ZedAndroid:
Google Android porting on ZedBoard

Ph.D Student Mario Barbareschi
Prof. Antonino Mazzeo
firstname.lastname at unina.it
Ing. Antonino Vespoli
ant.vespoli at studenti.unina.it

13/05/13

http://wpage.unina.it/mario.barbareschi/zedroid/index.html

Contents

This guide explains how to port the OS Android on Zedboard FPGA.

Version

Version 1.0:
How boot android on ZedBoard.
Android is equipped with video, keyboard and mouse drivers.
Version 1.1:
How to connect android to the developer machine from adb
Version 1.2:
How to connect android to Internet by ethernet.

Introduction

In this guide there are 4 parts to the Android porting on ZedBoard. It will be explained step by step how to do the porting Google Android OS. For this aim, It will use the VHDL sources and Ubuntu 12.04 LTS-based distribution provided by Xilinux. [1]

We have implemented a primordial version of the operating system with basic facilities (screen, keyboard etc etc). We want, in the following releases, add the support to audio, Ethernet etc etc.

1 Hardware requirements

- The ZedBoard: is an evaluation and development board based on the Xilinx Zynq 7000. Combining a dual Corex-A9 Processing System (PS) equipped with 85,000 Series-7 Programmable Logic (PL) cells;
- A monitor capable of displaying VESA-compliant 1024x768 @ 60Hz with an analog VGA input (i.e. PC monitor);
- An analog VGA cable for the monitor;
- A USB keyboard;
- A USB mouse;
- A USB hub recognized by Linux 3.3.0, if the keyboard and mouse are not combined in a single USB dongle;
- A SD card with 4GB or more (we suggest a Sandisk one);
- A PC with slot for SD cards and Xilinx ISE Design Suite release 14.2 with a license to use Xilinx series 7 tools.

2 Generate VHDL and boot.bin image

To generate all files to boot a linux flavour on ZedBoard you have to download the Xillinux distribution. It consists of two parts: a raw image of the SD card containing the file system needed by Linux at bootup, and a set of files to create a first boot image. To obtain the image we'll use Xilinx tools.
The distribution includes a demo of the Xillybus IP core for easy communication between the processors and logic fabric. This demo bundle includes a specific configuration of the Xillybus IP core, having a relatively low bandwidth performance, as it's intended to use for simple tests.

The Xillinux distribution is available for download at Xillybus site. [2]

Unzip the previously downloaded xillinux-eval-zedboard-XXX.zip file into a working directory.

2.1 Generating the processor netlist.

Within the boot image kit, double-click the system.xmp file in the "system" directory. This opens the Xilinx Platform Studio (XPS). Click "Generate Netlist" to the left (as shown in the image below).

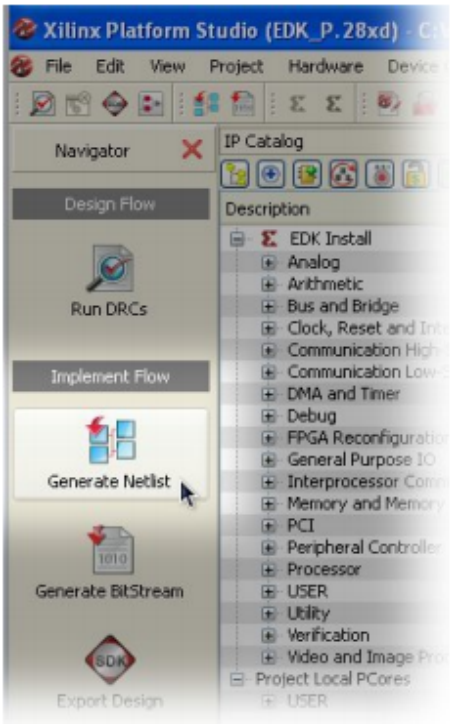

Figure 1:

The process takes up to 10 minutes. Several warnings are generated, but no errors should be tolerated (which is unlikely to happen).

The console output reports "XST completed" and "Done!" when the process complete successfully. At this point, close the XPS completely because you have not to use it again in the normal use of Xillinux.

2.2 Generating Xilinx IP cores.

Within the boot image kit, double-click the runonce.xise file in the "runonce" directory. This opens the Xilinx ISE Project Navigator. On the top left of opened windows, click on fifo 32x512, then expand the "CORE Generator" line in the process window below (clicking on the "+"), and finally, double click "Regenerate Core", as shown in the image below.

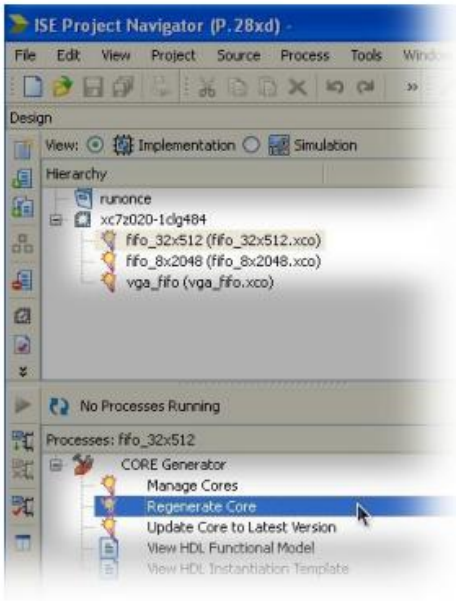

Figure 2:

The process produces a tolerable warnings, but no errors, and ends with this message: "Process Regenerate Core completed successfully". You have to repeat this for the two other IP cores: fifo 8x2048 and vga fifo. At this point, close the ISE Project Navigator completely.

2.3 Generating the bitstream file.

You can choose either VHDL or Verilog sources. So double-click the 'xilly-demo.xise' file in either the 'verilog' or 'vhdl' subdirectory. Both directories are

in the boot image kit. We choose the 'vhdl' subdirectory. The Project Navigator will launch and open the project with the correct settings. Just click "Generate Programming File".

Figure 3:

The procedure will produce several warnings but no errors. The process will end with the output "Process Generate Programming File completed successfully". The result can be found as xillydemo.bit in the 'verilog' or 'vhdl' directory (whatever was chosen) with several other files.
Close the ISE Project Navigator completely when this task is completed successfully.

2.4 Creating the boot.bin image.

The boot.bin is a binary image containing the FSBL and u-boot images produced by bootgen.
Copy the xillydemo.bit file (from the 'verilog' or 'vhdl' subdirectory, whichever was chosen) into the 'boot' directory . Next, make sure that the ISE tools' directory is in the execution path. Alternatively, makeboot.bat can be edited to call the "bootgen" utility with a full path.
The boot image is created as boot.bin in the same directory from the xillydemo.bit file and boot loader ELF files that are already in the 'boot' directory.

If a boot.bin file isn't generated, check the error message.

2.5 Loading the SD with the image of Xillinux.

The image was downloaded as a file named xillinux.img.gz (or similar), and is a gzip-compressed image of the SD card (the boot.bin file, which was generated, is missing in the image though). This image should be uncompressed and then written to the SD card's first sector and on. There are several ways and tools to accomplish this. The image contains a partition table, a partly populated FAT file system for placing the boot image, and the Linux root file system of ext4 type. The linux file system is not necessary to boot Android, but is useful to test the functionality of the linux kernel on the board. The second partition is ignored by almost all Windows computers, so the SD card will appear to be very small in capacity (16 MB or so). Writing a full disk image is not an operation intended for plain computer users, and therefore requires special software on Windows computers like USB Image Tool and extra care on Linux.

Copying the boot.bin file into the SD card.

3 Run linux on Zedboard

3.1 Jumper settings.

To boot by SD on the board is necessary to set the jumpers appropriately. The correct setting is depicted in the following image. Typically, the following jumper changes are necessary (be careful with a different board may be a different setting):

Figure 4:

- Install a jumper for JP2 to supply 5V to USB device.
- JP10 and JP9 moved from GND to 3V3 position, the three others in that row are left connected to GND.
- Install the jumper for JP6.

  The required setting differs from the one detailed in the Zedboard Hardware User Guide in that JP2 is jumpered, so that the USB devices attached to the board (USB keyboard and mouse) receive their 5V power supply.

3.2 Attaching peripherals.

The following general-purpose hardware should be attached at the board:

- A computer monitor to the analog VGA connector.
- A mouse and keyboard to the USB OTG connector, through the USB female cable that came with the board. The system will boot in the absence of these, and there is no problem connecting and dis-connecting the keyboard and mouse as the system runs. Note that JP2 on the board must be installed for this USB port to function.

- The Ethernet port is optional for common network tasks. The Linux machine configures the network automatically if the attached network has a DHCP server.
- The UART USB port is optionally connected to a PC, but is redundant in most cases.

3.3 Powering up the board.

- Plug the SD card into the Zedboard, and power it on. The following sequence is expected:
- Only the green "POWER" LED goes on. Nothing else happens.
- About 8 seconds later, the blue "DONE" LED goes on, and a red LED starts blinking. Other LEDs go on as well. The VGA monitor displays a screensaver pattern with Xillybus' logo moving on white background. All these indicate that the logic fabric (PL, "FPGA") has been loaded properly with the bitstream (xilly-demo.bit).
- After some additional 14 seconds (22 seconds from power-up), Linux bootup text appears rapidly on the VGA monitor. This looks exactly like a PC booting, with white text on black background.
- A login prompt should appear no later than 10 seconds after the bootup text was first seen on the VGA monitor. The system auto-logins as root, presenting a greeting message and a shell prompt.
- Type "startx" at command prompt to launch a Gnome graphical desktop. The desktop takes some 15-30 seconds to initialize.
- Now is possible to use the graphical desktop to check effectively that the linux kernel works on the ZedBoard.

3.4 Resize the file system.

The root file system image is kept small so that writing it to the device is as fast as possible. On the other hand, there is no reason not to use the SD card's full capacity. Under linux os is possible to use for this purpose gparted.

- Open the shell and type:
  sudo apt-get install gparted
- Run GParted type: sudo gparted from the shell
- Select on the top right the SD device
- Now is possible choose the favorite size for the file system partition

4 How to set up the local work environment

For the smooth running of the project is necessary to prepare the development environment providing all the required software components. For the purposes of this guide have been used Ubuntu 12.04, but is possible use different versions.

4.1 ARM tool-chain.

It is fundamental to the implementation of the project have a tool-chain in order. For the purposes of this guide have been used the Sourcery G++ Lite tool-chain, but is possible use others like Linaro.

4.2 JDK.

JDK is required to build Android. To install it download the jdk binary from oracles website and follow the steps below:

- chmod u+x jdk-xyz-linux-i586.bin
- ./jdk-xyz-linux-i586.bin
- sudo mkdir -p /usr/lib/jvm
- sudo mv jdk.xyz /usr/lib/jvm/
- sudo update-alternatives –install "/usr/bin/java" "java" "/usr/lib/jvm/jdk.xyz/bin/java" 1
- sudo update-alternatives –install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk.xyz/bin/javac" 1
- sudo update-alternatives –install "/usr/lib/mozilla/plugins/libjavaplugin.so" "mozilla-javaplugin.so" "/usr/lib/jvm/jdkxyz/jre/lib/i386/libnpjp2.so" 1
- sudo update-alternatives –config java
- sudo update-alternatives –config javac
- sudo update-alternatives –config mozilla-javaplugin.so
- sudo update-alternatives –config javaws

4.3 Installing required packages.

These are the packages required for Ubuntu 12.04, for different versions maybe there are difference

sudo apt-get install git 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 tofrodos \ python-markdown libxml2-utils xsltproc zlib1g-dev:i386

Create the following link

```
sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

4.4 Android Source.

- Installing Repo (Repo is a tool that makes it easier to work with Git in the context of Android).
- Make sure you have a bin/ directory in your home directory:
  `mkdir ~/bin`
- Include it in your path:
  `PATH=~/bin:$PATH`
- Download the Repo script:
  `curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo`
- Ensure it is executable:
  `chmod a+x ~/bin/repo`
- Create an empty directory to hold your working files:
  `mkdir WORKING_DIRECTORY`
- `cd WORKING_DIRECTORY`
- Run repo init to bring down the latest version of Repo with all its most recent bug fixes. You must specify a URL for the manifest, which specifies where the various repositories included in the Android source will be placed within your working directory:

  ```
  repo init -u https://android.googlesource.com/platform/manifest
  ```

  To check out a branch other than "master", specify it with -b (ex):

  ```
  repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1
  ```

- To pull down files to your working directory from the repositories as specified in the default manifest, run:

  ```
  repo sync
  ```

  The Android source files will be located in your working directory under their project names. The initial sync operation will take an hour or more to complete.

4.5 Linux Source.

It is possible to obtain the linux sources from:

```
git clone https://github.com/Digilent/linux-digilent.git
```

The used kernel is based upon the commit tagged v3.3.0-digilent-12.07-zed-beta

Downloading the Android kernel. The kernel that you can get from Google's Android Open Source Project repository. It contains Google's changes to the Mainline kernel, to support Android. You can get it from :

```
git clone https://android.googlesource.com/kernel/common -b android-3.3
```

5 Prepare the kernel for Android

5.1 Installing xillinux patch.

In the directory /usr/src/xillinux/kernel-patches are provided as git patches, the patch to add at the source to have the xillinux developed driver for linux kernel. Go on the top level of the linux source after have unzipped it and run:

```
sudo git apply –ignore-whitespace –ignore-space-change –check name-patch
```

It is used –ignore-whitespace –ignore-space-change to ignore few not relevant difference in the code
It is used –check for the check of potentials merge-conflict. After it makes sure that all the merge-conflicts are resolved run:

```
sudo git apply –ignore-whitespace –ignore-space-change –apply name-patch
```

Make the same for all the patches.

5.2 Obtain the Android patches to apply at Linux kernel.

It is not necessary obtain the Android patches to apply at the reference-kernel (the kernel that you have for your board) to obtain the merged-kernel(the kernel that is the result of merging Android patches to the Reference kernel. A Merged kernel should support Android.)
The expected size of the patch is about 2MB. To extract the patches, find the point in time (using "git log") where the Mainline kernel was imported to the Android kernel source tree.
Go to the top level of the Android kernel source tree and run:

```
git log –pretty=oneline –format="%Cgreen%h %Creset%s" \ –grep="Linux 3.3." -n 20
```

which will generate 20 one-line log entries that have the expression "Linux 3.3." in the subject or in the commit message. The output should look like:

```
192cfd5 Linux 3.3-rc6
d5a74af Merge tag 'iommu-fixes-v3.3-rc5' of git://git.kernel.org/pub/scm/linux/kernel/git

f2273ec Merge branch 'lpc32xx/fixes' of git://git.antcom.de/linux-2.6 into fixes

6b21d18 Linux 3.3-rc5
b01543d Linux 3.3-rc4 ffafe77 Merge branch 'v3.3-samsung-fixes-3' of git://
git.kernel.org/pub/scm/linux/kernel/
d65b4e9 Linux 3.3-rc3
62aa2b5 Linux 3.3-rc2
f94f72e Merge tag 'nfs-for-3.3-3' of git://git.linux-nfs.org/projects/trondmy/linux-
nfs
6c02b7b Merge commit 'v3.3-rc1' into stable/for-linus-fixes-3.3 dcd6c92 Linux
3.3-rc1
a12587b Merge tag 'nfs-for-3.3-2' of git://git.linux-nfs.org/projects/trondmy/linux-
nfs
93b8a58 xfs: remove the deprecated nodelaylog option
```

Since you need the patches to be on top of the 3.3.0 Mainline kernel release, run the following command:

```
git diff 192cfd5 HEAD > 3.3-rc6-to-Android.patch
```

This gives you a patch file (named 3.3-rc6-to-Android.patch) containing the changes to be merged. Check for merging conflicts, before attempting to perform the actual merge, by running:

```
git apply –ignore-whitespace –ignore-space-change –check [path/to/3.3-rc6-
to-Android.patch]
```

which produces a list of merging conflicts (if any). Make sure that you resolve these conflicts and that the patch applies cleanly to your tree before you continue. After solved all the merge-conflicts type:

```
git apply –ignore-whitespace –ignore-space-change –apply [path/to/3.3-rc6-
to-Android.patch]
```

5.3 Kernel configuration .

To have a correct configuration of the kernel is necessary to work on the .config file present in the main directory of the linux kernel source.

- Add the following rows
  CONFIG_PANIC_TIMEOUT=0
  #CONFIG_FIQ_DEBUGGER is not set
  #CONFIG_ARM_FLUSH_CONSOLE_ON_RESTART is not set
  CONFIG_KEXEC=y
  CONFIG_ATAGS_PROC=y
  CONFIG_HAS_WAKELOCK=y
  CONFIG_WAKELOCK=y
  #CONFIG_SUSPEND_TIME is not set
  CONFIG_ANDROID_PARANOID_NETWORK=y
  #CONFIG_NET_ACTIVITY_STATS is not set
  #CONFIG_NETFILTER_XT_MATCH_QUOTA2 is not set
  #CONFIG_IP_NF_TARGET_REJECT_SKERR is not set
  #CONFIG_IP6_NF_TARGET_REJECT_SKERR is not set
  #CONFIG_CFG80211_ALLOW_RECONNECT is not set
  CONFIG_RFKILL_PM=y
  CONFIG_MTD_NAND_IDS=y
  CONFIG_SENSORS_AK8975=m
  CONFIG_UID_STAT is not set
  CONFIG_MAX8997_MUIC=m
  # CONFIG_WL127X_RFKILL is not set
  # CONFIG_PPPOLAC is not set
  # CONFIG_PPPOPNS is not set
  #CONFIG_WIFI_CONTROL_FUNC is not set
  # CONFIG_BCMDHD is not set
  #CONFIG_INPUT_KEYRESET is not set
  #CONFIG_TOUCHSCREEN_SYNAPTICS_I2C_RMI is not set
  #CONFIG_INPUT_KEYCHORD is not set
  #CONFIG_INPUT_GPIO is not set
  CONFIG_DEVMEM=y
  #CONFIG_DCC_TTY is not set
  #CONFIG_ION is not set
  CONFIG_HID_BATTERY_STRENGTH=y
  #CONFIG_USB_OTG_WAKELOCK is not set
  # CONFIG_MMC_EMBEDDED_SDIO is not set
  # CONFIG_MMC_PARANOID_SD_INIT is not set
  #CONFIG_MMC_BLOCK_DEFERRED_RESUME is not set
  CONFIG_SWITCH=y
  # CONFIG_SWITCH_GPIO is not set
  CONFIG_ANDROID_BINDER_IPC=y
  CONFIG_ASHMEM=y
  CONFIG_ANDROID_LOGGER=y

CONFIG_ANDROID_PERSISTENT_RAM=y
CONFIG_ANDROID_RAM_CONSOLE=y
 # CONFIG_PERSISTENT_TRACER is not set
CONFIG_ANDROID_TIMED_OUTPUT=y
 # CONFIG_ANDROID_TIMED_GPIO is not set
CONFIG_ANDROID_LOW_MEMORY_KILLER=y
 # CONFIG_ANDROID_SWITCH is not set
 # CONFIG_ANDROID_INTF_ALARM_DEV is not set
 # CONFIG_YAFFS_FS is not set
CONFIG_STACKTRACE=y
 # CONFIG_CPU_NOTIFIER_ERROR_INJECT is not set
 # CONFIG_DEBUG_RODATA is not set
CONFIG_REED_SOLOMON_ENC8=y
CONFIG_REED_SOLOMON_DEC8=y

- remove the following rows

  CONFIG_MTD_NAND_IDS=y
  CONFIG_MAX8997_MUIC=m
  CONFIG_SENSORS_AK8975=m
  CONFIG_PMODCLS=m
  CONFIG_PMODCLP=m
  CONFIG_PMODDA1=m
  CONFIG_PMODAD1=m

- modify the following rows

  CONFIG_INET_TUNNEL=m -> CONFIG_INET_TUNNEL=y
  CONFIG_IPV6=m -> CONFIG_IPV6=y
  CONFIG_POWER_SUPPLY=m -> CONFIG_POWER_SUPPLY=y
  CONFIG_PDA_POWER=m -> CONFIG_PDA_POWER=y
  CONFIG_IPV6_SIT=m -> CONFIG_IPV6_SIT=y
  # CONFIG_ANDROID is not set -> CONFIG_ANDROID=y
  CONFIG_REED_SOLOMON=m ->CONFIG_REED_SOLOMON=y

Note that the names above are those which can be found in the 3.3.0 Android kernel configuration - be aware that these tend to change from version to version! We have also found that in multiprocessor systems, CPU hotplugging support is required. You should enable this if your ARM based SoC has more than one core in order to boot a kernel with multiprocessor support.

CONFIG_HOTPLUG_CPU is not set -> CONFIG_HOTPLUG_CPU=y

5.4 Build the Merged kernel.

Build the Merged kernel with:

```
make ARCH=arm CROSS_COMPILE=[path-to-arm-gcc] uImage
```

which will produce a kernel Image, uImage and zImage in arch/arm/boot directory in the Merged kernel source tree.

5.5 Know Issues.

In the following part are shown some possible problems/bugs

- Error message:

  ```
  In file included from drivers/video/xylon/xylonfb/xylonfb-main.c:45:0:
  drivers/video/xylon/xylonfb/xylonfb-main.c:
  In function 'xylonfb_set_timings':
  drivers/video/xylon/xylonfb/xylonfb-pixclk.h:19:12:
  error: inlining failed in call to always_inline 'pixclk_change':
  function body not available drivers/video/xylon/xylonfb/xylonfb-main.c:1368:21:

  error: called from here
  ```

- Solution:
  comment the following row in the file /drivers/video/xylon/xylonfb/xylonfb-pixclk.c

  ```
  /*inline int pixclk_change(struct fb_info *fbi)
  {
  #if HW_PIXEL_CLOCK_CHANGE_SUPPORTED == 0
  return 0;
  #elif HW_PIXEL_CLOCK_CHANGE_SUPPORTED == 1
  return 1;
  #endif
  }
  */
  ```

  modify the following row in the file /drivers/video/xylon/xylonfb/xylonfb-pixclk.h

  ```
  inline int pixclk_change(struct fb_info *fbi);
  ```

  with

  ```
  inline int pixclk_change(struct fb_info *fbi)
  {
  #if HW_PIXEL_CLOCK_CHANGE_SUPPORTED == 0
  ```

```
return 0;
#elif HW_PIXEL_CLOCK_CHANGE_SUPPORTED == 1
return 1;
#endif
};
```

- error message:

  .tmp_vmlinux1 arch/arm/kernel/built-in.o: In function '__cpu_disable':
  :(.text+0x5658):
  undefined reference to 'platform_cpu_disable' arch/arm/kernel/built-in.o:

  In function '__cpu_die': :(.text+0x576c): undefined reference to 'platform_cpu_kill' arch/arm/kernel/built-in.o:
  In function 'handle_IPI': :(.text+0x5ac8): undefined reference to 'platform_cpu_kill' arch/arm/kernel/built-in.o:
  In function 'cpu_die': :(.ref.text+0x40): undefined reference to 'platform_cpu_die'
  make: *** [.tmp_vmlinux1] Errore 1

- solution: This error occurs when is checked CONFIG_HOTPLUG_CPU
  in the .config file, to solved this problem either unchecked CONFIG_HOTPLUG_CPU
  or add the following code in the file /arch/arm/kernel/smp.c
  under the row "#ifdef CONFIG_HOTPLUG_CPU" add

```
#include <asm/smp.h>
static inline void cpu_enter_lowpower(void)
{
unsigned int v;
flush_cache_all();
asm volatile(
" mcr p15, 0, %1, c7, c5, 0|n" " dsb|n" /* * Turn off coherency */ "
mrc p15, 0, %0, c1, c0, 1|n" " bic %0, %0, #0x40|n"
" mcr p15, 0, %0, c1, c0, 1|n" "
mrc p15, 0, %0, c1, c0, 0|n" " bic %0, %0, %2|n"
" mcr p15, 0, %0, c1, c0, 0|n" : "=&r" (v) : "r" (0), "Ir" (CR_C) :
"cc");
}

static inline void cpu_leave_lowpower(void)
{
unsigned int v;
asm volatile( " mrc p15, 0, %0, c1, c0, 0|n" " orr %0, %0, %1|n"
" mcr p15, 0, %0, c1, c0, 0|n" " mrc p15, 0, %0, c1, c0, 1|n"
" orr %0, %0, #0x40|n" " mcr p15, 0, %0, c1, c0, 1|n" : "=&r" (v) :
```

```
"Ir" (CR_C) : "cc");
}
static inline void platform_do_lowpower(unsigned int cpu, int *spurious)
{
/*
* there is no power-control hardware on this platform, so all
* we can do is put the core into WFI; this is safe as the calling
* code will have already disabled interrupts
*/
for (;;)
{
dsb();
wfi();
/*
* Getting here, means that we have come out of WFI without
* having been woken up - this shouldn't happen
*
* Just note it happening - when we're woken, we can report
* its occurrence.
*/
(*spurious)++;
}
}
int platform_cpu_kill(unsigned int cpu)
{ return 1; }
/*
* platform-specific code to shutdown a CPU
*
* Called with IRQs disabled
*/
void platform_cpu_die(unsigned int cpu)
{
int spurious = 0;
/*
* we're ready for shutdown now, so do it
*/
cpu_enter_lowpower();
platform_do_lowpower(cpu, &spurious);
/*
* bring this CPU back into the world of cache
* coherency, and then restore interrupts
*/
cpu_leave_lowpower();
if (spurious) pr_warn("CPU%u: %u spurious wakeup calls|n", cpu, spu-
rious);
}
```

```
int platform_cpu_disable(unsigned int cpu)
{
/*
* we don't allow CPU 0 to be shutdown (it is still too special * e.g. clock
tick interrupts)
*/
return cpu == 0 ? -EPERM : 0;
}
```

6 Android filesystem

6.1 Add a device in the tree.

Each build target defines the configuration of the ARM based SoC/board and selects which sources should be built for Android. The build target directory that the Android build system uses depends on the Android version. In Froyo and Gingerbread the location is:
[android_root]/device whereas in Eclair it is:
[android_root]/vendor

In this guide, we assume that you are using Froyo.
Adding a build target.

Below is an example of adding a couple of ARM generic Android build targets, along with a mock target.
An example is downloadable here. [3]

- Create a directory in
  [android_root]/device,
  with the desired name.
  E.g.:

  mkdir [android_root]/device/arm

  Enter the new directory and create a products folder:

  mkdir [android_root]/device/arm/products

  In the products folder you need an AndroidProducts.mk file that lists all of the products that you have under the arm folder.
  This should be as follows:

  PRODUCT_MAKEFILES := \
  $(LOCAL_DIR)/armboards_v7a.mk \

$(LOCAL_DIR)/another_product.mk \

As this implies, you also need to have one "Product_Makefile" per product in the
[android_root]/device/arm/products/ folder.
Therefore, for our example you need:

armboards_v7a.mk
another_product.mk

Each of these files must contain the following, adapted to match the product and device naming :

$(call inherit-product, $(SRC_TARGET_DIR)/product/generic.mk)
#
#
Overrides
PRODUCT_NAME := [product_name]
PRODUCT_DEVICE := [board_name]
where PRODUCT_NAME is the build target name used by the Android build system, and PRODUCT_DEVICE defines the name of the directory that contains the files which describe the device.

For our example the first product is :
$(call inherit-product, $(SRC_TARGET_DIR)/product/generic.mk)
#
#
Overrides PRODUCT_NAME := armboard_v7a
PRODUCT_DEVICE := armboard_v7a
Note that the PRODUCT_NAME does not have to be the same as the PRODUCT_DEVICE.

Following the example above, create the required directories:

mkdir [android_root]/device/arm/armboard_v7a/
mkdir [android_root]/device/arm/another_product/

and populate them accordingly.

Each one of these directories should contain at least the "Android.mk" and "BoardConfig.mk" files. The first one is the makefile for the product and the second, as the name implies, is the config file for the product.

make file for new hardware from
#
LOCAL_PATH := $(call my-dir)

```
#
#
this is here to use the pre-built kernel

ifeq ($(TARGET_PREBUILT_KERNEL),)
TARGET_PREBUILT_KERNEL := $(LOCAL_PATH)/kernel
endif
#
file := $(INSTALLED_KERNEL_TARGET)
ALL_PREBUILT += $(file)
$(file): $(TARGET_PREBUILT_KERNEL) | $(ACP) $(transform-prebuilt-
to-target)
#
# no boot loader, so we don't need any of that stuff..
#
LOCAL_PATH := vendor/[company_name]/[board_name]
#
include $(CLEAR_VARS)
#
# include more board specific stuff here? Such as Audio parameters.
#
```

which, when adapted to our example, becomes:

```
#make file for ARMv7-A based SoC
# LOCAL_PATH := $(call my-dir)
#
# this is here to use the pre-built kernel
ifeq ($(TARGET_PREBUILT_KERNEL),)
TARGET_PREBUILT_KERNEL := $(LOCAL_PATH)/kernel
endif
#
file := $(INSTALLED_KERNEL_TARGET)
ALL_PREBUILT += $(file)
$(file): $(TARGET_PREBUILT_KERNEL) | $(ACP)
$(transform-prebuilt-to-target)
#
# no boot loader, so we don't need any of that stuff..
#
LOCAL_PATH := device/arm/armboard_v7a
#
include $(CLEAR_VARS)
#
# include more board specific stuff here? Such as Audio parameters.
#
PRODUCT_COPY_FILES += \
```

$(LOCAL_PATH)/armboard_v7a.kl:system/usr/keylayout/armboard_v7a.kl

Note the added line in the PRODUCT_COPY_FILES definition, that contains two locations separated by a colon. The first is the location of the keyboard file in the local Android source tree, and the second is where it will be installed on the target. This is added because we wanted to have our own, tweaked, keyboard layout.
The BoardConfig.mk file for armboard_v7a product is:

These definitions override the defaults in config/config.make for armboard_v7a
#
TARGET_NO_BOOTLOADER := false
TARGET_HARDWARE_3D := false
#
TARGET_CPU_ABI := armeabi-v7a
TARGET_CPU_ABI2 := armeabi
TARGET_NO_KERNEL := true
TARGET_ARCH_VARIANT := armv7-a-neon

BOARD_USES_GENERIC_AUDIO := true
USE_CAMERA_STUB := true

where you define that you are using the armeabi-v7a and that the target architecture variant is ARMv7-A with NEON.
*Note the support for 2 ABIs that was introduced in Froyo.
For any further system properties configuration, add a system.prop file containing the things you need, at the same location as the above files (device/arm/[DEVICE_NAME]).
The template for this is :

system.prop for
This overrides settings in the products/generic/system.prop file
#
rild.libpath=/system/lib/libreference-ril.so
rild.libargs=-d /dev/ttyS0

You should now be ready to build the Android file system for your SoC, with the device folder containing all the necessary configuration and build files. For our example, the device folder should look like this:

```
arm/
+- armboard_v7a
|       +- Android.mk
|       +- armboard_v7a.kl
```

```
|        +- BoardConfig.mk
|        \- system.prop
+- another_product
|        +- Android.mk
|        +- another_product.kl
|        +- BoardConfig.mk
|        \- system.prop
\- products
+- AndroidProducts.mk
+- armboard_v7a.mk
\- another_product.mk
```

6.2 Obtain the Android patches for Froyo version.

It is now required to apply some patch to the Android kernel source tree:

- Type in the shell the following comand:

  sudo git clone git://linux-arm.org/armdroid.git

- Go in the armdroid/fs/src/Froya directory
- Apply the four patches present in this directory

6.3 Build Android for your SoC.

Build the Android filesystem for one of your added targets, by going to the top directory of the Android source tree, and entering:

make PRODUCT-armboard_v7a-eng

Note: You do not need to build Android as the root user.

After some time, and if everything was set up correctly, you will get the Android root filesystem and a compressed version of it at:

[android_root]/out/target/product/armboard_v7a

6.4 Prepare the SD to boot Android

- Create a new partition on the SD of dimension more than 1 GB, necessary to put the Android file system.

- Copy [android_root]/out/target/product/armboard_v7a/root directory content in the new partition to create the [android_root_filesystem]

- Copy [android_root]/out/target/product/armboard_v7a/system directory content in [android_root_filesystem]/system

- For this filesystem to work, you also need to make changes in the [android_root_filesystem]/init.rc file. For Froyo you need to do the following changes:
  Locate and comment out the following lines:

  mount rootfs rootfs /ro remount
  mount yaffs mtd@system /system
  mount yaffs2 mtd@system /system ro remout
  mount yaffs2 mtd@cache /cache nosuid nodev

  To get root privileges in the shell, change:
  ...
  service console /system/bin/sh
  console
  disabled
  user shell
  group log
  ...
  to:
  ...
  service console /system/bin/sh
  console
  disabled
  user root
  group log
  ...
  To get DNS working, add the indicated line
  # basic network init
  ifup lo
  hostname localhost
  domainname localdomain
  —> setprop net.dns1 [DNS]
  and replace [DNS] with your local DNS server.

6.5 Set the appropriate boot arguments.

- Go in /arch/arm/boot/dts directory

- Edit the digilent-zed.dts file
- Set the boot argoment as:

  bootargs = "consoleblank=0 root=/dev/mmcblk0p3 fsroot=jffs2, nolock, wsize=1024 rw init=init noinitrd mem=1024M user_ debug=31 earlyprintk; linux,stdout-path = "/amba@0/uart@E0001000";

- On the top level of Linux Kernel source tree type in the shell the following command:

  sudo make ARCH=arm CROSS_ COMPILE= [path-to-arm-gcc] digilent-zed.dtb

- It is now generated the digilent-zed.dtb file, change it name in device-tree.dtb and put it in the SD card, in the partition (16 M dimentioned) where they are located boot.bin and zImage files.
- It is now possible boot your ZedAndroid.

6.6 Connect Android at Internet by ethernet

- Connect Lan cable to the board
- Type ALT + left arrow to obtain the command line.
- Type netcfg to obtain information about the connections configuration
- Type netcfg eth0 up
- Type netcfg eth0 dhcp, now the board should be connects to the network Type netcfg eth0 dhcp, now the board should be connects to the network
- If you run the browser it will appear a error message that says that Internet doesn't work, but is not true. You now can use internet.

6.7 ADB

For the developer of Android applications is really important to connect the android platform to the development machine. For this purpose is possibile to use ADB. ADB by OTG usb is not works yet, but it is possible use ADB by ethernet.
Below is explained how to carry out it.

- Turn on "USB Debugging" on your board.
- Go to home screen, press MENU, Select Applications, select Development, then enable USB debugging.
- Type ALT + left arrow to obtain the command line.

- Connect Android at Internet by ethernet
- Type netcfg and take note of ip address
- Type setprop service.adb.tcp.port 5555
- Type stop adbd
- Type start adbd

From the host machine carry out the following steps:

- The adb tool is a part of Android SDK package. Once you install Android SDK export the platform-tools and tools directory path as shown below.
  export PATH=<android_sdk_path>/platform-tools/:<android_sdk_path>/tools/:$PATH
- export ADBHOST=<target's ip address>
- adb kill-server
- adb start-server
- adb devices

Now it should appear something like this:
List of devices attached
emulator-5554 device

6.8 Known issues.

- Issue: Double screen on your lcd display, probably is configured 32 BPP for your lcd panel.
- Solution: in the WORKING_DIRECTORY/hardware/libhardware/modules/gralloc/framebuffer.cpp file modify the following row
  const_cast<int&>(dev->device.format) = HAL_PIXEL_FORMAT_RGB_565;

  with

  const_cast<int&>(dev->device.format) = HAL_PIXEL_FORMAT_RGBX_8888

6.9 Additions.

- Mouse. It is also possible add a patch to use the mouse on Android. This patch is available to download from here. [4]

7 Bibliography and reference

The following guide is based on the following sources. Thank you to the respective authors.

- http://blogs.arm.com/software-enablement/498-from-zero-to-boot-porting-android-to-your-arm-platform/
- http://source.android.com/source/initializing.html
- http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf
- http://linux-arm.org/LinuxKernel/LinuxAndroidPlatform
- http://processors.wiki.ti.com/index.php/Android_ADB_Setup
- http://stackoverflow.com

References

[1] http://xillybus.com/

[2] http://xillybus.com/zynq-download

[3] http://goo.gl/bxE2N

[4] https://patch-hosting-for-android-x86-support.googlecode.com/files/0001-1.-added-mouse-cursor.patch