的文件：





需要根据你的 Windows 版本来确定下载 32-bit 版本还是 64-bit 版本,下载的好处是保证能获得最新版本，但不想或无法下载的用户，也可以使用光盘上的版本，位于光盘 B 的 tools 目录下，文件名为 adt-bundle-windows-x86_64-20131030.zip，是 64-bit 的版本。

将下载得到的压缩包解压，可以看到包含以下目录:

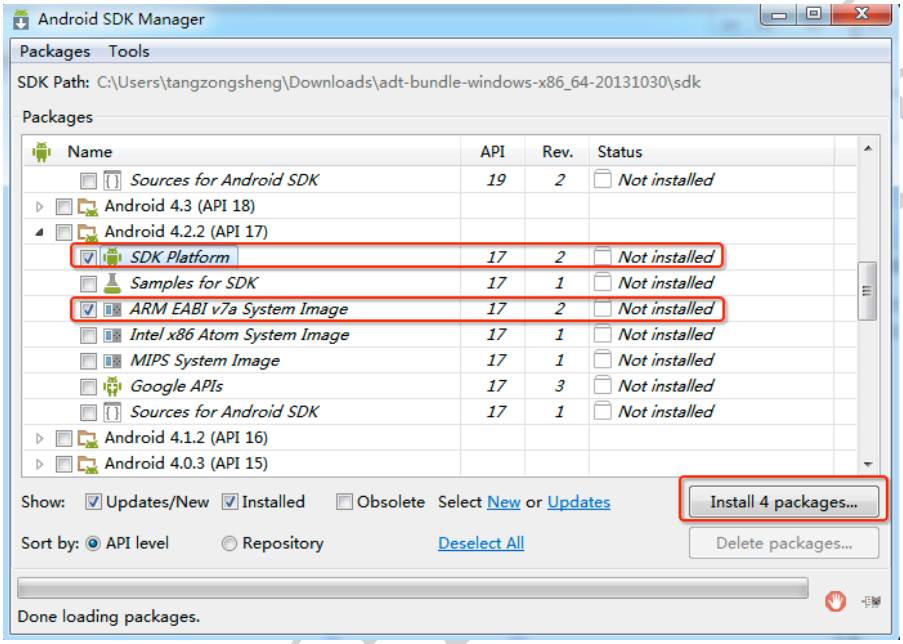


可以看到，压缩包中包含了 eclipse 和 Android SDK，Google 使用 eclipse 为基础打造了一

个 Android 集成开发环境，命名为 Android Developer Tools，简称 ADT。

1.2.1 下载Android SDK (API 17)

由于 Tiny4412 的 Android 版本是 4.2.2，为了方便在 Tiny4412 上调试程序，我们需要下载安装 Android 4.2.2 的 SDK (若开发板的 Android 系统有升级，请下载相应版本的 SDK,这里不再另行说明)，方法是双击 SDK Manager 启动 SDK 管理程序 Android SDK Manager，在 Android SDK Manager 中选中 Android 4.2.2 (API 17) 中的 SDK Platform 组件和 ARM EABI v7a System Image 组件,如下图所示:

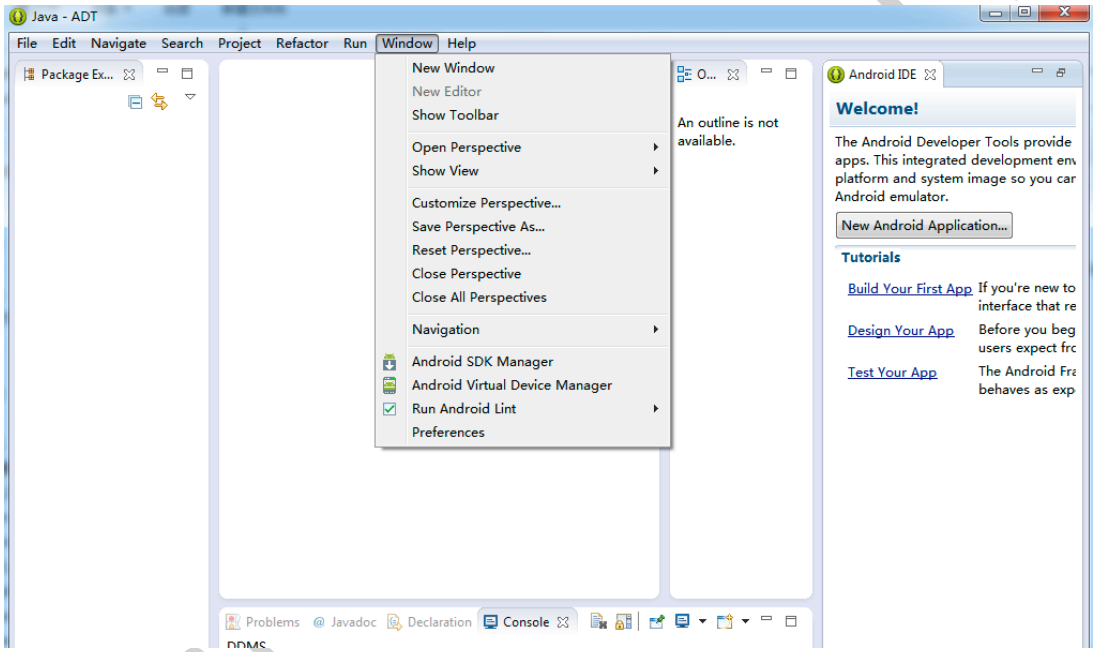


在上面的对话框中点击“Install 4 packages…”按钮，在弹出的 Choose Packages to Install”对话框中，选中“Accept License”单选框，点击“Install”按钮，将进入下载过程，下载速度视你的网速而定，请耐心等待下载完成。

1.2.2 启动ADT集成开发环境 (Android Developer Tools)

进入上一章节解压得到的 **adt-bundle-windows-x86_64-20131030 目录，再进入** eclipse 目录，双击 eclipse.exe 即可启动 ADT，启动封面如下图所示：

启动时，首先会让你设置 Workspace 的路径，这个路径用于保存你所创建所有程序的源代码，不修改直接点击 OK，之前会询问你是否发送使用数据给 Google，可以选择 No，然后点击 Finish 进入 ADT 的主界面，如下图所示：



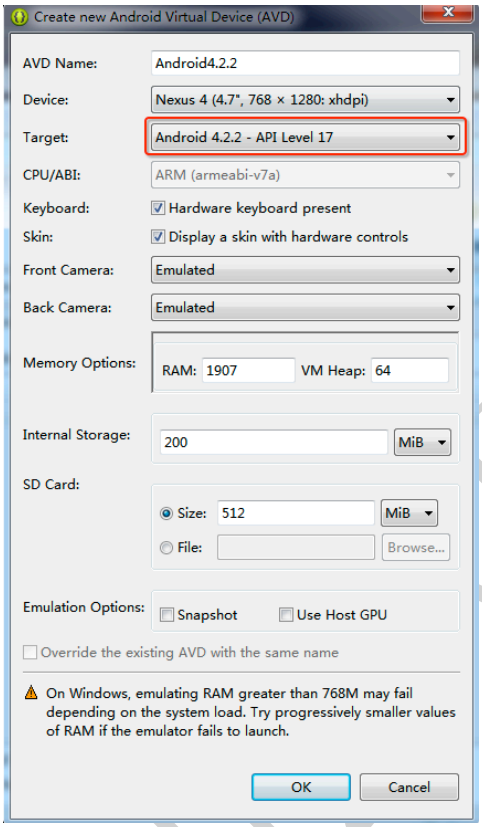
Eclipse 还是 ADT？

ADT 是 Google 使用 Eclipse 来构建的，它在 Eclipse 上面为我们安装并配置好了 Android SDK 以及 Android 开发所需的 Eclipse 插件，因此，ADT 本质上就是 Eclipse+Android SDK+ADT Plugin，因此不引起混乱，本文在接下来的内容中，使用 ADT 这个名字。

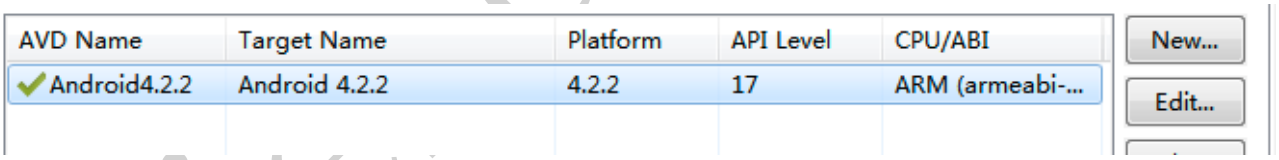
1.3 步骤三：创建Android模拟器

在 ADT 中，点击 Windows 菜单，然后点击 Android Virtual Device Manager 来启动模拟器管理器，点击“New...”按钮，将弹出“Create new Android Virtual Device(AVD)”对话框。

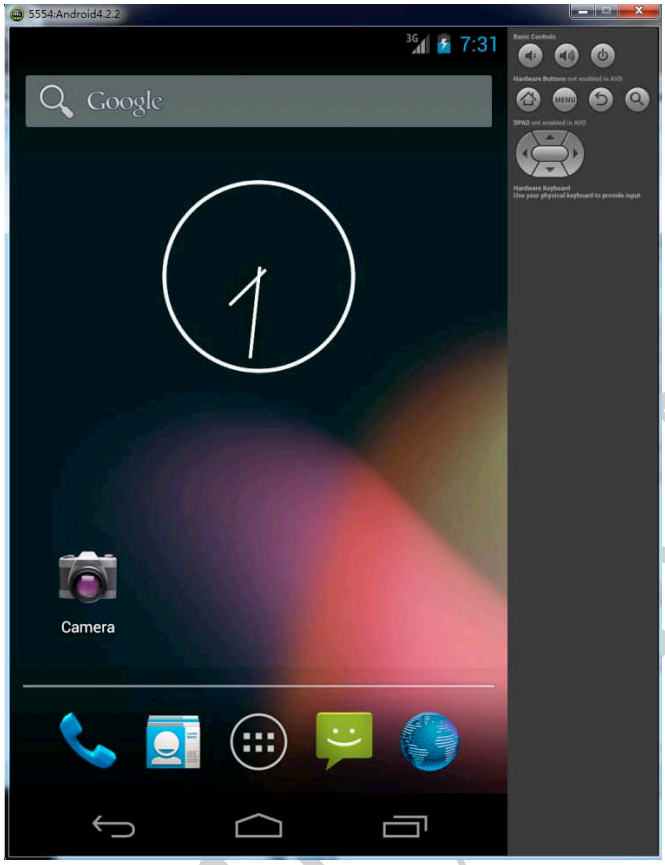
在“Create new Android Virtual Device(AVD)”对话框中，你可以根所你的喜欢进行配置，本文测试时使用的配置如下图所示：



点 OK 即可创建模拟器，完成后将在列表中列出该模拟器，如下图所示：



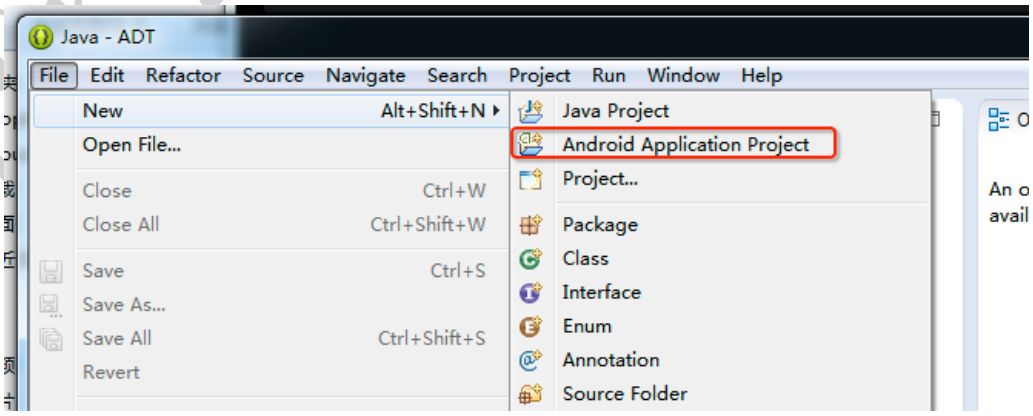
可在列表中选中它，然后点击 Start，在弹出的 Launch Option 对话框中点击 Launch 启动模拟器，效果如下所示：



1.4 步骤四：开发第一个Android程序 (验证开发环境是否搭建成功)

1.4.1 创建HelloWorld工程

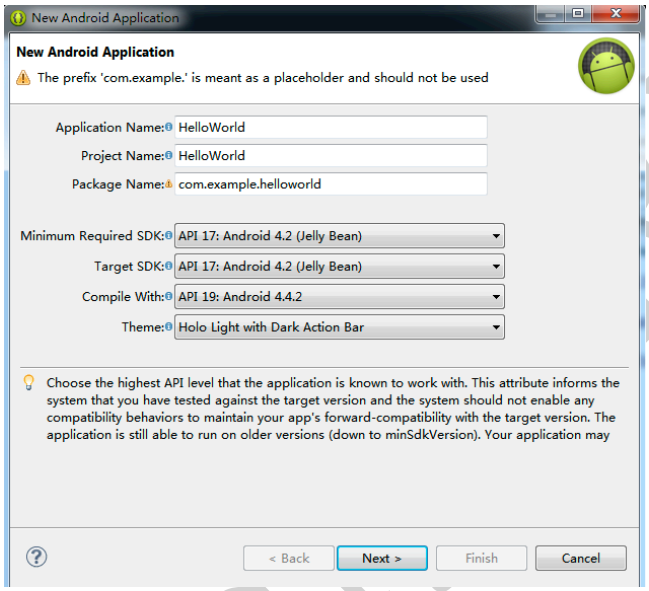
接下来我们新建一个 Android 项目 HelloWorld，以验证开发环境是否搭建成功。在 ADT 主界面上依次选择菜单：File->New->Android Application Project 如下图所示：



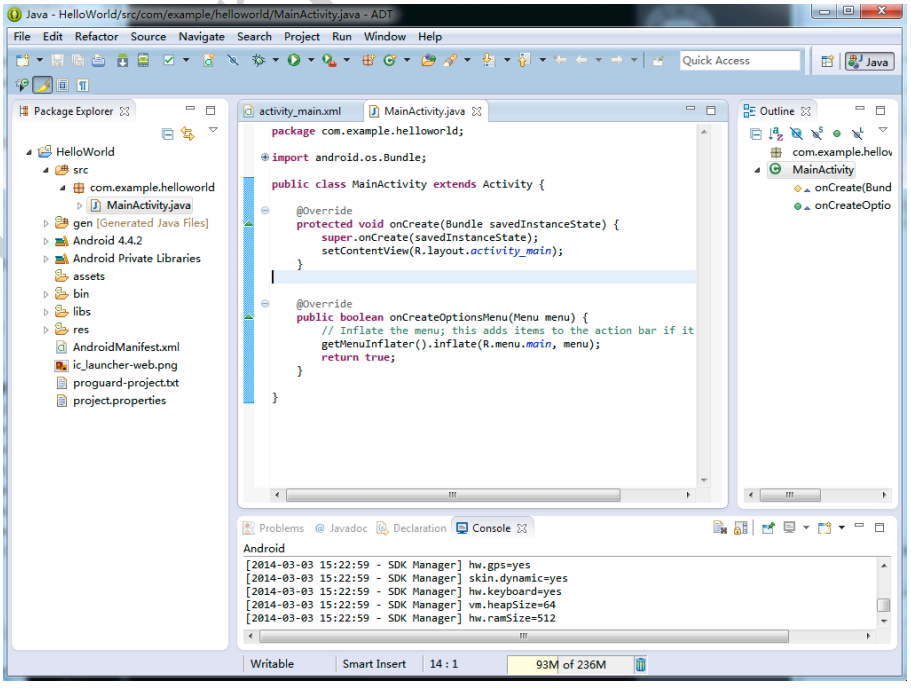
界面上将弹出“New Android Project”对话框，在对话框中进行如下输入：

1) Appication name 中输入：HelloWorld
2) Project Name 中输入：HelloWorld
3) Package name 中输入：com.example.helloworld
4) Build Target 中选择：API 17: Android 4.2 (Jelly Bean)
5) Target SDK 中选择：API 17: Android 4.2 (Jelly Bean)

如下图所示:



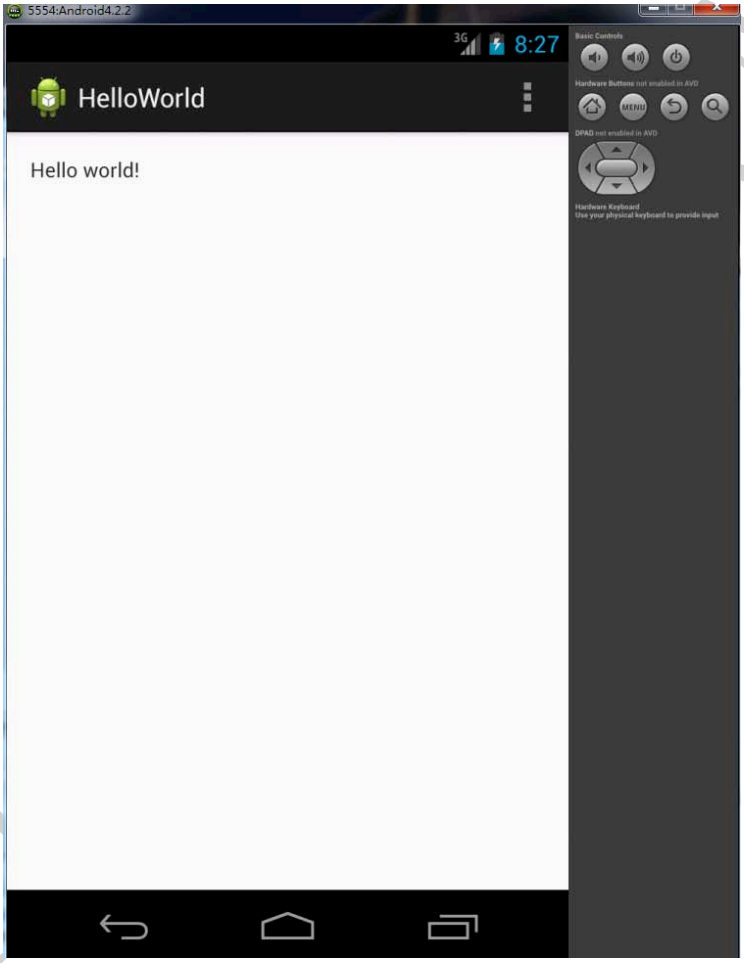
输入完成后，一路点 Next 最后点“Finish”完成新建项目向导并回到主界面：



1.4.2 在模拟器运行Android程序

要编译并运行 HelloWorld 程序，先在 Package Explorer 中选中 HelloWorld 工程名称，然后点击工具栏的运行按钮，或选择菜单：Run->Run As->Android Application 即可。

ADT 将会自动启动 Android 模拟器，启动过程因为需要启动 Android 系统，所以要耐心等待，Android 系统启动完成后，将会自动运行 HelloWorld 程序，如下图所示：



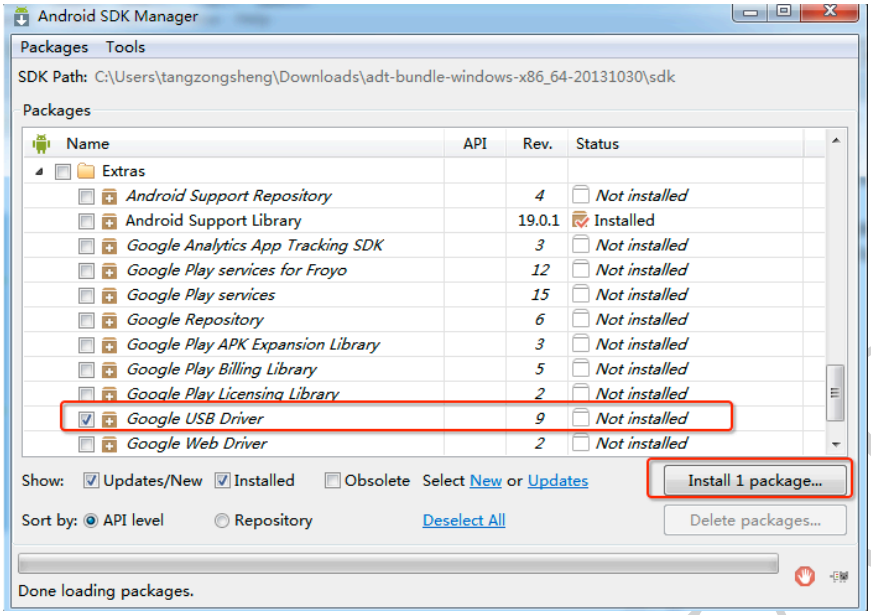
在接下来的章节，我们将实现把这个 HelloWorld 程序在开发板上调试和运行。

1.5 步骤五：建立Tiny4412 调试环境

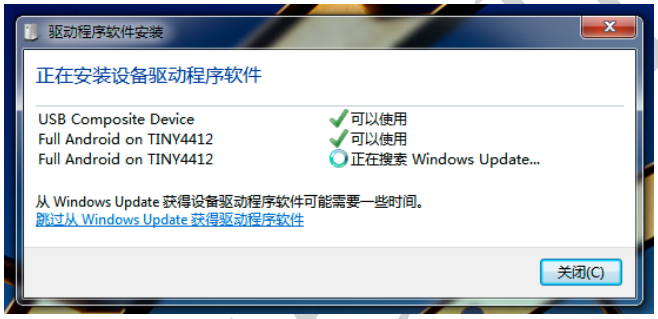
1.5.1 安装USB ADB驱动程序

在 ADT 界面上，点击 Window 菜单中的 Android SDK Manager 启动 SDK Manager，在

Android SDK Manager 的主界面上找到 Extras，查看一下 Google Usb Driver 的状态，如是不是 Installed 而是 Not installed，则参考下图选中 Google USB Driver，再点击 Install 1 package…进行安装：



安装完成后，将 Tiny4412 开机，在 Android 启动完毕后，插入 MiniUSB 线与 PC 相连，这时，Windows7 会提示正在安装驱动程序：



稍等片刻即可安装完成。

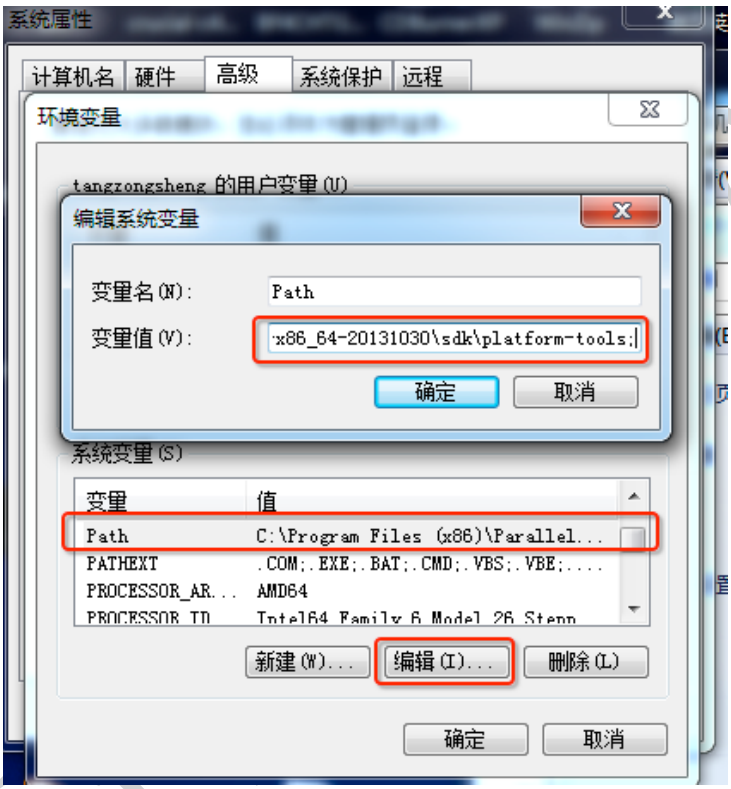
如果安装失败，可到我的电脑设备管理器中，找到 Tiny4412 的设备然后右击，选择“更新驱动程序软件”，在弹出的对话框中选择“浏览计算机上的驱动程序文件”，再点击“浏览”，在 Android SDK 安装路径中选择 USB 驱动程序的路径，位于你安装的 SDK 的以下子目录：“sdk\extras\google\usb_driver”，选择路径后点击“下一步”顺着向导进行安装即可。

1.5.2 在Tiny4412 上测试ADB功能

1.5.2.1 将adb命令添加到Path环境变量中

通过下面的方法将 adb 命令所在的路径添加到 Path 环境变量中：

1) 右击“我的电脑”->属性，再选择左边导航的“高级系统设置”选项。
2) 点击右下角的“环境变量”选项。
3) 在"系统变量"中，找到 Path 环境变量，双击它，在变量值前面追加 **adt-bundle 的路径，由于我将压缩包在 c:\根目录解压，因此，我添加的**路径如下：“C:\adt-bundle-windows-x86_64-20131030\sdk\platform-tools;”，注意后面有一个分号，如下图所示：



4) 点击“确定”完成环境变量设置。

顺便提一下，如果你将 ADT 放在 C 盘根目录而不是用户目录，则运行 SDK 管理器和 ADT 都需要以管理员的身份启动。

测试一下是否找到 adb 命令

通过点击开始菜单，在开始菜单下方的搜索框中输入 cmd，在 cmd.exe 上按回车来启动 DOS 窗口，在 DOS 窗口中，输入 adb 按回车，如果显示以下信息表示环境变量设置 OK：

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\tangzongsheng>adb
Android Debug Bridge version 1.0.31

 -a                            - directs adb to listen on all interfaces for a
onnection
 -d                            - directs command to the only connected USB devi
e
                                 returns an error if more than one USB device i
 present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is
unning.
 -s <specific device>          - directs command to the device or emulator with
the given
                                 serial number or qualifier. Overrides ANDROID_
ERIAL
                                 environment variable.
```

1.5.2.2　测试ADB的功能

查看设备连接状态

在 Tiny4412 上启动 Android，然后用 mini USB 线将 Tiny4412 与 PC 相连，在 DOS 窗口上输入以下命令验证开发板是否已连接：

```
# adb devices
```

显示以下内容表示成功连接到 Tiny4412 设备：

```
C:\>adb devices
adb server is out of date.  killing...
* daemon started successfully *
List of devices attached
0123456789ABCDEF        device


C:\>_
```

进入 ADB Shell

使用以下命令可以进入 Tiny4412 终端：

```
# adb shell
```

终端上输入 exit 可退回到 DOS 提示符。

用 ADB 安装软件

以安装 D:\sinaweibo_2.0.4.apk 的程序为例，在 DOS 窗口中输入 **adb install D:\sinaweibo_2.0.4.apk** 进行安装。

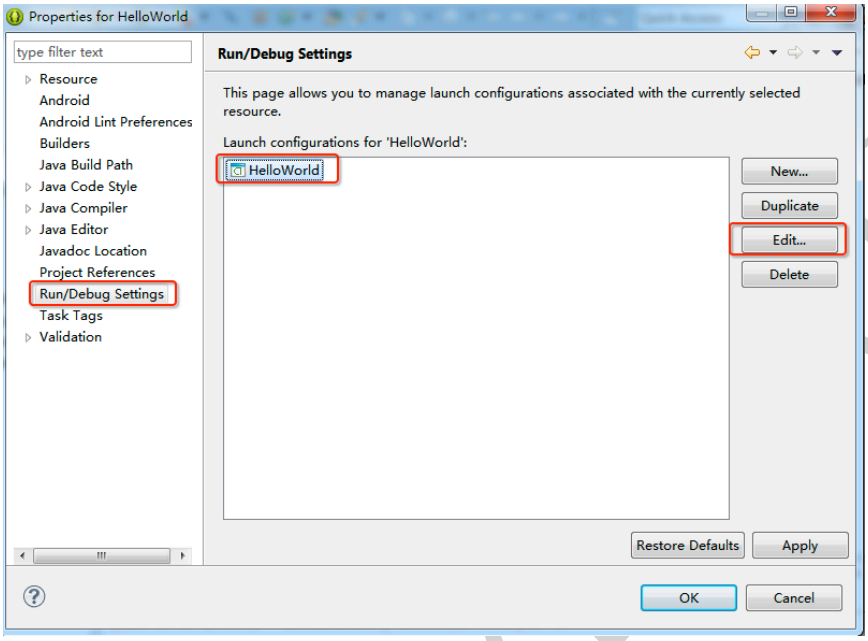
其它功能

ADB 功能非常强大，除了安装软件、调试、Shell 功能外，还可以往开发板上传送文件等，读者可以自己挖掘。

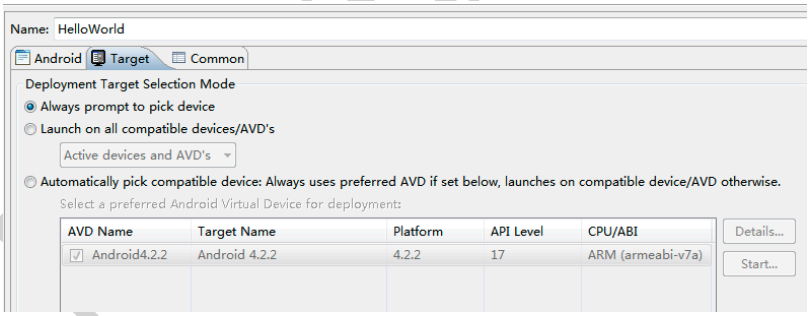
1.5.3 通过USB ADB在Tiny4412 上运行程序

双击 eclipse/eclipse.exe 启动 ADT，将自动打开 HelloWorld 工程，如果没有打开，可手动打开。

在 ADT 主界面左侧的 Package Explorer 中右击 HelloWorld 项目，点 Properties，将弹出 Properties for HelloWorld 窗口，在窗口中点击“Run/Debug Settings”，选择中间列表中的 HelloWorld，然后在右边点击“Edit…”按钮:

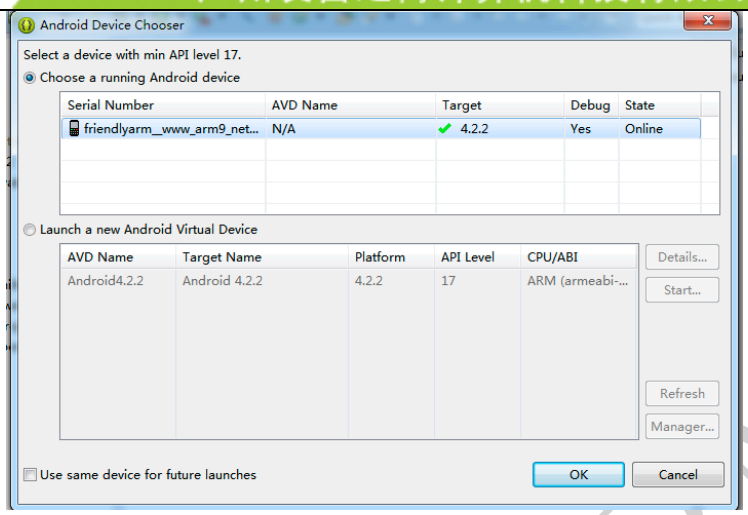


将弹出 Edit Configuration 窗口，点击“Target”，在 Deployment Target Selection Mode 上选择 Always prompt to pick device，如下所示：

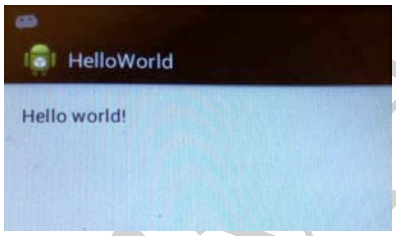


点击 OK 保存并退出。

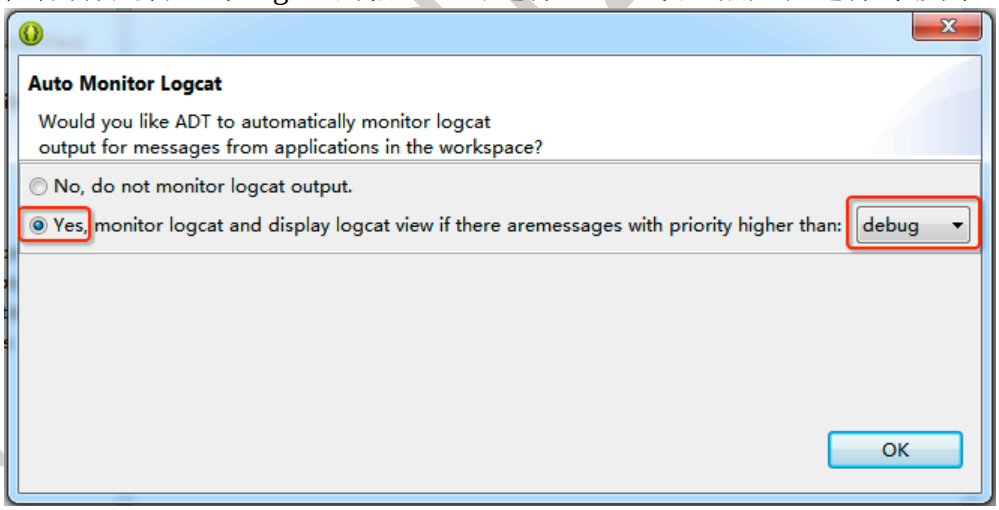
接下来是激动人心的时刻，选中 HelloWorld 工程名称，然后点击工具栏的运行按钮，或选择菜单：Run->Run，会弹出“Android Device Chooser”对话框，在其中选择“Choose a running Android device”，然后在列表中选中 Target 为 4.2.2 的设备(也就是 Tiny4412 啦)，如下图所示，完成后点击“OK”按钮：



略等片刻，HelloWorld 就在 Tiny4412 上运行起来啦：



ADT 会询问你是否跟踪 logcat 的信息，可选择 Yes 一项，然后在选择等级为 debug：



1.5.4 在Tiny4412 上调试Android程序

先在 Tiny4412 上退出 HelloWorld 程序，然后在 ADT 主界面上，点击菜单“Open Perspective”下的“Debug”，即可进入调试视图，我们尝试在源代码上设置一个断点，最后点击菜单“Run”

下的“Debug”重新运行程序，如下图所示，程序运行在我们设置的断点就暂停了，表示真机调试环境建立成功。

FriendlyARM

第二章　在Andorid程序中访问硬件

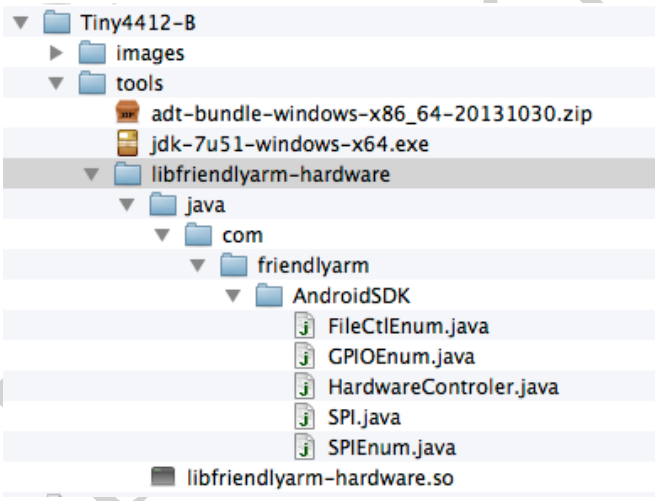
为方便用户开发需要访问开发板硬件资源的 Android 应用程序，友善之臂为用户开发了一个函数库(命名为 libfriendlyarm-hardware.so)，用于访问 Tiny4412 上的硬件资源，目前支持的硬件设备包括：串口设备、蜂鸣器设备、EEPROM、ADC、LED 设备等，可以访问的硬件接口包括：I2C, SPI, GPIO 常用的接口。

iTest 应用程序是使用该函数库来进行开发的，你可以通过运行 Android 上的 iTest 应用程序来了解这个函数库的功能，**本章节提供的示例实际上已经涵盖了 iTest 的所有硬件操作相关的功能**。

本章节主要介绍如何在 Android 应用程序中使用 libfriendlyarm-hardware.so 函数库来操作硬件设备。

2.1 如何使用函数库(libfriendlyarm-hardware.so)?

在 Tiny4412 光盘 B 的 tools\libfriendlyarm-hardware 目录下，包含了开发所需的.so 文件和 Java 对 JNI 接口的封装：



正如上图所示，目录下包含了一系列的 Java 文件和 libfriendlyarm-hardware.so 文件。

另外，由于 iTest 是使用 libfriendlyarm-hardware.so 开发的，因此在官方提供的 Android 版本中都内置了 libfriendlyarm-hardware.so 库，该库文件位于 Android 源代码目录的以下路径:

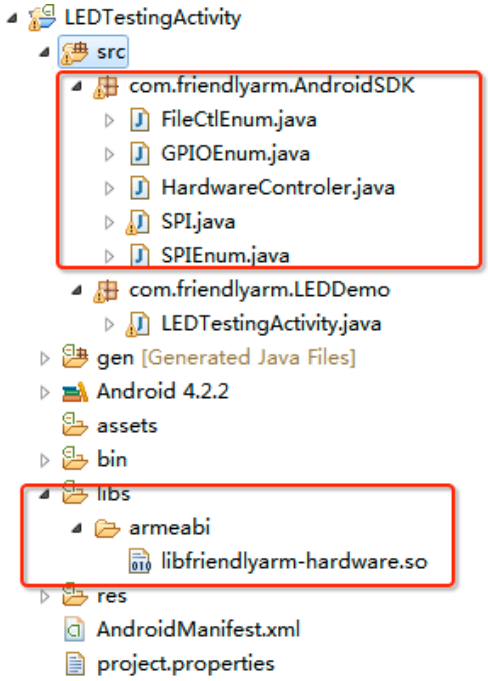
vendor/friendly-arm/exynos4412/rootdir/system/lib/libfriendlyarm-hardware.so

在开发板上位于 /system/lib/libfriendlyarm-hardware.so 目录下。

如果你是参考本文档的方法用 ADT 开发 Android 应用程序，可以通过以下方法使用 libfriendlyarm-hardware.so:

1) 定位到你的应用程序目录，在应用程序目录下创建 libs 目录，再进入 libs 目录下创建 armeabi 目录，然后将 libfriendlyrm-hardware.so 库文件拷贝到 armeabi 目录下。
2) 再回到你的应用程序目录，进入 src 目录下分别创建 com\friendlyarm\AndroidSDK 三层目录。
3) 然后将 B 光盘下的 tools/libfriendlyarm-hardware/java/com/friendlyarm/AndroidSDK 目录下的所有.java 文件复制到你工程内的 src\com\friendlyarm\AndroidSDK 目录下。

部署完毕后，启动 ADT，在 ADT 左侧右击你的项目列表，选择“Refresh”刷新一下项目，这时，项目的相关的目录应该如下图红色标出的部分所示方为正确：



主要的接口都在 HardwareControler 类中封装，其它 Enum 结尾的都是一些常量定义，在使用之前首先需要在代码中导入它们：

```
import com.friendlyarm.AndroidSDK.HardwareControler;
import com.friendlyarm.AndroidSDK.SPIEnum;
import com.friendlyarm.AndroidSDK.GPIOEnum;
import com.friendlyarm.AndroidSDK.FileCtlEnum;
```

然后直接调用 HardwareControler 类的接口即可，下个章节将逐个说明 HardwareControler 类中的函数接口。

2.2 函数库(libfriendlyarm-hardware.so)接口说明

在应用层，可透过上一章节中的 HardwareControler 类来调用 libfriendlyarm-hardware.so 库中的接口，下面中列出 HardwareControler 类中的接口的定义，这些接口都是类方法，因此不需要创建 HardwareControler 对象实例：

2.2.1 通用的输入输出接口

接口名称	参数与返回值说明	功能说明
int open(String devName , int flags)	参数说明： devName：要写入数据的 flags：打开文件的方式，例如可读可写还是只读打开，可选值需参考 FileCtlEnum.java 返回值说明： 成功返回文件描述符，出错返回-1。	打开设备。
int ioctlWithIntValue (int fd, int cmd, int value)	参数说明： fd:设备文件描述符 cmd：ioctl 命令 value:命令参数，限整数 返回值说明： 成功返回 0，出错返回-1。	执行设备的 ioctl 操作
int write(int fd, byte[] data)	参数说明： fd：要写入数据的文件描述符 data：要写入的数据 返回值说明： 成功返回写入的字节数，出错返回-1。	向打开的设备或文件中写数据。
int read(int fd, byte[] buf, int len)	参数说明： fd：要读出数据的文件描述符 buf：存储数据的缓冲区 len：要读取的字节数 返回值说明： 成功返回读取的字节数，出错返回-1，如果在调 read 之前已到达文件末尾，则这次 read 返回 0。	从打开的设备或文件中读取数据。
int select(	参数说明：	查询打开的设备或文件是否有数据可读。

int fd, int sec, int usec)	fd: 要查询的文件描述符 sen: 阻塞等待数据多长时间（单位：秒） usec: 阻塞等待数据多长时间 （单位：纳秒，1 毫秒=1000 纳秒） 返回值说明： 如果 fd 有数据可读，返回 1，如果没有数据可读，返回 0，出错时返回-1。	
void close(int fd)	参数说明： fd: 要关闭的文件描述符 返回值说明： 无	关闭指定的文件描述符

2.2.2 串口通讯的接口说明

与串口相关的接口如下表所示：

接口名称	参数与返回值说明	功能说明
int openSerialPortEx(String devName, long baud, int dataBits, int stopBits, String parityBit, String flowCtrl)	参数说明： devName: 串口设备文件名，可选的值有： /dev/s3c2410_serial1 /dev/s3c2410_serial2 /dev/s3c2410_serial3 /dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3 baud: 波特率 dataBits: 数据位 (取值 5~8，一般用 8) stopBits: 停止位 (取值 1~2，一般用 1) parityBit: 奇偶校验位(取值为单个字母，O 表示奇校验，E 表示偶校验，N 表示无校验) flowCtrl: 数据流控制(取值为单个字母，H 表示硬件流控制，S 表示软件流控制，N 表示不使用数据流控制) 返回值说明： 成功打开串口时，将返回串口的文件描述符，用该描述符可进行 read、write 和 select 等操作，如果打开失败，则返回 -1。	打开指定的串口设备，并返回文件描述符。

接口的使用说明：

先通过调用 openSerialPortEx 打开串口设备，然后可以在线程中、或者用 timer 通过调用 select

接口轮询串口设备是否有数据到来，如果有，则调用 read 接口读取数据。

要往串口中写入数据，调用 write 接口即可。
串口使用完毕后，需要调用 close 关闭串口。

2.2.3 开关LED的接口说明

LED 操作的接口如下表所示：

接口名称	参数与返回值说明	功能说明

`int setLedState(`
`int ledID,`
`int ledState )`
	参数说明：
	ledID： 指定要开关哪一个 LED（取值 0~3）
	ledState：1 表示亮，0 表示灭
	
	返回值说明：
	成功返回 0，失败返回-1
		该接口用于开关 LED 灯。

2.2.4 让PWM蜂鸣器发声和停止发声的接口说明

蜂鸣器操作的接口如下表所示：

接口名称	参数与返回值说明	功能说明
`int PWMPlay(int frequency);`	参数说明： frequency：要发声的频率 返回值说明： 成功返回 0，失败返回-1	按指定的频率让蜂鸣器发声
`int PWMStop();`	参数说明： 无 返回值说明： 成功返回 0，失败返回-1	让蜂鸣器停止发声

2.2.5 读取ADC的转换结果的接口说明

ADC 操作的接口如下表所示：

接口名称	参数与返回值说明	功能说明
`int readADC()`	参数说明： 无 返回值说明： 成功返回 ADC 转换的结果，失败返	读取第一个通道的 ADC 转换结果

	回-1	
int readADCWithChannel(int channel)	参数说明： channel：读取指定通道的 ADC 的值 返回值说明： 成功返回 ADC 转换的结果，失败返回-1	读取指定通道的 ADC 转换的结果
int[] readADCWithChannels(int[] channels);	参数说明： channels：要读取通道的 ADC 频道数组 返回值说明： 成功返回多个 ADC 结果（数组），错误返回空	一次性读取多个频道的结果，性能好

2.2.6 I2C接口说明

在使用如下接口之前，首先需要使用 open 接口打开 I2C 设备，如下所示，操作完成后，记得用 HardwareControler.close 关闭：

```
int fd = HardwareControler.open("/dev/i2c-0", FileCtlEnum.O_RDWR);
```

以下接口需要该 fd 作来参数进行操作：

接口名称	参数与返回值说明	功能说明
int setI2CSlave(int fd, int slave);	参数说明： fd：I2C 设备的文件描述符 slave：I2C 设备地址，例如 EEPROM 设备一般是 0x50 返回值说明： 成功返回 0，失败返回-1	设置要操作的 I2C 设备地址，例如 EEPROM 设备一般是 0x50
int setI2CTimeout(int fd, int timeout)	参数说明： fd：I2C 设备的文件描述符 timeout：超时时间 返回值说明： 成功返回 0，失败返回-1	设置超时时间(ioctl I2C_TIMEOUT)
int setI2CRetries(int fd, int retries)	参数说明： fd：I2C 设备的文件描述符 retries：重试次数 返回值说明：	设置重试次数(ioctl I2C_RETRIES)

	成功返回 0，失败返回-1	
int writeByteToI2C(int fd, int pos, byte byteData, int wait_ms)	参数说明： fd：I2C 设备的文件描述符 pos：字节位置 byteData:要写入的数据 wait_ms：等待指定的时间(毫秒) 返回值说明： 成功返回 0，失败返回-1	写一个字节的数据到 I2C 设备的指定位置,并等待指定的时间(毫秒)
int readByteFromI2C(int fd, int pos, int wait_ms);	参数说明： fd：I2C 设备的文件描述符 pos：字节位置 wait_ms：等待指定的时间(毫秒) 返回值说明： 成功返回 0，失败返回-1	从 I2C 设备指定的位置读一个字节的数据，并等待指定的时间(毫秒)

2.2.7 SPI接口说明

在使用如下接口之前，首先需要使用 open 接口打开 SPI 设备，如下所示，操作完成后，记得用 HardwareControler.close 关闭：

```
intspi_fd = HardwareControler.open("/dev/spidev1.0", FileCtlEnum.O_RDWR );
```

以下接口需要该 spi_fd 作来参数进行操作：

接口名称	参数与返回值说明	功能说明
int setSPIWriteBitsPerWord(int spi_fd, int bits)	参数说明： spi_fd：SPI 设备的文件描述符 bits：字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	设置每次读 SPI 设备的字长,单位是比特。虽然大部分 SPI 接口的字长是 8 或者 16，仍然会有一些特殊的例子。需要说明的是，如果这个成员为零的话，默认使用 8 作为字长 (ioctl SPI_IOC_WR_BITS_PER_WORD)
int setSPIReadBitsPerWord(int spi_fd, int bits)	参数说明： spi_fd：SPI 设备的文件描述符 bits：字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	设置每次写 SPI 设备的字长，单位是比特 (ioctl SPI_IOC_RD_BITS_PER_WORD)
int setSPIBitOrder(int spi_fd, int order)	参数说明： spi_fd：SPI 设备的文件描述符 order：传	设备 SPI 传输时是先传输低比特位还是高比特位，可选的参数有 SPIEnum.MSBFIRST 和 SPIEnum.LSBFIRST

	SPIEnum.MSBFIRST 或 SPIEnum.LSBFIRST 返回值说明： 成功返回 0，失败返回负数	
int setSPIClockDivider(int spi_fd, int divider)	参数说明： spi_fd：SPI 设备的文件描述符 divider：分频系数，传入在 SPIEnum.java 中定义的以 SPI_CLOCK_开头的常量，例如：SPIEnum.SPI_CLOCK_DIV128 返回值说明： 成功返回 0，失败返回负数	设置 SPI 的分频系数
int setSPIDataMode(int spi_fd, int mode)	参数说明： spi_fd：SPI 设备的文件描述符 mode：SPI 设备的模式，可传入 SPIEnum.SPI_MODE0 ~ SPIEnum.SPI_MODE3 返回值说明： 成功返回 0，失败返回负数	设置 SPI 设备的模式
int SPItransferOneByte(int spi_fd, byte byteData, int spi_delay, int spi_speed, int spi_bits)	参数说明： spi_fd：SPI 设备的文件描述符 byteData:要写入 SPI 设备的数据 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明： 成功返回读到的数据，失败返回负数	同时发送与接收一个字节的数据，调用示例： int byteRet = SPItransferOneByte(spi_fd , 0xAA , 0 /*delay*/ , 500000/*speed*/ ,8/*bits*/) ;
int SPItransferBytes(int spi_fd, byte[] writeData, byte[] readBuff, int spi_delay, int spi_speed, int spi_bits)	参数说明： spi_fd：SPI 设备的文件描述符 writeData:要写入的数据 readBuff：存放读取数据的缓冲区 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明：	同时发送与接收多个字节的数据

	成功返回 0，失败返回负数	
int writeBytesToSPI(int spi_fd, byte[] writeData, int spi_delay, int spi_speed, int spi_bits)	参数说明： spi_fd：SPI 设备的文件描述符 writeData:要写入的数据 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	写多个字节的数据到 SPI 设备
int readBytesFromSPI(int spi_fd, byte[] readBuff, int spi_delay, int spi_speed, int spi_bits)	参数说明： readBuff：存放读取数据的缓冲区 spi_delay:延时 spi_speed:传输速度 spi_bits:字长，单位是比特 返回值说明： 成功返回 0，失败返回负数	从 SPI 设备读取多个字节

2.2.8 GPIO接口说明

libfriendlyarm-hardware.so 函数库在操作 GPIO 时，使用了 sysfs 的 GPIO 接口，程序需要访问文件系统中 GPIO 相关的文件节点，因此，你的 Android 应用程序需要以 system 权限来运行，否则无法没有权限操作 GPIO，有两种方法可以让我们的 Android 程序获得 system 权限：

1) 在 Android 源代码中编译你的工程，并指定使用系统签名；

2) 用 ADT 编译工程，但在打包 apk 时自行加上 system 签名；

详情可参考后面章节中的 GPIO 示例。

下面是 GPIO 操作的相关接口：

接口名称	参数与返回值说明	功能说明
int exportGPIOPin(int pin)	参数说明： pin：GPIO 引脚编号 返回值说明： 成功返回 0，失败返回负数	通知系统需要导出控制的 GPIO 引脚编号，相当于执行命令 echo pin > /sys/class/gpio/export
int unexportGPIOPin(int pin)	参数说明： pin：GPIO 引脚编号 返回值说明：	通知系统取消导出某个 GPIO 引脚，相当于执行命令 echo pin > /sys/class/gpio/unexport

	成功返回 0，失败返回负数	
int setGPIOValue(int pin, int value)	参数说明： pin：GPIO 引脚编号 value：传入 GPIOEnum.LOW 表示输出低电平，传入 GPIOEnum.HIGH 表示输出高电平 返回值说明： 成功返回 0，失败返回负数	对某个引脚输出高或低电平
int getGPIOValue(int pin)	参数说明： pin：GPIO 引脚编号 返回值说明： 成功返回 GPIOEnum.LOW 表示输出低电平，返回 GPIOEnum.HIGH 表示输出高电平，失败返回负数	查询某个引脚的状态（高或低电平）
int setGPIODirection(int pin, int direction)	参数说明： pin：GPIO 引脚编号 direction：传入 GPIOEnum.IN 表示输入，GPIOEnum.OUT 表示输出 返回值说明： 成功返回 0，失败返回负数	配置引脚功能为输出或者输入
int getGPIODirection(int pin)	参数说明： pin：GPIO 引脚编号 成功返回 GPIOEnum.IN 表示输入，返回 GPIOEnum.OUT 表示输出，失败返回负数	查询引脚功能（为输出或者输入）

2.3 示例程序说明

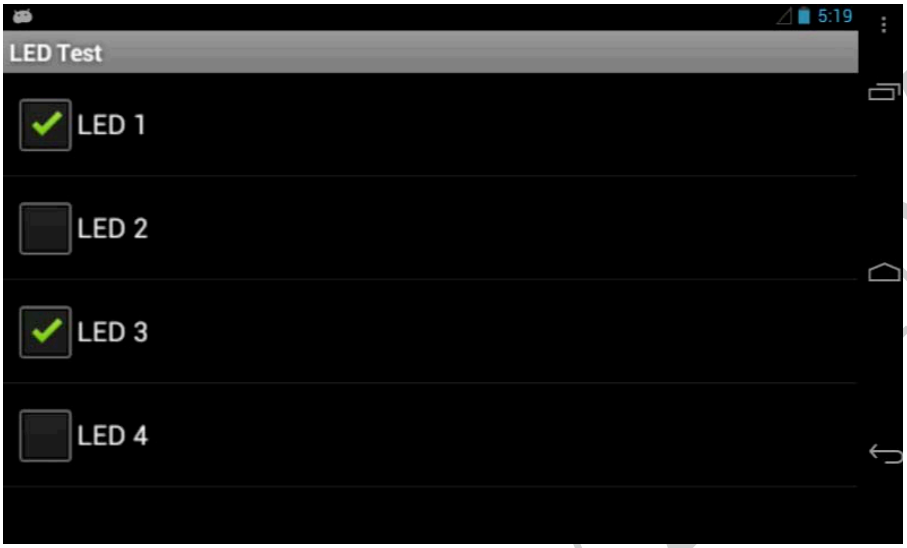
2.3.1 在板LED示例

操作在板 LED 的代码非常简单，只需调用接口 HardwareControler.*setLedState 开关指定的 LED* 即可，下面的代码演示开关 LED1：

```
import com.friendlyarm.AndroidSDK.HardwareControler;

//将 LED1 开启：

HardwareControler.setLedState(0,1);

//将 LED1 关闭：

HardwareControler.setLedState(0,0);
```

本示例仅适用于控制 Tiny4412 在板的 LED，如果你外接的 LED，可以参考下一个章节的内容：用 GPIO 来操作 LED.

完整的示例工程代码位于 A 光盘的 Android/examples/目录下，工程目录名：LEDDemo，示例运行界面如下：



2.3.2 GPIO示例

本示例演示 GPIO 接口的用法，将以操作在板 LED 为例，如果你外接的 LED，只需更换 pin 脚的序号即可。

与上一例子不同，本示例通过 GPIO 直接操作 LED，不再使用 Linux Kernel 的 LED 驱动，而且为了避免冲突，我们还需要将 Linux Kernel 的 LED 驱动先禁用，才能运行本示例。

2.3.2.1 以system权限运行Android应用程序

由于 libfriendlyarm-hardware.so 函数库在操作 GPIO 时，使用了 sysfs 的 GPIO 接口，程序需要访问文件系统中 GPIO 相关的文件节点，因此，你的 Android 应用程序需要以 system 权限来运行，否则无法没有权限操作 GPIO，有两种方法可以让我们的 Android 程序获得 system 权限：

1） 在 Android 源代码中编译你的工程，并指定使用系统签名；

2） 用 ADT 编译工程，但在打包 apk 时自行加上 system 签名；

由于我们的源代码是开放的，因此我们采用第一种比较简单的方法，即将示例工程放在 Android 源代码目录去编译，请将 GPIO_LED_Demo 目录拷到 Android 源代码的 packages/apps 目录下，以 Tiny4412 的 Android-4.2.2_r1 为例，假设你已经将 “Tiny4412 光盘 A” mount 到 /mnt/iso 目录下，则以下步骤是复制代码并编译：

```
定位到 Android 4.2.2 源代码根目录
# cd /opt/FriendlyARM/tiny4412/android-4.2.2_r1/
设置一下 Android 编译所需要环境变量
# . setenv
复制 GPIO_LED_Demo 代码到 packages/apps 目录
# cd packages/apps
# cp /mnt/iso/Android/examples/GPIO_LED_Demo . –a
执行 mm 编译工程
#cd GPIO_LED_Demo
# mm
```

编译生成的 apk 文件在 out/target/product/tiny4412/system/app/GPIODemo.apk 目录下，我们用 adb 安装并运行它：

```
# adb install –r out/target/product/tiny4412/system/app/GPIODemo.apk
```

如果你自己的工程，要注意以下两个注意事项：

1) 工程的 AndroidManifest.xml 文件需要指定：android:sharedUserId="android.uid.system",如下所示：

```
<manifestxmlns:android="http://schemas.android.com/apk/res/android"
package="com.friendlyarm.GPIODemo"
android:sharedUserId="android.uid.system"
android:versionCode="1"
android:versionName="1.0">
```

2) 工程的 Android.mk 文件需要加上一行：LOCAL_CERTIFICATE := platform

2.3.2.2 禁用内核中的LED驱动

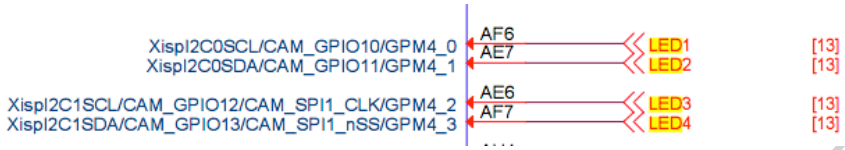
另外，要运行本示例，我们还需要用以下方法禁用 LED 驱动：

你需要重新配置 Android 的内核 Linux Kernel，在 make menuconfig 之后，进入以下路径 **Device Drivers ->Character devices**，找到 **LED Support for FriendlyARM Tiny4412 GPIO LEDs** 驱动，去掉前面的钩选，然后重新编译内核并烧写到板子上。

2.3.2.3　获悉LED的GPIO引脚的定义和内核编号

要通过 GPIO 操作 LED，我们还需要获悉 LED 的引脚定义，我们可以查询 Tiny4412 核心板的原理图来获得。

Tiny4412 核心板原理图位于光盘：**Schematic-PCB/Tiny4412/Tiny4412_1306_sch.pdf**，打开它，搜索”LED”关键字，找到如下引脚定义：



可见 LED1~4 管脚资源为 GPM4(0~3) 。

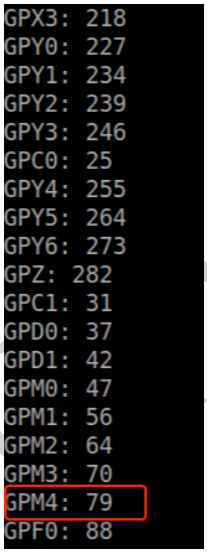
由于我们需要通过 sysfs 的方式来操作 GPIO，因此，我们还需要找到这组引脚在内核中所对应的编号（每一个 GPIO 引脚，内核都会赋于它一个编号）,查询内核编号的方法如下：

1） 将 Tiny4412 连接串口并开机，并进入 Android 系统；

2） 在超级终端（或 minicom）中输入以下命令（因命令太长符号太多，建议直接复制运行）：

```
# cd /sys/class/gpio
# for i in gpiochip* ; do echo `cat $i/label`: `cat $i/base` ; done
```

最后一行的命令会输出所有 GPIO 引脚，在内核中的对应编号，如下图所示，LED 所在的 GPM4 引脚的在内核中的编号是从 79 开始的：



LED1~4 的引脚 GPM4(0 ~ 3)在内核中的实际编号，还需要加上子编号 0~3 进行换算，因此，LED1 就是 79+0 即 79，LED4 就是 79+3=82。

FriendlyARM

2.3.2.4 通过GPIO来控制LED

有了 LED 的内核编号，就可以通过 sysfs 来操作它了，在操作之前，我们需要先调用 exportGPIOPin 导出引脚的文件系统节点 (相当于命令行执行 `echo pin > /sys/class/gpio/export`)，注意文件系统节点的生成可能有一定的延迟，所以在导出引脚后，需要再启动一个 100ms 的 timer 来等待，延时一会再执行接下来的初始化操作：

```
for (int i = 0; i <4; i++) {
int pin = 79 + i;
HardwareControler.exportGPIOPin(pin);
}
step = STEP_INIT_GPIO_DIRECTION;
timer.schedule(init_task, 100, 100);
```

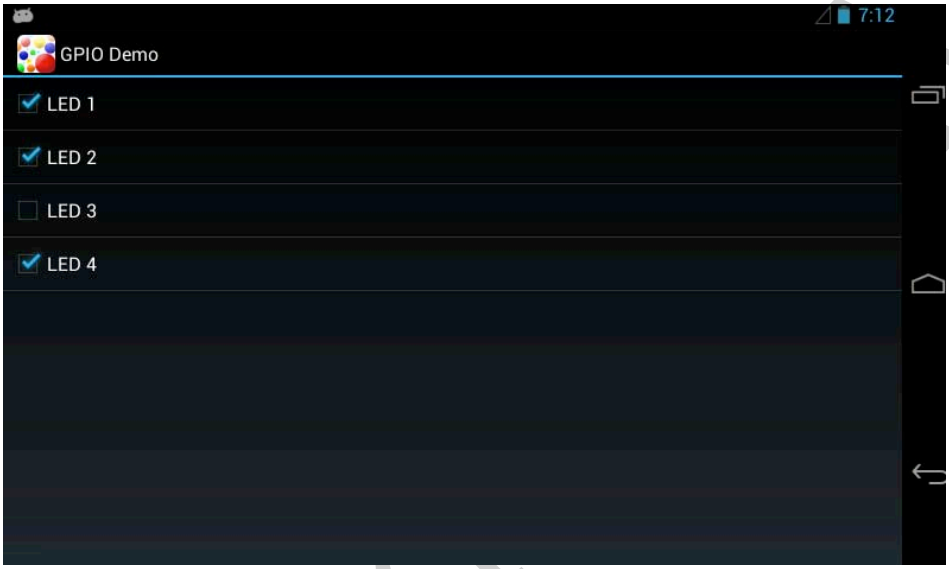
100 毫秒后，设置引脚的模式为输出，并对引脚设置高电平来熄灭所有 LED：

```
private TimerTask init_task = new TimerTask() {
publicvoid run() {
Log.d(TAG, "init_task " + step);
if (step == STEP_INIT_GPIO_DIRECTION) {
for (int i=0; i<4; i++) {
if (HardwareControler.setGPIODirection(79+i, GPIOEnum.OUT) == 0) {
                        } else {
Log.v("TimerTask", "setGPIODirection failed");
                        }
                }
step ++;
            } elseif (step == STEP_CLOSE_ALL_LED) {
for (int i=0; i<4; i++) {
if (HardwareControler.setGPIOValue(79+i, GPIOEnum.HIGH) == 0) {
                        } else {
Log.v(TAG, "setGPIOValue failed");
                        }
                }
step ++;
            } elseif (step == STEP_INIT_VIEW) {
                Message message = new Message();
message.what = 1;
handler.sendMessage(message);
            }

    }
};
```

最后处理界面操作，当点击界面上的 CheckBox 时，就对相应的 LED 引脚输出高低电平，如下代码所示：

```
HardwareControler.setGPIOValue(79+led.code,led.isSelected() ?
GPIOEnum.LOW:GPIOEnum.HIGH)
```

完整的示例工程代码位于 A 光盘的 Android/examples/目录下，工程目录名: GPIO_LED_Demo，示例运行界面如下：



2.3.3 串口通讯示例

本示例是一个串口通迅程序，程序在设置串口参数时，调用如下代码初始化串口：

```
devfd = HardwareControler.openSerialPort( devName, speed, dataBits, stopBits );
```

如果要操作使用流控制和奇偶校验，可使用如下函数：

```
devfd = HardwareControler.openSerialPortEx( devName, speed, dataBits, stopBits,
parityBit, flowCtrl );
```

串口通讯时，我们 需要轮询从串口另一端发来过来的数据，并显示到界面，在本示例中，启动了一个 500ms 的 Timer，每 500ms 调用 HardwareControler.select 接口检查串口是否有新数据，如果有，就调用 HardwareControler.read 来读取，查询和读取的相关代码如下所示：

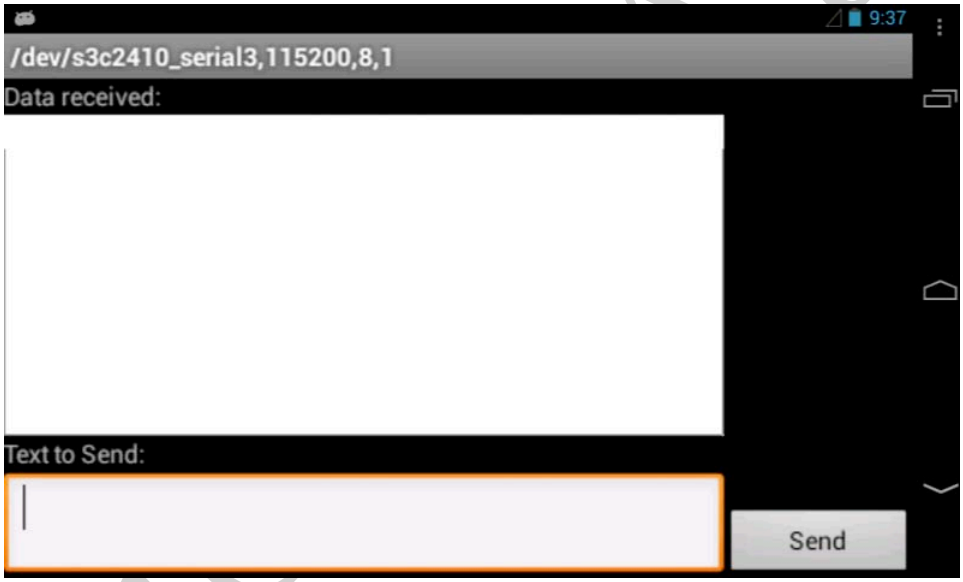
```
private final int BUFSIZE = 512;
private byte[] buf = new byte[BUFSIZE];
if (HardwareControler.select(devfd, 0, 0) == 1) {
```

```
    int retSize = HardwareControler.read(devfd, buf, BUFSIZE);
    String str = new String(buf, 0, retSize);
}
```

写数据到串口，用 HardwareControler.*write:*

```
String str = toEditor.getText().toString();
if (str.length() > 0) {
    ret = HardwareControler.write(devfd, str.getBytes());
}
```

完整的串口示例代码位于 A 光盘的 Android/examples/目录下，工程目录名：SerialPortDemo，示例运行界面如下：



2.3.4 PWM示例

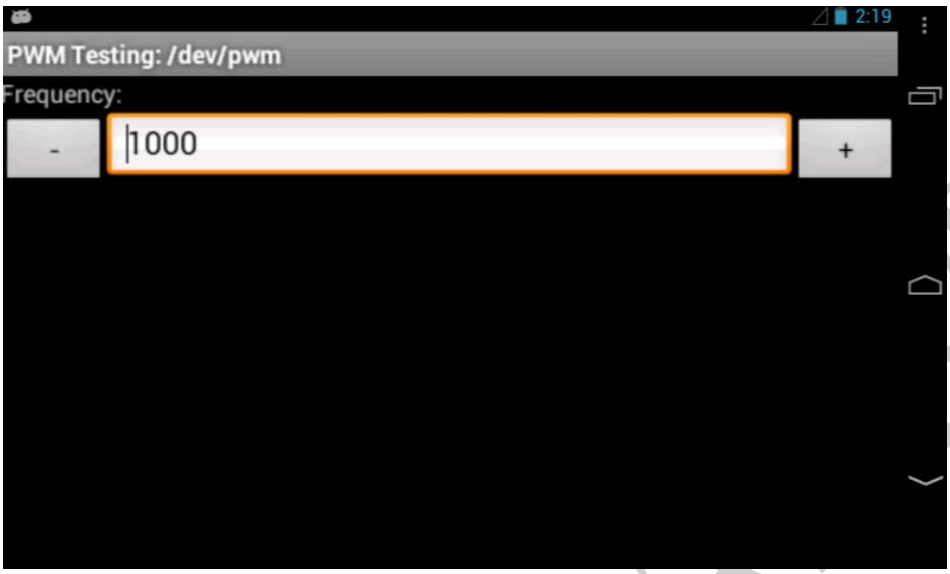
PWM 蜂鸣器的操作比较简单，只有一个接口，下面的代码演示以 1000 的频率让蜂鸣器响铃:

```
HardwareControler.PWMPlay(1000);
```

要停止响铃调用 PWMStop 即可:

```
HardwareControler.PWMStop();
```

完整的示例工程代码位于 A 光盘的 Android/examples/目录下，工程目录名为：PWMDemo，示例运行界面如下：

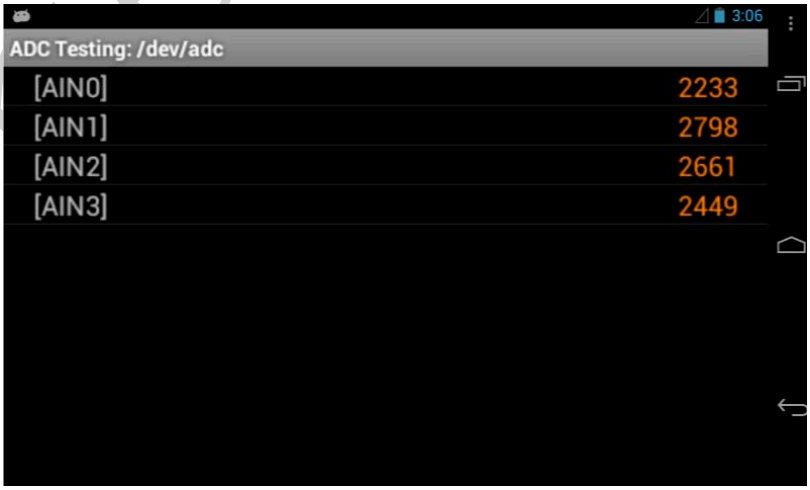


2.3.5 A/D转换示例

下面的代码演示用 A/D 采集的接口进行多个通道的采集：

```
int[] channels = {0,1,2,3};

int[] adc = HardwareControler.readADCWithChannels(channels);
```

完整的示例工程代码位于 A 光盘的 Android/examples/目录下，工程目录名为： ADCDemo，示例运行界面如下：



2.3.6 I2C& EEPROM示例

本示例演示 I2C 的操作，利用 I2C 的接口来操作 EEPROM，下面的代码演示向 EEPROM 中写入/读取一个字节的数据，首先是打开设备文件 /dev/i2c-0，接着设置 EEPROM 的设备地址 0x50，然后用 writeByteToI2C 接口写一个字节到 EEPROM，或用 readByteFromI2C 从 EEPROM 读一个字节:

```
devFD = HardwareControler.open("/dev/i2c-0", FileCtlEnum.O_RDWR);
HardwareControler.setI2CSlave(devFD, 0x50);
HardwareControler.writeByteToI2C(devFD, 0, 0x20, 10);
byte ret = HardwareControler.readByteFromI2C(devFD, 0, 10);
```

EEPROM 可存储 256 个字节的数据，所以在读写时需要指定的位置范围是 0~255，每次只能读写一个字节，每次读写之间需要作一定的延时，一般为 10ms，在 writeByteToI2C 中指定延时参数即可。

完整的示例工程代码位于 A 光盘的 Android/examples/目录下，工程目录名为：I2CDemo，示例运行界面如下：

2.3.7 SPI示例

本示例尚未提供。

2.3.8 看门狗示例

看门狗的操作比较简单，只要打开 /dev/watchdog0 设备文件，则意味着看门狗开始工作，打开设备文件后，程序必须 15 秒内写一个字节的数据进去这个设备文件，否则设备会重启。

使用 libfriendlyarm-hardware.so 库 标准的 open 和 write 接口就能完成看门狗的操作，如下所示：启动看门狗使用如下代码：

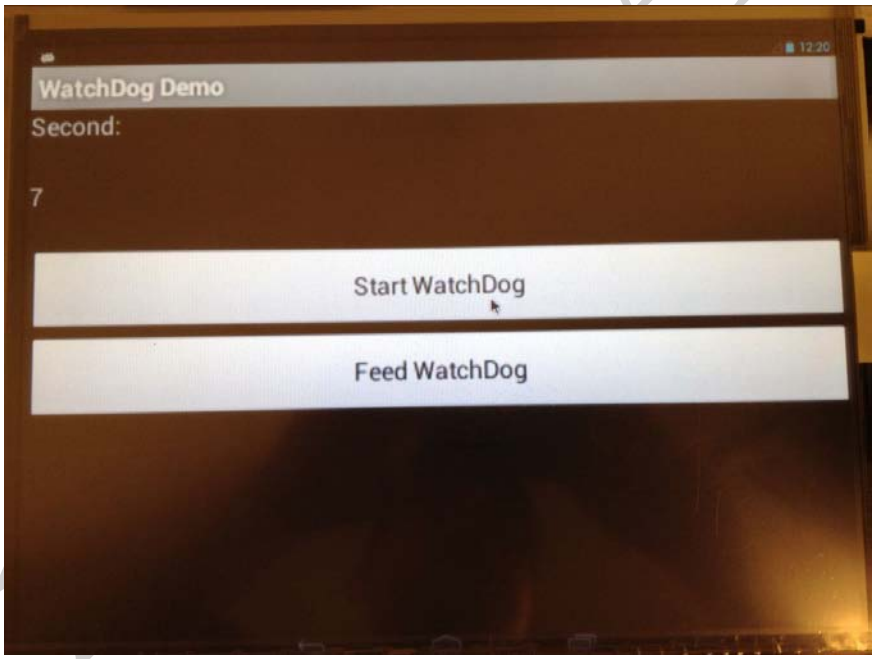
```
fd = HardwareControler.open( "/dev/watchdog0", FileCtlEnum.O_WRONLY );
```

这时，看门狗内部就开始计时了，通过写一个字节的数据进去，重置计时：

```
final String str = "W";
byte[] writeBytes = str.getBytes();
HardwareControler.write(fd, writeBytes);
```

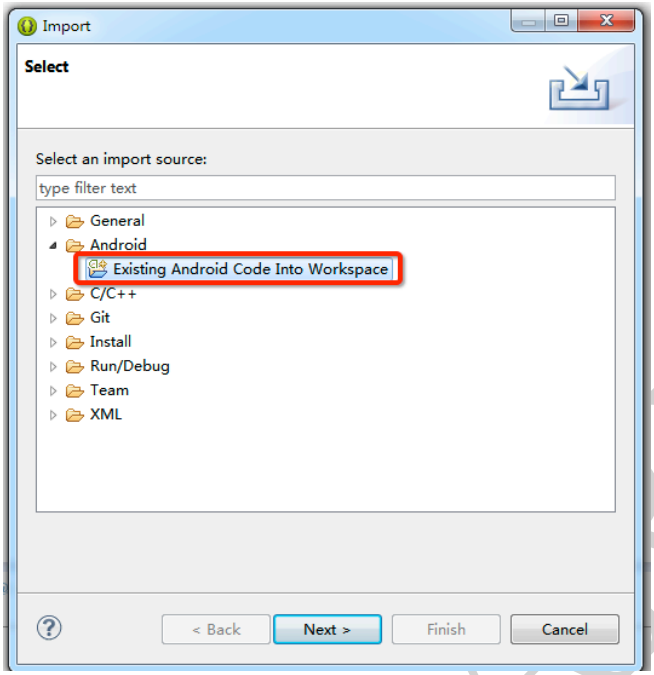
完整的代码请参考 WatchDogDemo 示例，该示例运行如下图所示:

只要你点击 Start WatchDog 按钮，时间就开始计时，中途点击 Feed WatchDog 重新计时，如果计时到达 15 秒，则设备重启：



2.4 在ADT中导入示例工程

在 PWMDemo 为例，我们先从光盘把 PWMDemo 拷到本地，然后启动 ADT，在 ADT 界面上通过菜单 File -> Import，在导入工程的界面上，选择 Existing Android Code Into Workspace，如下图所示：



然后选择工程目录 PWMDemo，再点 Finish 即可：