1. 下載 Android for BBB

http://www.ti.com/tool/androidsdk-sitara

2. 論壇 TI E2E Android

http://e2e.ti.com/support/embedded/android/default.aspx

3. BeagleBoneBlack Wiki

http://www.elinux.org/Beagleboard:BeagleBoneBlack

http://www.elinux.org/Beagleboard:Android

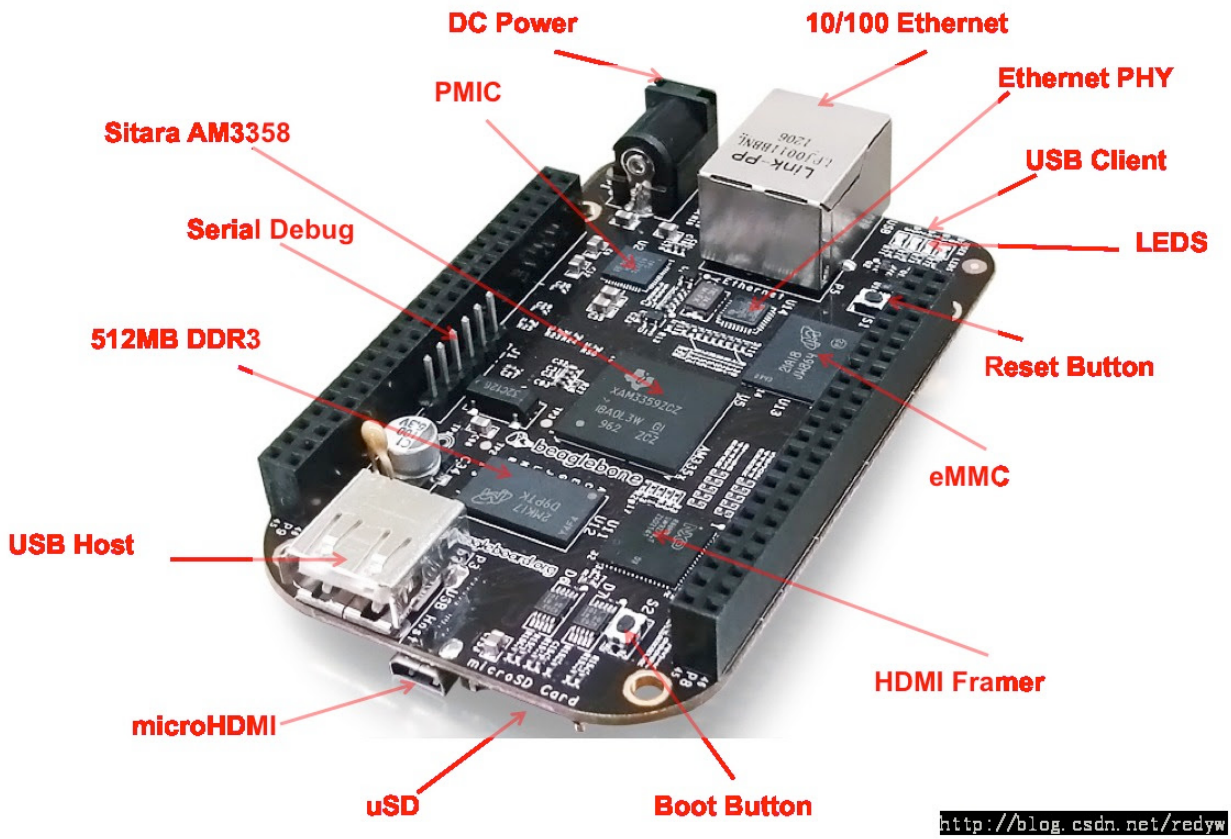
4. 手冊 Android for BBB

http://processors.wiki.ti.com/index.php/TI-Android-JB-4.2.2-DevKit-4.1.1_UserGuide

http://processors.wiki.ti.com/index.php/TI-Android-JB-4.2.2-DevKit-4.1.1_PortingGuide

一：硬件介紹

參考官方網站：http://beagleboard.org



二：獲取源代碼

從網址http://www.ti.com/tool/androidsdk-sitara 下載Android Jelly Bean 4.2.2 - Dev Kit for AM335x

可以用git下載源碼也可以直接下載源碼包，此處是直接下載的源碼包。下面是該源碼包的連接地址，包含了各種文檔。

http://downloads.ti.com/sitara_android/esd/TI_Android_DevKit/TI_Android_JB_4_2_2_DevKit_4_1_1/index_FDS.html

三：搭建主機平台

根據官網上的Developer Guide推薦，安裝的是Ubuntu12.04-64Bit

安装虚擬機中遇到的問題以及需要注意的地方如下：

1. 開機出現錯誤：piix4_smbus 0000:00:007.3: Host SMBus controller not enabled

解決辦法：sudo vim /etc/modprobe.d/blacklist.conf，在末尾加入blacklist i2c-piix4

2. Ubuntu的界面登陸後報錯 Vmware .... (SAGV)

解決辦法：

a)在VM設備配置中關閉3D加速

b)在Ubuntu登陸界面中點擊登陸框旁的小按鈕選擇2D界面

3. 關閉防火牆 sudo ufw disable

4. 更新源文件：sudo apt-get update

5. 安装ssh

server：sudo apt-get install openssh-server

client：sudo apt-get install openssh-client

6. samba

apt-get install samba

7. 命令行啟動

sudo gedit /etc/default/grub

找到這一行

GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"

改成

GRUB_CMDLINE_LINUX_DEFAULT="quiet splash text"

在輸入命令：

sudo update-grub

8. android編譯對內存要求很高，內存/swap分區太少，編譯會報錯」分配虛擬機內存失敗「

http://source.android.com/source/building.html

16GB RAM/swap

30GB disk(free)

修改交換分區到16GB

free // 查看系統內存

sudo mkdir /usr/swap

sudo dd if=/dev/zero of=/usr/swap/swapfile bs=1024 count=16777216

sudo mkswap -v1 /usr/swap/swapfile

sudo swapon /usr/swap/swapfile

sudo vim /etc/fstab

寫入/usr/swap/swapfile swap swap defaults 0 0

==========

系統安裝完成後，按照guide做如下設置：

1. 安裝必須的包

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
  libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
  libgl1-mesa-dev g++-multilib mingw32 openjdk-6-jdk tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev:i386 \
  minicom tftpd uboot-mkimage expect libgl1-mesa-dri libglapi-mesa:i386
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

注意：官網上少了libglapi-mesa:i386，不添加上會安裝失敗。

2. 安裝JDK6

```
$ chmod a+x jdk-6u45-linux-x64.bin
$ ./jdk-6u45-linux-x64.bin
$ sudo mkdir -p /usr/lib/jvm
$ sudo mv jdk1.6.0_45 /usr/lib/jvm/
$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.6.0_45/bin/java"
1
$ sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/lib/jvm/jdk1.6.0_45/bin/javac" 1
$ sudo update-alternatives --config java
```

$ sudo update-alternatives --config javac

$ sudo java -version

注意：必須下載JDK安裝，Ubuntu自帶的open jdk不行。

四：編譯源碼

1. 源碼下載回來為.bin格式的，需要安裝源碼包

$ chmod a+x TI_Android_JB_4.2.2_DevKit_4.1.1.bin

$ ./TI_Android_JB_4.2.2_DevKit_4.1.1.bin

2. 設置toolchain

添加下面的內容到.bashrc

exportPATH=~/BeagleBoneBlack/source/TI_Android_JB_4.2.2_DevKit_4.1.1/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin:$PATH

3. 編譯bootload

make CROSS_COMPILE=arm-eabi- distclean

make CROSS_COMPILE=arm-eabi- am335x_evm_config

make CROSS_COMPILE=arm-eabi-

===== OR =====

make TARGET_PRODUCT=beagleboneblack OMAPES=4.x u-boot_build

4. 編譯kernel

內核的默認配置文件是：arch/arm/configs/am335x_evm_android_defconfig

make ARCH=arm CROSS_COMPILE=arm-eabi- distclean

make ARCH=arm CROSS_COMPILE=arm-eabi- am335x_evm_android_defconfig

make ARCH=arm CROSS_COMPILE=arm-eabi- uImage

===== OR =====

make TARGET_PRODUCT=beagleboneblack OMAPES=4.x kernel_build

5. 編譯Android

make TARGET_PRODUCT=beagleboneblack OMAPES=4.x -j4

注意：-j4 是指定編譯的線程數，4是由CPU的個數（CPU核數） X 2 得來的。

如果是在虛擬機中最好不要使用該參數，容易出錯。（也可能是我的機器性能太差了）

6. 創建rootfs

make TARGET_PRODUCT=beagleboneblack fs_tarball

===== OR =====

$ cd out/target/product/beagleboneblack

$ mkdir android_rootfs

$ cp -r root/* android_rootfs

$ cp -r system android_rootfs

$ ../../../../build/tools/mktarball.sh ../../../host/linux-x86/bin/fs_get_stats android_rootfs . rootfs rootfs.tar.bz2

7. clean

make TARGET_PRODUCT=beagleboneblack clean

8. 編譯完成後，相關的編譯過程產物和結果都存放在out目錄中，可以考慮備份該目錄，防止失誤造成需要重新編譯android。

9. 燒寫SDCard

拷貝編譯生成的文件到目錄image_folder

cp external/ti_android_utilities/am335x/u-boot-env/uEnv_beagleboneblack.txtimage_folder/uEnv.txt

cp kernel/arch/arm/boot/uImageimage_folder

cp u-boot/u-boot.img image_folder

cp u-boot/MLO image_folder

cpout/target/product/beagleboneblack/rootfs.tar.bz2 image_folder

cp -rexternal/ti_android_utilities/Media_Clips image_folder

cpexternal/ti_android_utilities/am335x/mk-mmc/mkmmc-android.sh image_folder

cpexternal/ti_android_utilities/am335x/mk-mmc/README.txt image_folder

插入USB 讀卡器到PC上，用fdisk –l命令查看磁盤信息，發現SDCard設備為/dev/sdb

進入到目錄image_folder中執行下面的命令

sudo ./mkmmc-android.sh /dev/sdbMLO u-boot.img uImage uEnv.txt rootfs.tar.bz2 Media_Clips

五： 模塊編譯

這個在開發的時候比較常用

1. 設置環境變量

. build/envsetup.sh

===== or =====

source build/envsetup.sh

2. 用lunch命令指定product

3. 如何使用命令 mm，mmm 請查考help

輸入hmm 即可獲得help

注意： 以上使用的default配置，目標產品為generic，而在TI中編譯使用的是beagleboneblack，因此在mm時會報錯。

如果要指定beagleboneblack 則需要修改build/envsetup.sh，在lunch選項中加入add_lunch_combo beagleboneblack-eng

並使用lunch選擇beagleboneblack-eng，模塊編譯就會尋找正確的product。(TARGET_PRODUCT=beagleboneblack)

==========

PLATFORM_VERSION_CODENAME=REL

PLATFORM_VERSION=4.2.2

TARGET_PRODUCT=beagleboneblack

TARGET_BUILD_VARIANT=eng

TARGET_BUILD_TYPE=release

TARGET_BUILD_APPS=

TARGET_ARCH=arm

TARGET_ARCH_VARIANT=armv7-a-neon

HOST_ARCH=x86

HOST_OS=linux

HOST_OS_EXTRA=Linux-3.11.0-15-generic-x86_64-with-Ubuntu-12.04-precise

HOST_BUILD_TYPE=release

BUILD_ID=JDQ39

OUT_DIR=out

==========

4. 刪除指定模塊

mm clean-module_name

注意：module_name與每個程序中的Android.mk裡的LOCAL_MODULE是一致的。

如果沒有寫module_name會導致out中的東東被刪除了。

六：源碼介紹

1. 目錄結構

Android/abi (abi相關代碼。ABI：application binary interface，應用程序二進制接口)

Android/bionic（bionic C庫）

Android/bootable（啟動引導相關代碼）

Android/build（存放系統編譯規則及generic等基礎開發配置包）

Android/cts（Android兼容性測試套件標準）

Android/dalvik（dalvik JAVA虛擬機）

Android/development（應用程序開發相關）

Android/device （設備相關代碼）

Android/docs (介紹開源的相關文檔)

Android/external（android使用的一些開源的模組）

Android/frameworks（核心框架——java及C++語言，是Android應用程序的框架。）

Android/hardware（主要是硬件適配層HAL代碼）

Android/libcore（核心庫相關）

Android/ndk (ndk相關代碼。AndroidNDK（Android NativeDevelopment Kit）是一系列的開發工具，允許程序開發人員在Android應用程序中嵌入C/C++語言編寫的非託管代碼。)

Android/out（編譯完成後的代碼輸出與此目錄）

Android/packages（應用程序包）

Android/prebuilt（x86和arm架構下預編譯的一些資源）

Android/sdk（sdk及模擬器）

Android/system（文件系統、應用及組件——C語言）

Android/Makefile

2. Android.mk 結尾include各種模板文件

模板文件的定義在build/core/config.mk文件

BUILD_SHARED_LIBRARY：指向build/core/shared_library.mk。

BUILD_PACKAGE：指向build/core/package.mk，用來編譯APK文件。

BUILD_JAVA_LIBRARY：指向build/core/java_library.mk，用來編譯Java庫文件。

BUILD_STATIC_JAVA_LIBRARY：指向build/core/tatic_java_library.mk，用來編譯Java靜態庫文件。

BUILD_STATIC_LIBRARY：指向build/core/static_library.mk，用來編譯靜態庫文件。也就是.a文件。

BUILD_EXECUTABLE：指向build/core/executable.mk，用來編譯可執行文件。

BUILD_PREBUILT：指向build/core/prebuilt.mk。用來編譯已經預編譯好的第三方庫文件，實際上是將這些預編譯好的第三方庫文件拷貝到合適的位置去，以便可以讓其它模塊引用。

七：實例hello

這部分的代碼是從網上找的，參考的

http://blog.csdn.net/luoshengyang/article/details/6567257

下面簡單的介紹一下：

該實例是一個從上（UI）到下（硬件）的一整套開發。

從硬件設備（虛擬設備）中讀取數值顯示在UI上或者在UI上寫入數值並保存到硬件中，/proc/hello可以查看寫入硬件的數值。

實例包含了Kernel的驅動，HAL層，JNI，Java程序。

1. 虛擬設備

在內核中添加一個虛擬設備/dev/hello,代碼如下：

hello.h：

[cpp] view plaincopy

```
1. #ifndef _HELLO_ANDROID_H_
2. #define _HELLO_ANDROID_H_
3.
4. #include <linux/cdev.h>
5. #include <linux/semaphore.h>
6.
7. #define HELLO_DEVICE_NODE_NAME  "hello"
8. #define HELLO_DEVICE_FILE_NAME  "hello"
9. #define HELLO_DEVICE_PROC_NAME  "hello"
10. #define HELLO_DEVICE_CLASS_NAME "hello"
```

```
11.
12. struct hello_android_dev {
13.     int val;
14.     struct semaphore sem;
15.     struct cdev dev;
16. };
17.
18. #endif
```

hello.c：

[cpp] view plaincopy

```
1.  #include <linux/init.h>
2.  #include <linux/module.h>
3.  #include <linux/types.h>
4.  #include <linux/fs.h>
5.  #include <linux/proc_fs.h>
6.  #include <linux/device.h>
7.  #include <asm/uaccess.h>
8.
9.  #include "hello.h"
10.
11. /*主設備和從設備號變量*/
12. static int hello_major = 0;
13. static int hello_minor = 0;
14.
15. /*設備類別和設備變量*/
16. static struct class* hello_class = NULL;
17. static struct hello_android_dev* hello_dev = NULL;
18.
19. /*傳統的設備文件操作方法*/
20. static int hello_open(struct inode* inode, struct file* filp);
21. static int hello_release(struct inode* inode, struct file* filp);
22. static ssize_t hello_read(struct file* filp, char __user *buf, size_t count,
    loff_t* f_pos);
23. static ssize_t hello_write(struct file* filp, const char __user *buf, size_t
    count, loff_t* f_pos);
24.
```

```
25. /*設備文件操作方法表*/
26. static struct file_operations hello_fops = {
27.     .owner = THIS_MODULE,
28.     .open = hello_open,
29.     .release = hello_release,
30.     .read = hello_read,
31.     .write = hello_write,
32. };
33. 
34. /*訪問設置屬性方法*/
35. static ssize_t hello_val_show(struct device* dev, struct device_attribute* attr,
    char* buf);
36. static ssize_t hello_val_store(struct device* dev, struct device_attribute* attr,
    const char* buf, size_t count);
37. 
38. /*定義設備屬性*/
39. static DEVICE_ATTR(val, S_IRUGO | S_IWUSR, hello_val_show, hello_val_store);
40. 
41. /*打開設備方法*/
42. static int hello_open(struct inode* inode, struct file* filp) {
43.     struct hello_android_dev* dev;
44. 
45.     /*將自定義設備結構體保存在文件指針的私有數據域中，以便訪問設備時拿來用*/
46.     dev = container_of(inode->i_cdev, struct hello_android_dev, dev);
47.     filp->private_data = dev;
48. 
49.     return 0;
50. }
51. 
52. /*設備文件釋放時調用，空實現*/
53. static int hello_release(struct inode* inode, struct file* filp) {
54.     return 0;
55. }
56. 
57. /*讀取設備的寄存器val的值*/
58. static ssize_t hello_read(struct file* filp, char __user *buf, size_t count,
    loff_t* f_pos) {
59.     ssize_t err = 0;
60.     struct hello_android_dev* dev = filp->private_data;
```

```
61.
62.     /*同步訪問*/
63.     if(down_interruptible(&(dev->sem))) {
64.         return -ERESTARTSYS;
65.     }
66.
67.     if(count < sizeof(dev->val)) {
68.         goto out;
69.     }
70.
71.     /*將寄存器val的值拷貝到用戶提供的緩衝區*/
72.     if(copy_to_user(buf, &(dev->val), sizeof(dev->val))) {
73.         err = -EFAULT;
74.         goto out;
75.     }
76.
77.     err = sizeof(dev->val);
78.
79. out:
80.     up(&(dev->sem));
81.     return err;
82. }
83.
84. /*寫設備的寄存器值val*/
85. static ssize_t hello_write(struct file* filp, const char __user *buf, size_t
    count, loff_t* f_pos) {
86.     struct hello_android_dev* dev = filp->private_data;
87.     ssize_t err = 0;
88.
89.     /*同步訪問*/
90.     if(down_interruptible(&(dev->sem))) {
91.         return -ERESTARTSYS;
92.     }
93.
94.     if(count != sizeof(dev->val)) {
95.         goto out;
96.     }
97.
98.     /*將用戶提供的緩衝區的值寫到設備寄存器去*/
```

```
99.     if(copy_from_user(&(dev->val), buf, count)) {
100.            err = -EFAULT;
101.            goto out;
102.        }
103.
104.        err = sizeof(dev->val);
105.
106.    out:
107.        up(&(dev->sem));
108.        return err;
109.    }
110.
111.    /*讀取寄存器val的值到緩衝區buf中，內部使用*/
112.    static ssize_t __hello_get_val(struct hello_android_dev* dev, char* buf) {
113.        int val = 0;
114.
115.        /*同步訪問*/
116.        if(down_interruptible(&(dev->sem))) {
117.            return -ERESTARTSYS;
118.        }
119.
120.        val = dev->val;
121.        up(&(dev->sem));
122.
123.        return snprintf(buf, PAGE_SIZE, "%d\n", val);
124.    }
125.
126.    /*把緩衝區buf的值寫到設備寄存器val中去，內部使用*/
127.    static ssize_t __hello_set_val(struct hello_android_dev* dev, const char* buf,
    size_t count) {
128.        int val = 0;
129.
130.        /*將字符串轉換成數字*/
131.        val = simple_strtol(buf, NULL, 10);
132.
133.        /*同步訪問*/
134.        if(down_interruptible(&(dev->sem))) {
135.            return -ERESTARTSYS;
136.        }
```

```
137.
138.        dev->val = val;
139.        up(&(dev->sem));
140.
141.        return count;
142.    }
143.
144.    /*讀取設備屬性val*/
145.    static ssize_t hello_val_show(struct device* dev, struct device_attribute*
    attr, char* buf) {
146.        struct hello_android_dev* hdev = (struct
    hello_android_dev*)dev_get_drvdata(dev);
147.
148.        return __hello_get_val(hdev, buf);
149.    }
150.
151.    /*寫設備屬性val*/
152.    static ssize_t hello_val_store(struct device* dev, struct device_attribute*
    attr, const char* buf, size_t count) {
153.        struct hello_android_dev* hdev = (struct
    hello_android_dev*)dev_get_drvdata(dev);
154.
155.        return __hello_set_val(hdev, buf, count);
156.    }
157.
158.    /*讀取設備寄存器val的值，保存在page緩衝區中*/
159.    static ssize_t hello_proc_read(char* page, char** start, off_t off, int count,
    int* eof, void* data) {
160.        if(off > 0) {
161.            *eof = 1;
162.            return 0;
163.        }
164.
165.        return __hello_get_val(hello_dev, page);
166.    }
167.
168.    /*把緩衝區的值buff保存到設備寄存器val中去*/
169.    static ssize_t hello_proc_write(struct file* filp, const char __user *buff,
    unsigned long len, void* data) {
```

```
170.        int err = 0;
171.        char* page = NULL;
172.    
173.        if(len > PAGE_SIZE) {
174.            printk(KERN_ALERT"The buff is too large: %lu.\n", len);
175.            return -EFAULT;
176.        }
177.    
178.        page = (char*)__get_free_page(GFP_KERNEL);
179.        if(!page) {
180.            printk(KERN_ALERT"Failed to alloc page.\n");
181.            return -ENOMEM;
182.        }
183.    
184.        /*先把用戶提供的緩衝區值拷貝到內核緩衝區中去*/
185.        if(copy_from_user(page, buff, len)) {
186.            printk(KERN_ALERT"Failed to copy buff from user.\n");
187.            err = -EFAULT;
188.            goto out;
189.        }
190.    
191.        err = __hello_set_val(hello_dev, page, len);
192.    
193.    out:
194.        free_page((unsigned long)page);
195.        return err;
196.    }
197.    
198.    /*創建/proc/hello文件*/
199.    static void hello_create_proc(void) {
200.        struct proc_dir_entry* entry;
201.    
202.        entry = create_proc_entry(HELLO_DEVICE_PROC_NAME, 0, NULL);
203.        if(entry) {
204.            //entry->owner = THIS_MODULE;
205.            entry->read_proc = hello_proc_read;
206.            entry->write_proc = hello_proc_write;
207.        }
208.    }
```

```
209.
210.    /*刪除/proc/hello文件*/
211.    static void hello_remove_proc(void) {
212.        remove_proc_entry(HELLO_DEVICE_PROC_NAME, NULL);
213.    }
214.
215.
216.    /*初始化設備*/
217.    static int  __hello_setup_dev(struct hello_android_dev* dev) {
218.        int err;
219.        dev_t devno = MKDEV(hello_major, hello_minor);
220.
221.        memset(dev, 0, sizeof(struct hello_android_dev));
222.
223.        cdev_init(&(dev->dev), &hello_fops);
224.        dev->dev.owner = THIS_MODULE;
225.        dev->dev.ops = &hello_fops;
226.
227.        /*註冊字符設備*/
228.        err = cdev_add(&(dev->dev),devno, 1);
229.        if(err) {
230.            return err;
231.        }
232.
233.        /*初始化信號量和寄存器val的值*/
234.        //init_MUTEX(&(dev->sem));
235.        sema_init(&(dev->sem), 1);
236.        dev->val = 0;
237.
238.        return 0;
239.    }
240.
241.    /*模塊加載方法*/
242.    static int __init hello_init(void){
243.        int err = -1;
244.        dev_t dev = 0;
245.        struct device* temp = NULL;
246.
247.        printk(KERN_ALERT"Initializing hello device.\n");
```

```
248.
249.        /*動態分配主設備和從設備號*/
250.        err = alloc_chrdev_region(&dev, 0, 1, HELLO_DEVICE_NODE_NAME);
251.        if(err < 0) {
252.            printk(KERN_ALERT"Failed to alloc char dev region.\n");
253.            goto fail;
254.        }
255.
256.        hello_major = MAJOR(dev);
257.        hello_minor = MINOR(dev);
258.
259.        /*分配helo設備結構體變量*/
260.        hello_dev = kmalloc(sizeof(struct hello_android_dev), GFP_KERNEL);
261.        if(!hello_dev) {
262.            err = -ENOMEM;
263.            printk(KERN_ALERT"Failed to alloc hello_dev.\n");
264.            goto unregister;
265.        }
266.
267.        /*初始化設備*/
268.        err = __hello_setup_dev(hello_dev);
269.        if(err) {
270.            printk(KERN_ALERT"Failed to setup dev: %d.\n", err);
271.            goto cleanup;
272.        }
273.
274.        /*在/sys/class/目錄下創建設備類別目錄hello*/
275.        hello_class = class_create(THIS_MODULE, HELLO_DEVICE_CLASS_NAME);
276.        if(IS_ERR(hello_class)) {
277.            err = PTR_ERR(hello_class);
278.            printk(KERN_ALERT"Failed to create hello class.\n");
279.            goto destroy_cdev;
280.        }
281.
282.        /*在/dev/目錄和/sys/class/hello目錄下分別創建設備文件hello*/
283.        temp = device_create(hello_class, NULL, dev, "%s",
    HELLO_DEVICE_FILE_NAME);
284.        if(IS_ERR(temp)) {
285.            err = PTR_ERR(temp);
```

```
286.            printk(KERN_ALERT"Failed to create hello device.");
287.            goto destroy_class;
288.        }
289.
290.        /*在/sys/class/hello/hello目錄下創建屬性文件val*/
291.        err = device_create_file(temp, &dev_attr_val);
292.        if(err < 0) {
293.            printk(KERN_ALERT"Failed to create attribute val.");
294.            goto destroy_device;
295.        }
296.
297.        dev_set_drvdata(temp, hello_dev);
298.
299.        /*創建/proc/hello文件*/
300.        hello_create_proc();
301.
302.        printk(KERN_ALERT"Succedded to initialize hello device.\n");
303.        return 0;
304.
305.    destroy_device:
306.        device_destroy(hello_class, dev);
307.
308.    destroy_class:
309.        class_destroy(hello_class);
310.
311.    destroy_cdev:
312.        cdev_del(&(hello_dev->dev));
313.
314.    cleanup:
315.        kfree(hello_dev);
316.
317.    unregister:
318.        unregister_chrdev_region(MKDEV(hello_major, hello_minor), 1);
319.
320.    fail:
321.        return err;
322.    }
323.
324.    /*模塊卸載方法*/
```

```
325.  static void __exit hello_exit(void) {
326.      dev_t devno = MKDEV(hello_major, hello_minor);
327.
328.      printk(KERN_ALERT"Destroy hello device.\n");
329.
330.      /*删除/proc/hello文件*/
331.      hello_remove_proc();
332.
333.      /*銷毀設備類別和設備*/
334.      if(hello_class) {
335.          device_destroy(hello_class, MKDEV(hello_major, hello_minor));
336.          class_destroy(hello_class);
337.      }
338.
339.      /*删除字符設備和釋放設備內存*/
340.      if(hello_dev) {
341.          cdev_del(&(hello_dev->dev));
342.          kfree(hello_dev);
343.      }
344.
345.      /*釋放設備號*/
346.      unregister_chrdev_region(devno, 1);
347.  }
348.
349.  MODULE_LICENSE("GPL");
350.  MODULE_DESCRIPTION("First Android Driver");
351.
352.  module_init(hello_init);
353.  module_exit(hello_exit);
```

Kconfig:

[plain] view plaincopy

```
1.  config HELLO
2.      tristate "First Android Driver."
3.      default y
4.      help
5.        This is the first android driver.
```

Makefile:

[plain] view plaincopy

```
1. obj-$(CONFIG_HELLO) += hello.o
```

代碼放在kernel/drivers

在kernel/drivers/Kconfig 中加入 source "drivers/hello/Kconfig"

在kernel/drivers/Makefile中加入obj-$(CONFIG_HELLO) += hello/

讓編譯內核的時候能夠編譯加入的代碼。

可以通過cat /proc/hello 來讀取，或者用echo 10 > /proc/hello來寫入

2. HAL層

hello.h

[cpp] view plaincopy

```
1. #ifndef ANDROID_HELLO_INTERFACE_H
2. #define ANDROID_HELLO_INTERFACE_H
3. #include <hardware/hardware.h>
4.
5. __BEGIN_DECLS
6.
7. /*定義模塊ID*/
8. #define HELLO_HARDWARE_MODULE_ID "hello"
9.
10. /*硬件模塊結構體*/
11. struct hello_module_t {
12.     struct hw_module_t common;
13. };
14.
15. /*硬件接口結構體*/
16. struct hello_device_t {
17.     struct hw_device_t common;
```

```
18.     int fd;
19.     int (*set_val)(struct hello_device_t* dev, int val);
20.     int (*get_val)(struct hello_device_t* dev, int* val);
21. };
22. 
23. __END_DECLS
24. 
25. #endif
```

hello.c

[cpp] view plaincopy

```
1.  #define LOG_TAG "HelloStub"
2.  
3.  #include <fcntl.h>
4.  #include <errno.h>
5.  #include <cutils/log.h>
6.  #include <cutils/atomic.h>
7.  #include <hardware/hardware.h>
8.  #include "hello.h"
9.  
10. #define DEVICE_NAME "/dev/hello"
11. #define MODULE_NAME "Hello"
12. #define MODULE_AUTHOR "shyluo@gmail.com"
13. 
14. /*設備打開和關閉接口*/
15. static int hello_device_open(const struct hw_module_t* module, const char* name,
    struct hw_device_t** device);
16. static int hello_device_close(struct hw_device_t* device);
17. 
18. /*設備訪問接口*/
19. static int hello_set_val(struct hello_device_t* dev, int val);
20. static int hello_get_val(struct hello_device_t* dev, int* val);
21. 
22. /*模塊方法表*/
23. static struct hw_module_methods_t hello_module_methods = {
24.     open: hello_device_open
```

```
};

/*模塊實例變量*/
struct hello_module_t HAL_MODULE_INFO_SYM = {
    common: {
        tag: HARDWARE_MODULE_TAG,
        version_major: 1,
        version_minor: 0,
        id: HELLO_HARDWARE_MODULE_ID,
        name: MODULE_NAME,
        author: MODULE_AUTHOR,
        methods: &hello_module_methods,
    }
};

static int hello_device_open(const struct hw_module_t* module, const char* name,
struct hw_device_t** device)
{
    struct hello_device_t* dev;

    dev = (struct hello_device_t*)malloc(sizeof(struct hello_device_t));
    if(!dev) {
        ALOGE("Hello Stub: failed to alloc space.\n");
        return -EFAULT;
    }

    memset(dev, 0, sizeof(struct hello_device_t));
    dev->common.tag = HARDWARE_DEVICE_TAG;
    dev->common.version = 0;
    dev->common.module = (hw_module_t*)module;
    dev->common.close = hello_device_close;
    dev->set_val = hello_set_val;
    dev->get_val = hello_get_val;

    if((dev->fd = open(DEVICE_NAME, O_RDWR)) == -1) {
        ALOGE("Hello Stub: failed to open /dev/hello -- %s.\n", strerror(errno));
        free(dev);
        return -EFAULT;
    }
```

```
63.
64.     *device = &(dev->common);
65.     ALOGE("Hello Stub: open /dev/hello successfully.\n");
66.
67.     return 0;
68. }
69.
70. static int hello_device_close(struct hw_device_t* device)
71. {
72.     struct hello_device_t* hello_device = (struct hello_device_t*)device;
73.
74.     if(hello_device) {
75.         close(hello_device->fd);
76.         free(hello_device);
77.     }
78.
79.     return 0;
80. }
81.
82. static int hello_set_val(struct hello_device_t* dev, int val)
83. {
84.     ALOGE("Hello Stub: set value %d to device.\n", val);
85.
86.     write(dev->fd, &val, sizeof(val));
87.
88.     return 0;
89. }
90.
91. static int hello_get_val(struct hello_device_t* dev, int* val)
92. {
93.     if(!val) {
94.         ALOGE("Hello Stub: error val pointer.\n");
95.         return -EFAULT;
96.     }
97.
98.     read(dev->fd, val, sizeof(*val));
99.
100.        ALOGE("Hello Stub: get value %d from device.\n", *val);
101.
```

```
102.        return 0;
103.    }
```

Android.mk

[plain] view plaincopy

```
1. LOCAL_PATH := $(call my-dir)
2. include $(CLEAR_VARS)
3. LOCAL_MODULE_TAGS := optional
4. LOCAL_PRELINK_MODULE := false
5. LOCAL_MODULE_PATH := $(TARGET_OUT_SHARED_LIBRARIES)/hw
6. LOCAL_SHARED_LIBRARIES := liblog
7. LOCAL_C_INCLUDES += hardware/tecom/include
8. LOCAL_SRC_FILES := hello.c
9. LOCAL_MODULE := hello.default
10. include $(BUILD_SHARED_LIBRARY)
```

代碼放在hardware/tecom

3. JNI層

com_android_server_HelloService.cpp

[cpp] view plaincopy

```
1. #include "jni.h"
2. #include "JNIHelp.h"
3. #include "android_runtime/AndroidRuntime.h"
4. #include <utils/misc.h>
5. #include <utils/Log.h>
6. #include <stdio.h>
7. #include <hardware/hardware.h>
8. #include "hello.h"
9.
10. #define LOG_TAG "HelloService"
11.
```

```
12. namespace android
13. {
14.     /*在硬件抽象層中定義的硬件訪問結構體，參考<hardware/hello.h>*/
15.     struct hello_device_t* hello_device = NULL;
16.     /*通過硬件抽象層定義的硬件訪問接口設置硬件寄存器val的值*/
17.     static void hello_setVal(JNIEnv* env, jobject clazz, jint value) {
18.         int val = value;
19.         ALOGE("Hello JNI: set value %d to device.\n", val);
20.         if(!hello_device) {
21.             ALOGE("Hello JNI: device is not open.\n");
22.             return;
23.         }
24.
25.         hello_device->set_val(hello_device, val);
26.     }
27.         /*通過硬件抽象層定義的硬件訪問接口讀取硬件寄存器val的值*/
28.     static jint hello_getVal(JNIEnv* env, jobject clazz) {
29.         int val = 0;
30.         if(!hello_device) {
31.             ALOGE("Hello JNI: device is not open.\n");
32.             return val;
33.         }
34.         hello_device->get_val(hello_device, &val);
35.
36.         ALOGE("Hello JNI: get value %d from device.\n", val);
37.
38.         return val;
39.     }
40.         /*通過硬件抽象層定義的硬件模塊打開接口打開硬件設備*/
41.     static inline int hello_device_open(const hw_module_t* module, struct
    hello_device_t** device) {
42.         return module->methods->open(module, HELLO_HARDWARE_MODULE_ID, (struct
    hw_device_t**)device);
43.     }
44.         /*通過硬件模塊ID來加載指定的硬件抽象層模塊並打開硬件*/
45.     static jboolean hello_init(JNIEnv* env, jclass clazz) {
46.         hello_module_t* module;
47.
48.         ALOGE("Hello JNI: initializing......\n");
```

```
        if(hw_get_module(HELLO_HARDWARE_MODULE_ID, (const struct
    hw_module_t**)&module) == 0) {
            ALOGE("Hello JNI: hello Stub found.\n");
            if(hello_device_open(&(module->common), &hello_device) == 0) {
                ALOGE("Hello JNI: hello device is open.\n");
                return 0;
            }
            ALOGE("Hello JNI: failed to open hello device.\n");
            return -1;
        }
        ALOGE("Hello JNI: failed to get hello stub module.\n");
        return -1;
    }
        /*JNI方法表*/
    static const JNINativeMethod method_table[] = {
        {"init_native", "()Z", (void*)hello_init},
        {"setVal_native", "(I)V", (void*)hello_setVal},
        {"getVal_native", "()I", (void*)hello_getVal},
    };
        /*註冊JNI方法*/
    int register_android_server_HelloService(JNIEnv *env) {
            return jniRegisterNativeMethods(env,
    "com/android/server/HelloService", method_table, NELEM(method_table));
    }
};
```

代碼放在frameworks/base/services/jni/com_android_server_HelloService.cpp

修改該目錄下的Android.mk，將文件加入到參數LOCAL_SRC_FILES中。

修改onload.cpp，聲明並調用cpp中定義的函數

int register_android_server_HelloService(JNIEnv *env);

（這個可以在onload.cpp找到參考，是像系統註冊JNI方法。）

4. Service

HelloService.java

[java] view plaincopy

```java
package com.android.server;

import android.content.Context;
import android.util.Log;
import android.os.IHelloService;
import android.util.Slog;

public class HelloService extends IHelloService.Stub
{
    private static final String LOG_TAG = "HelloService";
    HelloService() {
        Log.i(LOG_TAG, "call init_native");
        init_native();
    }
    public void setVal(int val) {
        Log.i(LOG_TAG, "call setVal_native");
        setVal_native(val);
    }
    public int getVal() {
        Log.i(LOG_TAG, "call getVal_native");
        return getVal_native();
    }

    private static native boolean init_native();
    private static native void setVal_native(int val);
    private static native int getVal_native();
};
```

代碼放在frameworks/base/services/java/com/android/server/HelloService.java

需要修改該目錄中的SystemServer.java，添加如下代碼：

[java] view plaincopy

```
1. try {
2. <span style="white-space:pre">  </span>Slog.i(TAG, "Hello Service");
3. <span style="white-space:pre">  </span>ServiceManager.addService("hello", new
   HelloService());
4. } catch (Throwable e) {
5. <span style="white-space:pre">  </span>reportWtf("starting Hello Service", e);
6. }
```

向系統添加服務hello

5. AIDL

[cpp] view plaincopy

```
1. package android.os;
2.
3. interface IHelloService
4. {
5.         void setVal(int val);
6.         int getVal();
7. }
```

代碼是：frameworks/base/core/java/android/os/IHelloService.aidl

編譯後有IHelloService.Stub

6. Java程序

Hello.java

[java] view plaincopy

```
1. package com.tecom.hello;
2.
3. //import com.tecom.hello.R;
4. import android.app.Activity;
5. import android.os.ServiceManager;
6. import android.os.Bundle;
```

```
7.  import android.os.IHelloService;
8.  import android.os.RemoteException;
9.  import android.util.Log;
10. import android.view.View;
11. import android.view.View.OnClickListener;
12. import android.widget.Button;
13. import android.widget.EditText;
14.
15. public class Hello extends Activity implements OnClickListener {
16.     private final static String LOG_TAG = "com.tecom.Hello";
17.
18.     private IHelloService helloService = null;
19.
20.     private EditText valueText = null;
21.     private Button readButton = null;
22.     private Button writeButton = null;
23.     private Button clearButton = null;
24.
25.     /** Called when the activity is first created. */
26.     @Override
27.     public void onCreate(Bundle savedInstanceState) {
28.         super.onCreate(savedInstanceState);
29.         setContentView(R.layout.activity_hello);
30.
31.     helloService = IHelloService.Stub.asInterface(
32.         ServiceManager.getService("hello"));
33.
34.         valueText = (EditText)findViewById(R.id.edit_value);
35.         readButton = (Button)findViewById(R.id.button_read);
36.         writeButton = (Button)findViewById(R.id.button_write);
37.         clearButton = (Button)findViewById(R.id.button_clear);
38.
39.     readButton.setOnClickListener(this);
40.     writeButton.setOnClickListener(this);
41.     clearButton.setOnClickListener(this);
42.
43.         Log.i(LOG_TAG, "Hello Activity Created");
44.     }
45.
```

```
46.     @Override
47.     public void onClick(View v) {
48.         if(v.equals(readButton)) {
49.         try {
50.             Log.i(LOG_TAG, "call getVal.");
51.                 int val = helloService.getVal();
52.                 String text = String.valueOf(val);
53.                 valueText.setText(text);
54.         } catch (RemoteException e) {
55.             Log.e(LOG_TAG, "Remote Exception while reading value from device.");
56.         }
57.         }
58.         else if(v.equals(writeButton)) {
59.         try {
60.                 String text = valueText.getText().toString();
61.                 int val = Integer.parseInt(text);
62.             Log.i(LOG_TAG, "call setVal.");
63.             helloService.setVal(val);
64.         } catch (RemoteException e) {
65.             Log.e(LOG_TAG, "Remote Exception while writing value to device.");
66.         }
67.         }
68.         else if(v.equals(clearButton)) {
69.             String text = "";
70.             valueText.setText(text);
71.         }
72.     }
73. }
```

AndroidManifest.xml

[html] view plaincopy

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.     package="com.tecom.hello"
4.     android:versionCode="1"
5.     android:versionName="1.0" >
```

```
6.
7.      <uses-sdk
8.          android:minSdkVersion="8"
9.          android:targetSdkVersion="17" />
10.
11.     <application
12.         android:allowBackup="true"
13.         android:icon="@drawable/ic_launcher"
14.         android:label="@string/app_name"
15.         android:theme="@style/AppTheme" >
16.         <activity
17.             android:name="com.tecom.hello.Hello"
18.             android:label="@string/app_name" >
19.             <intent-filter>
20.                 <action android:name="android.intent.action.MAIN" />
21.
22.                 <category android:name="android.intent.category.LAUNCHER" />
23.             </intent-filter>
24.         </activity>
25.     </application>
26.
27.     <uses-permission android:name="android.permission.HARDWARE_TEST" />
28.
29. </manifest>
```

注意：指定改程序有訪問硬件的權限 <uses-permission android:name="android.permission.HARDWARE_TEST" />

activity_hello.xml

[html] view plaincopy

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.      <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.          android:orientation="vertical"
4.          android:layout_width="fill_parent"
5.          android:layout_height="fill_parent">
```

```
6.          <LinearLayout
7.              android:layout_width="fill_parent"
8.              android:layout_height="wrap_content"
9.              android:orientation="vertical"
10.             android:gravity="center">
11.             <TextView
12.                 android:layout_width="wrap_content"
13.                 android:layout_height="wrap_content"
14.                 android:text="@string/value">
15.             </TextView>
16.             <EditText
17.                 android:layout_width="fill_parent"
18.                 android:layout_height="wrap_content"
19.                 android:id="@+id/edit_value"
20.                 android:hint="@string/hint">
21.             </EditText>
22.         </LinearLayout>
23.         <LinearLayout
24.             android:layout_width="fill_parent"
25.             android:layout_height="wrap_content"
26.             android:orientation="horizontal"
27.             android:gravity="center">
28.             <Button
29.                 android:id="@+id/button_read"
30.                 android:layout_width="wrap_content"
31.                 android:layout_height="wrap_content"
32.                 android:text="@string/read">
33.             </Button>
34.             <Button
35.                 android:id="@+id/button_write"
36.                 android:layout_width="wrap_content"
37.                 android:layout_height="wrap_content"
38.                 android:text="@string/write">
39.             </Button>
40.             <Button
41.                 android:id="@+id/button_clear"
42.                 android:layout_width="wrap_content"
43.                 android:layout_height="wrap_content"
44.                 android:text="@string/clear">
```

```
45.           </Button>
46.         </LinearLayout>
47.     </LinearLayout>
```

Android.mk

[plain] view plaincopy

```
1. LOCAL_PATH:= $(call my-dir)
2. include $(CLEAR_VARS)
3. LOCAL_MODULE_TAGS := optional
4. LOCAL_SRC_FILES := $(call all-subdir-java-files)
5. LOCAL_PACKAGE_NAME := Hello
6. include $(BUILD_PACKAGE)
```

代碼放在packages/apps/tecom

java程序可以在windows下使用eclipse來開發，開發完成後要刪除gen目錄後編譯。