TI-Android-FroYo-DevKit-V2 UserGuide

TI Android FroYo DevKit V2 User Guide

User Guide - October 25, 2010

About this manual

This document describes how to install and work with Texas Instruments' Android FroYo DevKit release for OMAP35x, AM37x, AM35x platforms running Android. This release package provides a stable Android distribution with integrated SGX (3D graphics) drivers, TI hardware abstraction for video overlay and standard applications from Android. The package also includes Linux Android kernel, tools and documentation to ease development, deployment and execution of Android based systems. The product forms the basis for all Android application development on OMAP35x, AM37x, AM35x platforms. In this context, the document contains instructions to:

- Install the release
- Setting up the hardware
- Steps to use pre-built binaries in the package
- Running Android on the supported platforms
- Setting up the Android debugger “adb” with the hardware platform
- Installing and executing Android (out of market) applications hardware platforms

Installation

This section describes the list of Software and Hardware requirements to evaluate the DevKit release.

Hardware Requirements

This release of Android DevKit V2 is evaluated on the below given list of platforms. This package should be easily portable on other platforms on similar TI devices.

TI Device	Platform Supported	Version	Other Accessories
OMAP35x			
	OMAP35x EVM [1]	Rev G	DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min)
	Beagleboard [2]	Rev Cx	DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min)
AM35x			
	AM3517 Evaluation Module [3]	Rev C	DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min)
AM37x			
	AM37x Evaluation Module [4]	Rev C	DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min)

	BeagleBoard [2]	XM	DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min)

Software Host Requirements

If you are a Android application developer or would like to use Android SDK Tools then refer to http://developer.android.com/sdk/requirements.html for Host PC requirements.

To evaluate this release we recommend you to have a Linux "Ubuntu 8.04 or above" Host machine, See Ubuntu Linux installation notes [5]

Package Content

```
TI_Android_FroYo_DevKit-V2
      |-- Android_Source_Manifest
      |-- Documents
      |   |-- RowboPERF_User_Guide.pdf
      |   |-- Software_Manifests
      |   |-- Test_Performance_Results
      |   |   |-- CTS_Report.tar.gz
      |   |   |-- Performance_Results.pdf
      |   |   `-- Test_Results.pdf
      |   |-- TI-Android-FroYo-DevKit-V2_ReleaseNotes.pdf
      |   `-- TI-Android-FroYo-DevKit-V2_UserGuide.pdf
      |-- OMAP35x_Android_SGX_SDK.tar.gz
      |-- Performance_Apps
      |   |-- 0xbench
      |   |-- rowboatBench
      |   |-- RowboPERF
      |   `-- StorageIO
      |-- Prebuilt_Images
      |   |-- AM35X
      |   |-- AM37X
      |   |-- beagleboard-rev-c4
      |   |-- beagleboard-xm
      |   `-- OMAP35X
      |-- Sources
      |   |-- Android_Linux_Kernel_2_6_32.tar.gz
      |   |-- u-boot-03.00.02.07.tar.gz
      |   `-- x-loader-03.00.02.07.tar.gz
      `-- Tools
          |-- flash-utility.tar.gz
          |-- mk-bootscr
          |-- mk-mmc
          |-- pinmux-utility.tar.gz
          `-- signGP
```

Out of the Box Demo

This section gives the instructions to quickly prepare an SD Card image and get an experience of Android DevKit V2 on TI platforms/devices.

- Download the pre-built Image from http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_00_00/index_FDS.html for the platform you own, can be AM35x EVM, AM37x EVM, Beagleboard Rev C4/XM, OMAP35x EVM.
- Get an SD Card of minimum size 2GBytes and a USB Card reader
- Insert the USB SD Card reader (with SD Card) in your host Linux PC
- Prepare the MMC/SD card Image

```
tar -xzvf <Board name>.tar.gz
cd <Board name>
sudo ./mkmmc-android /dev/sd<device>
```

- The above step prepares the SD Card.
- Setup the board/platform
  - Do the DIP switch settings to boot from SD Card, see the section on DIP switch setting below.
  - Insert the SD Card into the Board
  - Switch on the platform
  - Wait for 35sec to get Android up on the UI screen

```
NOTE: For the first time boot the System might take 4 min to boot.

NOTE: If your NAND flash is not empty the system might not boot with MMC,
      in that case do the following with Serial Console / Terminal prompt in u-boot

#> mmc init
#> fatload mmc 0 0x82000000 boot.scr
#> source 0x82000000
```

  - Click on the TI logo and run the apps you are interested in, look at the RowboPerf user guide for more details about this app http://processors.wiki.ti.com/index.php/RowboPERF_User_Guide.

Android Booting Procedure

Booting Android on any TI platform requires following software components

- Kernel Image (uImage)
- Bootloader (u-boot.bin)
- Bootstrapping (x-load.bin.ift for NAND or MLO for MMC)
- Filesystem (rootfs)

The above listed software components or images can be populated by

- Building sources from this package
- Using the pre-built images in this package

```
NOTE:

- To build software components using sources require "ARM cross compiler tool chain".
```

```
 - The ARM tool chain can be downloaded from Android pre-built repository. Tool Chain [6]
```

Software Integration

This section describes the procedure to compile and integrate all the required software components to boot Android on TI platforms.

Toolchain

Download the tool chain and export it in the default Linux Path.

```
 Example:

#> export PATH=<tool chain install path>/linux-x86/toolchain/arm-eabi-4.4.0/bin/:$PATH
```

Kernel

Untar the kernel source located in the sources directory

```
 #> tar -xzvf Android_Linux_Kernel_2_6_32.tar.gz
```

Execute the following commands to the kernel sources

```
 #> make CROSS_COMPILE=arm-eabi- distclean
```

```
 #> make CROSS_COMPILE=arm-eabi- <default config>

Where default config is

   omap3_evm_android_defconfig         : For OMAP35x, AM37x EVM
   am3517_evm_android_defconfig        : For AM35x EVM
   omap3_beagle_android_defconfig      : For Beagleboard Rev Cx, XM
```

```
 #> make CROSS_COMPILE=arm-eabi- uImage

This command will build the Linux Kernel Image in arch/arm/boot "uImage"
```

u-boot

Untar the u-boot sources located in the sources directory

```
 #> tar -xzvf u-boot-03.00.02.07.tar.gz
```

Execute the following commands to the kernel sources

```
 #> make CROSS_COMPILE=arm-eabi- distclean
```

```
 #> make CROSS_COMPILE=arm-eabi- <default config>

Where default config is

   omap3_evm_config         : For OMAP35x, AM37x EVM
   am3517_evm_config        : For AM35x EVM
   omap3_beagle_config      : For Beagleboard Rev Cx, XM
```

```
 #> make CROSS_COMPILE=arm-eabi-

This command will build the u-boot Image "u-boot.bin"
```

x-loader

Untar the x-loader sources located in the sources directory

```
 #> tar -xzvf x-loader-03.00.02.07.tar.gz
```

Execute the following commands to the kernel sources

```
 #> make CROSS_COMPILE=arm-eabi- distclean
```

```
 #> make CROSS_COMPILE=arm-eabi- <default config>

Where default config is

   omap3evm_config          : For OMAP35x, AM37x EVM
   am3517evm_config         : For AM35x EVM
   omap3beagle_config       : For Beagleboard Rev Cx, XM
```

```
 #> make CROSS_COMPILE=arm-eabi-

This command will build the x-loader Image "x-load.bin"
```

To create the MLO file used for booting from a MMC/SD card, sign the x-loader image using the signGP tool found in the Tools directory of the Devkit.

```
 #> ./signGP ./x-load.bin
```

The signGP tool will create a .ift file, that can be renamed to MLO.

```
NOTE:


- The Pre-built images are provided in this package to help users boot android without building the sources
```

Setting up Hardware

This DevKit release supports five different platforms, OMAP35x EVM, AM37x EVM, AM35x EVM, Beagleboard Rev Cx, Beagleboard XM. While they are different devices the hardware setup will almost remain the same.

- Connect the UART port of the platform to the Host PC and have a Terminal software like TeraTerm, Minicom or Hyperterminal.
- Connect the Ethernet (on Beagle Rev C4 we don't have an Ethernet port)
- Connect Audio Speakers
- For Beagle boards you need to connect DVI Monitor through HDMI connector.
- Use self powered USB port and connect it to USB Host port of the platform, mainly for Beagle and AM35x EVM. For AM37x and OMAP35x EVM the onboard keypad can be used
  - Connect a USB keyboard and USB Mouse to the USB HUB

```
 NOTE:


- The AM35x EVM and Beagleboard have no keypad mappings, user is recommended to use USB Keyboard over a
- Self powered USB HUB connected to the Host port of AM35x EVM or Beagleboard.
```

- Select Appropriate DIP Switch settings on EVM(s) to boot over MMC/SD

For MMC/SD boot - On OMAP35x and AM37x EVM the DIP switch SW4 should be set as shown below

Switch	1	2	3	4	5	6	7	8
State	OFF	ON	ON	ON	OFF	OFF	OFF	OFF

For MMC/SD boot - On AM35x EVM the DIP switch S7 should be set as shown below

Switch	1	2	3	4	5	6	7	8
State	ON	OFF	OFF	ON	OFF	OFF	OFF	ON

Booting Android

TI platforms (Beagle or EVM) can be booted over MMC or NAND or UART. We follow and prefer MMC based booting of platforms.

Procedure to populate MMC/SD Card

Use the mk-mmc utility provided in the tools folder of this package to populate the SD Card. This utility helps users create a MMC/SD Card with required Images to boot Android on any given TI platform.

This will partition the SD card to three partitions namely boot, rootfs and data. 1) The boot partition will get populated with the images required for booting. 2) The rootfs partition will be used as android root filesystem partition. 3) The Media inside the folder Media_Clips will get copied to the data partition. The data partition will get mounted as EXTERNAL storage when Android boots up.

Execute the following command

```
  Example:

  #>./mkmmc-android /dev/sdc MLO u-boot.bin uImage boot.scr rootfs.tar.bz2 Media_Clips

 This populates the SD/MMC card with all the images.
```

```
 NOTE:

   To create the boot.scr boot script use the mkbootscr tool found in the Tools
directory provided in the DevKit.
```

If you want to use the pre built images in the SDK you have to adjust the above mentioned command to take them into account, as a more direct example the commands below will generate a SD card for an OMAP3 EVM. Ensure you have your SD card connected to the Linux machine you are using and that it is in /dev/sdb for this command otherwise adjust the command accordingly (**WARNING**: if you get this wrong it can wipe your HDD). Note that this assumes you installed the SDK in your home (~) directory and that the command is run with sudo (or your preferred way of getting super user privileges) to allow for the reformatting of the SD card.

```
 HOST $ cd ~/TI_Android_FroYo_DevKit-V2/OMAP35X
 HOST $ sudo ../../Tools/mk-mmc/mkmmc-android.sh /dev/sdb
```

Procedure to add Video, Audio and other media

To play media after booting Android on any platform, the content must be included in the MMC/SD card's FAT32 partition. If you use the mk-mmc script included in the release package then it creates 3 partitions. The media content should be placed into the 3rd (FAT32) partition.

```
  Example:

#> sudo mount /dev/sdd3 /mnt
#> sudo cp <all media files> /mnt
#> sudo umount /mnt
```

```
NOTE:


- This release supports all the standard Android media formats, listed here http://developer.android.com/guide/appendix/media-formats.html
```

Booting the platform

Booting over MMC using boot.scr

```
NOTE:


- If the board has bootargs configured already, then the board will not boot for Android automatically,
- It is suggested to either delete the bootargs or use the following commands on u-boot prompt through UART console.
```

```
#> mmc init
#> fatload mmc 0 0x82000000 boot.scr
#> source 0x82000000
```

If the board is not configured for bootargs, then it automatically boots.

Keypad mappings

The below table lists the keypad and USB Keyboard mappings to Android UI functionality.

Functionality	USB Keyboard/Mouse	Keypad on OMAP35x/AM37x EVM	Keypad on AM35x EVM
Home Screen	Home	R3C2	S3
Left	Left Arrow	R2C1	S6
Right	Right Arrow	R0C2	S5
Up	Up Arrow	R1C3	S7
Down	Down Arrow	R2C0	S9
Volume Up	Volume Up	R1C2	S4
Volume Down	Volume Down	R0C1	S1
Contacts	F3		
Select	Enter	R3C1	S2
Back	Mouse right	R2C3	S8
Menu	F1	R3C3	S10

Using DVI Monitor

On OMAP35x, AM37x and AM35x EVMs the on board LCD is used as output device by default. User is allowed to configure DVI port as output device, by changing the boot arguments as shown below.

Append the boot arguments with following text

```
   omapfb.mode=dvi:1280x720MR-16 omapdss.def_disp="dvi"
```

```
   Example:

  To boot over MMC and use DVI at resolution 1024x768 on OMAP35x EVM,
the complete bootargs would be,

  setenv bootargs init=/init console=ttyS0,115200n8 ip=dhcp rw
root=/dev/mmcblk0p2 rw init=/init rootwait mem=256M
androidboot.console=ttyS0 omapfb.mode=dvi:1024x768MR-16
omapdss.def_disp="dvi"
```

We have noticed on few monitors the below bootargs works for DVI.

```
  setenv bootargs console=ttyS0,115200n8 androidboot.console=ttyS0
mem=256M root=/dev/mmcblk0p2 rw rootfstype=ext3 rootdelay=1 init=/init
ip=off mpurate=1000 omap_vout.vid1_static_vrfb_alloc=y vram=16M
omapfb.vram=0:8M,1:4M,2:4M omapfb.mode=dvi:hd720 omapdss.def_disp=dvi
```

```
  NOTE:

  - On beagleboard the DVI port is configured as default output device.
```

Using Network Filesystem

Android filesystem can be mounted over network, the bootargs for doing the same should include below text instead of MMC

```
   ip=dhcp rw root=/dev/nfs nfsroot=<your NFS server ipaddr>:/home/USER/FILESYSTEM_DIR,nolock noinitrd
```

```
   Example: Complete bootargs for OMAP35x board using NFS and LCD
output

   setenv bootargs init=/init console=ttyS0,115200n8 ip=dhcp rw
root=/dev/nfs nfsroot=192.168.133.01:/home/user/targetfs,nolock
mem=256M noinitrd androidboot.console=ttyS0
```

Building Android Sources

Android sources (filesystem) can be built by following the instructions documented here

As Google we recommend to use Ubuntu 8.04+ , but also you can use CentOS 5.3 (32 bit).

- Install DHCP server
- Install tftp server (actual for OMAP3EVM)
- Install repo tool
- Set up your Linux development environment, make sure you have the following:

Required packages:

Git 1.5.4 or newer and the GNU Privacy Guard. JDK 5.0, update 12 or higher. Java 6 is not supported, because of incompatibilities with @Override. flex, bison, gperf, libsdl-dev, libesd0-dev, libwxgtk2.6-dev (optional), build-essential, zip, curl, minicom, tftp-server, uboot-mkimage

For Ubuntu 32-bit use such command:

```
  $ sudo apt-get install git-core gnupg sun-java5-jdk flex bison gperf
libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl
libncurses5-dev zlib1g-dev minicom tftpd uboot-mkimage
```

Ubuntu Intrepid (8.10) users may need a newer version of libreadline:

```
  $ sudo apt-get install lib32readline5-dev
```

Configure your network¶

- Configure your host Ethernet adapter

```
  sudo ifconfig ethX 10.10.10.1 netmask 255.255.255.0 up
```

- Configure DHCP server for target's network

```
    ; Example dhcpd.conf
    default-lease-time 600;
    max-lease-time 7200;
    subnet 10.10.10.0 netmask 255.255.255.0 {
      range 10.10.10.2 10.10.10.10;
```

```
    host beagleboard_rev_c3 {
      hardware ethernet 00:80:C8:xx:xx:xx;
      fixed-address 10.10.10.2;
    }
```

```
    host omap3evm_rev_d {
      hardware ethernet 00:50:c2:xx:xx:xx;
      fixed-address 10.10.10.3;
      filename "evm/uImage"; <- this string actual for boot with tftp
    }
```

- Configure TFTP-server

```
    /etc/xinet.d/tftpd:
```

```
    service tftp
    {
```

```
    protocol        = udp
    port            = 69
    socket_type     = dgram
    wait            = yes
    user            = nobody
    server          = /usr/sbin/in.tftpd
    server_args     = /tftpboot
    disable         = no
  }
```

- Clone the Sources:

```
#> mkdir rowboat-android
#> cd rowboat-android
#> repo init -u git://gitorious.org/rowboat/manifest.git -m TI-Android-FroYo-DevKit-V2.xml
#> repo sync
```

- Build the root file system for OMAP35x and AM37x

```
#> make TARGET_PRODUCT=omap3evm TARGET_BUILD_VARIANT=tests -j8

TARGET_PRODUCT should be am3517evm for AM35x.
```

- Install the SGX (Open GL drivers) libraries and package into filesystem

```
Follow the instructions provided here to install SGX (OpenGL) package into the filesystem

http://code.google.com/p/rowboat/wiki/ConfigureAndBuild#Install_the_Android_Graphics_SGX_SDK_on_Host_Machine
```

- Prepare the root filesystem

```
Follow the steps below to populate the Android filesystem.

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

The rootfs.tar.bz2 is the android filesystem, it can be put on a SD/MMC Card or used our NFS.
```

ADB Android Debugger & Downloader

Android Debug Bridge (adb) is a versatile tool lets you manage the state of the Android-powered device. For more information about what is possible with adb, see Android Debug Bridge page at http://developer. android. com/ guide/developing/tools/adb.html.The ADB tool can be used to

- Download an application from a host machine, install & run it on the target board.
- Start a remote shell in the target instance.
- Debug applications running on the device using the debugging tool DDMS (Dalvik Debug Monitor Server) which runs on top of adb connection.
- Copy files to and from the board to host machine

Downloading "ADB" & Host setup

The adb tool is a part of Android SDK package located at http://developer.android.com/sdk/index.html.For an overview of how to install and set up the Android SDK, follow download & setup instructions from http://developer.android.com/sdk/index.html.Once you install Android SDK, the directory contents look like this.

```
  add-ons/
  docs/
  platforms/
    <platform>/
       data/
       images/
       skins/
       templates/
       tools/
       android.jar
  samples/
  tools/
SDK Readme.txt
```

The adb tool is located in tools/ directory under the Android SDK installation. Export the tools directory path as shown below.

```
  $> export PATH=${PATH}:<your_sdk_dir>/tools
```

Connecting Host machine & board through adb

This release of DevKit has been tested for three different methods of connecting a given board with host machine

- adb over USB
- adb over USB Ethernet
- adb over Ethernet

The below sections describe each of these methods and provides necessary instructions for the same.

adb over USB

- Make sure that the mini-usb cable is connected between the host usb port and the target's USB OTG port
- Turn on "USB Debugging" on your board. On the board (UI screen)-
  - Go to home screen, press MENU,
  - Select Applications, select Development, then enable USB debugging.
  - Alternatively, you can navigate to Settings->Applications->Development and then enable the "USB debugging" option.
- Setup host machine to detect the board. On Ubuntu Linux host machines this is done by adding a rules file to configure device vendor ID of on-board OMAP device.
- For the EVMs and Boards covered here, the vendor ID is "18d1".
  - Log in as root and create this file: **/etc/udev/rules.d/51-android.rules**

```
    For Gusty/Hardy, edit the file to read:
    SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", MODE="0666"
```

```
    For Dapper, edit the file to read:
    SUBSYSTEM=="usb_device", SYSFS{idVendor}=="18d1", MODE="0666"
```

- Execute the following to change the user mode for the rules file.

```
$> chmod a+r /etc/udev/rules.d/51-android.rules
```

- Verify the adb connectivity between host and target board

```
$> adb devices
```

```
If device is connected, then output on screen should list the device, example:

   List of devices attached
   20100720    device
```

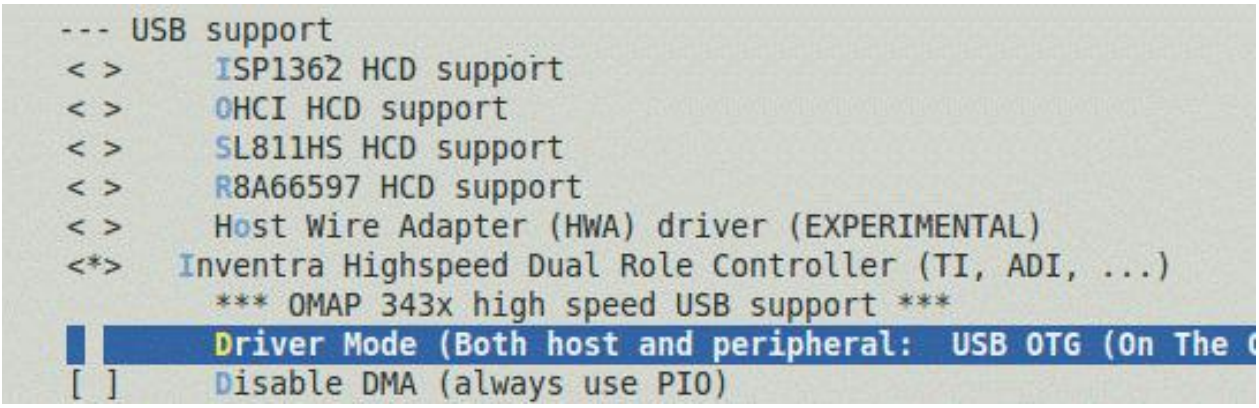
adb over USB Ethernet (Ethernet over USB)

- Make sure that the mini-usb cable is connected between the host usb port and the target's USB OTG port.
- Configure the Linux kernel to use Ethernet gadget. Enable USB support, configure the Inventra controller, and add USB gadget support.

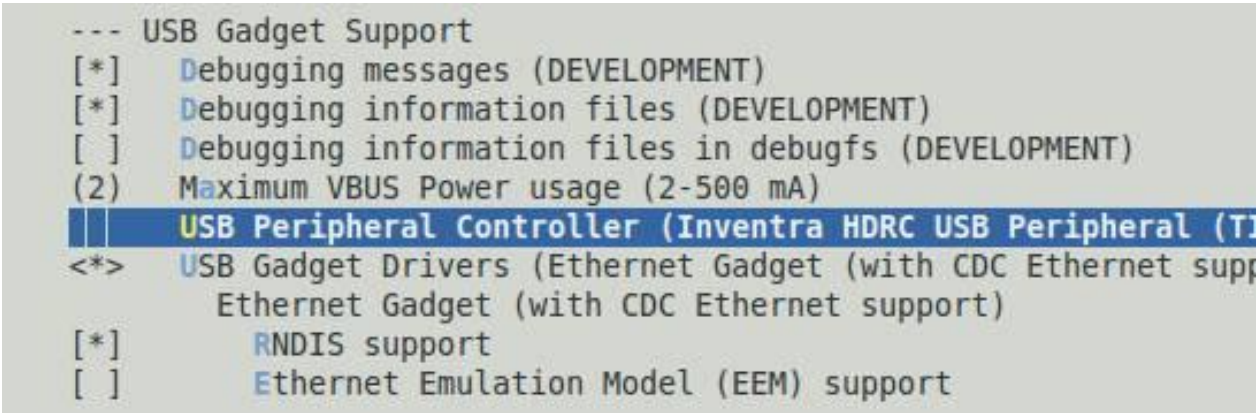
IMPORTANT NOTE: Inventra configuration must occur in two places as shown in non-highlighted lines of the screen shots below.

```
#> make ARCH=arm CROSS_COMPILE=arm-eabi- menuconfig
```

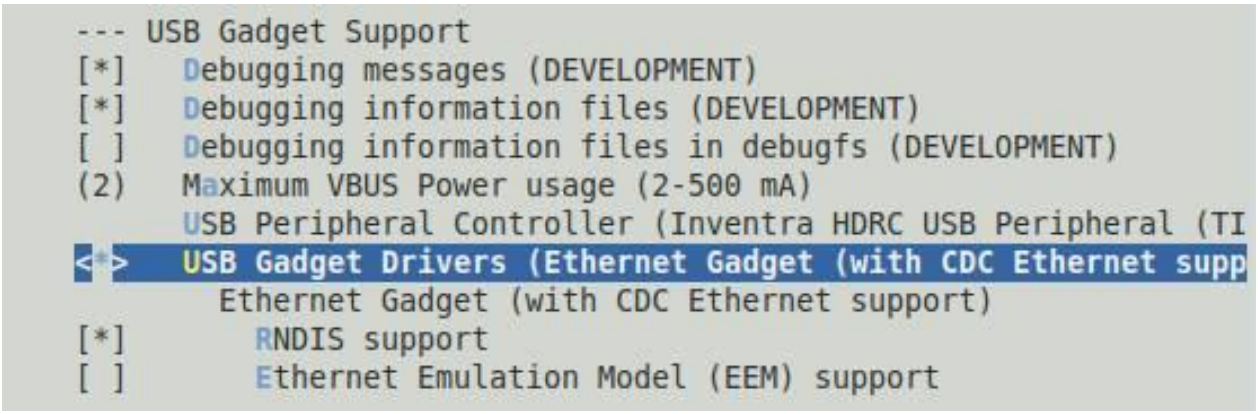
Device Drivers --- USB Support



Device Drivers --- USB Support --- USB Gadget Support



Device Drivers --- USB Support --- USB Gadget Support --- Enable Gadget Ethernet support



- Build the Kernel with the above configuration changes and use the uImage to boot the board. Refer to Kernel compiling instructions above.
- Establish network connection
  - Assign an IP address to the usb ethernet adapter.

The USB network gadget g_ether is named usb0 (instead of eth0 or other network interface names). The normal set of Ethernet configuration tools should work, such as ifconfig, netstat, and route.

For example, the following commands will assign the network address 192.168.194.2 to the target. Run this on the target:

```
$> ifconfig usb0 192.168.194.2 netmask 255.255.255.224 up
```

On Host machine, run the following commands to establish the connection to the target:

```
$> sudo ifconfig usb0 192.168.194.1 netmask 255.255.255.224 up
$> sudo route add 192.168.194.2 dev usb0
```

The target and the host machine should be connected, run ping command to test the same:

```
$ ping -c 3 192.168.194.2
PING 192.168.194.2 (192.168.194.2) 56(84) bytes of data.
64 bytes from 192.168.194.2: icmp_seq=1 ttl=64 time=6.08 ms
64 bytes from 192.168.194.2: icmp_seq=2 ttl=64 time=0.511 ms
64 bytes from 192.168.194.2: icmp_seq=3 ttl=64 time=0.485 ms
--- 192.168.194.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.485/2.361/6.089/2.636 ms
```

- Establish ADB connection

On the host machine execute following commands to establish adb connection

```
$ export ADBHOST=<target's ip address>
$ adb kill-server
$ adb start-server
```

Verify the connection by executing

```
$ adb devices
```

If connected, device name should be listed as a "emulator"

```
$ adb devices
List of devices attached
```

```
emulator-5554    device
$ adb shell
```

adb over Ethernet

- Make sure Ethernet port on board and host machine are connected to the network
- Check Ethernet configuration for the board

```
target #> netcfg
   lo       UP    127.0.0.1       255.0.0.0       0x00000049
   eth0     UP    172.24.190.59   255.255.252.0   0x00001043
```

- If Ethernet was not configured, configure Ethernet of the board using ifconfig/netcfg as shown below.

```
target #> netcfg eth0 dhcp
```

- Configure the ADB Daemon to use an ethernet connection using setprop as shown below.

```
target #> setprop service.adb.tcp.port 5555
```

- If network is configured successfully (above steps) then Restart service adbd on the target,

```
target #> stop adbd
target #> start adbd
```

- On the host machine use following commands to establish adb connection

```
$> export ADBHOST=<target's ip address>
$> adb kill-server
$> adb start-server
```

- Verify for device connectivity, by executing the following commands

```
$> adb devices If connected, you'll see the device name listed as a "emulator"
```

```
$> adb devices
```

```
If connected, find the device name listed as a "emulator"
```

```
List of devices attached
emulator-5554    device
$ adb shell
```

For more information about adb commands, see Android Debug Bridge page at http:// developer. android. com/ guide/developing/tools/adb.html

adb over USB on Windows Machine

Follow the below instructions to get ADB over USB work on a Windows PC

- Download latest Android SDK

(http://developer.android.com/sdk/index.html) and uncompress it in a local folder (i.e. c:\android_sdk).

- Optionally, you may want to add the location of the SDK's primary tools directory to your system PATH. Right-click on My Computer, and select Properties. Under the Advanced tab, hit the Environment Variables button, and in the dialog that comes up, double-click on Path (under System Variables). Add the full path to the tools\ directory to the path.
- Download Android USB Driver

(https://dl-ssl.google.com/android/repository/usb_driver_r03-windows.zip) and uncompress it in a local folder (i.e. c:\android_sdk\usb_driver)

- Edit (or create and then edit if it doesn't already exist) file in

"%USERPROFILE%\.android\adb_usb.ini":

```
echo 0x18D1 > "%USERPROFILE%\.android\adb_usb.ini"
```

- Edit android_winusb.inf to match EVM/Beagle vendor and product ids:

Under [Google.NTx86] section add:

```
;TI EVM
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_9018
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_9018&MI_01
```

Note: Be careful to add it under Google.NTx86 and not under Google.NTamd64 unless your machine is AMD 64 bits. If you skip this step you won't be able to later install the driver as windows will reject it.

- Boot the board as normal and wait until shell prompt is available (micro-B USB cable must be disconnected).
- Connect micro-B USB cable between board and Windows PC.
- If it is proceeding as planned, Windows will tell you it found a new hardware asks you to install the driver. Install driver that was downloaded as described in step 3 above:

Answer "No, not this time" to the question about running Windows Update to search for software.

  - Choose "Install the hardware that I manually select from a list (Advanced)" this is the 2nd option, then click "Next"
  - Select "Show All Devices", then click "Next"
  - You are going to see a grayed-out text box with "(Retrieving a list of all devices)", click the "Have Disk..." button
  - Browse" to your driver folder (c:\android_sdk\usb_driver). It will be looking of a .inf file so select "android_winusb.inf" and click "Open" then "OK". It's the only file there so you shouldn't go wrong.
  - Select "Android ADB Interface" then click the "Next" button.
  - A warning will appear, answer "Yes" but read the warning anyway.
  - Click the "Close" when the wizard is completed.
- Disconnect and reconnect micro-B USB cable from Board(probably reboot it as well).
- Open command prompt and restart adb server just to make sure it is in a proper state:

```
adb kill-server
adb start-server
```

- List the attached devices with "adb devices". It should show your board/device with a random number.
- Type "adb shell". You should see the "#" indicating it works.

Running Applications

The root File System provided in this DevKit releases contains only standard Android components and applications. User might be interested to download & run android applications (.apk) available in the market. The below procedure gives the steps to be followed to download any .apk file to the board and run it on the platform.

Installing (.apk files) application on Target Platform

- From the host: You can use adb tool for package installation.

```
$> adb install <package>.apk.
```

NOTE: Use -s option with the adb tool, to install the package on external storage.

On successful installation adb tool will report SUCCESS on host terminal, and the application would be listed on the android main menu.

Un-installing applications (.apk) using adb

- To un-install non-default components (that were installed later)
  - Method 1: On the host machine execute the following

```
$> adb uninstall <package>.apk
```

  - Method 2: On target:

```
Main menu -> Menu -> Settings -> Applications -> Manage applications -> Find the package
Tap on it -> Uninstall -> OK -> OK
```

  - On successful removal, the application would have been removed from the android main menu. All the short-cuts to the application also removed.
- To un-install default components, use the following commands from abd on host machine

```
$ adb shell
#rm /system/app/app.apk
```

On successful removal, the application would have been removed from the android main menu.

Setup ADB for application Debugging

ADB and Eclipse, with ADT(Android Development Tools plug-in) allow users to create and debug Android applications. Follow Developing In Eclipse, with ADT at http:/ / developer. android. com/ guide/ developing/ eclipse-adt.html

Steps to connect Eclipse to the board.

- Setup the adb connection with the board by following the instructions given above in connecting board ...

```
Verify the connectivity by executing
$ adb devices
```

- Open Eclipse IDE. Eclipse, with ADT plugin enable users to
  - Create an android project.
  - Build and Run the project on a connected board.
  - Debug the project using the Debug perspective.
  - Use DDMS (Dalvik Debug Monitor Server) to monitor the connected board.

For more detailed and complete information on the above follow Developing In Eclipse, with ADT at http://developer.android.com/guide/developing/eclipse-adt.html

- Open DDMS(Dalvik Debug Monitor Server) perspective. This DDMS perspective can be opened from the eclipse menu via:

```
Window -> Open Perspective -> Other -> DDMS;
Click on OK
```

- DDMS provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information,incoming call and SMS spoofing, location data spoofing, and more.
- For more information on DDMS and to use it, follow Using the Dalvik Debug Monitor page at http://developer.android.com/guide/developing/tools/ddms.html

Copy any files to and from the board over ADB

- Using the adb commands "pull" and "push" user can copy files to and from the board.
- Unlike the install command, which only copies an .apk file to a specific location, the pull and push commands let you copy arbitrary directories and files to any location on the board.
- To copy a file or directory (recursively) from the board, use

```
adb pull <remote> <local>
```

- To copy a file or directory (recursively) to the board, use

```
adb push <local> <remote>
```

In the commands, <local> and <remote> refer to the paths to the file or directory on your development host (local) and on the target instance (remote).

```
Here's an example:
adb push foo.txt /sdcard/foo.txt
```

Adobe Flash 10 Integration

The Android version of Flash10 [7] that runs on Froyo is now available for customer download (by registration) at, http://focus.ti.com/docs/toolsw/folders/print/adobeflash-a8.html

The below steps give the procedure to download the Adobe Flash 10 library for Android FroYo and installing the same in File system.

- Download the flashplayer installer “Flash10.1_Android_Webkit_Plugin-0.4-Linux-x86-Install.bin” from http://focus.ti.com/docs/toolsw/folders/print/adobeflash-a8.html
- Execute the installer

```
#> ./ Flash10.1_Android_Webkit_Plugin-0.4-Linux-x86-Install.bin
Will result in following instruction, press "Y"
```

```
This will install Flash10.1 Android Webkit Plugin on your computer.  Continue? [n/Y] Y
Select the source install location
Where do you want to install Flash10.1 Android Webkit Plugin?
[/home/user/flash10_androidplugin] /home/user/flash10_androidplugin
```

```
Installing Flash10.1 Android Webkit Plugin...

Installing Program

Files...

Installation complete.

After Installation the following directory structure is resulted
```

- Change to Flash installed directory on Host PC

```
#> cd flash10_androidplugin
#> ls
install_flash_player.apk  uninstall
```

- Install flash player plug in on target via adb

```
#> adb install install_flash_player.apk
```

- Do the browser configuration
  - Explained in the below section
- Test the Adobe Flash installation
  - Browse the link http://www.adobe.com/software/flash/about/
  - Should display Adobe Flash Player Successfully Installed

Compatibility Test Suite (CTS)

This section describe the procedure to run CTS on any platform.

- Pre-requisites
  - Download and extract (untar) the CTS package from http://dl.google.com/dl/android/cts/android-cts-2.2_r3-x86.zip
  - Download and extract (untar) Google Android SDK from http://dl.google.com/android/android-sdk_r07-linux_x86.tgz
- Setup an ADB connection between Host and platform as mentioned in ADB section above.
- Setup your platform to run the accessibility tests:
  - adb install -r android-cts/repository/testcases/CtsDelegatingAccessibilityService.apk
  - On the device enable Settings > Accessibility > Accessibility > Delegating Accessibility Service
- Launch the CTS.
  - Edit android-cts/tools/startcts to point SDK_ROOT to android sdk installation location.
  - Run ./tools/startcts
  - On CTS prompt check the available plans

```
cts_host > ls –plan
```

  - Start a specific Test Plan

```
cts_host > start --plan <plan name>
```

Once all the tests are executed, the results can be browsed in an browser by opening [android-cts/repository/results/session-name/testResult.xml] and use the results to adjust your design.

Configuring Android Applications

Browser Configuration

To browse web pages user should configure the Internet connection as given below.

```
#> netcfg eth0 dhcp
#> getprop net.eth0.dns1
```

This prints the dns for the ethernet port, do the following to configure the DNS entries on board.

```
#> setprop net.dns1 <your_dns_server_ip>
```

If the platform is behind proxy, then following command should be executed to set the proxy settings

```
#> setprop net.gprs.http-proxy http://proxyurl:80
```

NOTE: If network is behind a proxy, in this DevKit release, we have NOT found a method to set the proxy server. We tried using "setprop net.eth0.http-proxy hostname:port" and "setprop net.gprs.http-proxy hostname:port", but neither could get us through the proxy. Also, the option of adding an entry of (99,'http_proxy','hostname:port") to the 'system' and 'secure' tables in the /data/data/com.android.providers.settings/databases/settings.db database has also been tried, but failed.

USB Mass Storage

The Android FroYo 2.2 supports USB Mass storage functionality, however the external storage can be mounted either on SD Card or a USB mass storage device. The user is allowed to choose one of the two options.

By default the TI Android DevKit chooses SD card as the external storage device.

If a user is interested to use storage over USB (USB mass storage) then following changes have to be done in the default root filesystem

1)Create some directory on Android Filesystem via adb shell or console

```
# mkdir /partition
```

2) Insert the usb mass storage device, Assume the device node created at /dev/block/sda1 and the device has fat partition.

```
#mount –t vfat /dev/block/sda1 /partition
```

3) Now usb mass storage device is mounted at /partition. You can careate browse the filr

Limitation 1) Auto mounting usb mass storage device not supported 2) The gallery app/Android Settings doesn't recognize the mass storage mounted

Versioning

This is Release DevKit-V2. The release is available from http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_00_00/index_FDS.html

Technical Support and Product Updates

For further information or to report any problems, contact http://e2e.ti.com "Android" or http://support.ti.com.
For community support join http://groups.google.com/group/rowboat
For IRC #rowboat on irc.freenode.net

References

[1] http://focus.ti.com/docs/toolsw/folders/print/tmdsevm3530.html

[2] http://beagleboard.org

[3] http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3517.html?DCMP=dsps_arm_102109&HQS=Other+OT+sitara-prtools

[4] http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3715.html?DCMP=am37x_060710&HQS=Other+OT+am37xprtf

[5] http://developer.android.com/sdk/installing.html#troubleshooting

[6] http://android.git.kernel.org/?p=platform/prebuilt.git;a=tree;f=linux-x86/toolchain/arm-eabi-4.4.0;h=0f1763c115040381493278dde810a69e4bc37a76;hb=HEAD

[7] http://processors.wiki.ti.com/index.php/TI_Flash10_Graphics_Framework

Article Sources and Contributors

TI-Android-FroYo-DevKit-V2 UserGuide *Source*: http://processors.wiki.ti.com/index.php?oldid=114445 *Contributors*: AJ, AnantGole, Anveshg, Arunjoseph05, Kevinsc, Khasim

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

Image:Android USB ADB ENABLE.JPG *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USB_ADB_ENABLE.JPG *License*: unknown *Contributors*: Khasim

Image:Android USBGadget ADB.JPG *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USBGadget_ADB.JPG *License*: unknown *Contributors*: Khasim

Image:Android USBEthernet ADB.JPG *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USBEthernet_ADB.JPG *License*: unknown *Contributors*: Khasim

License

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.