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

October 29, 2013

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

Contents

Version

Version 1.0:
    How boot android on ZedBoard.
    Android is equipped with video and keyboard 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.
Version 1.3
    How to obtain android without the use of the xillinux platform

Introduction

This guide allow to make the porting of OS Android on the ZedBoard. The result is a properly working platform with high potential, there are a lot of project that could find fulfillment through this platform. For example is possible to develop hardware accelerators managed by App Android, it is also possible to replace software modules heavy computationally with the mutual in hardware version.

We 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 etc etc. We used as permanent memory support a SD Flash, it is used to store the boot files and the Android file system.

This guide explains step by step how to do the porting of the Google Android OS. For this aim, it will use the Xilinx tools, VHDL sources and Ubuntu 12.04 LTS-based.

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.

2 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 for the kernel compiling part, but is possible use different versions. It is required Xilinx ISE Design Suite release 14.2 (or newer) also, with a license to use Xilinx series 7 tools.

2.1 ARM tool-chain.

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

2.2 Installing required packages.

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

```
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 uboot-mkimage gparted screen
```

Create the following link

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

2.3 JDK.

JDK is required to build Android. To install it download the jdk binary from oracles website[?] and follows 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

2.4 Linux hardware design.

It is needed to use the Xilinx's tools XPS(Xilinx platform studio) and SDK (Software Development Kit) to have the right configuration of the FPGA and PS, in fact these allows to obtain the boot files that configure the board.

The Linux hardware design is available on the digilent web site[2].

Once you have downloaded the .rar file, you should unzip it and put in a folder with path without spaces.

2.5 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
```

2.6 Android Source.

- You have to use Repo, it 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 http://commondatastorage.googleapis.com/git-repo-downloads/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.

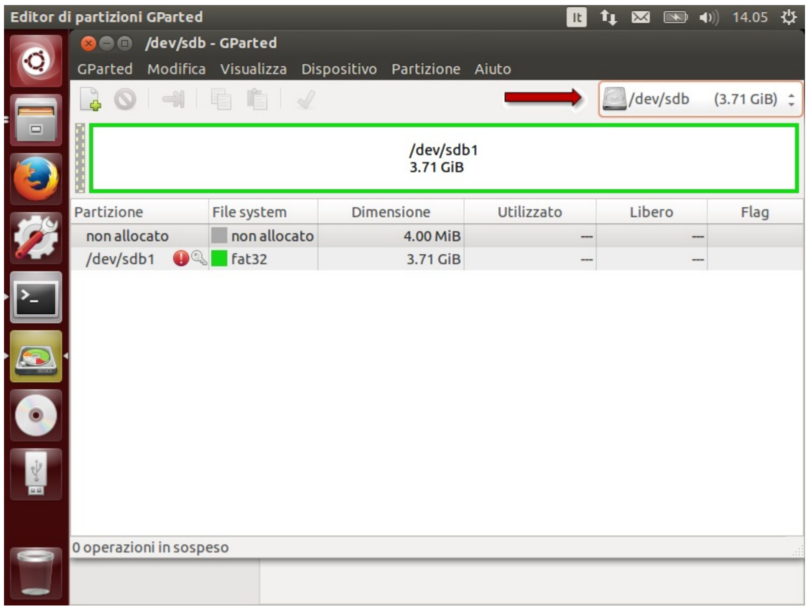
2.7 Prepare the SD memory

We needed a SD memory as permanent memory support. It has to split in two or three parts, two are mandatory, another is facultative. A partition contains the boot files, the second contains the Android file system, the facultative partition contains the linux file system. You could put the Linux file system in a partition to boot Linux OS, it allows you to make some useful tests, but it is not necessary to Android boot.

We use for our purpose GParted application to format and create the partitions.

- Open GParted.

- Select the peripheral in the showed dialog windows in the following photo.



Partizione	File system	Dimensione	Utilizzato	Libero	Flag
non allocato	▨ non allocato	4.00 MiB	—	—	
/dev/sdb1	🟩 fat32	3.71 GiB	—	—	

0 operazioni in sospeso

- In each partition, with mouse right button open the menu and select "unmount".
- In each partition, with mouse right button open the menu and select "delete".
- Create a partition with "FAT 16" filesystem and 16 MB size. This is for the boot files.
- Create a partition with "EXT4" filesystem for Linux filesystem, you can choose your desiderata size.
- Create a partition with "EXT4" filesystem for Android filesystem, you can choose your desiderata size (We recommend a size of at least 1 GB).

3 Generate boot.bin file

The BOOT.bin file allows to configure the two parts of the Zynq, programmable logic(PL) and processing system(PS). This file include the first stage boot loader(FSBL), the second stage boot loader(SSBL), and bitstream file. The bitstream file contains all the information to the programmable logic configuration. The FSBL loads the bitstream, configure the PS and after load the SSBL in memory. The SSBL takes a compressed version of the kernel and loads it in the main memory, after the control is leaved to the operative system. The needed sources to obtain this file are in the linux hardware design folder, we call it Hardware Design Source.

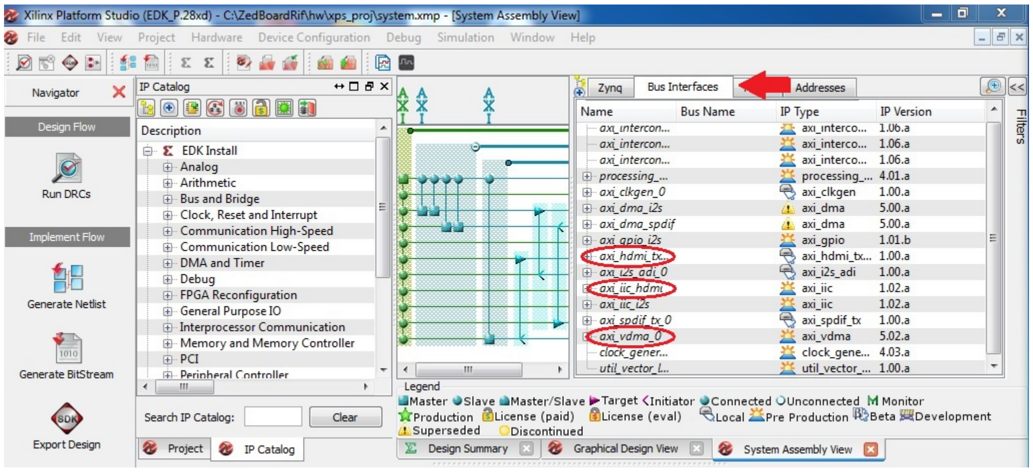
3.1 Include the VGA core

We use for our purpose the VGA support, but it could be substituted with the HDMI support. The VGA core is developed by Xillinux[4], that provide all the requests sources. Below is explained step by step the right procedure.

- From Xillinux website, download the requested sources[5], (we call their [Xillinux sources]).
- Make sure that XPS is closed.
- Copy the [Xillinux sources]\system\pcores\xillyvga_v1_00_a folder in the [Hardware Design Source]\hw\xps_proj\pcores path.
- In the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a path add a folder called netlist.
- Go in the [Xillinux sources]\runonce, and do double click on runonce.xise file.
- Click on vga_fifo, and in below windows do double click on Regenerate Core.
- It will be generate in the runonce folder several files, you have to copy the file vga_fifo.v in the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\hdl\verilog folder.
- Copy the file [Xillinux sources]\runonce\vga_fifo.ngc in the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\netlist folder.
- Copy the file [Xillinux sources]\cores\xillyvga_core.ngc in the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\netlist folder.
- In the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\data\xillyvga_v2_1_0.pao file add the following row:
  lib xillyvga_v1_00_a vga_fifo verilog
- In the file [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\hdl\verilog\xillyvga.v add the following row (under the row N°83) :
  FDCE vga_iob_ff [13:0] (.Q({ vga_red, vga_green, vga_blue, vga_hsync, vga_vsync}),
  .D({ vga_red_app[7:4], vga_green_app[7:4], vga_blue_app[7:4], vga_hsync_app, vga_vsync_app }),
  .C(vga_clk_app), .CE(1'b1), .CLR(1'b0));
- In the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\hdl\verilog\xillyvga.v file modify the following rows
  - (row N°69) output [7:0] vga_blue
    with
    output [3:0] vga_blue

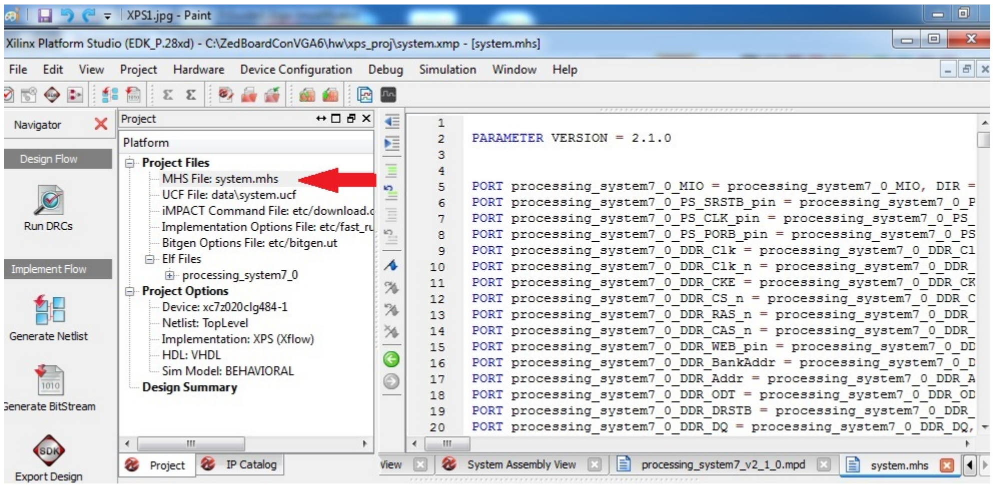
- (row N°70) output [7:0] vga_green
  with
  output [3:0] vga_green
- (row N°72) output [7:0] vga_red
  with
  output [3:0] vga_red

- In the directory [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\data create a file called xillyvga_v2_1_0.bbd
- In the xillyvga_v2_1_0.bbd file add the following text:
  Files
  #####
  vga_fifo.ngc,xillyvga_core.ngc
- In the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\data\xillyvga_v2_1_0.mpd file modify the following rows
  - (row N°90) PORT vga_red = "", DIR = O, VEC = [7:0]
    with
    PORT vga_red = "", DIR = O, VEC = [3:0]
  - (row N°91) PORT vga_green = "", DIR = O, VEC = [7:0]
    with
    PORT vga_green = "", DIR = O, VEC = [3:0]
  - (row N°92) PORT vga_blue = "", DIR = O, VEC = [7:0]
    with
    PORT vga_blue = "", DIR = O, VEC = [3:0]
- In the [Hardware Design Source]\hw\xps_proj\pcores\xillyvga_v1_00_a\data\xillyvga_v2_1_0.mpd file add the following rows(under the row N°7)
  OPTION STYLE = MIX
  OPTION RUN_NGCBUILD = TRUE
- Open the project at [Hardware Design Source]\hw\xps_proj\system.xmp
- Click on Bus Interfaces button and delete in the photo circled components, to delete they do right click on the component and select Delete Instance, in the next windows click on “Delete instance and all its connections” and click Ok. Do the same for all the circled components.



Name	Bus Name	IP Type	IP Version
axi_intercon...		axi_interco...	1.06.a
axi_intercon...		axi_interco...	1.06.a
axi_intercon...		axi_interco...	1.06.a
processing_...		processing_...	4.01.a
axi_clkgen_0		axi_clkgen	1.00.a
axi_dma_i2s		axi_dma	5.00.a
axi_dma_spdif		axi_dma	5.00.a
axi_gpio_i2s		axi_gpio	1.01.b
axi_hdmi_tx...		axi_hdmi_tx...	1.00.a
axi_i2s_adi_0		axi_i2s_adi	1.00.a
axi_iic_hdmi		axi_iic	1.02.a
axi_iic_i2s		axi_iic	1.02.a
axi_spdif_tx_0		axi_spdif_tx	1.00.a
axi_vdma_0		axi_vdma	5.02.a
clock_gener...		clock_gene...	4.03.a
util_vector_l...		util_vector_...	1.00.a

- Open the file system.mhs (showed in the picture)
  - change the row (row N°152)
    PORT IRQ_F2P = axi_vdma_0_mm2s_introut & axi_dma_spdif_mm2s_introut & hdmi_int & axi_dma_i2s_mm2s_introut & axi_dma_i2s_s2mm_introut & axi_iic_i2s_IIC2INTC_Irpt
    with
    PORT IRQ_F2P = axi_vdma_0_mm2s_introut & axi_dma_spdif_mm2s_introut & axi_dma_i2s_mm2s_introut & axi_dma_i2s_s2mm_introut & axi_iic_i2s_IIC2INTC_Irpt



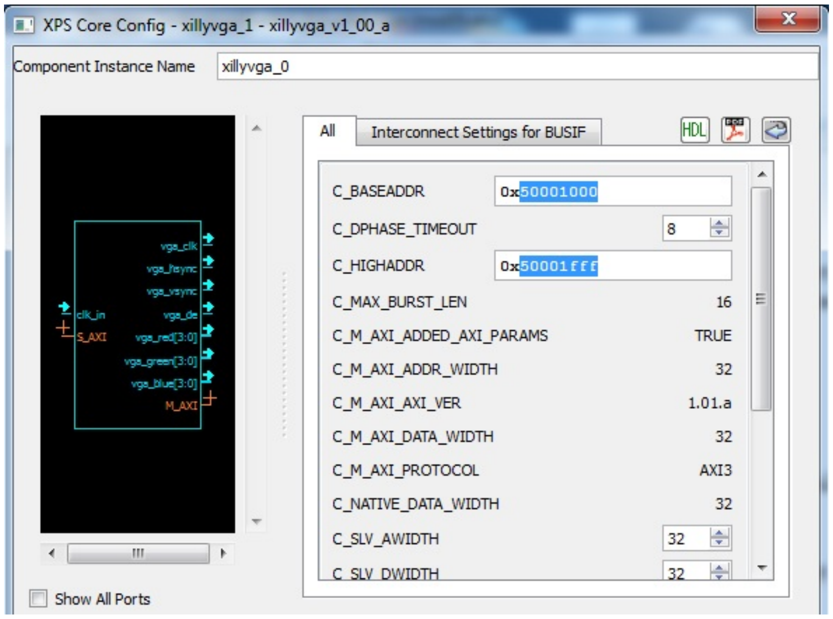
```

PARAMETER VERSION = 2.1.0


PORT processing_system7_0_MIO = processing_system7_0_MIO, DIR =
PORT processing_system7_0_PS_SRSTB_pin = processing_system7_0_P
PORT processing_system7_0_PS_CLK_pin = processing_system7_0_PS_
PORT processing_system7_0_PS_PORB_pin = processing_system7_0_PS
PORT processing_system7_0_DDR_Clk = processing_system7_0_DDR_Cl
PORT processing_system7_0_DDR_Clk_n = processing_system7_0_DDR_
PORT processing_system7_0_DDR_CKE = processing_system7_0_DDR_CK
PORT processing_system7_0_DDR_CS_n = processing_system7_0_DDR_C
PORT processing_system7_0_DDR_RAS_n = processing_system7_0_DDR_
PORT processing_system7_0_DDR_CAS_n = processing_system7_0_DDR_
PORT processing_system7_0_DDR_WEB_pin = processing_system7_0_DD
PORT processing_system7_0_DDR_BankAddr = processing_system7_0_D
PORT processing_system7_0_DDR_Addr = processing_system7_0_DDR_A
PORT processing_system7_0_DDR_ODT = processing_system7_0_DDR_OD
PORT processing_system7_0_DDR_DRSTB = processing_system7_0_DDR_
PORT processing_system7_0_DDR_DQ = processing_system7_0_DDR_DQ,
```

- Click on the IP Catalog manu, click on Project local PCores submanu, click on User and then on XILLYVGA.

- In the XPS Core Config windows set the C_BASEADDR and C_HIGHADDR fields as the picture.

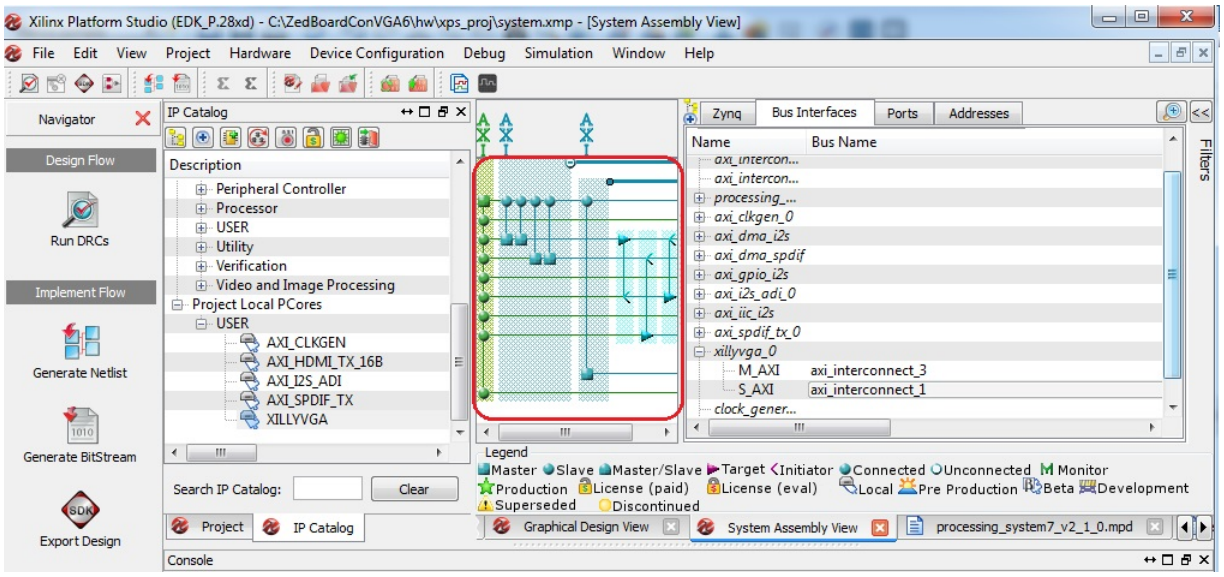


Component Instance Name: xillyvga_0

C_BASEADDR	0x50001000
C_DPHASE_TIMEOUT	8
C_HIGHADDR	0x50001fff
C_MAX_BURST_LEN	16
C_M_AXI_ADDED_AXI_PARAMS	TRUE
C_M_AXI_ADDR_WIDTH	32
C_M_AXI_AXI_VER	1.01.a
C_M_AXI_DATA_WIDTH	32
C_M_AXI_PROTOCOL	AXI3
C_NATIVE_DATA_WIDTH	32
C_SLV_AWIDTH	32
C_SLV_DWIDTH	32

- In the Instantiate and Connect IP windows without change nothing click Ok.

- Make sure that all the connection are as showed in the next picture.



- Click on Ports manu, with right button click on the clk_in xillyvga_0's signal and select "make external", do the same for the signals:
  - vga_hsync
  - vga_vsync
  - vga_red
  - vga_green
  - vga_blue
- Click on the left side on Generate Bitstream

Now the bitstream file have been generate. You can find it in [Hardware Design Source]\hw\xps_proj\implementation\system.bit path. Now to generate the FSBL you have to follow these steps:

- Click on Export Hardware Design
- Check "include bitstream and BMM file" in the open dialog box.
- Press Export Only
- Once the process is finished, open SDK and create a new workspace. We call it WorkSpace.
- Import the Hardware Profile
- Open the File menu, click New and go on "Xilinx Hardware Platform Specification" to open a dialog window

- Click the browse button
- Select the hardware platform specification file. You can find it in [Hardware Design Source]\hw\xps_proj\SDK\SDK_Export\hw\system.xml path.
- Under the File menu, click New and then "Xilinx C Project" to open the dialog window.
- In the dialog window, choose "Zynq FSBL" under the "Select Project Template" header. Leave all the other options as defaults, press Next.
- Leave all options as defaults again and click Finish.
- Replace [Work Space]\zynq_fsbl_0\src\main.c with [Hardware Design Source]\sw\zynq_fsbl\src\main.c.
- Under the Project menu, click Clean then select "Clean all projects," and then press OK. Pressing OK should clean and rebuild the project.

After the rebuild has completed, you find the FSBL binary at [Work Space]\zynq_fsbl_0\Debug\zynq_fsbl_0.e
The SSBL is the u-boot.elf file, you can find it in the [Hardware Design Source]\boot_image\u-boot.elf. path. If you would like to customize it you should:

- Download the source files from the git repository with this command
  git clone https://github.com/Digilent/u-boot-digilent
- Find the board setting, for the Zedboard you should find in include/configs/zynq_zed.h file.
- Configure U-Boot through a series of macros.

Now you have all the necessary files to generate the boot.bin file.

- Open Xilinx SDK and set a workspace.
- Click on Xilinx Tools menu, and then Create Boot Image.
- In the dialog window, set the Bif file to "Create a new Bif file…"
- Browse to select the your choosing FSBL.
- Click on the Add button and select you choosing bitstream.
- Click on the Add button and select the SSBL (u-boot.elf file).
- Choose the Output Folder.
- Click Create Image, you should find the boot.bin file in the output folder.

4 Prepare the kernel for Android.

4.1 Obtain the Android patches to apply at Linux kernel.

It is necessary to apply a patch from google source to obtain a fit kernel for our purposes. This patch is derivable via git repository. The expected patch size is about 2MB. To extract the patches we using "git log".
Go to the top level of the Linux 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 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]

4.2 Obtain the xillinux patches to use VGA.

From the Xillinux Web site download the Xillinux file system[6]. You have to mount the .img file only to obtain the Xilly patches, you can find them in /usr/src/xillinux/kernel-patches path.

You will find six patches but you have to apply for VGA core only:

- 0002-FPGA1-clock-frequency-and-others-settings-to-match-h.patch
- 0004-Added-xillyvga-driver-to-kernel-tree.patch
- 0005-To-be-removed-HP2-AXI-bus-width-set-to-32-bits-in-VG.patch

To apply you have to use the following command:
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.

4.3 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.

- *Make you sure to have these configuration modules*
  CONFIG_ARCH_ZYNQ=y
  CONFIG_ANDROID_PARANOID_NETWORK=y
  CONFIG_ANDROID=y
  CONFIG_ANDROID_BINDER_IPC=y
  CONFIG_ASHMEM=y
  CONFIG_ANDROID_LOGGER=y
  CONFIG_ANDROID_PERSISTENT_RAM=y
  CONFIG_ANDROID_RAM_CONSOLE=y
  CONFIG_ANDROID_TIMED_OUTPUT=y
  CONFIG_ANDROID_LOW_MEMORY_KILLER=y
  CONFIG_CLKDEV_LOOKUP=y

```
CONFIG_HAVE_CLK_PREPARE=y
CONFIG_COMMON_CLK=y
CONFIG_HAS_WAKELOCK=y
CONFIG_WAKELOCK=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
```

4.4 Build the patched and configurated 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 Linux kernel source tree.

4.5 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, spurious);
}
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;
}
```

4.6 Prepare the device tree

The Device Tree is a data structure for describing hardware. Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time.

You can find a device tree source in [Hardware Design Source]/linux path. You to apply the following modifications

- comment these rows
  - From 213 to 293
  - From 300 to 329
  - From 343 to 355
- modify these rows
  - modify the row (467) compatible = "xlnx,ps7-sdio-1.00.a", "generic-sdhci";
    with
    compatible = "xlnx,ps7-sdio-1.00.a", "generic-sdhci";

  - modify the row (469) interrupts = < 0 24 4 >;
    with
    interrupts = < 0 24 0 >;
  - modify the row (50) bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x800000,8M earlyprintk rootwait devtmpfs.mount=1";
    linux,stdout-path = "/axi@0/serial@e0001000";
    for Linux boot with
    bootargs = "console=ttyPS0,115200 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk rootfstype=ext4";
    for Android boot with
    bootargs = "consoleblank=0 root=/dev/mmcblk0p3 fsroot=jffs2, nolock, wsize=1024, resize=1024 rw init=init noinitrd mem=1024M user_debug=31 earlyprintk";

- add these rows

  - xillyvga@50001000 {
    compatible = "xlnx,xillyvga-1.00.a";
    reg = < 0x50001000 0x1000 >; } ;

Now to compile the device tree you have to copy the modified file in [Linux Kernel Source]/arch/arm/boot/dts folder, then in the [Linux Kernel Source] path type from the Linux shell:

```
make ARCH=arm CROSS_COMPILE=[path-to-arm-gcc] devicetree.dtb
```

You will find in [Linux Kernel Source]/arch/arm/boot/ folder the devicetree.dtb file.

4.7 Booting Linux OS

It could be useful to boot linux for test reason or to check the previous steps correctness. To do this you have to follow these steps:

- Copy the linux file system in the partition created for this purpose.
- Copy in the boot files partition the boot.bin, devicetree.dtb and zImage files.
- Set the board jumpers like is shown in the picture.
- Connect the usb cable to the UART and a VGA monitor.
- Put the sd card in the reader.
- Turn on the board.
- Type on the Linux shell this command:
  screen /dev/ttyACM0 115200

If all works well you will see the linux shell in the terminal, the Linux symbol on the VGA monitor. This thing is important because we use the VGA monitor to show the Android GUI.

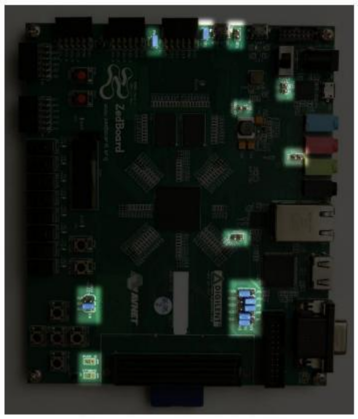


Figure 1: Board jumpers configuration

5 Android filesystem

5.1 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.
Below there is an example of adding a couple of ARM generic Android build targets, along with a mock target.
An example is downloadable here. [?]

- 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 arm-
board_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
```

5.2 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

5.3 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
```

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

5.4 Prepare the SD to boot Android

- Copy [android_root]/out/target/product/armboard_v7a/root directory content in the Android 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.

- Now all is ready to boot Android. Keep in mind to change the boot args to be suitable to Android boot. If all works properly you will see the Android GUI on VGA monitor, you have to connect your keyboard via

USB OTG to use Android.

5.5 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.

5.6 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
```

5.7 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 Bibliography and reference

This 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
- ZedBoard™ Linux Hardware Design Project Guide made by digilent
- http://linux-arm.org/LinuxKernel/LinuxAndroidPlatform
- http://processors.wiki.ti.com/index.php/Android_ADB_Setup
- http://stackoverflow.com

References

[1] http://www.oracle.com/us/downloads/index.html

[2] http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,1028&Prod=ZEDBOARD

[3] http://linux-arm.org/git?p=armdroid.git;a=blob_plain;f=fs/src/Froyo/Froyo-device.tar.bz2;hb=6706399926a700f2f9b939277349f7196671af21

[4] https://www.xillybus.com/xillinux

[5] http://www.xillybus.com/downloads/xillinux-eval-zedboard-1.2.zip

[6] http://www.xillybus.com/downloads/xillinux-eval-zedboard-1.2.zip