1 Configuring Linux

1.1 Introduction

This section will explain how to set up a linux distribution from scratch on the gumstix that is compatible with the project board. The process of creating a working linux system is tricky and requires some knowledges on how to use command line based commands. The instructions will explain how to set up **OpenEmbedded** cross compilation system, how to configure and compile the linux kernel and how to configure the outcoming linux system. The compilation of everything from scratch is time consuming and the initial compilation of the OpenEmbedded packages took about 24hours on a virtual machine.

1.2 OpenEmbedded

1.2.1 What is OpenEmbedded ?

Openembedded is a build framework for embedded devices running Linux. It offers a suit of tools and packages to allow developers to create a complete Linux distribution for the target device. Bitbake "recipes" are included to provide a configuration-less deployment of packages to different architectures.

1.2.2 Prerequisites

- A recent Linux distribution, Linux Mint 10 has been used during the project
- Distribution packages: git, subversion, gcc, patch, help2man, diffstat, texi2html, makeinfo (texinfo on Linux Mint/Ubuntu), ncurses-devel (libncurses5-dev on Linux Mint/Ubuntu), cvs, gawk, python-dev, python-pysqlite2, unzip, chrpath, ccache and python-psyco (recommended).

Note: On some distributions the **/bin/sh** file is a symbolic link to **/bin/dash**, to restore sh to the standard shell run **dpkg-reconfigure dash** as super-user and answer **NO** when asked whether you want to install dash as **/bin/sh**.

1.2.3 Retrieving the source code

The Gumstix documentation on OpenEmbedded recommend at least 10GB of free space on the hard disk however during the project a dedicated linux install has been used and 50GB of hard disk were required for the compilation plus the Linux distribution.

The default gumstix OpenEmbedded configuration uses the **overo-oe** folder in the home folder of the current user this can be modified but will not be explain in this document.
First create the "overo-oe" folder and cd into it.

```
$ mkdir -p ~/overo-oe
$ cd ~/overo-oe
```

Now to retrieve the Gumstix Overo OpenEmbedded solution, this will take some time depending on your internet connection.

```
$ git clone git://gitorious.org/gumstix-oe/mainline.git\
        org.openembedded.dev
$ cd org.openembedded.dev
$ git checkout --track -b overo origin/overo
```

Next step is to install BitBake.

```
$ cd ..
$ git clone git://git.openembedded.net/bitbake bitbake
$ cd bitbake
$ git checkout 1.10.2
```

1.2.4 Environment setup

Create the OpenEmbedded configuration files and profile based on the provided files.

```
$ cp -r org.openembedded.dev/contrib/gumstix/build .
```

Setup the environment variables on the bash profiles (assuming bash is the shell used).

```
$ cat ~/overo-oe/build/profile >> ~/.bashrc
```

1.2.5 First Build

You can build a basic kernel and non-gui root file system now, this task will take a long time depending on the internet connection as the packages required will be downloaded on the fly and also depending on the computer speed. Only the first compilation will take so much time and the next ones will used the precompiled packages. One issue has been identifier on the University network at this stage, wget program is used to download some of the source code and does not apply the proxy settings. (a workaround can be used but is not in the scope of this document).

Run the build process

```
$ cd ~/overo-oe
$ bitbake omap3-console-image
```

Once the compilation is over the root file system and kernel image will be located in **/overo-oe/tmp/deploy/glibc/images/overo**

2 Preparing the microSD Card

2.1 Partitioning the card

The microSD card have to be formatted in a specific way to allow the gumstix to boot on it.
Identify the micro sd device

```
$ fdisk -l
```

Assuming the micro sd device is **/dev/sdb**
Unmount the device

```
$ umount /dev/sdb
```

Create an empty partition table

```
$ sudo fdisk /dev/sdb
Command (m for help): o
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
```

Look at the current card informations

```
Command (m for help): p
Disk /dev/sdb: 2032 MB, 2032664576 bytes
64 heads, 63 sectors/track, 984 cylinders
Units = cylinders of 4032 * 512 = 2064384 bytes
Disk identifier: 0x00aa8e5c
Device Boot      Start         End      Blocks   Id  System
```

Note the card size in bytes (i.e 2032664576)
Switch to "Expert" mode:

```
Command (m for help): x
```

To allow the gumstix to boot from the sd card the geometry needs to be 255 heads and 63 sectors with 512 bytes per sector therefore the number of cylinders have to be calculated.

card_number_of_bytes / 255 / 63 / 512 = number_of_cylinders

This value have to be round **down** and not up. In the case of the SD card shown above the number_of_cylinders is 247.

```
Expert command (m for help): h
Number of heads (1-256, default 4): 255
Expert command (m for help): s
Number of sectors (1-63, default 62): 63
Warning: setting sector offset for DOS compatiblity
Expert command (m for help): c
Number of cylinders (1-1048576, default 984): number_of_cylinders
```

Return to fdisk main mode and create the two partitions that will be used by the kernel and by the root file system (rootfs).

Create a 32MB FAT partition

```
Expert command (m for help): r
Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-247, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-247, default 15): +32M
```

Change the partition to FAT32

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

Make it bootable

```
Command (m for help): a
Partition number (1-4): 1
```

Create the ext3 partition for the rootfs

```
Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (6-247, default 6): 6
Last cylinder or +size or +sizeM or +sizeK (6-247, default 247): 247
```

Verify the new partition info

```
Command (m for help): p
Disk /dev/sdb: 2032 MB, 2032664576 bytes
255 heads, 63 sectors/track, 247 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x00aa8e5c
Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1           5       40131    c  W95 FAT32 (LBA)
/dev/sdb2               6         247     1943865   83  Linux
```

Write the new partition table onto the card

```
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks
```

2.2 Formatting the new partitions

Create the two paritions file systems

```
$ sudo mkfs.vfat -F 32 /dev/sdb1 -n FAT
        ...
$ sudo mkfs.ext3 /dev/sdb2
        ...
```

2.3 Installing the system on the SD card

There are three files required of the FAT partition to boot the gumstix that should have been compiled during the OpenEmbedded compilation stage.

- MLO: the boot-loader loader
- u-boot.bin: the boot-loader
- uImage: the Linux Kernel

Mount the FAT partition and copy the MLO **first**.

```
$ sudo mount /dev/sdb1 /media/card
$ sudo cp MLO-overo /media/card/MLO
$ sudo cp u-boot.bin /media/card/u-boot.bin
$ sudo cp uImage /media/card/uImage
```

Unmount the FAT partition and mount the ext3 one

```
$ sudo umount /dev/sdb1
$ sudo mount /dev/sdb2 /media/card
```

Untar the rootfs

```
$ cd /media/card
$ sudo tar xvaf /path/to/console-image.tar.bz2
```

Unmount the ext3 partition

```
$ umount /dev/sdb2
```

At this point you should have a bootable linux on the gumstix however the screen will not work without configuration but the linux can be accessed through the serial communication.

3 Compiling a custom kernel on the Gumstix

This section assumes that a Linux distribution is already up and running on the Gumstix and that a serial communication through the TTY connection is established.

3.1 Retrieving the kernel source code

The easiest way to retrieve the last version with the already applied patches for the gumstix is to use there GIT repository to checkout the files.
Open the file **linux-omap3_2.6.36.bb** from the OpenEmbedded files and look for SRC_URI the address next to it. During the project development the URI was git://www.sakoman.com/git/linux-omap-2.6.git therefore open a Terminal clone the repository.

```
$ git clone -n git://www.sakoman.com/git/linux-omap-2.6.git linux-omap
$ cd linux-omap
$ git branch -a
 ...
$ git checkout -b omap-2.6.36 origin/omap-2.6.36
```

Next copy the OpenEmbedded kernel config in the linux kernel source directory.

```
$ cp ~/overo-oe/org.openembedded.dev/recipes/linux/linux-omap3-2.6.36/defconfig .config
```

Copy this files into a removeable media and connect it to the gumstix.

3.2 Configuring the Kernel

The Linux kernel have to be configured before it can be compiled

```
$  make menuconfig
```

This will run the Linux kernel configuration menu, for this board to work with the USB screen the displaylink kernel have to be added as modules. Other options such as the CAN support have to be enabled.

To compile the kernel with the DisplayLink drivers for the screen

```
Device Drivers > Staging Drivers > Untick Exclude Staging drivers from being built
Device Drivers > Staging Drivers > Press M on Displaylink USB Framebuffer support
```

To compile the kernel with the CAN support

```
Networking Support > press M on CAN bus subsystem support
Netowrking Support > CAN bus subsystem support > press M on Raw CAN Protocol
Netowrking Support > CAN bus subsystem support > press M on Broadcast Manager CAN Protocol
```

To add the support for the push buttons two files have to be modified, the u-boot (linux bootloader) multiplexing file to select the function of the microcontroller pins and the board configuration in the kernel.

U-Boot Modifications: On the OpenEmbedded system open /overo-oe/tmp/work/overo-angstrom-linux-gnueabi/u-boot-omap3-(version+git-commit)/git/board/overo/overo.h and change the pin definition. The different options for each pin are defined OMAP35X Technical Reference Manual in Section 7.4.4.3

Kernel Modifications: Open **arch/arm/mach-omap2/board-overo.c** with a text editor and search for the definition of gpio_buttons[] an entry such as the following should be added to the array for a button to work. A pin can have only one function at a time so make sure that the pin is not already assigned.

```
{
        .code                   = KEY_UP,       // Send a KEY_UP event
        .gpio                   = 19,           // Use pin 19
        .active_low             = 1,            // The button is active low
        .desc                   = "key up",
        .wakeup                 = 1,            // Allow this button to wake from sleep
        .decounce_interval      = 20,           // 20ms Decounce delay
},
```