Android 於 ARM VersatilePB 之移植技術探討

Porting Android to ARM VersatilePB

林國弘 Kuo-Hung Lin 張世學 Shih-Hsueh Chang
陳仕杰 Shih- Chieh Chen 曾紹崟 Shau-Yin Tseng
嵌入式系統軟體部

摘要

自從Google發表手機開發軟體平台-Android後，馬上引起全世界自由軟體的社群積極參予和產學界的高度注視。在 e-linux 的新聞「2009 年 Linux 與 open source 十大預測」[1]更是指出，2009 年將是 Android 一個重要的里程碑，即將在手機領域大放異彩。隨著首支 Android 手機(T-Mobile G1)的上市以及 Android 原始碼releae-1.0 正式釋出，因而得以更深入 Android 的研究以及擴展其功能。本文論述如何將 Android 一步一步移植至 ARM 的 Versatile 開發平台，以拋磚引玉精神使國內學業界能更快速的學習與發展 Android 平台。

1. 簡介

以往的手機系統往往是手機製造商所自行研發之封閉性系統，但隨著市場的需求，希望能使其有更大彈性，開放系統也開始應用於手機上(如：WinCE、Symbian)，通常稱此類型手機為智慧型手機(Smartphone)。

隨著智慧型手機在手機市場上開始佔有一席之地，Linux 也將其觸角伸進智慧型手機。部份廠商開始研製以 Linux 為基礎之智慧型手機。手機開發軟體社群也一一出現，如: LiMO[2]、Openmoko[3]。這個趨勢更因 Google 於 2007 年發表以 Linux 為基礎之平台—Android 而受到更大的矚目[4]。

Android 不同於眾多開放式手持 Linux 開放平台，Android 的重點不在於 Linux kernel 本身。Android 最大特色為"完整且開放的軟體研發 SDK、Framework"。Openmoko 手機上可以執行 Android[5]，由此可見其定位方向的不同。另外，近期 Intel 主導之 Moblin[6]計畫，其架構與應用對象與 Android 略有相似。Moblin 是 Mobile + Linux 的簡寫。Intel 針對 MID(mobile internet device)架構，所設計的 Linux 作業系統，並對 Intel Atom 做了最佳化。與 Android 不同之處是，Moblin 並未針對手機的特性做設計，如通訊、簡訊、GPRS/CDMA 上網等等並無特別的支援。Android 提供完善的開發工具與環境，開發手機之廠商可因此降低其軟體授權之成本而受惠，而各大研究單位也可藉由此機會踏入智慧型手機

之領域。

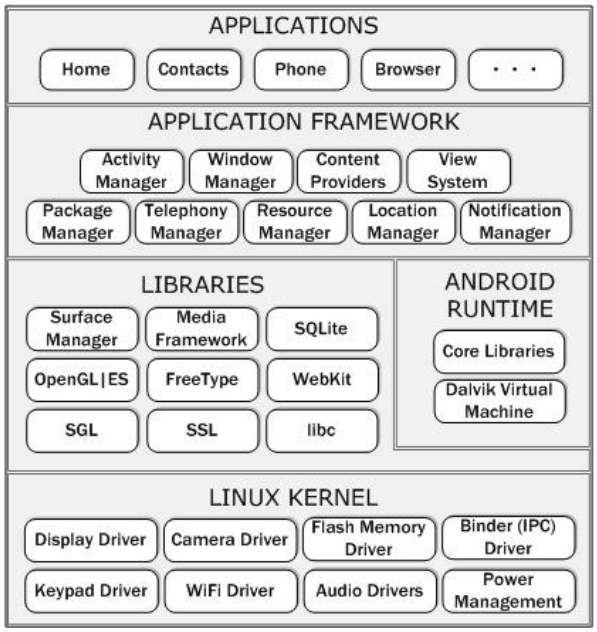
本文將以拋磚引玉精神，詳細的論述如何將 Android 一步一步的移植至 ARM 的開發平台 Versatile，希望能引起漣漪效應，使國內學業界能更快速的學習與發展 Android 平台。

本文的架構如後，在第 2 節將會對 Android 之架構做一初淺之介紹，有了對 Android 初步認識後，將在第 3 節詳細描述移植之過程，在第 4 節介紹 STC 在 Versatile 平台的移植成果，最後在第 5 節對本文做一簡短之總結。

2. Android 系統架構

如圖 1 之 Android 系統架構所示，共分為五個部份: (1)應用程式(Applications)、(2)應用程式架構、(Application Framework)、(3)函式庫(Libraries)、(4)Android 執行環境(Android Runtime)、(5)Linux Kernel。Android 之應用程式利用其所提供之應用程式框架完成各項應用程式之功能。應用程式與應用框架皆是由 Java 語言所開發，運行於 Google 所改良之虛擬機器 Dalvik 之上，它使 Android 更適合運作於嵌入式系統上。Dalvik 不同於一般 JVM(Java Virtual Machine)，最大不同之處在 Dalvik 為 register-based，而 JVM 為 Stack-based。此外，Dalvik 有了許多的改進，提高了執行效率和系統資源需求也較低。另外，Dalvik 也避免商業行為上使用 Java ME 授權等問題[7][8]。Android 上所有與使用者互動之程式皆是由 Java 所撰寫而成，譬如：啟始畫面(Home)、瀏覽器、撥號器等。應用程式框架提供許多不同的系統服務供使用者應用程式使用，譬如：活動管理、視窗管理、套件管理等，其中也包含了不同的硬體管理服務以供程式操作。

除了上述由 Java 所建構之執行環境，底層仍然是需要一些系統函式庫與其配合。這些函式庫包含了多媒體、資料庫、畫面管理等，用以協助應用框架。最底層則是由 Linux Kernel 提供各式的驅動程式與最基本的系統資源管理。而針對不同的硬體平台，所需要修改的部份主要就是 Linux Kernel。



圖一 Android 架構

3. Android 系統移植

自 Google 於 2007 年 11 月公佈 Android 系統，並釋出 SDK 及部份程式原始碼(包含 Linux Kernel、Emulator 及 Webkit Browser 等)開始，世界各地紛紛有人嘗試將 Android 系統移植到實體的手機平台，在只有部分程式原始碼公佈的情況下，這不是一項容易的工作。第一位在這方面取得進展的是 Open Kernel Labs 的 Benno[9]，他在他的部落格描述了嘗試將 Android 移植到 Neo

1973 的過程，雖然因為硬體相容性的問題，當時他並還沒成功，但他所使用的方法及嘗試過程的心得卻對整個社群產生了漣漪效應，之後陸陸續續有人成功將 Android 移植到各種實際的硬體平台，STC 也成功地將 Android 移植到 ARM 的標準開發板— Versatile Platform Baseboard for ARM926EJ-S，簡稱 VersatilePB[10]。到了 2008 年 10 月，Google 正式釋出其 Android 1.0 版的程式原始碼，移植的工作變得更加容易，每個人均可以在自己的開發環境中重新編譯出 Android 系統，甚至修改程式原始碼以符合自己的硬體需求。

本節將對移植的方法做一詳細解說，在說明之前，將先介紹移植時開發環境與實際硬體平台所需要軟硬體需求，之後將對移植平台做一簡介，最後才詳細的論述如何將 Android 一步一步的移植至 ARM 的開發平台 VersatilePB。

3.1 移植所需的軟硬體環境

移植時開發環境與實際硬體平台所需要軟硬體需求如下：

開發環境(host 端):

- 硬體：PC
- 軟體：OS: Linux Ubuntu 8.10。

實際硬體平台(target 端，本文欲移植對象):

- 硬體：ARM versatilePB
- 軟體：Bootloader: U-Boot

3.2 實際硬體平台簡介

ARM VersatilePB 所使用的主處理器核心為 ARM926EJ-S，其指令集架構為 ARMv5T。由於 Android 係以 ARMv5T 之指令集架構為基礎來開發，因此須注意所選用之 ARM 處理器是否支援 ARMv5T 指令集。若是選用不支援 ARMv5T 之處理器，只能在取得完整程式原始碼的狀況下進行移植，在移植時對程式原始碼中使用到 ARMv5T 專有指令的地方做修改。本文所使用之移植平台 ARM VersatilePB 支援 ARMv5T，因此可不必修改 Android 之程式原始碼，亦可直接使用由 SDK 取得的二位元映像檔來進行移植。另外，為了多媒體運用的加速，增加了工研院晶片中心自行研發的 PAC DSP。下面為實際硬體平台之規格：

ARM VersatilePB：

1. ARM926EJ-S MPU
2. 128MB SDRAM
3. 64MB NOR Flash (§128MB NOR Flash)
4. Input: PS/2 Keyboard, PS/2 Mouse
5. Video output: CLCD VGA
6. Audio output: AACI
7. Ethernet
8. PAC DSP

3.3 移植程序

將 Android 移植到實際硬體平台有三個主要步驟：1.準備 Android、2.準備 Linux Kernel、3.準備檔案系統內容，而檔案系統內包含了編譯完成的 Android library。移植流程如圖 2。底下詳細說明各步驟，但在此之前，預設開發環境 PC 已安裝好 Linux Ubuntu 8.10。預設實際硬體平台 VersatilePB 已經安裝好 U-Boot。

3.3.1 準備 Android

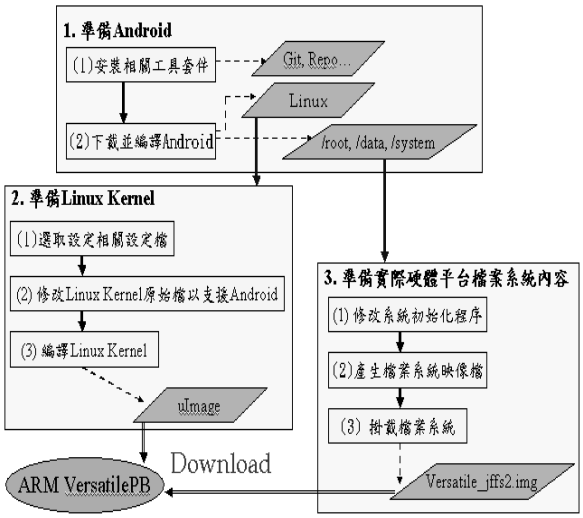
準備 Android 相關工作，主要分為兩個步驟: (1) 安裝相關工具套件，與(2)下載並編譯 Android。底下詳細說明各步驟，更進一步說明可

§ ARM VersatilePB Revision E 將 NOR Flash 擴充為 128MB

以參考 Android Open Source Project[11]。

(1) 安裝相關工具套件

為了下載及編譯 Android 原始碼，必須在開發環境 PC 上安裝相關工具套件，所需的套件及



圖二　移植之流程

其步驟說明如下:

(A) 安裝套件: Git 1.5.4

```
$ sudo apt-get install git-core gnupg
```

(B) 安裝套件: JDK 5.0 [建議使用 JDK5.0，若用 JDK6.0 在使用 Andorid SDK (Eclipse)時會出錯]

```
$ sudo apt-get install sun-java5-jdk
```

(C) 安裝套件: flex,bison,gperf,libsdl-dev, libesd0-dev,libwxgtk2.6-dev(optional),build-essential, zip, curl.

```
$ sudo apt-get install flex bison gperf
libsdl-dev  libesd0-dev  libwxgtk2.6-dev
build-essential zip curl libncurses5-dev
zlib1g-dev
```

(D) 安裝套件: Valgrind

```
$ sudo apt-get install valgrind
```

(E) 安裝套件: Intrepid (8.10)

```
$ sudo apt-get install libreadline5-dev
```

(F) 安裝套件: Repo

```
$ cd ~
$ mkdir bin
$ vim .bashrc
PATH=/home/xxx/bin:${PATH}
$ echo $PATH
$curl
http://android.git.kernel.org/repo>~/bi
n/repo
$ chmod a+x ~/bin/repo
```

(2) 下載並編譯 Android

安裝完成所需的工具套件後，接下來透過 Repo 下載 Android 原始碼並且編譯 Android。在 Android 原始碼中，除了 Android 本身 library 和檔案系統外，也包含針對 Android 修改的 Linux Kernel – linux2.6.25 以及 tool-chain – arm-eabi-4.2.1。詳細步驟如下:

(A) 建立 Android 工作資料夾

```
$ mkdir mydroid
$ cd mydroid
```

(B) 利用 Repo 下載 Android release-1.0

```
$repo init -u
git://android.git.kernel.org/platform/m
```

```
anifest.git –b release-1.0

$ repo sync
```

(C) 編譯 Android

```
$ cd ~/mydroid

$ make
```

因為 gcc 版本問題，編譯時產生找不到 header files 錯誤。此問題為使用 ubuntu 8.10(gcc 4.3.2)才會有的問題，不同的 gcc 版本不一定有此問題。

錯誤如下:

```
frameworks/base/tools/aidl/aidl.cpp: In function ‘int convert_direction(const char*)’:
frameworks/base/tools/aidl/aidl.cpp:69: error: ‘strcmp’ was not declared in this scope
frameworks/base/tools/aidl/aidl.cpp:72: error: ‘strcmp’ was not declared in this scope
frameworks/base/tools/aidl/aidl.cpp: In function ‘void main_import_parsed(buffer_type*)’:
frameworks/base/tools/aidl/aidl.cpp:100: error: ‘malloc’ was not declared in this scope
frameworks/base/tools/aidl/aidl.cpp:101: error: ‘memset’ was not declared in this scope
```

為了解決上面的錯誤，需在下列檔案，增加 include 相關 header files [12][13]][14]。如下：

(a)external/srec/tools/thirdparty/OpenFst/fst/lib/../../fst/lib/vector-fst.h

```
#include <stdlib.h>, #include <string.h>
```

(b)external/srec/tools/thirdparty/OpenFst/fst/lib/symbol-table.cpp

```
#include <stdlib.h>, #include <string.h>
```

(c) build/tools/atree/files.cpp

```
#include <stdlib.h>, #include <string.h>
```

(d) build/tools/atree/fs.cpp

```
#include <stdlib.h>, #include <string.h>
```

(e) frameworks/base/tools/localize/file_utils.cpp

```
#include <stdlib.h>, #include <string.h>
```

(f) frameworks/base/tools/localize/localize.cpp

```
#include <stdlib.h>
```

(g) frameworks/base/tools/localize/Perforce.cpp

```
#include <stdlib.h>, #include <string.h>
```

(h)frameworks/base/tools/localize/XLIFFFile.cpp

```
#include <algorithm>
```

(i)frameworks/base/tools/localize/XMLHandler.cpp

```
#include <algorithm>
```

(j) development/emulator/qtools/dmtrace.cpp

```
#include <unistd.h>
```

(k) frameworks/base/tools/aidl/search_path.h

```
#include <string>
```

(l) frameworks/base/tools/aidl/options.h

```
#include <string.h>
```

(m)frameworks/base/tools/aidl/generate_java.cpp

```
#include <string.h>
```

(n) frameworks/base/tools/aidl/aidl.cpp

```
#include <stdlib.h>, #include <string.h>
```

修改完畢後再重新編輯，即可以產生 Android 相關的檔案系統及 library。

```
$ cd ~/mydroid
$ make
```

make 完後產生的檔案存放在 out/target/product/generic 目錄下，共有 3 個目錄，root、system 與 data，root 為 Android 檔案系統根目錄的基本架構，system 即為/system 目錄的內容，data 即為/data 目錄的內容。

3.3.2 準備 Linux Kernel

Android 是在 Linux 上執行的一個系統，所以我們必須先為 Android 準備一個 Linux 環境，這個 Linux 環境必須符合 Android 的一些要求。在第 3.3.1 節中使用 Repo 下載 Android 原始碼，該份原始碼就包含 Linux Kernel 的程式原始碼。這一份程式原始碼已經經過 Google 增修了許多部分，與 Linux 官方網站的程式原始碼略有不同。主要增修的部份在 Binder 和 Power Management。此外，因為 Android 並非針對特定硬體而開發，所以 Google 也提供了一個虛擬的硬體平台，撰寫了一組裝置驅動程式，以便在 SDK 中的模擬器(Emulator)上執行。

準備 VersatilePB 的 Linux Kernel 主要分為三個步驟: (1) 選取設定相關設定檔、(2) 修改 Linux Kernel 原始檔以支援 Android、(3) 編譯 Linux Kernel。

(1)選取設定相關設定檔

在取得 Kernel 的程式原始碼後，需決定 Kernel 設定檔中哪些功能必須被選取，哪些功能必須被移除。先以硬體預設的組態檔 (config 檔) 為基礎，然後按 Android 需求修改。以 VersatilePB 為例，先以 versatile_defconfig 為基礎，比對 goldfish 的預設組態檔 goldfish_defconfig。

設定組態檔，指令如下:

```
$ cd ~/myandroid/kernel
$ make ARCH=arm versatile_defconfig
$ make menuconfig ARCH=arm
```

設定檔需選取項目和主要注意事項包含如下：

(A) 需選取 Input Event Interface；Android 是透過 event interface 來取得輸入的鍵值或位置，所以在 Kernel 中必須支援此功能。詳細路徑如下:

```
CONFIG_INPUT_EVDEV
  Device Drivers--->Input device
  support--><*> Event interface
```

(B) 選取 Android Binder IPC，移除 OpenBinder；Android 提供了自己的 Binder 介面，作為軟體元件相互溝通的機制，所以移除 OpenBinder 選項，以避免不必要的混淆。詳細路徑如下:

```
ANDROID_BINDER_IPC
  Device Drivers---> Android ---> [*] Binder
  IPC Driver
```

(C) 選取 Real Time Clock (RTC)；Android Power Management 需用到 RTC 的功能，因此 Kernel 中必須支援此功能。詳細路徑如下:

```
CONFIG_RTC_CLASS
  Device Drivers---> Android ---> <*>Real Time
  Clock
```

(D) 選取 Android's Shared Memory Subsystem; Android 所需的 Memory Subsystem。詳細路徑如下:

```
CONFIG_ASHMEM
  General setup--->[*] Enable Android's Shared
  Memory Subsystem
```

(E) 選取 Use the ARM EABI to compile the kernel；因為 Android 的 library 需透過 EABI 與 Linux Kernel 溝通。詳細路徑如下:

```
CONFIG_AEABI
  General setup--->[*] Enable Android's Shared
  Memory Subsystem
```

(F) 選取 Low Memory Killer；當系統 memory 過少時，執行此程序。詳細路徑如下:

```
CONFIG_LOW_MEMORY_KILLER
  Device Drivers--->[*]Misc devices--><*>
  Low Memory Killer
```

(G) 選取 Android power driver。詳細路徑如下:

```
ANDROID_POWER
  Device Drivers---> Android---> [*] Android
  power driver
```

(H) 選取 The DCCP Protocol。詳細路徑如下:

```
CONFIG_IP_PNP_DHCP
  Networking--->Networking options--->[*]IP:
  DHCP support
```

(I) 選取 VGA 8x8 font。詳細路徑如下:

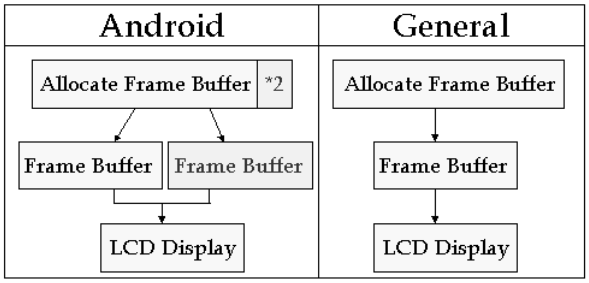
```
CONFIG_FONT_8x8
  Device Drivers--->Graphics support --->
  Console display driver support --->[*] VGA
  8x8 font
```

(J) 選取 VGA 8x16 font。詳細路徑如下:

```
CONFIG_FONT_8x16
  Device Drivers--->Graphics support ---> Console
  display driver support --->[*] VGA 8x16 font
```

(2) 修改 Linux Kernel 原始檔以支援 Android

Android 對於 Video(frame buffer)和 Audio 的裝置驅動程式也有些特殊要求。Video(frame buffer)裝置必須支援 double buffering 的功能，才能正常顯示。而其流程不同之處如圖 3。



圖三　Video (Frame buffer) Driver 之流程不同之處

以下是我們對 Linux Kernel 中與 VersatilePB Video 相關的程式的修改說明：

(A) 修改 arch/arm/mach-versatile/core.c 檔

```
framesize = SZ_1M    改成
framesize=640*480*2*2;
```

(B) 修改 include/linux/amba/clcd.h 檔

```
var->yres_virtual = var->yres = (var->yres + 1)
```

& ~1; 改成

```
var->yres = (var->yres + 1) & ~1;
var->yres_virtual = var->yres * 2;
```

(C) 修改 drivers/video/amba-clcd.c 檔，此檔有 5 個地方要修改

(a) fb->fb.fix.ypanstep=0; 改成

```
fb->fb.fix.ypanstep = 1;
```

(b)fb->fb.var.yres_virtual=fb->panel->mode.yres; 改成

```
fb->fb.var.yres_virtual=
fb->panel->mode.yre * 2;
```

(c) static int clcdfb_set_par(struct fb_info *info)改成

```
static int clcdfb_set_par(struct fb_info *info)
{
   return 0;
}
```

```
static int clcdfb_set_par_first(struct fb_info
*info)
{ ...
}
```

(d) len = info->fix.smem_len; 改成

```
len = info->fix.smem_len + 4096;
// since android will alloc more memory than
// framebuffer size, so we add 4096 for safe
```

(e) fb_set_var(&fb->fb, &fb->fb.var); 改成

```
clcdfb_set_par_first(&fb->fb);
```

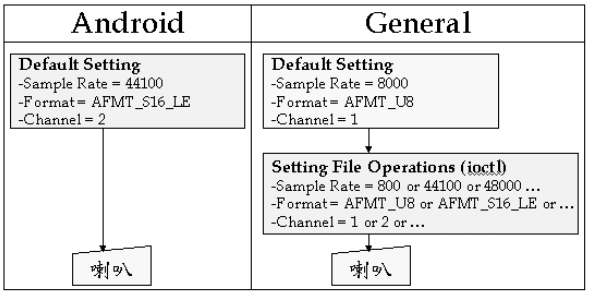
(f)新增 fb_pan_display()函式，實作兩個 buffer 的交互切換。

```
/* this is a new ops added for android display
*/
static int clcdfb_pan_display(struct
fb_var_screeninfo *var, struct fb_info *info)
{
   struct clcd_fb *fb = to_clcd(info);
        unsigned long ustart;
ustart = fb->fb.fix.smem_start +
fb->fb.var.yoffset * fb->fb.fix.line_length;

        writel(ustart, fb->regs + CLCD_UBAS);

        return 0;
}
static struct fb_ops clcdfb_ops = {
...
        .fb_pan_display =
clcdfb_pan_display,
...
};
```

完成上述的修改後，此時 Video 裝置可以支援 double buffering 的功能，正確顯示。

Audio 部分必須採用特定的設定，才能正常播音。Android 在 Audio 方面的著墨較少，由 goldfish_audio 可以得知，Android 並未支援多數標準的 ioctl command , 其播音功能事先已經設定好，並未在執行階段進行修改，其預設值可以從 Android 模擬器的程式原始碼中取得 , sample rate 為 44.1K , format 為 AFMT_S16_LE , channel

數目為 2，需將 Audio 驅動程式的預設值改成這些數值才能正常播音。而其流程不同之處如圖 4。



圖四　Audio Driver 流程之不同之處

以下是對 VersatilePB Audio 相關程式的修改：

(A) 修改 sound/core/oss/pcm_oss.c 檔

(a) `runtime->oss.rate = 8000;` 改成

```
runtime->oss.rate = 44100;
```

(b) `runtime->oss.format = AFMT_U8;` 改成

```
runtime->oss.format = AFMT_S16_LE;
```

(c) `runtime->oss.channels = 1;` 改成

```
runtime->oss.channels = 2;
```

(B) 修改 sound/arm/aaci.c 檔

`ret = snd_pcm_new(aaci->card, "AACI AC'97", 0, 1, 1, &pcm);` 改成

```
ret = snd_pcm_new(aaci->card, "AACI AC'97", 0, 5, 5, &pcm);
```

完成上述的修改後，此時 Audio 裝置可以正確撥音。

(3) 編譯 Linux Kernel

修改必要的檔案後就可編譯 Linux Kernel，指令如下:

```
$ make ARCH=arm
CROSS_COMPILE=../prebuilt/linux-x86/toolcha
in/arm-eabi-4.2.1/bin/arm-eabi- uImage
```

此時因為未安裝 mkimage，所以無法產生 uImage 產生下面警告:

```
Warning:
"mkimage" command not found - U-Boot images will
not be built
```

安裝 mkimage 且重新編譯 Kernel

```
$ sudo apt-get install uboot-mkimage
$ make ARCH=arm
CROSS_COMPILE=../prebuilt/linux-x86/toolcha
in/arm-eabi-4.2.1/bin/arm-eabi- uImage
```

最後產生的 Kernel 影像檔 arch/arm/boot/uImage。透過 u-boot 的 tftp，將 uImage 燒入到 VersatilePB。

```
# tftp
..
Filename '/tftpboot/uImage'.
Load address: 0x7fc0
Loading:
#################################################
#################################################
#####################################
done
Bytes transferred = 1681976 (19aa38 hex)
…


# protect off 0x34080000 0x3423ffff
# erase 0x34080000 0x3423ffff
```

```
# cp 0x7fc0 0x34080000 0x66A8E
# setenv bootcmd 'cp 0x34080000 0x7fc0 0x66A8E ;
bootm'
# saveenv
```

3.3.3 準備實際硬體平台檔案系統內容

經過這些修改所產生的 Linux Kernel 即可符合 Android 系統所需，接下來的步驟是準備開機過程所需的檔案系統，而此檔案系統亦包含了編譯完成的 Android library。修改步驟分別(1) 修改系統初始化程序、(2) 產生檔案系統映像檔、(3) 掛載檔案系統。

(1) 修改系統初始化程序

取得的檔案系統內容主要是針對模擬器的虛擬平台而設計，要移植到實體平台，還有許多工作要做，必須在系統初始化時將 Android 執行時所需的環境準備好，將導向虛擬裝置的設定導向實體裝置，並做必要的修改工作。

(A) 修改 init.rc 檔

init.rc 檔為 Android 初始化程序的設定檔，其中的內容包含環境變數的設定，檔案系統的掛載，必要目錄的建立，權限設定，程式啟動的順序與條件等等。在移植時，首先針對硬體平台的實際配置與使用者的需求調整掛載條件、權限設定，其次將模擬器 qemu 相關的設定均取消。此外，需要增加一些程式，用實體的設定取代模擬器的設定，譬如網路 IP、DNS 設定、時間、音效設定等等。

修改 /out/target/product/nfs/root/init.rc 檔

(註解掉下面幾行)

```
# mount rootfs rootfs / ro remount
# mount yaffs2 mtd@system /system
```

```
# mount yaffs2 mtd@system /system ro remount
# mount yaffs2 mtd@userdata /data nosuid nodev
# mount yaffs2 mtd@cache /cache nosuid nodev
# socket rild-debug stream 660 radio system
```

(B) 修改鍵盤對映檔

鍵盤對映檔的位置在/system/usr/keylayout /qwerty.kl。不同的硬體可能有不同的對映，必須透過 Android 提供的工具 getevent 取得實際的鍵值，然後修改鍵盤對映檔，如此鍵盤才能正確使用。

修改/out/target/product/generic/root/system/ usr/keylayout/ qwerty.kl 檔

key 158 BACK WAKE_DROPPED 改成

```
key 1 BACK WAKE_DROPPED
```

(C) 增加 Android Audio 對應的裝置節點

Android 虛擬硬體並未採用一般的/dev/dsp 做為 Audio 的裝置節點，而是創造了一個新的裝置節點/dev/eac 來對映 Audio，因此，我們有兩個方式來讓音效產生效果：

(a) 在 Linux Kernel 中修改 Audio 的對映裝置節點。

(b) 在/dev 下建立一個與/dev/dsp 相同的裝置節點。

為簡化問題，我們採用第二種方法。指令如下:

```
$ cd /out/target/product/nfs/root/dev
$ sudo mknod eac c 14 3
```

(D) 權限問題的處理

在 Android 系統中，應用程式並非以 root 的權限執行，因此可能產生問題，必須一一解決，修改檔案權限，譬如，由最新釋出的程式原始碼

編譯出來的檔案系統，我們必須修改 /system/usr/keychars 下檔案的權限，讓任何人都可以讀取，否則鍵盤將無法正常作用。

```
$ chmod a+r
/out/target/product/nfs/root/system/usr/keych
ars/qwerty2.kcm.bin
$ chmod a+r
/out/target/product/nfs/root/system/usr/keych
ars/ qwerty.kcm.bin
$ chmod a+r
/out/target/product/nfs/root/system/usr/keych
ars/ tuttle2.kcm.bin
```

(2)產生檔案系統映像檔

移植的最後步驟是產生一個可以放到硬體儲存裝置的檔案系統映像檔，所以需要就硬體的特性來決定該如何產生這個映像檔。Android 預設採用的檔案系統為 yaffs2，yaffs2 是針對 NAND Flash 發展出來的未壓縮檔案系統。用 yaffs2 映像檔有兩個問題，其一是 Linux Kernel 中的 yaffs2 檔案系統驅動程式並未支援 NOR Flash，其二是 yaffs2 並非壓縮的檔案系統，對於儲存空間較少的平台(譬如 VersatilePB)可能無法移植，Android 基本的容量需求約需 55MB，暫存空間的需求至少 13MB。針對第一個問題，我們需要對 Linux Kernel 做 patch；針對第二個問題，我們可能需要對檔案系統進行分割，或者採用有壓縮的檔案系統：

(A) 對 Linux Kernel 做 patch 需修改 fs/yaffs2/下的程式，修改 yaffs_mtdif 相關檔案，以 NOR Flash 的操作模擬 NAND Flash 的動作。

(B) 進行分割或選擇其他檔案系統，jffs2 與 yaffs2 相同，都是針對 Flash 發展的檔案系統，但它同時支援 NOR 和 NAND Flash，此外，jffs2 是壓縮的檔案系統，可以儲存更多的檔案，但是，jffs2 也有一項缺點，它並未實作出 mmap write 的功能。Android 為了增加效能，在程式中使用了許多 mmap 功能，jffs2 未支援 mmap write，確實可能造成問題，然而，Android 並非對所有檔案都採用 mmap，而是針對/data 目錄下的暫存檔採用 mmap，我們將/data 目錄分割出來，採用 tmpfs，tmpfs 為記憶體中的檔案系統，本身即支援 mmap 功能，所以，主要的檔案系統採用 jffs2 並不會造成任何影響，同時可以經由壓縮檔案系統，減少 Flash 儲存空間的容量需求。在 VersatilePB 採用這個方案。

(3) 掛載檔案系統

通常掛載 android 檔案系統的方法有透過 a. nfs 與 b.jffs2 兩種方法。

(A) nfs 掛載檔案系統

因為發展時期常會更換檔案系統，所以使用 nfs 掛載檔案系統。

(a) Host 端設定

(i)安裝 NFS

```
$ sudo apt-get install nfs-common
$ sudo apt-get install nfs-kernel-server
```

(ii)設定掛載點

```
$ sudo vi /etc/exports
/home/xxx/Android/v1.0/out/target/product
/nfs
140.96.28.0/24(rw,async,no_root_squash)
```

(iii)啟動 nfs server

```
$ sudo /etc/init.d/nfs-kernel-server start
```

(b) Target 端設定(透過 u-boot 設定開機參數)

```
# set bootargs=root=/dev/nfs rw
nfsroot=140.96.28.31:/home/xxx/Android/v1
.0/out/target/product/nfs/root init=/init
ip=dhcp mem=128M console=ttyAMA0
video=vc:1-2clcdfb:
# saveenv
```

(B) jffs2 掛載檔案系統

若發展完成，可以讓 ARM versatilePB 獨立發展(standalone)。

(a)Host 端設定

(i)安裝 mkfs.jffs2 [15]

```
$ cvs
-d :pserver:anoncvs@cvs.infradead.org:/ho
me/cvs login (password: anoncvs)
$ cvs
-d :pserver:anoncvs@cvs.infradead.org:/ho
me/cvs co mtd
$ cd /home/echoman/mkfs.jffs2/mtd/util
$ make
```

(ii)修改

/out/target/product/versatile_jffs2/root/init.rc 檔

```
mount tmpfs tmpfs /sqlite_stmt_journals size=4m
```

改成

```
mount tmpfs tmpfs /data size=20m
```

(iii)產生 jffs2 格式檔案系統

```
$ cd /out/target/product/generic/root
$ rm –rf data (將/data mount 成 mem fs)
$ mkdir  data
$ cd ..
$ mkfs.jffs2 -d root -o versatile_jffs2.img
$ make install
```

(b)Target 端設定

(i)下載檔案系統到 ARM VersatilePB

```
# tftp 0x04000000
/tftpboot/versatile_jffs2.img

Using MAC Address 00:02:F7:00:19:34
TFTP from server 140.96.28.78; our IP address
is 140.96.28.84
Filename '/tftpboot/versatile_jffs2.img'.
Load address: 0x4000000
Loading:
#############################################
#########################
done
Bytes transferred = 40281504 (266a5a0 hex)
# protect off 0x34380000 0x36ffffff
# erase 0x34380000 0x36ffffff
# cp 0x04000000 0x34380000 0x99A968
```

(ii)設定 u-boot 開機參數

```
# set bootargs root=/dev/mtdblock0
mtdparts=armflash.0:40M@0x380000(jffs2)
init=/init ip=dhcp mem=128M console=ttyAMA0
rootfstype=jffs2 video=vc:1-2clcdfb:
# saveenv
```

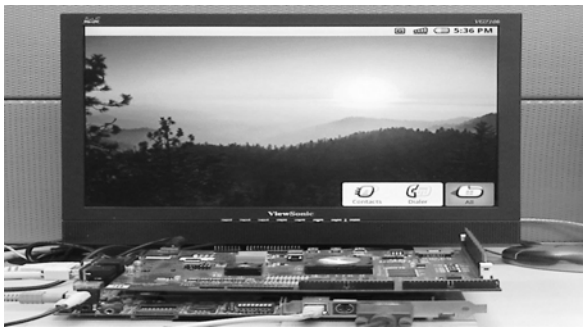
此時，檔案系統內已經包含編譯完成的 Android 相關的 library，開發版也已經完成燒入

kernel。所以，當完成透過 nfs 或 jffs2 掛載檔案系統後，我們將 VersatilePB 重新開機，即可正確開機且執行 Android。

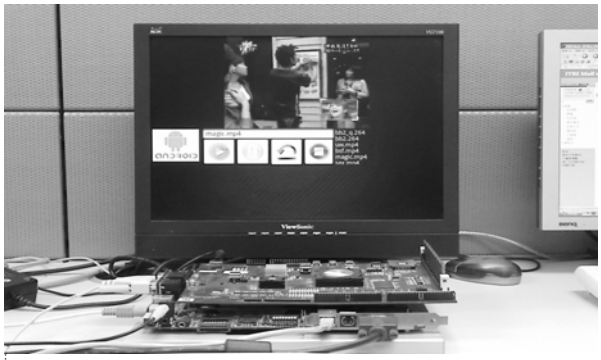
4. 移植成果

圖 5 是依照本文描述移植 Android 至 VersatilePB 平台執行之情境，圖 6 中有上下兩層疊板，下層為 VersatilePB 平台，用來執行 Android，其中 ARM 執行頻率 210MHz ，BUS 頻率 35MHz，SDRAM 128M。上層的板子是 STC 自行開發的 PACDSP 子板，用來做 H.264 的解碼。

如圖 1 所示，Android 是在 Linux 上執行的一個系統，Android 架構的最下層是許多的 Linux 驅動程式，在本文的移植過程中，僅修改與啟動了 memory controller、interrupt controller、timer、UART、flash、ethernet、video (LCD)、audio playback 等的驅動程式，因此將 VersatilePB 重新開機 Android 起來後，即可執行 Ethernet 上網、Browser 網頁瀏覽、電子郵件、行事曆、通訊錄、Audio 播放、MPlayer 播放、Google Map 等基本功能。若要有其他如錄音或是錄影功能，則必須要修改錄音或是錄影所需的驅動程式，這些並不在本文描述範圍。



圖五　Android 在 VersatilePB 平台



圖六　在 Android 平台(VersatilePB + PACDSP)上播放 H.264 影片

如圖 6 所示，STC 也成功修改 Android 原始碼，使播放 H.264 檔案時能透過 PACDSP 子板來解碼，提升 Android 播放 H.264 的效能，但此修改過程也不在本文描述範圍。

5. 結論

本文詳細介紹了 Android 針對 ARM VersatilePB 移植之方法。如何從無到有，下載、修改、編譯且執行在實際硬體平台上。Android 之移植可以分為三個步驟：準備 Android、準備 Linux Kernel、準備檔案系統內容。Android 和檔案系統主要修改地方為 ”與 Linux Kernel 溝通界面部份” 。Linux Kernel 主要修改地方為 ”LCD 和 Audio 支援 Android 部分” 。本文的移植經驗，可以運用於相關的各種平台。有助於產學界更快速學習且發展 Android 平台。隨著 Android 原始碼的開放，可以更深入其內部之架構加以分析或是修改其功能。也因此可以藉由修改其內部功能以符合各種不同硬體平台與應用之需求。在未來，本研究會朝著 Android 內部架構作更深入之研究。同時經由國科會、CIC 以及教育部 ESW 聯盟推廣到學校教學與研究，另外藉由各項技術轉移推廣到業界，協助業界開發以 Android 為基礎之各種行動設備。

6. 參考文獻

[1] 10 predictions for Linux and open source in 2009, http://www.e-linux.it/news_detail.php?id =7731

[2] LiMO, http://www.limofoundation.org/

[3] Openmoko, http://www.openmoko.com/

[4] http://code.google.com/android/

[5] Openmoko with Android, http://chinese.engadget.com/2008/11/06/openmoko-android/

[6] http://tw.myblog.yahoo.com/champ_yen/article?mid=329&sc=1

[7] http://www.lupaworld.com/viewnews-30186.html

[8] moblin, http://moblin.org/

[9] http://benno.id.au/blog/

[10] Platform Baseboard for ARM926EJ-S, http://www.arm.com/products/DevTools/VPB926EJ-S.html

[11] http://source.android.com/download

[12] Add a few missing headers Fixes build with gcc 4.3.2, http://review.source.android.com/Gerrit#change,57

[13] Howto build Android full source for X86 Architecture on Fedora distribution, http://www.mail-archive.com/android-porting@googlegroups.com/msg01066.html

[14] android-framework, http://www.mail-archive.com/android-framework@googlegroups.com/msg00296.html

[15] JFFS2: The Journalling Flash File System, version 2, http://sourceware.org/jffs2/