

INTRODUCTION

This document tries to describe the first steps when developing with TI's Sitara ARM Microprocessors. It was put together for personal use and is no substitution for official TI's documentation which can be found on the web.

Hardware and software used

- Host is running 32-bit Ubuntu 12.04.1 LTS on a 64bit machine (Intel® Core™2 Duo CPU T7300 @ 2.00GHz × 2)
- TI Linux SDK: ti-sdk-am335x-evm-05.06.00.00-Linux-x86
- TI Graphics SDK: Graphics_SDK_setuplinux_4_08_00_02
- Target: Beaglebone Rev A6
- LCD: Part of ChipSee's expansion which includes a 7" LCD with resistive touch (INNOLUX Display Model AT070TN92 V.), DVI output, accelerometer and audio I/O

PART 1: Linux EZ Software Development Kit (EZSDK) for Sitara™ ARM® Microprocessors - LINUXEZSDK-BONE

Downloading and Installing TI's Linux EZSDK on the Host

1. The SDK can be downloaded from TI's website: <http://www.ti.com/tool/linuxezsdk-sitara>
Versions of the SDK exist for various IC's and/or development boards. Based on used hardware¹ the LINUXEZSDK-BONE is to be downloaded. At the time of writing this document the latest version of the LINUXEZSDK-BONE is v05.06.00.00 (Linux kernel 3.2.0). The name of downloaded installer executable is ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install.
2. Move the downloaded executable to a working directory. Throughout the document this directory is /usr/local/LinuxDevelopment and is denoted as \$WORKDIR.
3. Before installing the just downloaded EZSDK make sure the file did not become corrupt during downloading or possible moving of the file to a working directory after downloading it. Whether the file is corrupt or not can be checked by calculating a md5sum like so:

```
md5sum ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install
```

The result will look like:

```
990e690eea37dacb911fd036dae5820c ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install
```

The first part is the checksum which can be compared with checksums inside md5sum.txt, found on the TI's website from which the EZSDK was downloaded.

4. When the installer is downloaded it is not executable by default so the file's permissions need to be changed first:

```
sudo chmod a=rwx ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install
```

5. Install EZSDK by moving to \$WORKDIR and issuing:

```
./ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install
```

The chosen installation directory was \$WORKDIR/ti-sdk-am335x-evm-05.06.00.00 and is referred to as

¹ See chapter Hardware and software used.

`$SDKHOME` from here on.

If one also downloaded TI's Code Composer studio from their web site and put the downloaded file with executable permissions into the same folder as `ti-sdk-am335x-evm-05.06.00.00-Linux-x86-Install` (`$WORKDIR`) then the option to also install CCS can be selected in SDK's installer. This was not done in this example.

6. When installation is complete the user is notified (in the last window of the installer) that an environment setup script needs to be run in order to correctly configure the host so that the SDK will be able to run on it. The `setup.sh` script is located in `$SDKHOME`. It must be run with root privileges so run it by issuing:

```
sudo ./setup.sh
```

Things done by the script (also have a look at `setup.sh`'s output in `Setup-Script-Output.txt`):

- a) Verify that you are running Ubuntu 10.04 LTS or Ubuntu 12.04 LTS. SDK was tested on these machines. If you are running a different Linux this check needs to be omitted by commenting out all lines in the `$SDKHOME/bin/setup-host-check.sh` script which, amongst others, gets called by the `setup.sh` script.
 - b) Install all needed packages (`xinetd tftpd nfs-kernel-server minicom build-essential libncurses5-dev uboot-mkimage autoconf automake`)
 - c) Set up a Network File System (NFS) and place a default filesystem in a directory of your choice. Default is `$SDKHOME/targetNFS`. Whatever you decide to change it to it will be referred to as `$TARGETNFS` from here on.
 - d) It exports the created filesystem for NFS access by adding below line to the `/etc/exports` file on your host:
`$SDKHOME/targetNFS *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)`
assuming `$SDKHOME/targetNFS` is the location of target's filesystem chosen during `setup.sh` execution.
 - e) Set up a tftp server and a tftp's folder (`$TFTPHOME`) which will hold target kernel image. The script also places a default image into this folder (`uImage-am335x-evm.bin`)
 - f) Set up minicom serial terminal (for connecting to the target through a serial port). Minicom setup can later be changed in `/etc/udev/`
 - g) Create a `uEnv.txt` file which contains u-boot variables. U-boot is a bootloader which searches for a `uEnv.txt` upon startup and if found, uses the variables from that file to boot Linux kernel. Variables among other things contain the boot method (in this case `tftpboot` and NFS filesystem), relevant ip addresses, ...
7. (This may be redundant.) Change permissions of `$SDKHOME` and all its contents and also of the tftp directory and its contents (`/tftpboot` is assumed):

```
sudo chmod -R a=rwx $SDKHOME
sudo chmod -R a=rwx /tftpboot
```

Testing SDK Installation: Compile and Deploy u-Boot and Linux Kernel, Deploy a Root Filesystem and Run Everything on Beaglebone

To test the just installed Linux EZSDK we will do the following:

- Compile and deploy u-Boot contained in the SDK,
- Compile and deploy Linux kernel contained in the SDK,
- Deploy a default target file system also contained in the SDK

Compile and deploy u-Boot

U-Boot is a third-stage bootloader used to load a Linux image and boot it. TI's EZSDK comes with u-Boot source (version 2012.10 in our case). First stage bootloader is already contained on the silicon of Sitara microprocessors (burnt in ROM). This first-stage bootloader loads a second-stage bootloader which comes in form of a MLO file. Both the MLO and u-Boot image are built when building u-Boot.

1. First navigate to u-Boot's source folder. It should be located at `$SDKHOME/board-support/u-boot-2012.10-psp05.06.00.00`. This folder is referred to as `SUBBOOTHOME` from here on.

2. Build u-Boot and MLO by issuing:

```
make O=NU-object_folder CROSS_COMPILE=$SDKHOME/linux-devkit/bin/arm-arago-linux-gnueabi- ARCH=arm am335x_evm
```

`$SDKHOME/linux-devkit/bin` is the location of the **arago toolchain** provided with TI's EZSDK.

`NU-object_folder` is a working folder for the build process and is also where the results will be saved. This folder can be anything you like. You can create it prior to issuing above command or let the build create it. Using a separate, so called object folder for building is advisable by TI as it makes cleaning the build easier – only the contents of this folder need to be deleted.

3. After the build is finished two relevant files have been placed into the working object folder (`NU-object_folder` in our case). These are `MLO` and `u-boot.img`. These two files need to be placed into a FAT32 boot partition on a micro SD card.
4. Additionally to `MLO` and `u-boot.img` another file is needed on the SD card. This is the `uEnv.txt` file which contains a sort of script for u-Boot to execute upon startup. U-Boot automatically searches for this file when it starts up. If it does not find it a default Linux boot sequence is carried out. The default boot sequence will only work in a specific set of circumstances so we will write our own Linux boot sequence and put it into `uEnv.txt`. Take a look at `uEnv.txt` which should be located in the same folder as this EZSDK installation manual document.

Make sure that `serveripaddress` is set to the ip address of your host, where you will be hosting the tftftp and NFS. `Hostnfsroot` needs to be set to the location of your NFS on the host (`$TARGETNFS`) and `bootfilelocation` must be set to the location and name of the Linux kernel image. This location is relative to the home folder of the tftp server - `$TFTPHOME`. We have not yet built the Linux kernel so there is no image in this folder yet, apart from the one the `setup.sh` script placed in it (I never found a need to use this image). See chapter Downloading and Installing TI's Linux EZSDK on the Host, point 6 for more details about the `setup.sh` script.

Deploy a Linux Filesystem

For development purposes it is usual to boot Linux via a tftp server and use a Network File System (NFS) to access its file system which is located on the host. So both the target's kernel image and its file system are located on the host and accessed over network by the target. When running the `setup.sh` script the NFS was already set up and a default target file system was already placed in the `$SDKHOME/targetNFS` folder. This folder was also marked as one which contains a file system which can be exported to remote hosts by appropriately modifying the `/etc/exports` configuration file on the host. All of this was done automatically. See chapter Downloading and Installing TI's Linux EZSDK on the Host, point 6 for more details about `setup.sh` script.

The file system which is placed into `$SDKHOME/targetNFS` by default is a fairly extensive file system. To start with a more “bare-boned” file system one can delete the contents of `$SDKHOME/targetNFS` and extract and unzip the `arago-`

`base-tisdk-image-am335x-evm.tar.gz` file system into this same folder instead. This archive is contained in `$SDKHOME/filesystem` and as its name suggests, it contains a more basic (base) file system.

If at any time you will change your target file system's directory (default is `$SDKHOME/targetNFS`) you must update its path in `/etc/exports` and issue the following commands: `/usr/sbin/exportfs -a`

The NFS based file system is now set up. We may proceed with compiling and deploying a Linux kernel.

Compile and deploy Linux Kernel

TI's EZ Linux SDK comes with a full Linux kernel source located in `$SDKHOME/board-support/linux-3.2.0-psp05.06.00.00`, referred to as `$LINUXHOME` from here on. The Linux kernel version contained by the used SDK is 3.2.0 and uses PSP version 05.06.00.00. PSP stands for Platform Support Products and is created with the intent to provide OS ports, device drivers and sample applications to kick start development on a TI SOC platform by customers.

1. First navigate to `$LINUXHOME`.
2. If needed change any source files that need changing to enable any capes and/or other functionality. See the `NU_Linux-Source-Code-Modifications` document for examples of modifying Linux kernel source code.
3. As when compiling u-Boot the Arago toolchain, which comes with the SDK, is used in Linux kernel compilation as well. If you remember we had to give the `make` utility a full path to this toolchain when compiling u-Boot (*Build u-Boot and MLO by issuing:*). At this point it may be wise to export this path to Linux' `PATH` environment variable by issuing the following command:

```
export PATH=$PATH:$SDKHOME/linux-devkit/bin
```

To export this permanently put the same command in the `.profile` file, usually located in the home directory of your Linux installation. Exporting the toolchain's location to Linux's `PATH` environment variable will do away with the need of writing `CROSS_COMPILE=$SDKHOME/linux-devkit/bin/arm-arago-linux-gnueabi-` as a parameter every time a kernel will be compiled using `make`. Instead only `CROSS_COMPILE=arm-arago-linux-gnueabi-` will suffice.

4. Before starting the compilation/build it is a good idea to clean up the build environment from the last build. This is obviously not necessary for the first build but it is done for subsequent builds if a major change to Linux sources is made. The clean is done by issuing `make mrproper` when located in `$LINUXHOME`.
5. Linux source has a very large code base because it has the ability to support numerous machines with many different configurations. To enable the ones needed for Linux to run on the Beaglebone only those parts of code which are needed for Beaglebone need to be compiled. To tell the compiler which these parts are a special `.config` configuration file needs to be created. This can be done manually, but this is never done. The Linux source includes many configuration files which were already written by the community and if you are using a device which requires the same configuration as in any of those files you can just use that configuration file. If using a device which does not have its own configuration file one of the included files can at least be taken as a base and modified accordingly (`menuconfig` utility which can be invoked by issuing `make menuconfig` is useful for modifying Linux' source configuration). Luckily, the Linux source already contains a configuration file which can be used to compile Linux kernel for the Beaglebone. It is called `tisdk_am335x-evm_defconfig`. To use this configuration file first issue the following command while in `$LINUXHOME`:

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- tisdk_am335x-evm_defconfig
```

This creates a `.config` file identical to `tisdk_am335x-evm_defconfig` in Linux source top directory - `$LINUXHOME`.

6. Now everything is ready for compilation of Linux kernel from its source. This is done by issuing the following command which builds a Linux kernel image suitable for mounting with u-Boot:

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage
```

After the build is done an `uImage` file is placed into `$LINUXHOME/arch/arm/boot` folder. This file contains the Linux kernel image which will be used by u-Boot to boot up Linux on the target.

7. To build any kernel modules which the kernel image needs the following command must now be issued:

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- modules
```

8. The built modules must now be installed (correctly placed into the target's file system directory tree):

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-  
INSTALL_MOD_PATH=$SDKHOME/targetNFS modules_install
```

9. As a last step the uImage created in step 6 needs to be placed in the folder used by the tftp server so that the Beaglebone will be able to find it. This folder was set up when running the setup.sh script as part of the SDK's installation – See chapter Downloading and Installing TI's Linux EZSDK on the Host, point 6. The default folder is /tftpboot. Both this folder and the kernel image inside /tftpboot (uImage) need to have read and write permissions!

After completing this last step the host is now set up, u-Boot and Linux kernel built and deployed, the file system is ready and all that is left is to start up the Beaglebone and see if everything works.

Test Everything on Beaglebone

If above steps were completed successfully firing up Beaglebone is easy. Simply connect its mini USB port to the PC, start minicom or similar serial terminal, insert the SD card which contains the MLO, u-Boot and uEnv.txt into the Beaglebone, connect the Beaglebone to your router and apply power. You should see the whole boot-up sequence debug messages being written down in minicom. When the Beaglebone is booted into Linux a prompt comes up, asking for the user of the current session. Type `root` and hit return. You are now running Linux on the Beaglebone!

PART 2: TI Linux Graphics SDK - gfxsdk 4_08_00_02

Beaglebone's AM3359 microprocessor contains, as many other TI's SOC's, an on chip graphics accelerator – SGX (based on SGX530 core, revision 1.2.5). It is based on Imagination Technologies Ltd. POWERVR® SGX530 core and enables 2D and 3D acceleration of graphics applications. In order for these applications to make efficient use of the SGX a suitable Linux drivers are needed. TI Linux Graphics SDK contains these drivers for different TI's SOC's. It also contains demos which can be used if drivers were properly installed on the target.

This chapter describes the steps needed to install proper graphics drivers on the Beaglebone and successfully test them by running a demo application.

Downloading and Installing TI Linux Graphics SDK

Before continuing make sure the following conditions are met:

- Ensure that toolchain installation is complete
- Ensure that the NFS target is setup
- Ensure that the setup and build for u-boot is complete
- Ensure that the setup and build for Linux Kernel is complete

Once all of the above is taken care of, proceed with downloading and installing the Graphics SDK.

1. The Graphics SDK can be downloaded from TI's website:

http://software-dl.ti.com/dsp/dsp_public_sw/sdo_sb/targetcontent/gfxsdk/latest/index_FDS.html

At the time of writing this the latest version is 4.08.00.02 and the installer's name is

Graphics_SDK_setuplinux_4_08_00_02.bin . This installer installs the full package. One with only a couple of demo applications also exists and so does one with no demos, only the drivers.

2. Move the downloaded executable to a working directory. I chose \$WORKDIR..
3. Before running the just downloaded installer make sure the file did not become corrupt during downloading or possible moving of the file to a working directory after downloading it. Whether the file is corrupt or not can be checked by calculating a md5sum like so:

```
md5sum Graphics_SDK_setuplinux_4_08_00_02.bin
```

The result will look something like:

```
7c29d8eb6631ee42317fdc4487cb6e68 Graphics_SDK_setuplinux_4_08_00_02.bin
```

The first part is the checksum which can be compared with checksums inside md5_SDK.txt under md5 Checksums, found on the TI's website from which the Graphics SDK was downloaded.

4. When the installer is downloaded it is not executable by default so the file's permissions need to be changed first:

```
sudo chmod a=rwx Graphics_SDK_setuplinux_4_08_00_02.bin
```

5. Install the Graphics SDK by moving to \$WORKDIR and issuing:

```
./Graphics_SDK_setuplinux_4_08_00_02.bin
```

The chosen installation directory was \$WORKDIR/Graphics_SDK_setuplinux_4_08_00_02.bin and is referred to as \$GSDKHOME from here on.

During the installation process one can select which components will be installed. Only the es8.x and sdk are needed for Beaglebone. Other components contain graphics drivers for other SOC's. es8.x contains drivers for am335x SOC's and sdk contains demo applications.

6. Change directory to \$GSDKHOME
7. Export the architecture to environment variables: export ARCH=arm

8. Open `$GSDKHOME/Rules.make` file and edit the following (in Ubuntu you must edit the file as a super user):

- Set `HOME` to `$WORKDIR`
- Set the toolchain installation directory path to where your toolchain resides (Arago):
`CSTOOL_DIR=$SDKHOME/linux-devkit/bin`
(`$SDKHOME` was defined earlier in chapter Downloading and Installing TI's Linux EZSDK on the Host, point 5)
- Set the toolchain prefix: `CSTOOL_PREFIX=arm-arago-linux-gnueabi-`
- Set the kernel installation path: `KERNEL_INSTALL_DIR=$LINUXHOME`
(`$LINUXHOME` was defined earlier in chapter Compile and deploy Linux Kernel)
- Set the path to the target's file system: `TARGETFS_INSTALL_DIR=$SDKHOME/targetNFS`
- Set the top-level graphics installation directory path: `GRAPHICS_INSTALL_DIR=$(HOME)/Graphics_SDK_4_08_00_02`
(This should actually already be set correctly by default.)

9. After modifying the `Rules.make` file a build process can be started which will build the demo applications and prepare everything needed for later installation of the drivers (see points 10 and also). Issue the following command to start the build process (we will be doing a build intended for target Linux without Xorg):

```
make BUILD=release OMAPES=8.x all (for development it is worth considering building a debug version)
```

This command will build the complete graphics SDK for AM335x. Options to only build the kernel modules also exists (make `BUILD=release OMAPES=8.x all_km`). If specifying `RELEASE=debug` a debug version can be built which is more suitable for development as it outputs many debug messages while running demos.

10. After building, the Graphics SDK needs to be installed to the target's filesystem:

```
make BUILD=release OMAPES=8.x install (for development it is worth considering building a debug version)
```

This command will install the complete graphics SDK to target file system as mentioned in `Rules.make`. A command to only install the graphics kernel modules to the target's file system also exists (make `BUILD=release OMAPES=8.x install_km`).

11. During this last step a notice was issued saying: "Please ensure that PSP Linux kernel is re-built at least once." So at this stage the Linux kernel needs to be rebuilt and re-deployed (see chapter Compile and deploy Linux Kernel).

12. Compare Linux boot arguments from `uEnv.txt` and the ones below. Make sure the relevant ones match:

```
setenv bootargs 'console=ttyO0,115200n8 root=/dev/nfs
nfsroot=<nfshostipaddress>:<rootpath>,nolock rw mem=256M ip=dhcp earlyprintk=serial
vram=50M'
```

13. Now boot into Linux on the target.

14. Run the script `/etc/init.d/335x-demo` at Linux command prompt after the Beaglebone boots up. The graphics SDK comes with a SOC specific startup script that takes care of installing the right set of graphics driver libraries and kernel modules. For am335x SOC's this is the `335x-demo` script.

See `335x-demo_script_output.txt` document for an output of the `335x-demo` script.

15. To check the status of the just installed GFX drivers on the target run the `gfx_check.sh` script. You can find it here: https://gforge.ti.com/gf/download/docmanfileversion/203/3715/gfx_check.sh and copy it somewhere into the target's filesystem (I choose `/opt`). When on the target move to the directory where the `gfx_check.sh` is located and issue the following command:

```
./gfx_check.sh (if needed make it executable first – chmod command)
```

See `gfx_check_script_output.txt` for an output of the `gfx_check.sh` script. The output should report that SGX drivers are correctly installed under SGX driver information.

16. The just installed drivers are actually kernel modules which were loaded by the `335x-demo` script as a last step of its execution. The Linux kernel will not load these modules by default. To avoid having to load modules manually after every start up a start-up script is needed which will do that for us. This script will be a simple and short script which will actually only call another script that was installed with on the target as a part of the graphics SDK. The script in question is `rc.pvr` which is located in `/etc/init.d` and loads the SGX driver.

So, the script we need to write is a simple one and looks like this:

```
#!/bin/sh
echo "Loading pvrsvkm kernel module (SGx PVR driver) "
if [[ -x /etc/init.d/rc.pvr ]]; then
    cd /etc/init.d
    ./rc.pvr start
else
    echo "pvrsvkm kernel module (SGx PVR driver) failed to load!"
fi
```

Named the script `load-pvr-driver` and place it into `/etc/init.d` folder of the target's file system. This script needs to be executed towards the end of initialization after every power-up. Since Linux is operating in user level 5 after booting successfully our script needs to be run towards the end of initialization scripts for run level 5. Hence, the script needs to have a symbolic link placed to it in the `/etc/rc5.d` folder. To do that change directory to `/etc/rc5.d` and issue the following command:

```
ln -s ../init.d/load-pvr-driver ./S98load-pvr-driver
```

After this is done, Beaglebone should automatically load SGX graphics drivers after every power on.

17. Again make sure that the entire target's filesystem has read, write and execute permissions by moving to `$SDKHOME` and issuing the following command:

```
sudo chmod -R a=rwx targetNFS
```

Test the Installed Graphics Drivers

To test the graphics drivers installed by following the instructions given in Downloading and Installing TI Linux Graphics SDK one must simply run one of the demos which were installed with the graphics SDK. To start one do the following:

1. If you have not done so yet, restart Beaglebone after finishing the last step described in Downloading and Installing TI Linux Graphics SDK
2. Change directory on the target to `/opt/gfxsddemos/ogles`
3. Run Skybox demo by issuing `./OGLESSkybox`
4. You should see an animation of a hot-air balloon flying over the sea on the LCD plugged to your Beaglebone.
5. To exit the demo press 'q'.

PART 3: Enabling LCD and TSC

To be able to use an LCD with touch screen controller (TSC) on the Beaglebone you must first connect it to appropriate pins (refer to AM335x Technical Reference Manual, your LCD's data sheet and your TSC data sheet). After doing so some minor modifications to Linux source code must be done. For my hardware (see chapter Hardware and software used) changes described in the `NU_Linux-Source-Code-Modifications` needed to be made.

After Linux source is modified it needs to be rebuilt as described in chapter Compile and deploy Linux Kernel.

Whether modifications and build were done successfully will be seen on next boot of the target. Proper operation of the LCD will be seen soon after power-up (Linux Penguin logo should be visible shortly after power-up).

The best way to test the touch screen is to calibrate it. Log in to Linux over the serial console so that you will be able to see debug outputs from the calibration utility while calibrating (LCD will be used by the utility so debug text will not be output there). After logging in run `ts_calibrate` utility by typing `ts_calibrate` into the command prompt and hitting enter (return). You should see “*TSLIB calibration utility. Touch crosshair to calibrate.*” Written on the LCD. Follow the instructions and touch the 5 crosshairs. Each time you touch it it jumps to a new location and you should see the coordinates of your last press written in the serial console.