AN3012
Application note

Getting started with uClinux™ for STM32F10x high-density devices

Introduction

uClinux, pronounced "you-see-Linux", literally means "microcontroller (µC) Linux®".
uClinux is a Linux kernel fork for microcontrollers (MCUs, embedded systems). It does not have a memory management unit (MMU). Originally derived from version 2.0 (1996) of the Linux kernel, it now has ports based on Linux 2.6. Since version 2.6, the major parts of uClinux have been integrated with the mainline kernel for a number of processor architectures. The project continues to develop patches and supporting tools to use Linux on microcontrollers. uClinux supports many architectures, and this new version supports the STM3210E-EVAL evaluation board.

The purpose of this application note is to explain you how to:

1. Install the uClinux operating system, the toolchain, and configure the kernel for the STM3210E-EVAL board
2. Build a uClinux image and download it to the STM3210E-EVAL board
3. Add applications to the kernel

Contents

List of tables

List of figures

1 Package description

This section presents all the files in the package, that are needed to get started with uClinux on STM32F101xC/D/E and STM32F103xC/D/E devices.

● *uClinux_on_stm32.patch.gz*: the uClinux patch, using which the ARM Cortex-M3 CPU and STM3210E-EVAL board are supported.

● *AN3012.pdf*: the purpose of this guide is to show all the steps necessary to successfully configure, build and run uClinux on the board. It also presents the procedure to add new hardware driver and applications to the kernel.

● *Tiny_kernel_boot_loader.hex / kernel_boot_loader.hex*: these are the boot loaders required to boot up the board and load uClinux (kernel loader / firmware updater).

● *uClinux_on_stm32-jffs.dfu / uClinux_on_stm32-MCU_Flash.dfu*: these are the binary image files that can immediately be downloaded to the board to get uClinux running.
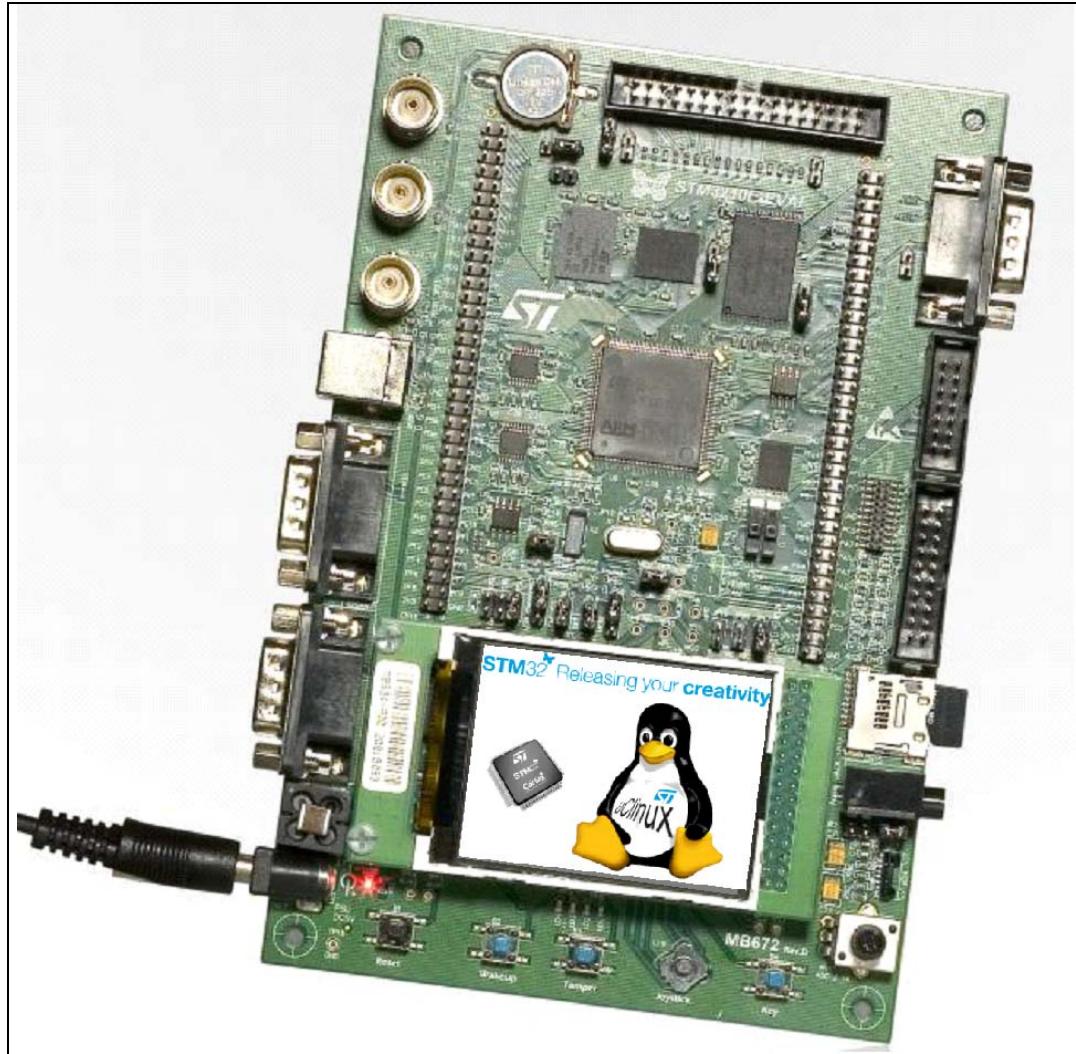
2   Hardware description

The STM3210E-EVAL evaluation board is designed as a complete development platform for STMicroelectronics's ARM™ Cortex®-M3 core-based STM32F103xx microcontrollers delivered in 144-pin packages. This device features: full speed USB2.0, CAN2.0A/B compliant interface, two I^2S channels, two I^2C channels, five USART channels with Smartcard support, three SPI channels, two DAC channels, FSMC interface, SDIO, 64 Kbyte of internal SRAM and 512 Kbyte of Flash memory, JTAG and SWD debug support.

The STM3210E-EVAL has the following onboard hardware features:

● Three 5 V power supply options: power jack, USB connector or daughterboard
● Boot from user Flash memory, system memory or SRAM
● I2S audio DAC, stereo audio jack
● 128 Mbyte MicroSD card
● A- and B-type Smartcard support
● 64 or 128 Mbit serial Flash memory, 512 Kbit × 16 SRAM, 512 Mbit or 1 Gbit NAND Flash and 128 Mbit NOR Flash memory
● I2C/SMBus compatible serial interface temperature sensor
● Two RS-232 channels with RTS/CTS handshake support on one channel
● IrDA transceiver
● USB2.0 full speed connection
● CAN2.0A/B compliant connection
● Inductor motor control connector
● JTAG and trace debug support
● 240 × 320 TFT color LCD
● Joystick with 4-direction control and selector
● Reset, wakeup, tamper and user buttons
● 4 color LEDs
● RTC with backup battery

Figure 1.   STM3210E-EVAL board

3 Getting the tools

This section explains how to get the software tools that are required to build and run uClinux on the STM3210E-EVAL evaluation board.

3.1 uClinux package

3.1.1 Standard distribution

You should first download the original uClinux source files from the uClinux project page (*http://www.uclinux.org/*), or simply follow the following direct link: *http://www.uclinux.org/pub/uClinux/dist/uClinux-dist-20080808.tar.bz2*

To get the *uClinux-dist-20080808.tar.bz2* file (281Mbytes), which is compatible with the STM3210E-EVAL uClinux kit.

Porting is based on the latest stable version of uClinux which implements the Linux kernel 2.6.26-uc0.

You also need the uClinux distribution update patch file: *uClinux-dist-20080808-20090112.patch.gz* (47.1 Mbytes) available from: *http://www.uclinux.org/pub/uClinux/dist/patches/uClinux-dist-20080808-20090112.patch.gz*

3.1.2 STM3210E-EVAL uClinux software kit

This package contains the kernel patch (update of 399 files) and the kernel boot loader (see *Section 1: Package description*). The package is available from http://*www.st.com/stm32*.

3.2 GNU toolchain

A toolchain known to successfully build the kernel for ARM Cortex-M3 targets can be downloaded from the CodeSourcery web site *http://www.codesourcery.com/*, or using the following link: *http://www.codesourcery.com/sgpp/lite/arm/portal/release827*

The G++ Lite 2009q1 toolchain is a free version of the CodeSourcery G++ toolchain, which is an improvement of the GNU toolchain for ARM processors. It supports ARM, thumb and thumb-2 compilation for all architectures, including Version 7 of the ARM Architecture.

Note: *1*   *This application note only shows how to Install the "easy-to-install recommended packages".*

          2   *You need a PC running on the Linux operating system to be able to install and compile uClinux. The Linux distribution (Fedora, Mandriva, Ubuntu, etc.) should have the kernel development kit.*

3.3 DFuSe & Flash loader demonstrator

These two tools are needed to load the final binary images to the board.

3.3.1 DFuSe applet

DFuSe stands for Device firmware upgrade application. This applet, coupled to the kernel boot loader firmware, allows the upgrade of the final *.dfu* file to the on-board Flash memory.

It can be downloaded from the STMicroelectronics website or directly, using the following link: *http://www.st.com/stonline/products/support/micro/files/um0412.zip*.

3.3.2 Flash loader demonstrator

This applet, together with the System memory boot loader capabilities, is used to upgrade *kernel_boot_loader.hex* to the STM32F101/103xC/D/E's 512-Kbyte internal Flash memory.

The Flash loader demonstrator is available from st.com at: *http://www.st.com/stonline/products/support/micro/files/um0462.zip*

Note: *You need a PC running on the Microsoft® Windows® operating system to be able to run and use these two applets.*

4   Building the uClinux image
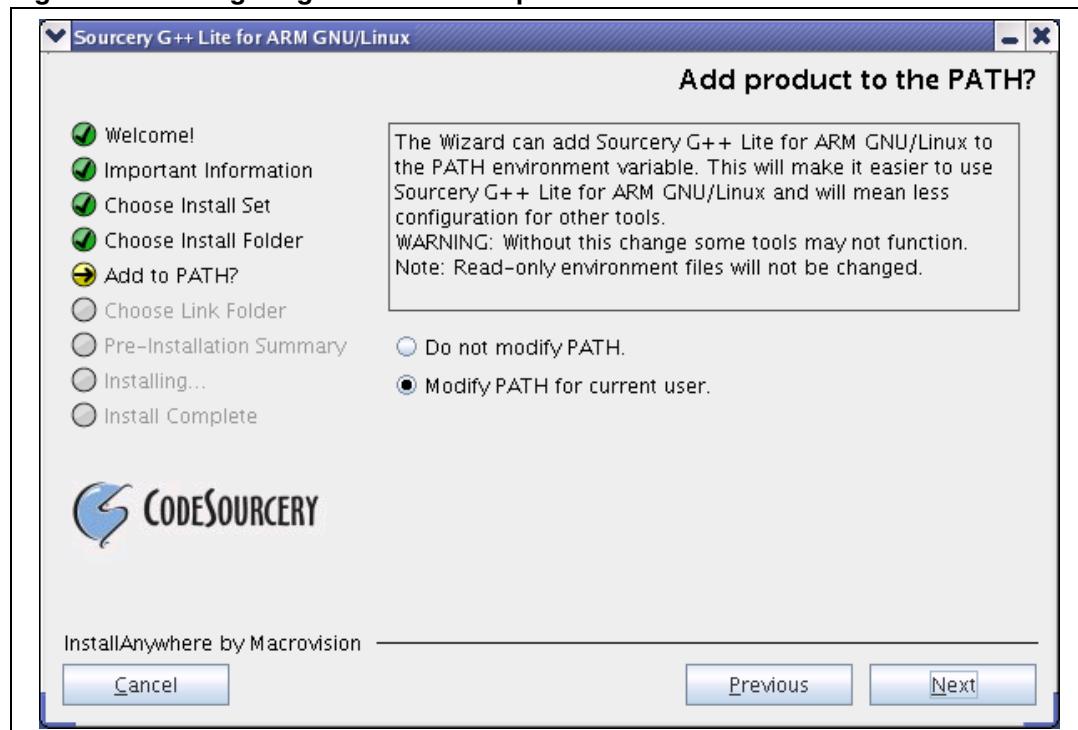
This section shows the kernel building process from scratch; to successfully build the uClinux image, you have to Install *GNU toolchain*, extract the uClinux source files, apply the path to the source, configure uClinux for the STM3210E-EVAL board and finally run the build process.

4.1   Installing the toolchain

On a host PC running with the Linux OS:

1.   Move to the directory containing the previously downloaded *arm-2009q1-163-arm-uclinuxeabi.bin* file.
2.   Execute the installer by double clicking on the bin file or from the command line by typing:

    #./arm-2009q1-163-arm-uclinuxeabi.bin

3.   Follow the installer Wizard to install the typical toolchain settings.

Figure 2.   Configuring the installation parameters

Note: *When no windowing server is running you can use the console installation mode, which is enabled with the "-i console" parameter:*

    # ./arm-2009q1-163-arm-uclinuxeabi.bin -i console

4.2 Preparing the source code tree

Let us assume that all the downloaded files are present under the **~/sources** directory.

Make a new workspace directory, where the build process will take place.

```
# mkdir ~/workspace
```
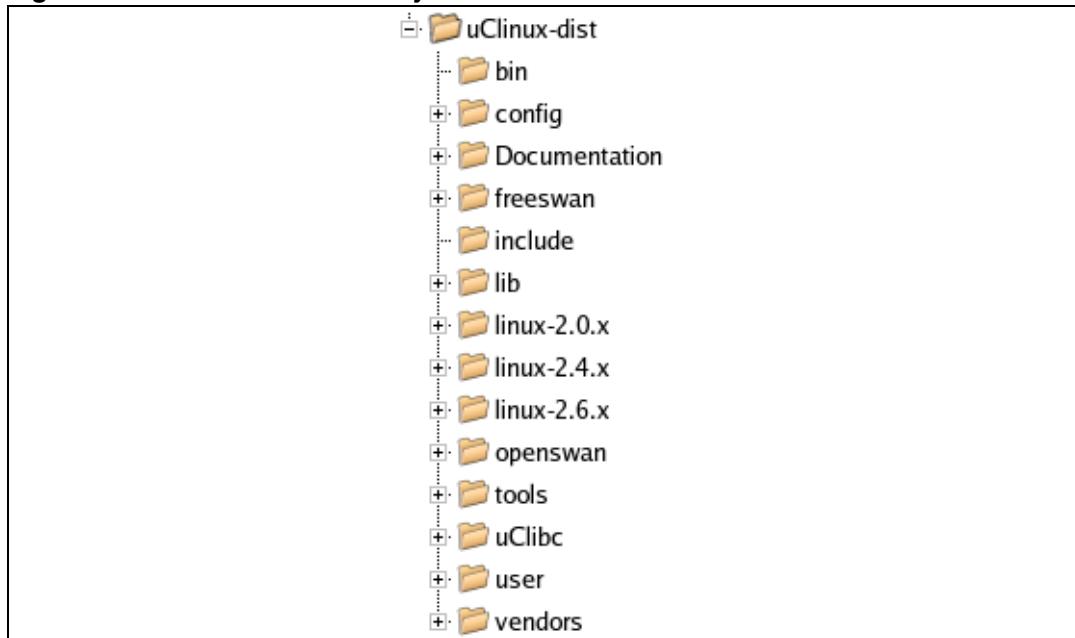
1. Extract the original uClinux distribution files: make sure you have enough free space (more than 2 Gbytes).
   ```
   # cd ~/workspace
   # tar -xjvf ~/sources/uClinux-dist-20080808.tar.bz2
   ```

   Figure 3 shows what is under the **uClinux-dist** directory.

Figure 3.   uClinux-dist directory.
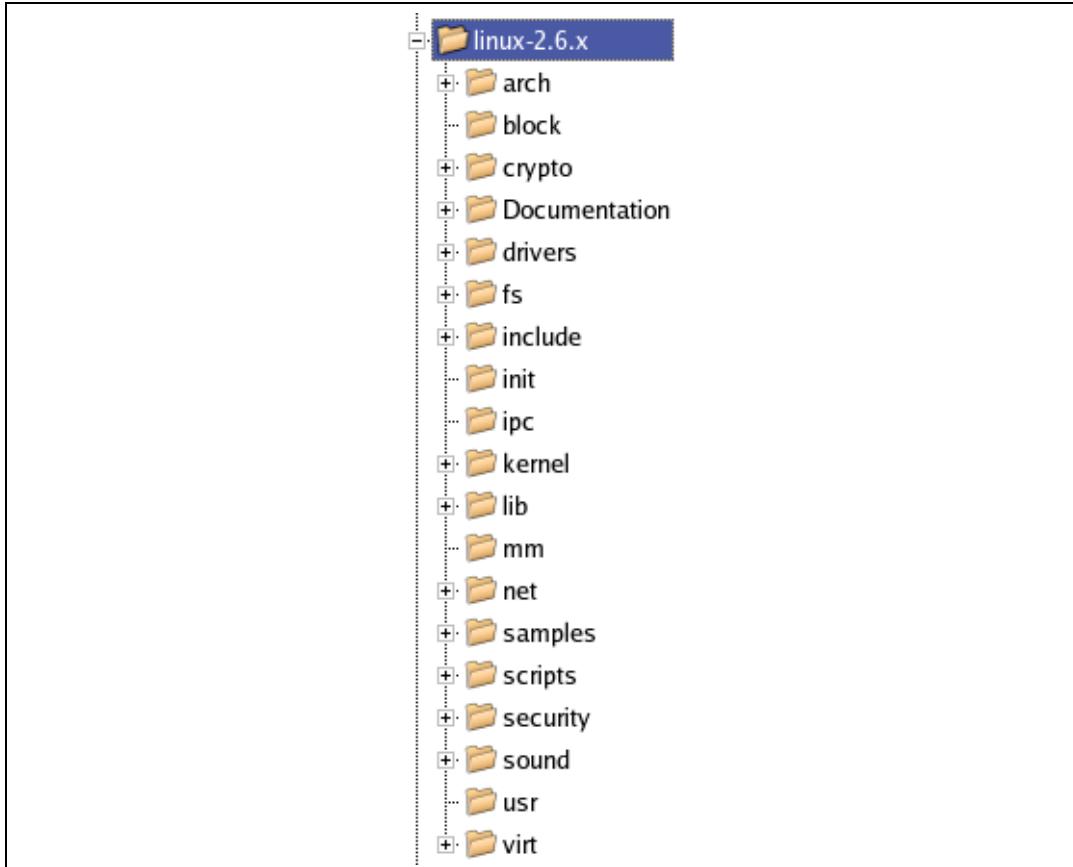
2. Update the uClinux distribution with the *uClinux-dist-20080808-20090112.patch.gz* patch:

   ```
   # cd ~/workspace/uClinux-dist/
   ```

   ```
   # zcat ~/sources/uClinux-dist-20080808-20090112.patch.gz | patch -p1
   ```

3. To enable support of STM32 MCUs, patch the uClinux distribution with *uClinux_on_stm32.patch.gz* from the STM3210E-EVAL uClinux kit:

   ```
   # zcat ~/sources/uClinux_on_stm32.patch.gz | patch -p1
   ```

Figure 4.   uClinux kernel directory.
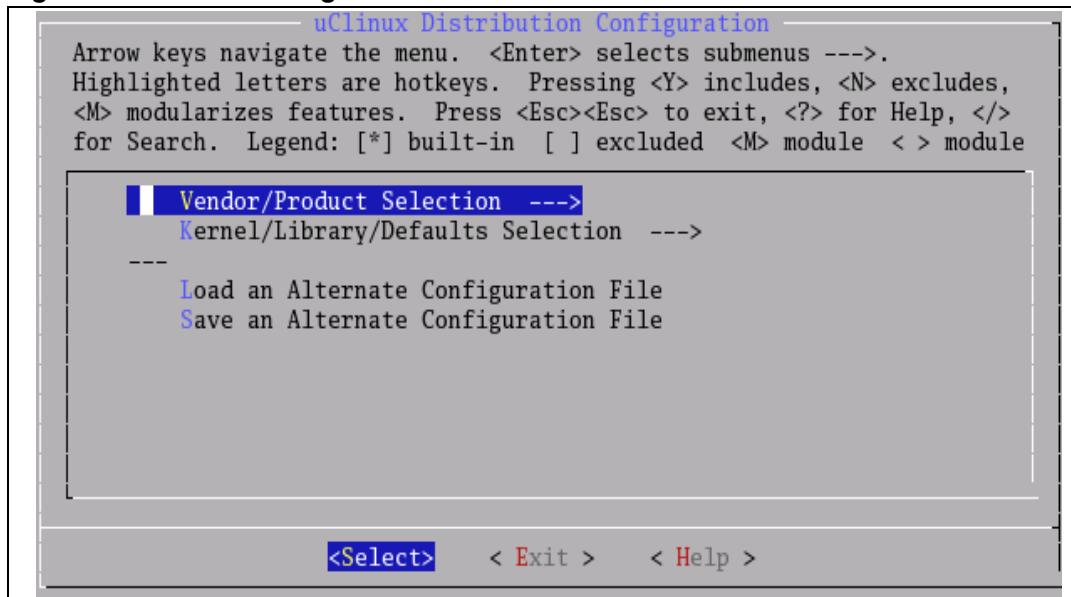
4.3   Configure uClinux for STM3210E-EVAL

Now that you have enabled support of the STM3210E-EVAL evaluation board by uClinux, you need to set up the configuration for the kernel and the final application. Many types of kernel binary images can be built depending on how you configure uClinux.

In this section you will see how to configure the uClinux kernel to run from the STM32's internal Flash memory or the external NOR Flash memory.

```
# make menuconfig
```
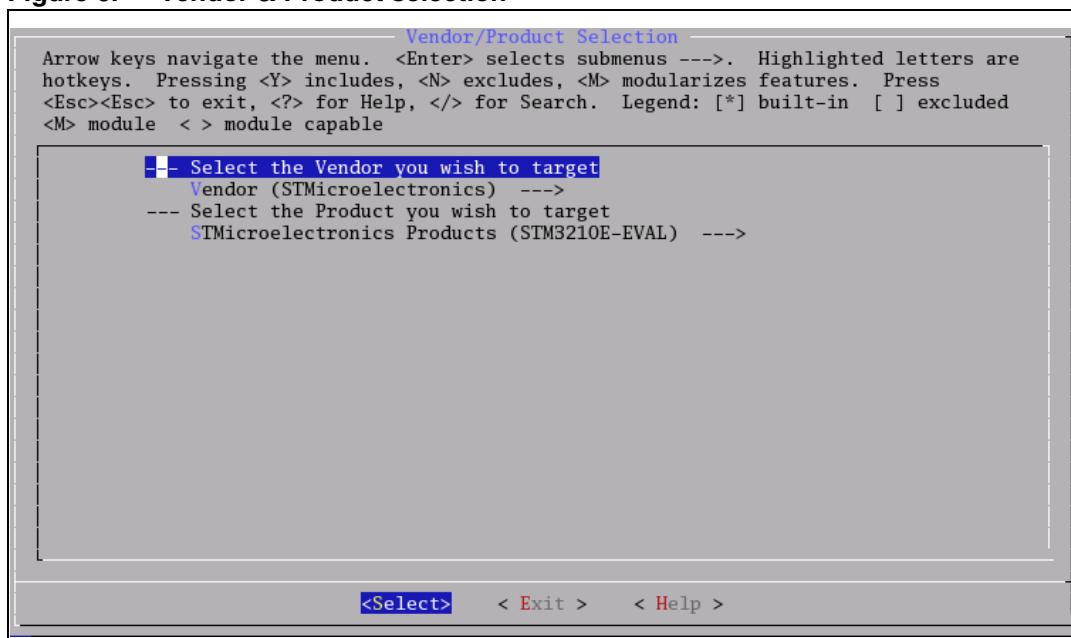
From the main menu, select the **vendor / product** as shown in *Figure 5*.

Figure 5.   uClinux configuration main menu

Select "STMicroelectronics" for "Vendor" and, depending on the type of kernel image you want to build, select "STM3210E-EVAL-jffs" or "STM3210E-EVAL-MCU_Flash" for "Product".

By selecting "STM3210E-EVAL-jffs" you apply the default configuration that enables support of an external Flash memory, and boot from it.

If you choose the "STM3210E-EVAL-MCU_Flash" configuration, the compilation script will build a mini image that runs from the microcontroller's internal Flash memory.
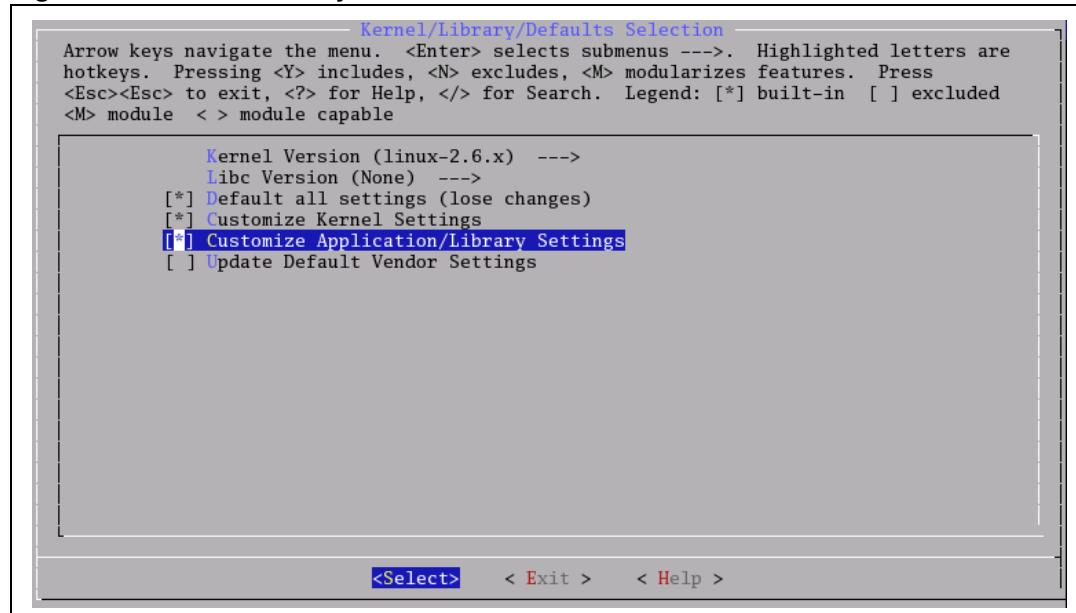
Figure 6.   Vendor & Product selection

Return to the main menu (with the "Exit" button or the "ESC" key).

Under the **Kernel / Library / Defaults Selection** menu, choose *Linux-2.6.x* for the "**kernel version**" and "*none"* for the **"libc version"** and toggle the next three menus as shown in *Figure 7*:

● "**Default all settings**" will apply the default vendor/product configuration to the kernel and application.

● "**Customize Kernel Settings**" is used to edit the kernel configuration.

● "**Customize Application/Library Settings**" is used to choose the application that will be add to the root file system.

● Finally "**Update Default Vendor Settings**" is used to save the new configuration as the vendor default configuration for future use.

Figure 7.   Kernel / Library / Defaults selection
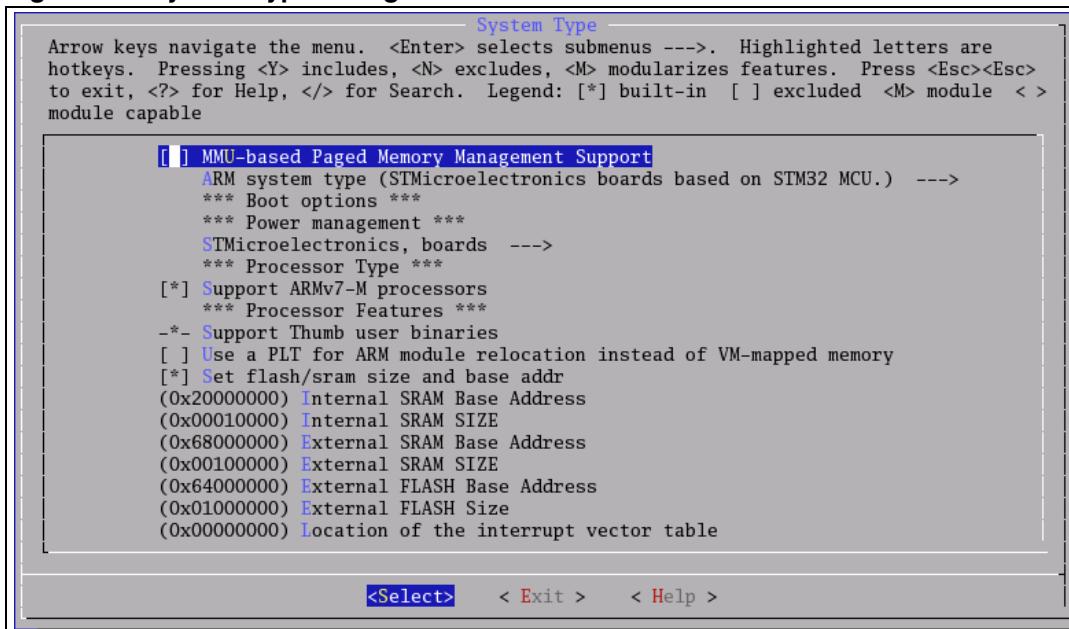
You can now exit the configuration menu. When prompted to save, choose "yes".

4.3.1   Configuring the kernel

The Linux kernel has many configuration items. If you have followed the above described steps, the kernel should be configured to meet the STM3210E-EVAL evaluation board specifications. Let us now look into some of the main configuration menus.

● The System Type menu (see *Figure 8*) shows the MMU setting, ARM System type, processor type, Flash and SRAM settings. All these options should coincide with the board specifications.

Figure 8.   System Type configuration

```
┌──────────────────────────── System Type ────────────────────────────┐
│ Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are │
│ hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> │
│ to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > │
│ module capable │
│ ┌──────────────────────────────────────────────────────────────────┐ │
│ │        [ ] MMU-based Paged Memory Management Support              │ │
│ │            ARM system type (STMicroelectronics boards based on STM32 MCU.)  ---> │ │
│ │            *** Boot options ***                                   │ │
│ │            *** Power management ***                               │ │
│ │            STMicroelectronics, boards  --->                       │ │
│ │            *** Processor Type ***                                 │ │
│ │        [*] Support ARMv7-M processors                             │ │
│ │            *** Processor Features ***                             │ │
│ │        -*- Support Thumb user binaries                            │ │
│ │        [ ] Use a PLT for ARM module relocation instead of VM-mapped memory │ │
│ │        [*] Set flash/sram size and base addr                      │ │
│ │        (0x20000000) Internal SRAM Base Address                    │ │
│ │        (0x00010000) Internal SRAM SIZE                            │ │
│ │        (0x68000000) External SRAM Base Address                    │ │
│ │        (0x00100000) External SRAM SIZE                            │ │
│ │        (0x64000000) External FLASH Base Address                   │ │
│ │        (0x01000000) External FLASH Size                           │ │
│ │        (0x00000000) Location of the interrupt vector table        │ │
│ └──────────────────────────────────────────────────────────────────┘ │
│                                                                      │
│              <Select>      < Exit >    < Help >                      │
└──────────────────────────────────────────────────────────────────────┘
```

●   The boot options menu is used to enable the XIP (execute in place) mode and set the physical location of the kernel xipImage. You can also specify the default kernel command line. The "XIP Kernel Physical Location" parameter points to the Flash memory address where the kernel image is located.
    For example, if "STM3210E-EVAL-jffs" was selected, the kernel binary will be written to the external Flash memory address 0x6400 0000 set in the configuration menu (as described in *Section 5.2.2: Generating the .dfu image and upgrading the board*).
    If "STM3210E-EVAL-MCU_Flash" was chosen, the XIP setting will point to 0x0800 3000.

    [*] Kernel Execute-In-Place from ROM

    (0x6400 0000) **XIP Kernel Physical Location**

    or

    (0x0800 3000) **XIP Kernel Physical Location**

●   For console input/output, enter "**Device Drivers**" -> "**Character Device**" -> "**Serial Drivers**". Set the two following options:

    [*] STM3210E-EVAL USART Port

    [*] Support for console on STM3210E-EVAL USART Port

●   Many file systems are supported by the uClinux kernel and can be enabled from the "**File systems**" menu. You can determine which file system will be supported according to your product memory size. For example if you need the jffs2 file system you have to enable it.

    In the case of the STM3210E-EVAL-jffs configuration, this file system is mandatory because it is used as the root file system.

    Under the "**File systems**" -> "**Miscellaneous File system**" submenu, toggle JFFS2 fs (with fs standing for file system).

    [*] Journalling Flash File System v2 (JFFS2) support

4.3.2 Application/library settings

In this menu, leave the vendor default settings. In "**Miscellaneous Applications**" -> "**STM3210E-EVAL Demo(s)**", you will find a few applications examples exploiting the STM32 hardware and the uClinux features and drivers.

Applications under development will also appear there.

Figure 9.   STM3210-EVAL demonstration menu

1.   If you use the STM3210E-EVAL-MCU_Flash configuration, the Show Logo application will not be selected because the LCD is not initialized during the boot process hardware initialization.

When exiting the configuration menu, save the settings.

4.4   Running the build process

You are now ready to start building the uClinux system. Depending on the previously selected configuration ("STM3210E-EVAL-jffs" or "STM3210E-EVAL-MCU_Flash"), you will use one of the two following processes to build the kernel image:

●   If you chose the STM3210E-EVAL-jffs configuration:

Simply type the "**make**" command and wait the end of the build process. At this step, the kernel and the selected application are built.

After this operation, the *xipImage.bin*, *rootfs.img.bin* and *logo.bin* files are generated and copied to the "**uClinux-dist/images/**" folder.

●   If you chose the STM3210E-EVAL-MCU_Flash configuration:

To build the minimal image, proceed as follows:

a)   Run the "**make ucfront**" command to prepare tools to build user applications.

b)   Run the "**make user_only**" command to prepare user applications.

c)   Run the "**make romfs**" command to build the **initramfs** folder tree.

d)   Run the "**make image**" command to copy the kernel image to the "**uClinux-dist/images/**" folder.

At the end of this step a single *xipImage.bin* file is generated.

Whatever the configuration you used ("STM3210E-EVAL-jffs" or "STM3210E-EVAL-MCU_Flash"), now all you need to do is copy the generated image(s) and jump to the next step: *Loading the images to the board*.
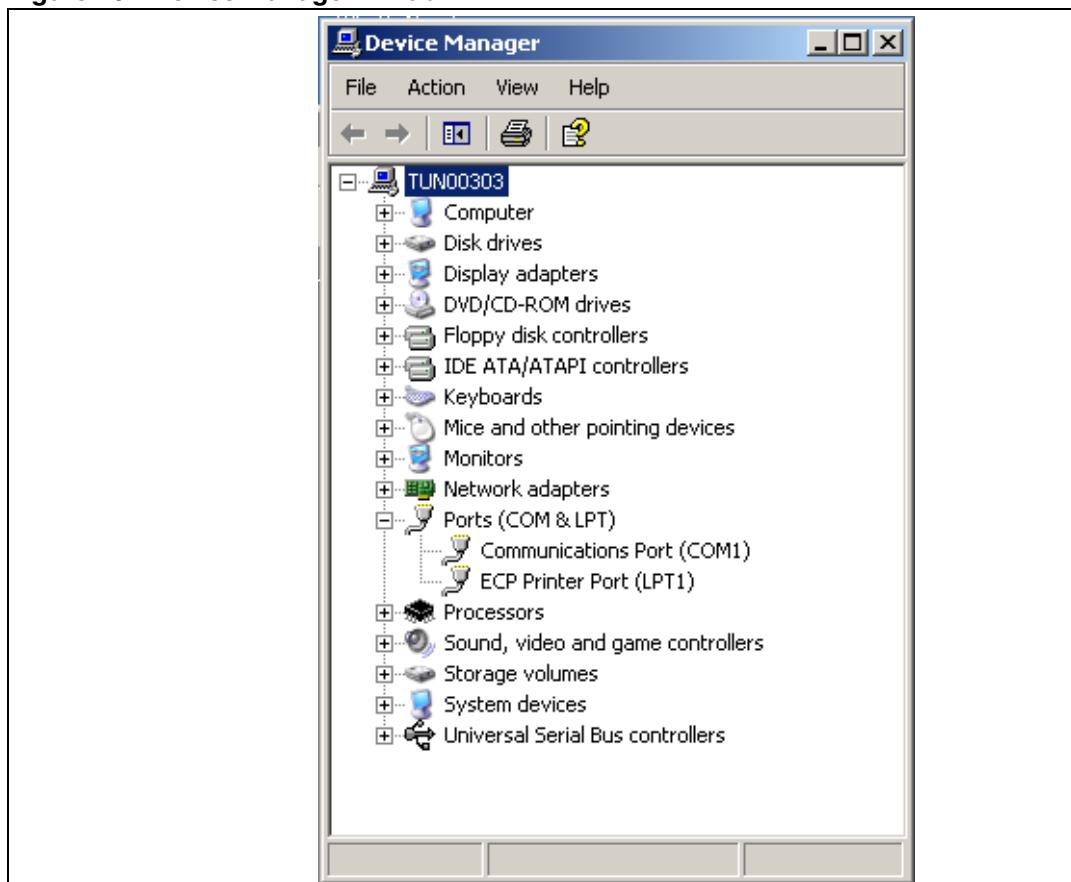
5 Loading the images to the board

5.1 Loading the "kernel loader / software updater"

On a host PC running with a recent version of Microsoft Windows, you need to verify that you have an available COM port to communicate with the board.

To check that you have an available communication port, right-click on the "My Computer" icon on the desktop and select "Properties" from the pop-up menu. The "System Properties" dialog box appears. Click on the "Hardware" tab, and then on the "Device manager" button to display the system hardware configuration. Available COM ports are grouped under the "Ports (COM & LPT)" node in the hardware tree as shown in *Figure 10*.

Figure 10. Device Manager window

5.1.1 Software installation

Run the *Flash_Loader_Demonstrator_VX.Y_Setup.exe* file: the InstallShield Wizard will guide you through the installation of the Flash loader demonstrator application on your computer. Once the software has been successfully installed, click on the "Finish" button.

5.1.2 Hardware installation

Connect the device to a spare COM port on your PC.

5.1.3 Load the hex file to the Flash memory
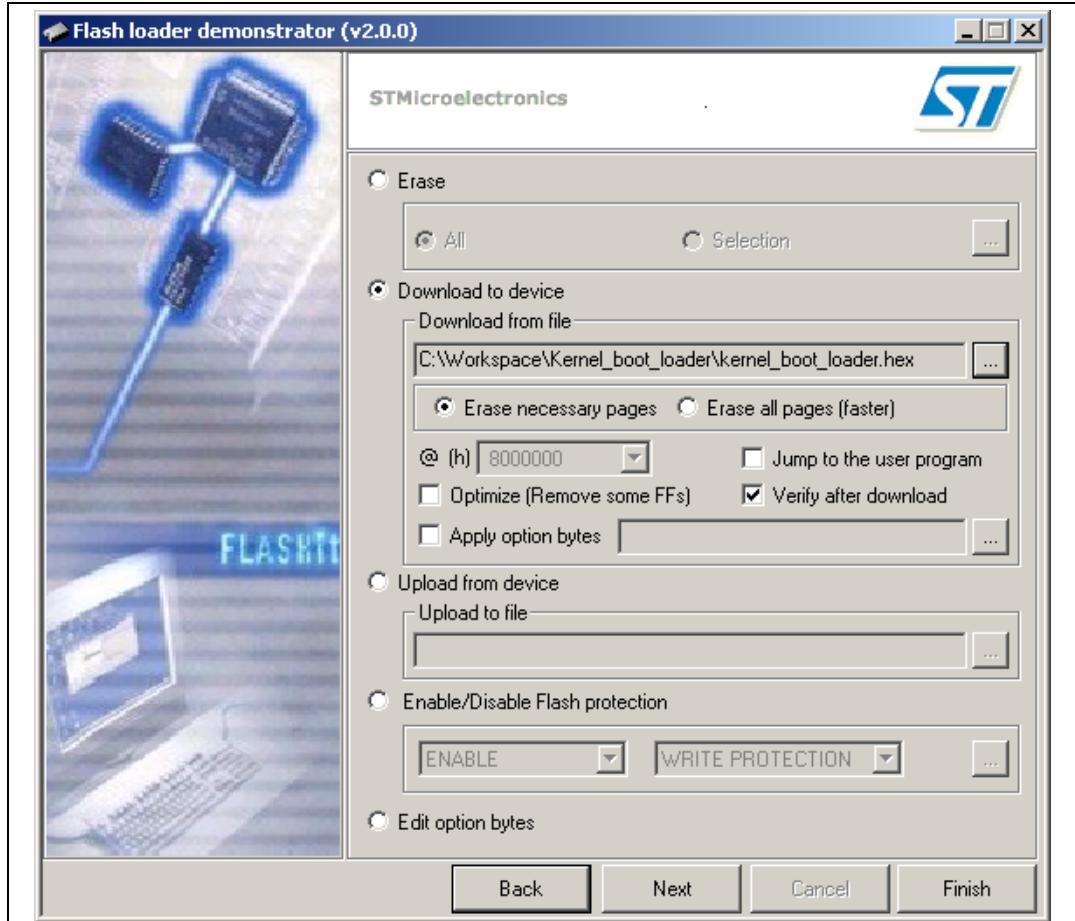
Before loading the file, move the boot jumpers to "System Memory Mode" then power up the board. Run the Flash loader demonstrator application from the "Programs" menu.

1. The first step consists in selecting the connection settings: UART port, baud rate and timeout.

2. In the second step, the connection is established and communication has started. This step consists in displaying the Flash memory status. If this status is read-protected, the "Next" button is disabled until the read protection is removed by clicking on the "Remove protection" button. Removing the read protection causes all the Flash memory pages to be erased.

3. At this step the Wizard displays the available device information such as the target ID, the firmware version, the supported device, the memory map and the memory protection status. Select the target from the target combobox and click on "Next".

4. At this step, select the Download operation and set the related parameters as shown in *Figure 11*. "**Download from file**" should point to the *Kernel boot loader* file from the STM3210E-EVAL uClinux kit.

   Depending on the board configuration selected in *Section 4.3: Configure uClinux for STM3210E-EVAL* when booting from the STM32's internal Flash memory, you require a tiny boot loader to preserve some Flash pages for the kernel. In the case when the kernel is placed in an external Flash memory you can use a standard boot loader.

   – Use *Tiny_kernel_boot_loader.hex* (< 12 Kbytes) when booting from the STM32's Flash memory (with the "STM3210E-EVAL-MCU_Flash" configuration).

   – Use *Kernel_boot_loader.hex* when booting from the NOR Flash memory (with the *STM3210E-EVAL-jffs* configuration).

Figure 11. Flash loader demonstrator download configuration

5. The last Wizard page shows the operation page. It gives the size of the data to be downloaded, the percent completed and the duration of the operation.

Note: *If an error message appears, you can consult the Flash Loader demonstrator user manual available from the "Programs" menu.*

5.2 Loading the kernel/root file system images

Move the boot jumpers to select the "User Flash" boot mode, set the JP14 board jumper to connected mode and reset the board.

5.2.1 DfuSe demonstration installation

Software Installation

Run the *DfuSe_Demo_VX.Y.Z_Setup.exe* file: the InstallShield Wizard will guide you to install DfuSe applications and source code on your computer. When the software is successfully installed, click on the "Finish" button. You can then explore the driver directory. The driver files are in the "Driver" folder located at your install path (C:\Program files\STMicroelectronics\DfuSe).

Hardware installation

● Connect the device to a spare USB port on your PC.

● The "Found New Hardware Wizard" then starts. Follow the wizard to install the new hardware driver available in the "Driver" folder at your install path.

5.2.2 Generating the .dfu image and upgrading the board

Let us assume that the files generated from *Running the build process* have been copied to a local folder on the MS Windows host (e.g. C:\Workspace\Images).

DFU file manager
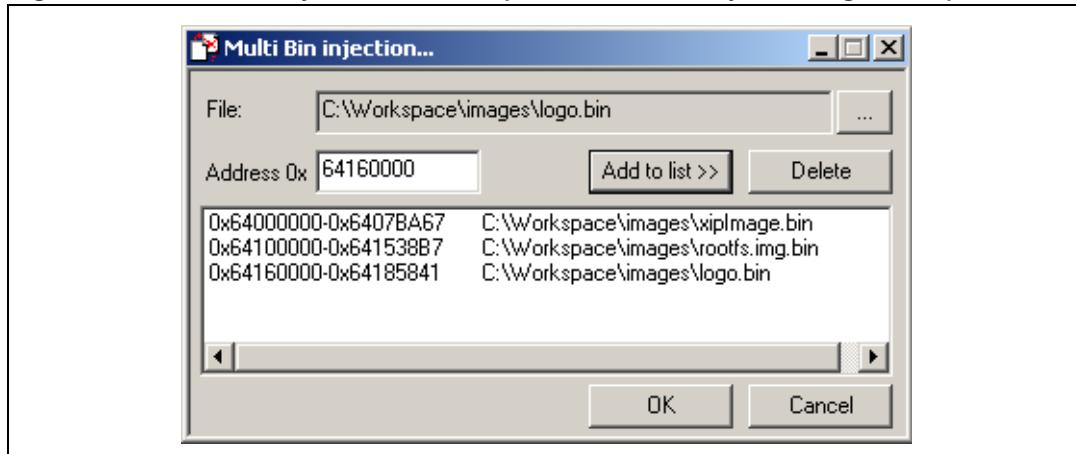
Start the DFU file manager application (Start -> All Programs -> STMicroelectronics -> DfuSe-> DFU File Manager), select "**i want to GENERATE a DFU file from S19, HEX or BIN files**" then click on "**OK**".

1. Set an unused **Target ID** number.

2. Fill the **VID**, **PID**, **Version** and **target name** fields

3. Click on the "**Multi Bin**" button to show the "**Multi Bin Injection**" dialog box.

   If you chose the "STM3210E-EVAL-jffs" configuration in *Section 4.3: Configure uClinux for STM3210E-EVAL* you have to:

   – Set the start address in the **Address** field to "6400 0000".

   – Click the **Browse** button to select the *xipImage.bin* file at C:\Workspace\Images\xipImage.bin.

   – Click on the **Add to list** button to add the selected binary file at the given address.

   – Redo the same sequence to add C:\Workspace\Images\rootfs.img.bin at address "6410 0000", and C:\Workspace\Images\logo.bin at address "6416 0000".

   – Click on "**OK**" to validate.
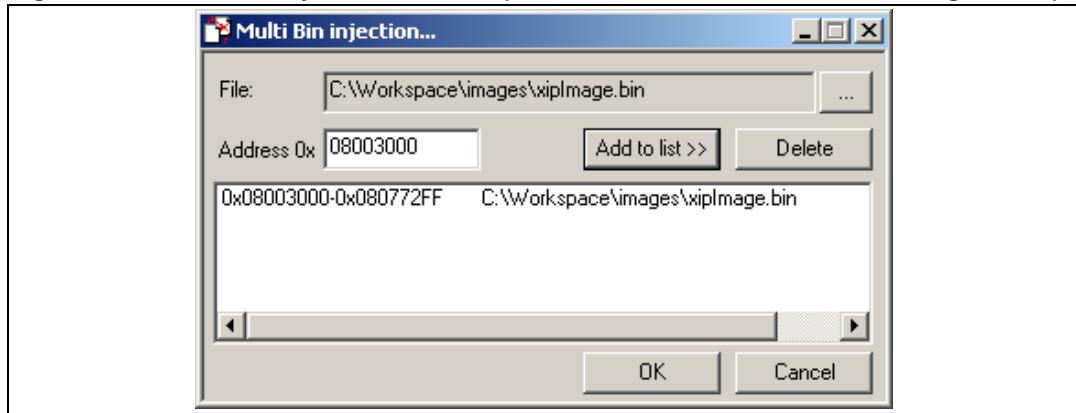
Figure 12. Multi Bin injection window (STM3210E-EVAL-jffs configuration)

If you chose the "STM3210E-EVAL-MCU_Flash" configuration in *Section 4.3: Configure uClinux for STM3210E-EVAL* you have to:

– Set the start address in the **Address** field to "0800 3000".
– Click the **Browse** button to select the *xipImage.bin* file at *C:\Workspace\Images\xipImage.bin*.
– Click on the **Add to list** button to add the selected binary file at the given address.
– Click on "**OK**" to validate.

Figure 13.   Multi Bin injection window (STM3210E-EVAL-MCU_Flash configuration)
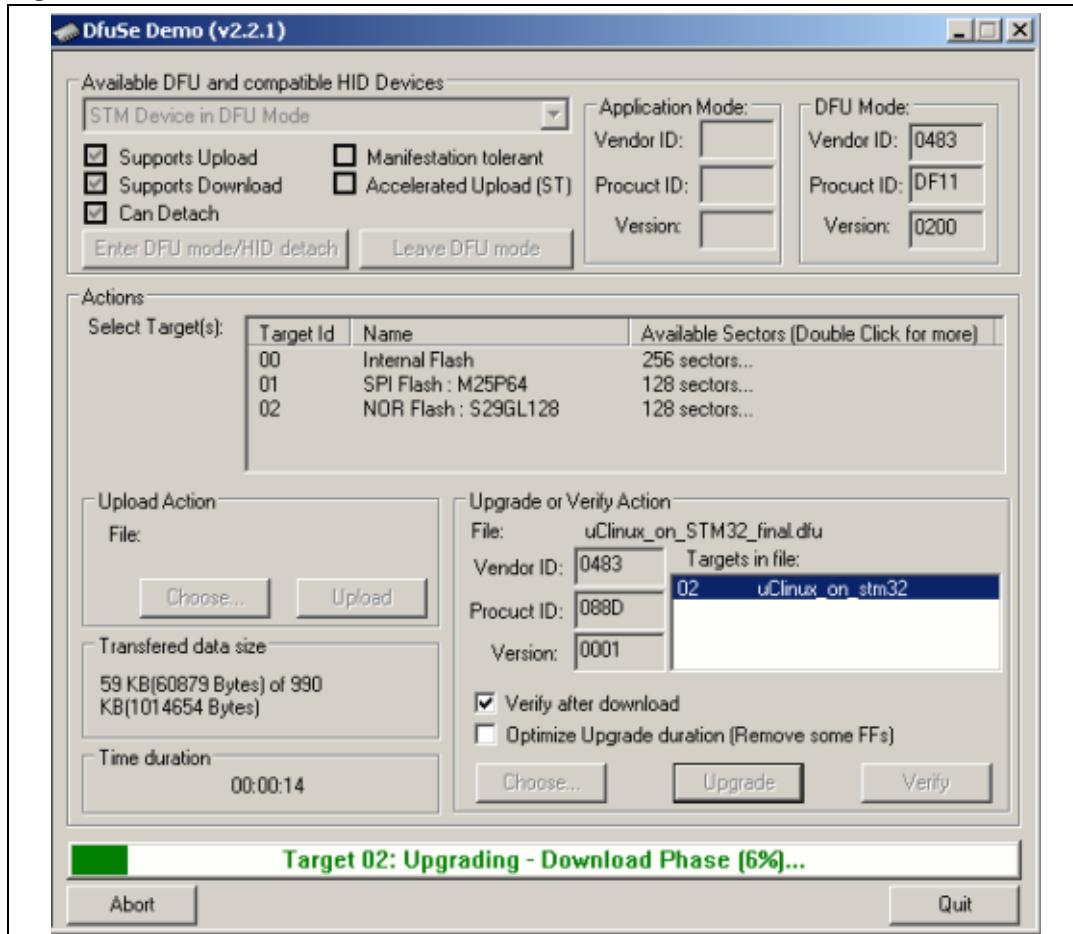
4.   To create the DFU file, click on "**Generate**".

How to download a DFU file

Run the "DfuSe demonstration" application (Start -> All Programs ->STMicroelectronics -> DfuSe -> DfuSe Demonstration).

1.   Click on the "**Choose**" button to select the previously generated DFU file. The displayed Information such as VID, PID, Version and target number is read from the DFU file.
2.   Choose "NOR Flash" or "internal Flash" in the "**Select target(s)**" area.
3.   Check the "**Verify after download**" checkbox if you want to launch the verification process after downloading data (*this step is optional*).
4.   Click on the "**Upgrade**" button to start upgrading the file content to the memory.
5.   Wait until the end of the upgrade process.

Figure 14. DfuSe demonstration window

6 First run

After completing the previous steps, you should be able to boot up uClinux on your board.

● Connect the board's USART1 to the PC COM port.

● Open a serial terminal on you PC (e.g.: MS HyperTherminal) and configure it as shown in *Figure 15*:

    – Bits per second: 19200

    – Data bits: 8

    – Parity: None

    – Stop bits: 1

    – Flow control: none

● Reset the board. You should then be able to see the boot log (see *Figure 16*) and finally, the shell prompt.

Figure 15. Example of MS HyperTherminal
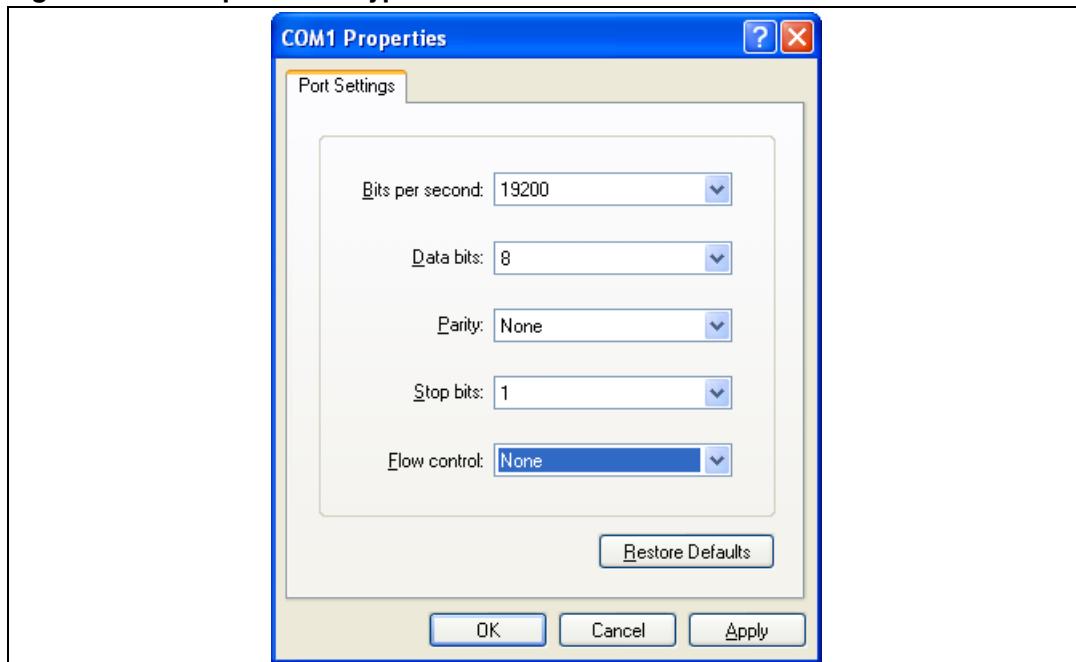
Figure 16. uClinux boot log

```
platform Initialisation finished jumping to kernel.
Linux version 2.6.26-uc0 (stm32@localhost.localdomain) (gcc
version 4.3.3 (Sourcery G++ Lite 2009q1-163) ) #1 Fri Jul 3
11:17:34 UTC 2009
CPU: ARMv7-M Processor [411fc231] revision 1 (ARMv?(11)M)
Machine: STM3210E-EVAL
SRAM Config: bank[0] @ 0x68000000 (size: 1024KB) - bank[1] @
0x20000000 (size: 64KB).
Built 1 zonelists in Zone order, mobility grouping off. Total
pages: 254
Kernel command line: noinitrd root=mtd1 ro rootfstype=jffs2
init=/linuxrc console=ttyS0
PID hash table entries: 16 (order: 4, 64 bytes)
console [ttyS0] enabled
Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory: 1MB 0MB = 1MB total
Memory: 980KB available (436K code, 55K data, 8K init)
Mount-cache hash table entries: 512
JFFS2 version 2.2. „¦. 2001-2006 Red Hat, Inc.
simple-gpio: now handling 16 GPIOs: 0 - 15
ttyS0 at MMIO 0x40013800 (irq = 37) is a STM32 USART1 Port
Probed and found the STM3210E-EVAL NOR flash chip
Creating 4 MTD partitions on "M29W128F NOR FLASH":
0x00000000-0x00100000 : "Kernel raw data"
0x00100000-0x00160000 : "rootfs"
0x00160000-0x00190000 : "rawdata"
0x00190000-0x001c0000 : "cramfs_partition"
rtc-stm3210e_eval rtc-stm3210e_eval.0: rtc core: registered rtc-
stm3210e_eval as rtc0
rtc-stm3210e_eval rtc-stm3210e_eval.0: setting system clock to
2009-07-03 12:00:26
VFS: Mounted root (jffs2 filesystem) readonly.
Freeing init memory: 8K
Mounting proc fs
Mounting sysfs
Welcome to
        ____ __ __
      /  __| |_|
 _ _ _|  |  | |  __ _ ___ \ ‾ _|  ‾ ‾\ \/ /
|  |_| | |__| | |  _| ___ _|  |_| |\ \/ \
|  ___ \____|_| |_| | | _| |\____|\_/\_/
|_|
For further information check:
http://www.uclinux.org/
http://www.st.com/stm32
/ #
```

7 Command examples/tutorials

1)

```
/ # ls -l
drwxr-xr-x    2 0           0            0 Jul  3  2009 bin
drwxr-xr-x    2 0           0            0 Jul  3  2009 dev
drwxr-xr-x    2 0           0            0 Jul  3  2009 etc
drwxr-xr-x    2 0           0            0 Jul  3  2009 home
drwxr-xr-x    2 0           0            0 Jul  3  2009 lib
-rwxr-xr-x    1 0           0          153 Jul  3  2009 linuxrc
drwxr-xr-x    2 0           0            0 Jul  3  2009 mnt
dr-xr-xr-x   19 0           0            0 Jan  1  1970 proc
drwxr-xr-x    2 0           0            0 Jul  3  2009 root
drwxr-xr-x    2 0           0            0 Jul  3  2009 sbin
drwxr-xr-x    9 0           0            0 Jan  1  1970 sys
drwxr-xr-x    3 0           0            0 Jul  3  2009 usr
```

2)

```
/ # cat /proc/meminfo
MemTotal:          988 kB
MemFree:           272 kB
Buffers:             0 kB
Cached:            128 kB
SwapCached:          0 kB
Active:            104 kB
Inactive:            8 kB
SwapTotal:           0 kB
SwapFree:            0 kB
Dirty:               0 kB
Writeback:           0 kB
AnonPages:           0 kB
Mapped:              0 kB
Slab:                0 kB
SReclaimable:        0 kB
SUnreclaim:          0 kB
PageTables:          0 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
WritebackTmp:        0 kB
CommitLimit:       492 kB
Committed_AS:        0 kB
VmallocTotal:        0 kB
VmallocUsed:         0 kB
VmallocChunk:        0 kB
```

3)

```
/ # ps
  PID USER        VSZ STAT COMMAND
    1 0          148 R    /bin/sh
    2 0            0 SW<  [kthreadd]
    3 0            0 SWN  [ksoftirqd/0]
    4 0            0 SW<  [events/0]
    5 0            0 SW<  [khelper]
    6 0            0 SW   [pdflush]
    7 0            0 SW   [pdflush]
    8 0            0 SW<  [kswapd0]
    9 0            0 SW<  [aio/0]
   15 0           32 S    Led_Show
   16 0          132 R    ps
```

4)

```
/# kill -ALRM 15
Pause signal received
/# kill -ALRM 15
Resume signal received
```

Using the Linux signal to interact with process 15: pauses and resumes the animation

5)
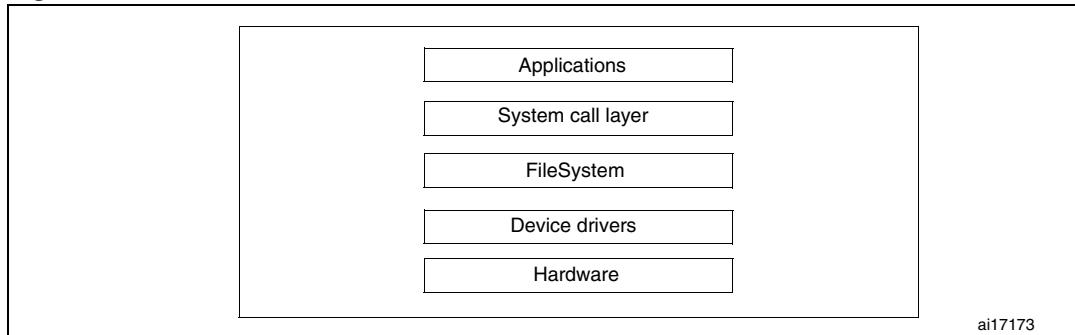
```
/ # kill -TERM 15: Ends the task with PID 15 (Led_Show application),
/ # ps
  PID USER     VSZ STAT COMMAND
    1 0        172 S    /bin/sh
    2 0          0 SW<  [kthreadd]
    3 0          0 SWN  [ksoftirqd/0]
    4 0          0 SW<  [events/0]
    5 0          0 SW<  [khelper]
    6 0          0 SW   [pdflush]
    7 0          0 SW   [pdflush]
    8 0          0 SW<  [kswapd0]
    9 0          0 SW<  [aio/0]
   18 0        132 R    ps
```

8 How to add new features

8.1 How to add a new hardware driver

The kernel device driver is the software support which allows the application to interact with the hardware via the kernel system call interface. It implements the "what capabilities are to be provided" mechanism and the "how these capabilities can be used" policy while respecting higher-layer specifications.

Figure 17. Linux device driver architecture

8.1.1 References

● A typical book is the third edition of "***Linux Device Drivers***":
   http://lwn.net/Kernel/LDD3/

● Also refer to the "**uClinux-dist/linux-2.6.x/Documentation/**" and "**uClinux-dist/linux-2.6.x/Documentation/driver-model/**" folders.

8.1.2 Example of a kernel driver

The purpose of this section is to present Linux device drivers, and to give two examples of drivers: the STM3210E-EVAL evaluation board's USART driver and the GPIO port driver.

USART driver

This section gives an overview of UART drivers.

First, you must know that serial drivers are not accessed directly by applications but through a software stack which exports the whole functionality of the driver (TTY Layer). You should therefore implement all the APIs needed by the higher layer, and register the driver.

The main structures needed for this purpose are:

● *struct uart_driver*: in this data struct you need to fill the name, major/minor numbers and the number of UARTs supported by the driver. The console file is only needed when you enable console Input/Output over the serial port.

Figure 18. STM3210E-EVAL uart_driver struct

```
static struct uart_driver stm3210e_eval_uart_driver = {
        .owner              = THIS_MODULE,
        .driver_name        = "STM32 USART DRIVER",
        .dev_name           = STM32_USART_NAME,
        .major              = STM32_USART_MAJOR,
        .minor              = STM32_USART_MINOR,
        .nr                 = STM32_NR_USARTS,
#ifdef CONFIG_SERIAL_STM3210E_EVAL_CONSOLE
        .cons               = &stm3210e_eval_console,
#endif
};
```

● *struct uart_port*: this struct exports the low-level configuration data of the hardware, like the memory base, IRQ number, I/O type, etc.

Figure 19. STM3210E-EVAL uart_port struct

```
static struct uart_port stm3210e_eval_usart1_port = {
        .membase = (u8*)USART1,
        .iotype = SERIAL_IO_MEM,
        .irq = USART1_IRQn,
        .fifosize = 0,
        .flags = UPF_BOOT_AUTOCONF,
        .line = 0,
        .ops = &stm3210e_eval_ops
        };
```

● *struct uart_ops*: the information stored in this data struct allows the higher layer to interact with the hardware. Most of the functions exported by this data struct are hardware dependent and should take care of hardware requirements.

Figure 20. STM3210E-EVAL uart_ops struct

```
static struct uart_ops stm3210e_eval_ops = {
        .tx_empty           = stm3210e_eval_tx_empty,
        .set_mctrl          = stm3210e_eval_set_mctrl,
        .get_mctrl          = stm3210e_eval_get_mctrl,
        .stop_tx            = stm3210e_eval_stop_tx,
        .start_tx           = stm3210e_eval_start_tx,
        .stop_rx            = stm3210e_eval_stop_rx,
        .enable_ms          = stm3210e_eval_enable_ms,
        .break_ctl          = stm3210e_eval_break_ctl,
        .startup            = stm3210e_eval_startup,
        .shutdown           = stm3210e_eval_shutdown,
        .set_termios        = stm3210e_eval_set_termios,
        .type               = stm3210e_eval_type,
        .release_port       = stm3210e_eval_release_port,
        .request_port       = stm3210e_eval_request_port,
        .config_port        = stm3210e_eval_config_port,
        .verify_port        = stm3210e_eval_verify_port
};
```

All these structures are declared and well commented in "***include/linux/serial_core.h***".

● *struct console*: in order to implement console support over the serial port, you have to fill this struct with the necessary data, like the name, the write function used to print the kernel message, various console flags, etc.

Figure 21. STM3210E-EVAL console struct

```
static struct console stm3210e_eval_console = {
        .name   = STM32_USART_NAME,
        .write  = stm3210e_eval_console_write,
        .device = uart_console_device,
        .setup  = stm3210e_eval_console_setup,
        .flags  = CON_PRINTBUFFER,
        .index  = -1, /* Specified on the cmdline (e.g. console=ttyST0 ) */
        .data   = &stm3210e_eval_uart_driver,
};
```

See the "***include/linux/console.h***" header file for more details.

Once you have implemented all the needed functions you have to call the:

● uart_register_driver: to bind the low-level driver with the SERIAL CORE
● uart_add_one_port: that lets the TTY layer know that a new device has been added

Note: 1   These two functions can be called from the module Init function.

● register_console: to insert the console in the list of active consoles. It can be exported with the console_initcall macro to be called during the initial stage of the kernel startup.

Figure 22. STM3210E-EVAL console Init function

```
static int __init stm3210e_eval_console_init(void)
{
        register_console(&stm3210e_eval_console);
        return 0;
}

console_initcall(stm3210e_eval_console_init);
```

The following configuration lines should now be added to the ***drivers/serial/Kconfig*** file to enable the driver selection from the kernel configuration menu.

```
config SERIAL_STM3210E_EVAL_USART
    bool "STM3210E-EVAL USART port."
    depends on CONFIG_ARCH_STM3210E_EVAL
    select SERIAL_CORE
    help
        This selects the STM3210E-EVAL USART port support.


config SERIAL_STM3210E_EVAL_CONSOLE
    bool "Support for console on STM3210E-EVAL USART port."
    depends on SERIAL_STM3210E_EVAL_USART=y
    select SERIAL_CORE_CONSOLE
    help
        This selects the console support on STM3210E-EVAL USART port.
```

Finally you need to add the following line to the ***drivers/serial/Makefile*** file, to ensure that the driver will be compiled when selected from the kernel configuration menu.

```
obj-$(CONFIG_SERIAL_STM3210E_EVAL_USART) += stm32_usart.o
```

Simple GPIO driver

This is another example of a character (char) device driver. It differs from the serial driver because it directly implements system calls on the device node (e.g. open, read, write, and so on).

The major struct is the file_operations data structure used to link the device node (major, minor numbers) to the driver. Each field in the structure points to a function in the driver that implements a specific operation.

Figure 23.   Simple GPIO file_operations struct

```
static struct file_operations simple_gpio_fops = {
        .owner      = THIS_MODULE,
        .read       = simple_gpio_read,
        .write      = simple_gpio_write,
        .open       = simple_gpio_open,
        .release    = simple_gpio_release,
        .ioctl      = simple_gpio_ioctl,
};
```

An example of implementation is the write operation. It consists in identifying the pin on which the changes will take effect via the device node minor number, then in parsing the buffer sent by the user application, and in applying the result to the hardware.

Figure 24. Simple GPIO write operation

```
static ssize_t
simple_gpio_write(struct file *file, const char __user *buf, size_t count, loff_t *pos)
{
        unsigned int gpio = iminor(file->f_path.dentry->d_inode);
        ssize_t ret;
        char dir = '?', uvalue = '?';
        int value;

        ret = 0;
        while (ret < count) {

        switch (uvalue) {
        case '?': value = gpio_get_value(gpio); break;
        case 'T': value = !gpio_get_output_value(gpio); break;
        default:  value = uvalue - '0'; break;
        }

        switch (dir) {
        case 'I': gpio_direction_input(gpio,value); break;
        case 'O': gpio_direction_output(gpio, value); break;
        }

        if (uvalue != '?')
                gpio_set_value(gpio, value);

        return ret;
}
```

Conclusion

An easy way of developing new device drivers is to avoid starting from scratch when possible, and to try and find some similar drivers in **linux-2.6.x/drivers** so that you can port them.

8.2 How to add a new application

To add a new application to the uClinux image:

1. Create a new folder named **myprog** in the **uClinux-dist/user/** directory for your application files.

2. Put your source and header files (***.c**, ***.h**) in the new directory: **uClinux-dist/user/myprog**.

3. Create a new **Makefile** for your application like the one shown below:

```
OBJS = file1.o file2.o
PROG = myexec
all: $(PROG)

$(PROG): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

romfs:
        $(ROMFSINST) /bin/$(PROG)

clean:
        -rm -f $(PROG) *.gdb *.o
```

4. Add a block to **uClinux-dist/user/Kconfig** under the STM3210E-EVAL Demo(s) menu as shown below:

```
menu "Miscellaneous Applications"


menu "STM3210E-EVAL Demo(s)"


config USER_myprog
     bool "my new application."
     help
          this is how to add a new application.
```

5. Add a line to **uClinux-dist/user/Makefile**, which adds **myprog** to the list of directories to be built:

```
dir_$(CONFIG_USER_myprog) += myprog
```

6. To build your application and to add it to the rootfs (root file system) image, run the following commands:

```
# make user_only

# make romfs

# make image
```

9   Revision history

Table 1.   Document revision history

Date	Revision	Changes
15-Sep-2009	1	Initial release.
29-Sep-2009	2	*Section 3.3.1: DFuSe applet* updated. Addresses corrected in *DFU file manager*. *Figure 12: Multi Bin injection window (STM3210E-EVAL-jffs configuration)* modified. Bits per second modified in *Section 6: First run*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2009 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com