UM2003
User manual

Getting started guide - STreamPlug ST2100 Linux support package

Introduction

This user manual provides application developers with a first introduction to the Linux based reference software installed in the Flash memory of the STreamPlug evaluation boards. It is not intended to be a tutorial on the Linux operating system or embedded software design. It only covers topics that are specific to the implementation on the STreamPlug SoC and boards.

The Linux support package (LSP) is a set of software provided free of charge by STMicroelectronics® for the STreamPlug SoC.

The LSP has the following objectives:

- Demonstrating capabilities of the STreamPlug device through a widely available high level operating system (Linux) executed as native or over the Microvisor (OKL4) provided by Open Kernel Labs (OKL), now General Dynamics Broadband. The LSP is only targeting STMicroelectronics evaluation boards.
- Providing a starting point for customers willing to accelerate a Linux porting to their proprietary STreamPlug based hardware platforms and products.

A detailed description of LSP contents can be found in the following companion document UM1942 - “Linux software user manual for STreamPlug ST2100”, while all the data about the system software and tools are in the UM2004 - “STreamPlug ST2100 SDK and quick start guide”.

Contents

1 About this manual

This manual focuses on software usage on STreamPlug development boards and is organized as a sequence of chapters going from a description of the first step and then covering more complex subjects.

This guide applies to all currently available STreamPlug evaluation boards. However, each different evaluation board provides, in general, a different selection and combination of hardware devices on the board (and companion boards, wherever applicable). For a detailed description of hardware features for each evaluation board, please refer to the corresponding evaluation board user manual.

2 Working with pre-flashed software

STreamPlug evaluation boards come with embedded software already stored in the (serial NOR) Flash memory, according to a predefined generic configuration. Using a STreamPlug board with pre-flashed software is initially useful to get familiar with the target hardware platform and the embedded Linux environment.

This activity does not strictly require the installation of the STreamPlug Linux support package (software development environment). This can be performed later, as described in the following section of this document.

Warning: RISK OF INCORRECT PROGRAMMING.
Carefully check that the specific hardware configuration (e.g.: DIP switch settings) has been set according to what is described in the relevant hardware manuals before powering the target hardware, and booting the pre-flashed software.

2.1 Host PC requirements

2.1.1 Windows PC

In order to control the target hardware, use a PC with a Microsoft® Windows® operating system (XP, Vista, and Windows 7).

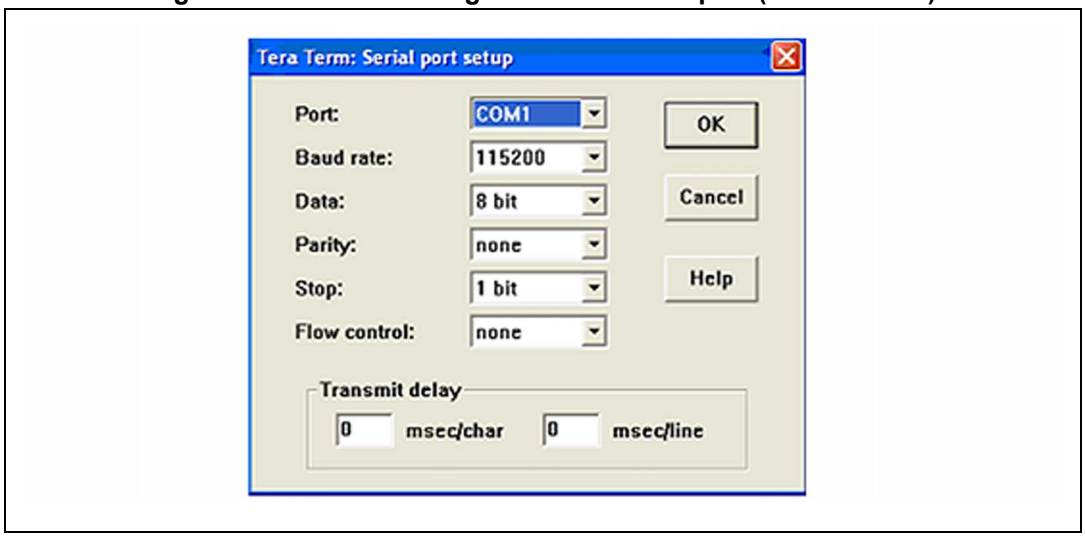
The first step is to set up a serial port for interacting with the embedded consoles (Linux shell and RTOS shell). If an RS232 serial port is not available on the PC, use a USB/RS232 adapter (not provided in the kit).

The second step is to obtain a terminal emulation program. Windows comes with the built-in HyperTerminal, but any equivalent tool can be used as an alternative. For instance, Tera Term is an open source free application with more features and higher flexibility, especially its scripting capability.

More technical information about Tera Term and a downloadable file may be found on the Internet . In order to configure the serial port with Tera Term:

1. Launch the tool.
2. Click on “Setup > serial port”. The configuration must reflect that shown in *Figure 1*.
3. To save the proper setting, click on “Setup > save setup”.

Figure 1. Tera Term configuration for serial port (Windows PC)



Using the HyperTerminal is very similar. To configure the serial port with the HyperTerminal:

1. Enter the “File > Properties” menu
2. Select the COM port (for example COM1) in the “Connect Using” dialog box
3. Press the “Configure” button
4. Enter the “Port Setting“ fields accordingly
5. To save the current configurations select the “File >Save As” menu item.

2.1.2 Linux PC

As an alternative to Windows, a PC with the Linux OS may be used.

The “$“ symbol represents a normal user prompt while a “#“ symbol means a root level prompt.

Caution: **IMPORTANT: READ/WRITE ACCESS IS REQUIRED**
Read/write access to the PC serial port is required. If necessary, check your distribution documentation to enable it. For example on Fedora Linux systems, add the user to the “dialout” system group.

Minicom is one of the most commonly used terminal emulators for Linux. Assuming a Fedora distribution for the host PC, to check the availability of the Minicom, execute the following command:

```
$ rpm -q minicom
```

On a Debian family distribution (Debian, Ubuntu, etc.) the command is:

```
$ apt-get -s install minicom
```

This will simulate the installation of the Minicom and so it will indicate if the Minicom is already installed or not.

To install the Minicom, if not found, execute this command from a root shell:

```
# yum install minicom
```

On Debian:

```
# apt-get install minicom
```

To start the Minicom, type the command:

```
$ minicom
```

To enter the configuration menu for the first time, press the key combination “Ctrl-A” and then “Z” (in sequence).

Note: If there is no global configuration file, the Minicom will not start. Create one by running the following command from a root shell:

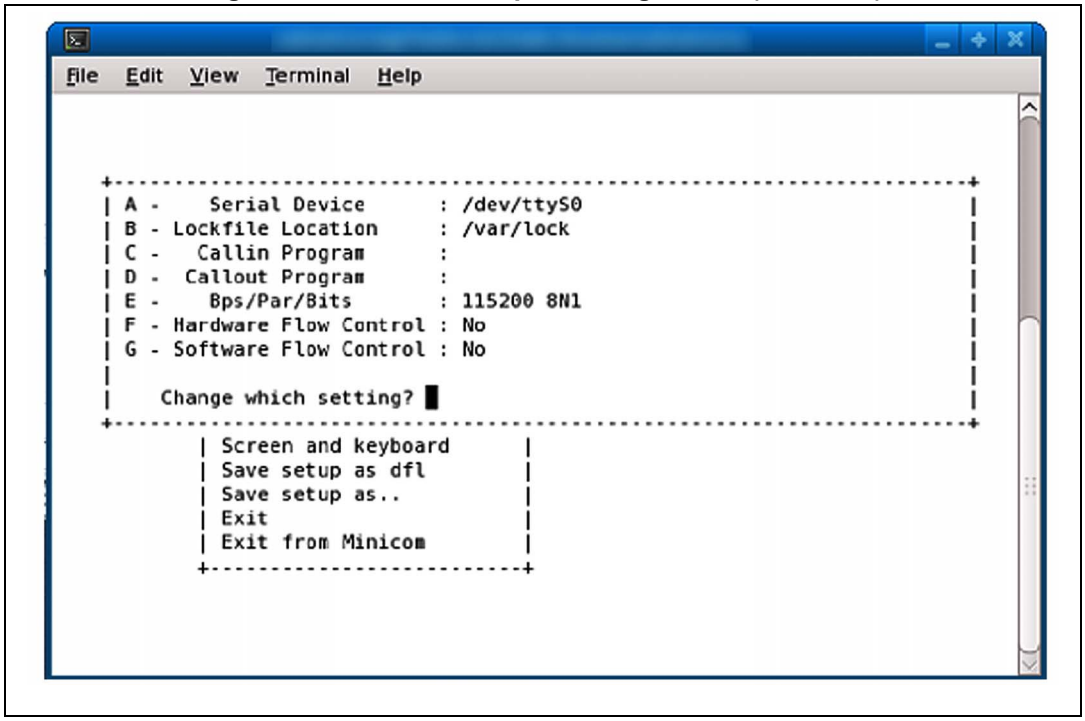
```
# minicom -s
```

and then follow the normal configuration procedure.

The serial connection information can be configured in the “configure minicom“ submenu and then “Serial port setup“. After that, the configuration must be saved using the “Save setup as“ option. The serial device name to be entered must match the one used for the link to the STreamPlug evaluation board. For example, the first serial port on the Linux PC is named /dev/ttyS0.

Select the serial speed as 115200 bps with 8 bits, no parity and 1 stop bit (115200 8N1) and disable both hardware and software flow control as shown in *Figure 2*.

Figure 2. Minicom serial port configuration (Linux PC)



To save a new default configuration which is automatically used by Minicom, select “Save setup as dfl”.

Alternatively, to create a new configuration file select “Save setup as …”. In order to use it, specify the configuration file name in the command line when invoking the Minicom.

Please note that by default the Minicom tries to initialize a “modem” connected to the chosen interface. To skip this step, invoke the Minicom with the -o option as follows:

```
$ minicom -o
```

As an alternative under GNU/Linux, use “GtkTerm” (see *Figure 3*) or “CuteCom” (see *Figure 4*).

Figure 3. GtkTerm screen

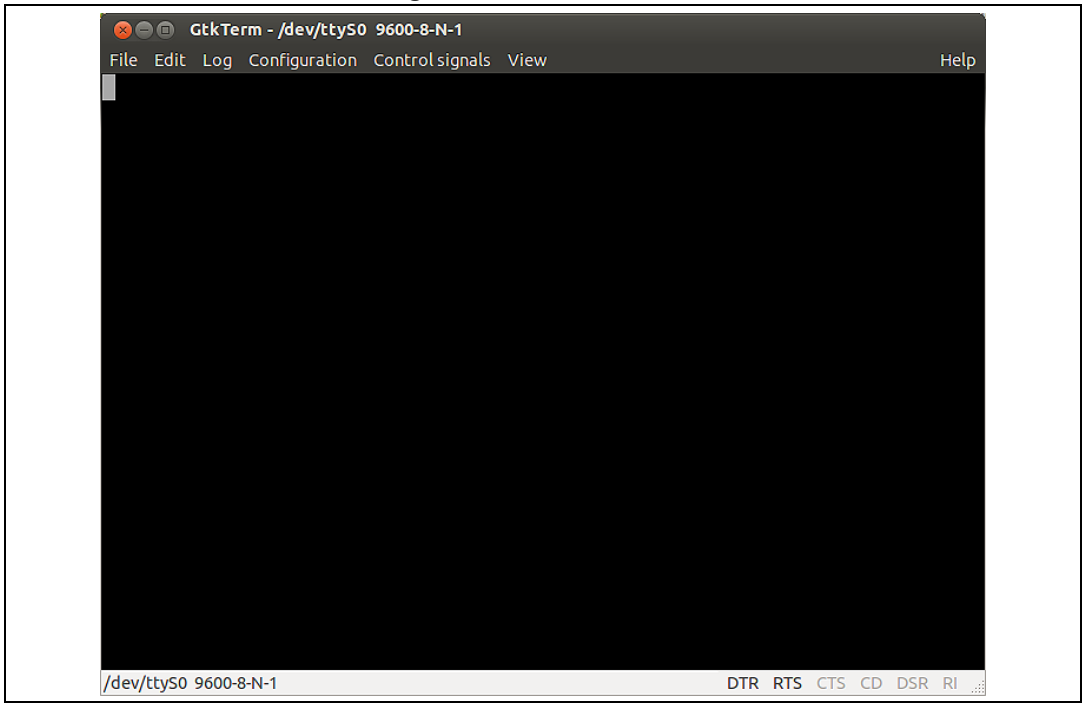
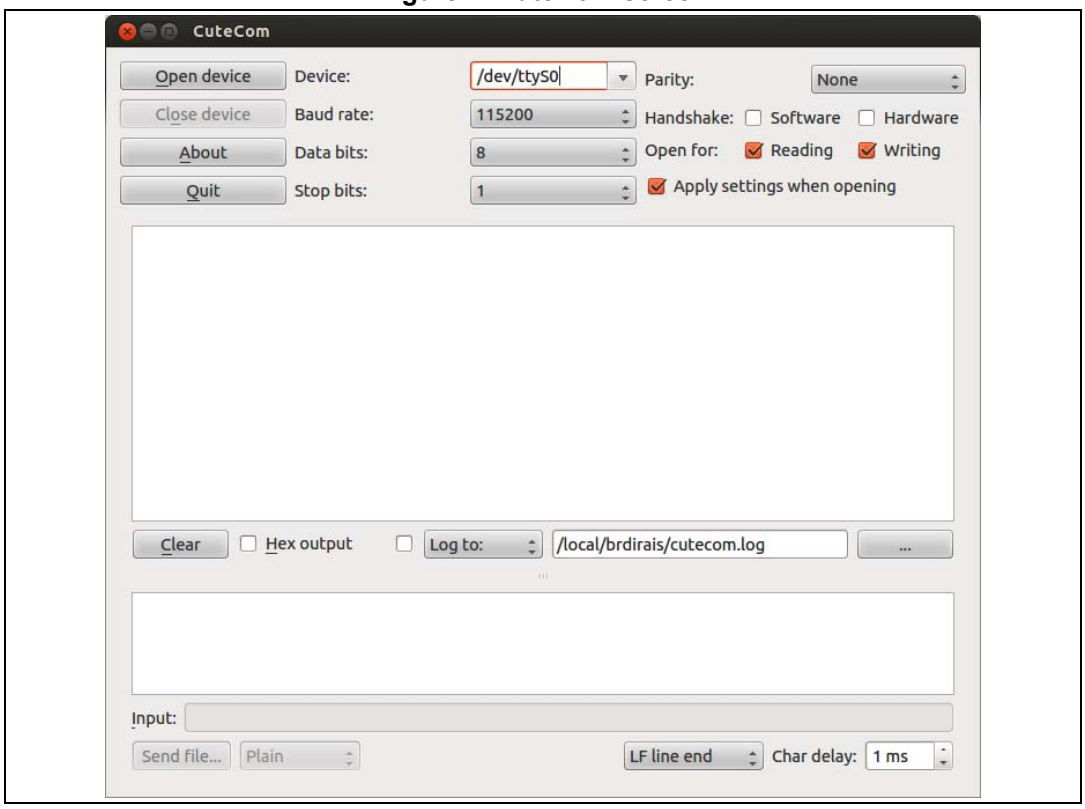


Figure 4. CuteCom screen



2.2 Overview of Flash contents and structure

On all evaluation boards, the default software is only pre-stored in the serial NOR Flash. The use of other memory types available on some evaluation boards (NAND, parallel NOR) may require the installation and usage of the STreamPlug additional file systems.

In the full configuration, the hypervisor and the system firmware start before the Linux kernel and provide to the proper configuration (via command line ATAG) to the Linux cell. The “Smartboot” becomes obsolete. In this full configuration the firmware is made of two images: a factory image that stores all the board dependent data including the command line and the upgrade image that includes the Linux kernel.

In the native Linux configuration the smartboot starts before the Linux kernel and provides the proper configuration via the command line ATAG.

The kernel, after further preliminary initialization, mounts the so called root file system. This is the binary image containing all software and data that comprise the operating system complementing the kernel. The root file system is the hierarchical file structure that end users see from the familiar Linux shell prompt. For pre-flashed software, it contains a generic subset of initialization scripts, system commands and runtime libraries.

There is a wide choice of file system standards for Flash memories in the Linux world (for examples, CRAMFS, JFFS2, YAFFS2, LogFS and UBIFS).

Refer to the UM2004 - “STreamPlug ST2100 SDK and quick start guide” for more details on the firmware images, Flash memory maps, etc.

2.3 Booting up to Linux prompt

To boot Linux just power up the board. The root file system is mounted automatically, the system shell is run and a prompt appears when it is ready to accept commands. If there is an LCD monitor connected to the color LCD (CLCD) interface and a keyboard to the USB host one, a shell login is automatically redirected to the LCD panel.

The pre-flashed file system provides a default set of system commands and runtime libraries.

System commands are the familiar basic utilities mainly provided to support the development and debugging stages. They are all stored under standard Linux paths. They may not be strictly needed in a final product; however their small size allows them to be kept in production devices without a significant penalty in terms of the memory footprint.

As is typical for embedded Linux environments, it uses “BusyBox”, an open source program combining tiny versions of many common user space GNU utilities, such as `ls`, `rm`, and others, into a single small executable; an important factor when the minimized Flash footprint is required. You can obtain more information about BusyBox on the web.

BusyBox has been developed with size optimization and limited resources in mind. For this reason the available commands typically have fewer options than their full-featured GNU counterparts. However, the most important options are still available with all the functions needed for developing and testing embedded products.

In addition to the BusyBox features, a number of other executable programs are also included in the RootFS images.

It can be also helpful to explore the Linux standard `/proc` and `/sys` pseudo-file systems. These subtrees contain user accessible entries that pertain to the runtime state of the kernel and, by extension, the executing processes that run on the top of it. The term “pseudo” is used because the proc and sys file systems exist only as a reflection of the in memory kernel data structures they display. This is why most files and directories within these directories are zero bytes in size.

In practice, the proc file system is intended to be populated at runtime with system information and statistics. Proc files may be either “read only” or “read write”. Each numerically named directory within `/proc` corresponds to the process ID (PID) of a process currently executing on the system. This part of the proc file system totally depends on the runtime state of the target. Each numeric entry contains subfiles that provide process-specific information. The other (non-numeric) entries describe some aspects of kernel operation.

The standard `/proc/bus/usb` subtree is also made available. This is used to access USB host controllers and plugged devices from user space applications.

For more details about the functionality provided by pseudo file systems, please refer to standard Linux documentation.

2.4 Using USB pen drive

USB pen drives can be accessed in a standard Linux way: connecting them to a USB host port and mounting their file system under the root file system. An example of the operational sequence for a standard pen drive connected as an “sda” device (only one pen drive present) is as follows:

1. Plug the pen drive into a USB port and wait for the Linux kernel to auto-detect it (active kernel messages will be visible on the debug terminal).
2. Mount the pen drive file system:
   a) `# mount /dev/sda1 /mnt`
   b) Now the pen drive file system can be accessed under the `/mnt` directory.
3. Transfer files as usual (for example: `cp` command).
4. When finished, `unmount` the pen drive:
   a) `# umount /mnt`

Now the pen drive may be physically unplugged.

2.5 Using SATA hard disk

The evaluation board supports the SATA.

An example of the operation sequence for a SATA hard disk is the following:

1. Mount the SATA file system:
   a) If FAT32:
   ```
   # mount /dev/sda1 /mnt
   ```
   b) If NTFS:
   ```
   # ntfs-3g /dev/sda1 /mnt
   ```
2. Now the SATA file system can be accessed under the `/mnt` directory.
3. Transfer files as usual (for example: `cp` command).
4. When finished, `unmount` the SATA:
   ```
   # umount /mnt
   ```
5. Now the SATA may be physically unplugged.

2.6 Connecting evaluation board to LAN

The evaluation board should be connected to a developer's host PC over a private LAN or even a point-to-point link.

By default the STreamPlug has no IP address. It can be configured via “ifconfig” from the console. In order to make the evaluation board asking for an IP address to the DHCP server during the boot the kernel command line can be customized.

Commonly used private IP addresses (Class C) are in the range of 192.168.0.0 to 192.168.255.255.

As an example, let's assume an IPv4 local area network with the following characteristics:

Network IPs:192.168.1.X

Host PC IP:192.168.1.1

Evaluation board IP:192.168.1.10

X = 0, 2 … 0, 11… 254

On a Linux PC, configure the host address as follows:

```
# ifconfig eth0 192.168.1.1 broadcast 192.168.1.255 netmask 255.255.255.0
```

In this example we are assuming that the evaluation board is connected to the Ethernet port 0 (eth0) of the host machine.

On the evaluation board console, configure the target address as follows:

```
# ifconfig eth0 192.168.1.10 broadcast 192.168.1.255 netmask 255.255.255.0
```

Note: When running the full STreamPlug software, the mapping of network interfaces to Linux devices varies depending on the Linux command line configuration (see Table 1).

Table 1. Ethernet mapping

Command line setting	Description
eth=off	Ethernet is not available to Linux. The device “eth0” corresponds to the PLC interface.
eth=on	Ethernet is available to Linux as a direct interface. Linux uses a native driver for Ethernet. The device “eth0” refers to the Ethernet, while the device “eth1” is mapped to PLC.
eth=rtos	Ethernet is managed by RTOS (STreamPlug firmware) and L2 bridging is implemented inside RTOS. A single device “eth0” is exposed to Linux that is the bridged PLC + Ethernet.

3 STreamPlug Linux distribution

The STreamPlug distribution provides all the host side (PC) and target side (evaluation board) software components enabling system designers to develop their own applications for STreamPlug based platforms, as well as customize the various aspects of the embedded software architecture.

3.1 Overview

The STreamPlug distribution is made by several different components that can be summarized as follows:

- Command line cross-development toolchain (compiler, linker, building tools, etc.) running on a Linux x86 PC.
- Buildroot with a set of open source user space ARM packages (programs and runtime libraries) to be promptly reused in root file systems as support to specific applications. The root file system also provides the compiler, linker and so on for compiling applications on target boards.
- Linux kernel 2.6.35, configurable for the different STreamPlug evaluation boards. It includes BSP/device drivers for the STreamPlug hardware features.
- System SDK including:
  - Hypervisor and system firmware, with added support for STreamPlug evaluation boards
  - Second and third level bootloaders firmware, plus boot tools

Most distribution components are also available as a source code.

Full details about STreamPlug Linux components are reported in the UM1942 Linux software user manual for STreamPlug ST2100 and third-party documentation (e.g.: GNU, General Dynamics Broadband, etc.).

The following sections detail what each component can do and how to use it. However, to simplify their usage a configuration file is provided with ready to use targets to build any combination of components.

3.2 Toolchain

The cross-development toolchain is a set of programs running on a host PC which are used to generate applications executable on an ARM based system. The tool set is usually composed of the following items:

- Cross-compilation of the source code to generate a native object code for the ARM CPU cores integrated into the STreamPlug embedded MPU family.
- Cross-linking of the ARM object code to generate executable programs or (dynamically linkable) shared libraries.
- Managing object code archives, incremental rebuilding and other auxiliary tasks.

The provided toolchain is based on the widely adopted open source GNU toolset.
A summary of the main available command line tools is reported in *Table 2*. For detailed documentation please consult the GNU website.

Table 2. List of main toolchain packages

Tool	Function
GCC	**GNU compiler collection includes front ends for C and C++ as well as libraries for these languages**
gcc C	Cross-compiler for ARM
gcov	Code coverage
Binutils	**GNU binutils are a collection of binary tools**
ar	Archiver
as	Cross-assembler for ARM
gprof	Profiling tool
ld	Cross-linker for ARM
nm	Lists symbols in object files
objcopy	Copies a binary file
objdump	Displays information from object files
ranlib	Generates an index to speed access to archives.
readelf	Displays the information about the contents of ELF format files
rtrip	Removes symbols and sections from files
GNU make	**Incremental build management**
GDB gdb	**Debugger**

The toolchain package installed for the STreamPlug Linux BSP is downloadable from: http://sources.buildroot.net/arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2.

The cross-development toolchain can be installed in `/opt` with the following commands:

```
$ cd /opt
$ tar -xjf arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
$ mv arm-2009q1 arm-none-linux-gnueabi
$ export PATH=$PATH:/opt/arm-none-linux-gnueabi//bin
$ export CROSS_COMPILE=arm-none-linux-gnueabi-
$ export ARCH=arm
```

Note that the environment variables should be defined in the configuration script of the used shell to make them permanent. For example: if the bash shell is used then the environment variables can be defined in the file `~/.bashrc`. It is then possible to check the toolchain by trying to run one of its tools. For example:

```
$ arm-none-linux-gnueabi-gcc --help
Usage: arm-none-linux-gnueabi-gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  --
help={target|optimizers|warnings|undocumented|params}[,{[^]joined|[^]separ
ate}]
                           Display specific types of command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search
                           path
  -print-libgcc-file-name  Display the name of the compiler's companion
                           library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component
                           <prog>
  -print-multi-directory   Display the root directory for versions of
                           libgcc
  -print-multi-lib         Display the mapping between command line options
                           and multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -print-sysroot           Display the target libraries directory
  -print-sysroot-headers-suffix Display the sysroot suffix used to find
                           headers
  -Wa,<options>            Pass comma-separated <options> on to the
                           assembler
  -Wp,<options>            Pass comma-separated <options> on to the
                           preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xassembler <arg>        Pass <arg> on to the assembler
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor
```

```
  -Xlinker <arg>           Pass <arg> on to the linker
  -combine                 Pass multiple source files to compiler at once
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of
                           <file>
  -std=<standard>          Assume that the input sources are for <standard>
  --sysroot=<directory>    Use <directory> as the root directory for headers
                           and libraries
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
  -V <version>             Run gcc version number <version>, if installed
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not
                           executed
  -E                       Preprocess only; do not compile, assemble or
                           link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input
                           files
                           Permissible languages include: c c++ assembler
                           none
                           'none' means revert to the default behavior of
                           guessing the language based on the file's
                           extension

Options starting with -g, -f, -m, -O, -W, or --param are automatically
 passed on to the various sub-processes invoked by arm-none-linux-gnueabi-
gcc.  In order to pass
 other options on to these processes the -W<letter> options must be used.

For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>.
```

3.3 Buildroot

The buildroot (<http://buildroot.net>) is a tool to handle almost everything of an embedded system development project: the cross-compiling toolchain, root file system generation, kernel image compilation and bootloader compilation. The buildroot is also sufficiently flexible that it can also be used for only one or several of these steps.

The most important buildroot features are that it:

- Can support the generation of all the main software components
- Is very easy to set up, thanks to its menuconfig, gconfig and xconfig configuration interfaces, which are familiar to all embedded Linux developers. Building a basic embedded Linux system with the buildroot typically takes 15 to 30 minutes.
- Supports several hundreds of packages for user space applications and libraries. X.org stack, gtk2, Qt, DirectFB, SDL, GStreamer and a large number of network related and system related utilities and libraries are supported.
- Supports multiple file system types for the root file system image: JFFS2, UBIFS, tarballs, romfs, cramfs, squashfs and more.
- Can generate a uClibc cross-compilation toolchain, or reuse the existing glibc, eglibc or uClibc cross-compilation toolchain
- Has a simple structure that makes it easy to understand and extend. It relies only on the well known Makefile language.

The released installed buildroot is downloadable from http://buildroot.net/downloads.

The buildroot can be installed into a local folder with commands like the following:

```
$ cd
$ tar -xjf buildroot.tar.bz2
$ export BUILD_ROOT_DIR=~/buildroot
```

The buildroot has a configuration tool similar to the Linux kernel tool. There is no need to be root in order to configure and use the buildroot.

The whole buildroot package, customized buildroot configuration files, STreamPlug devices architecture and new packages have been provided.

To configure the buildroot to a host modified/new application for the development board, it is necessary to follow the steps:

```
$ cd buildroot
$ make okl_streamplug_minimal_board_buildroot_SMI_defconfig
$ make
```

Customization to the build procedure of the buildroot may be done using the command:

```
$ make xconfig; Graphical interface
```

As an alternative, use:

```
$ make menuconfig; Pseudo-graphical interface
```

or:

```
$ make gconfig; Gnome graphical interface
```

3.4 Linux package

The Linux kernel running on the STreamPlug machine is based upon the open source Linux kernel software version 2.6.35. STreamPlug chip is based on the ARM926 architecture.

The LSP contains a copy of the Linux kernel ready to be used on a STreamPlug board as guest OS of the OKL4 Microvisor.

Table 3 is the list of the folders added or modified within the ARM Linux kernel tree in order to include the support of the STreamPlug machine and device drivers foreseen for this release.

Table 3. ARM Linux kernel tree updates

Folder	Status
arch/arm/mach-streamplug	New
arch/arm/plat-streamplug	New
arch/arm/boot/Makefile	Modified in order to support the build of an *.elf image compressed
arch/arm/configs	Modified in order to include machine configurations The default one is: okl4_hybrid_platform_streamplug_devel_board_defconfig
arch/arm/Kconfig	Modified in order to add the configuration for the STreamPlug platform
arch/arm/Makefile	Modified in order to set the Linux entry offset to 0x00048000 and to add help comment for the *.elf image
sound/soc/streamplug	New
drivers/char	New vlog device driver and new device for image validation
drivers/char/c3	Modified C3 device driver
drivers/gpio	New ArkGPIO device driver
drivers/media/video	New image sensor device driver
drivers/misc	New PCIe gadget device driver
drivers/mtd/devices	New SMI device driver
drivers/net/irda	New DICE FIrDA device driver
drivers/rtc	New RTC device driver

The ST STreamPlug Linux kernel supports the following device drivers:

- SMI
- UART1/UART2
- Watchdog
- Timers
- VIC
- Ethernet “Best Effort”
- USB host
- USB device Ethernet gadget

- USB device zero gadget
- USB device FS gadget
- DMA
- RTC
- I^2C
- SPI
- JPEG decoder
- GPIO
- JPEG encoder
- CAN
- CLCD
- PCIe RC and EP with MSI/HW interrupts
- FSMC
- FIrDA
- SATA
- C3
- SPORT audio out
- Ark GPIO
- SPORT audio in
- TS image capture
- KSP interface support
  - Miscellaneous register r/w access
  - Virtual interrupt dispatcher
- VLOG
- Flash controller shared access mechanism
- HPAV driver (PLC)
- Image validation device

The configuration of peripherals is implemented using boot parameters, and is not hard coded in the Linux sources.

3.5 SDK package

See the UM2004 - “STreamPlug ST2100 SDK and quick start guide” for details.

For updates and support please consult the IoTecha website: http://www.iotecha.com/.

4 Working at application level (userland)

When working at the application level (the so called “userland”) developers are only concerned with programs and libraries stored in a root file system. Bootloaders and the kernel are assumed to be stable and stored in the Flash memory.

4.1 Workflow models

There are many approaches (workflows) to modify/extend the root file system for specific application scenarios; the main ones are described in the following subsections.

4.1.1 Remote mounting of root file system (NFS)

If the board can be connected to the development PC through an Ethernet LAN, the most practical solution is to leave the root file system stored on the PC and remotely mount it on the target embedded Linux OS through the NFS protocol.

The advantages of this approach are:

- The root file system has no global size constraints. Developers can keep hundreds or thousands of packages (programs and libraries) in a directory of their PC disk. All file access from the Linux OS running on the board is performed over the network in a transparent way. Files are not copied to the Flash memory, but are loaded to the DDR RAM strictly on demand.
- A program or library can be simply built (compiled and linked) with the output file on the PC disk. The new version is then available for execution on the board without any need for the manual transfer or board reboot.

The drawbacks are:

- File access by NFS over the LAN can be slower than direct Flash memory access.
- There is no early assessment of which files are actually used and of the overall required size for future migration to the Flash memory.

To remotely mount the root file system, configure and start the NFS server on the Linux PC.

Assuming the NFS server functionality is already provided by the host, the only configuration is an entry for the target root directory to the `/etc/exports` file.

To check the NFS availability and start the services, use the following commands (from the user root account):

```
# rpm -q nfs-utils
```

After modifying the `/etc/exports` file, make sure the NFS system is notified about the change, for example by running the command:

```
# service portmap start
Starting portmap: [ OK ]
```

followed by the command:

```
# service nfs start
Starting NFS services: [ OK ]
Starting NFS quotas: [ OK ]
Starting NFS daemon: [ OK ]
Starting NFS mountd: [ OK ]
```

Each time this file is changed while the NFS service is already started, it either has to be restarted or the NFS daemon must be forced to reload the new configuration:

```
# exportfs -a
```

4.1.2 Incremental changes to Flash file system

If the root file system is stored on the Flash memory, it is possible to transfer files from the PC to the target board in the following ways:

- Transfer by USB pen drive
- Transfer by LAN (TFTP)

4.1.3 Flash file system full replacement

To replace the entire file system on the Flash memory:

- Rewrite the root file system partition using the access to the SPI connector.
- Rewrite the root file system from Linux by overwriting the mtdblock of the file system.

4.2 Command line cross-development

The most important item concerning host packages is the cross-development toolchain (the set of programs running on a host PC but targeting ARM-specific code output) that has support for:

- Cross-compilation of the source code to generate a native object code for the ARM CPU cores integrated into the STreamPlug SoC
- Cross-linking of the ARM object code to generate executable programs or (dynamically linkable) shared libraries
- Managing object code archives, incremental rebuilding and other auxiliary tasks.

4.3 Rebuilding the root filesystem

Once the application is working correctly, it is possible to add it to the buildroot build procedure in order to rebuild the root filesystem and then to make it available to the user permanently at every power-on of the target board.

The binary file of the application and its supporting libraries and data files must be copied in the template filesystem defined within the buildroot tool. In case of the STreamPlug the folder to place new and/or update applications is in the folder:

`<buildroot>/board/okl_streamplug/.`

There are two configuration files under the folder "`configs`" with the corresponding name:

- `okl_streamplug_devel_board_buildroot_defconfig`, to be used on the Flash memory of 16 MB or greater
- `okl_streamplug_minimal_buildroot_defconfig`, to be used on small partitions.

To rebuild the filesystem, it is possible to use the following commands:

```
$ make okl_streamplug_<fs_type>_buildroot_<SMI|NAND>_defconfig
$ make
```

and then perform the following steps:

1. Set target configuration for the ARM926T architecture
2. Set build options
3. Include the toolchain package that was just installed
4. Build/install selected target packages
5. Recover the skeleton of the filesystem, the devices table and the `inittab` customized for the STreamPlug from `<builroot>/board/okl_streamplug`.

At the end of build the filesystem image is created under the folder: `<buildroot>/output/images`. The *.ubi image can be burned in the target hardware's Flash.

Note: At startup the Linux kernel mounts the filesystem from the Flash bank 1 containing the root filesystem.

In the main filesystem are included:

- The kernel modules (c3, FSMC, USB device gadget, dmatest, etc.)
- The user space examples
- alsa-utils and alsa-lib
- iperf
- fbv package (for LCD controller)
- irda-utils
- mtd tools
- can-utils and ip-route2
- ntfs-3g package (for mounting NTFS SATA drive)
- Network packages (bridge-utils, etc.)
- gpm package (mouse support)

5 Working with customized kernels

When working with STreamPlug evaluation boards, modifications to the supplied Linux kernel are only needed when:

- Changing kernel configuration, by enabling or disabling some options or features
- Developing new drivers on the top of existing ones, in order to interface peripherals that can be added through custom add-on boards connected to STreamPlug evaluation boards (where applicable)
- Rewriting partially or totally some existing reference device drivers (for example, for further optimization or special needs)
- Developing custom kernel modules

The use of kernel level software with hardware platforms other that STreamPlug evaluation boards is not discussed in this section.

When working at the kernel level, developers are concerned with the kernel source code tree from which a single binary image file must be generated by “rebuilding” the kernel.

The main tasks to be performed are:

- Kernel reconfiguration and rebuild
- Firmware image rebuild
- Firmware image loading and execution on a target evaluation board

For general information about the Linux kernel, please refer to public Linux documentation.

In order to build each single Linux package, the following environment variables need to be set or passed to make commands:

```
$ export CROSS_COMPILE = arm-none-linux-gnueabi-
$ export PATH = $PATH:/opt/streamplug-toolchain/bin
$ export ARCH = arm
```

5.1 Reconfiguring and building Linux kernel

To rebuild the Linux kernel it is necessary to:

1. Install STreamPlug SDK
2. Setup the Linux kernel build environment
3. Configure and build the Linux kernel (vmlinux)
4. Pack the Linux kernel into an upgrade image
5. Updated the root filesystem with the Linux kernel modules.

5.1.1 Install STreamPlug SDK

See the UM2004 - “STreamPlug ST2100 SDK and quick start guide” for details.

5.1.2 Setup Linux kernel build environment

Before building the Linux kernel patched with OKL's support for the OKL4 Microvisor, it is necessary to locate a library folder where the build procedure can find “include files” and the library specific to support the OKL4 Microvisor.

The library folder is delivered within the LSP package in the SDK package on the okl4sdk/stage directory. Then the following Linux environment variable is used by the Linux kernel build procedure to locate the library folder

```
$ export OKL4_PROG_ENV= ${STP_SDK_DIR}/okl4sdk/stage
```

5.1.3 Configure and build Linux kernel

After the source code for the STreamPlug Linux has been prepared, it is necessary to configure it in order to activate the right functionalities for the corresponding type of the STreamPlug board (either debug or development).

Under the folder arch/arm/configs are available configuration files for each supported configuration:

- Full configuration (paravirtualised kernel)
  - `okl4_hybrid_platform_streamplug_devel_board_defconfig`
- Native Linux configuration
  - `streamplug_devel_board_defconfig`

The commands used to configure the kernel are:

```
$ make distclean
$ make okl4_hybrid_platform_streamplug_devel_board_defconfig
```

A finer grained configuration can be obtained by using the default tool provided by the Linux kernel using:

```
$ make menuconfig
```

or

```
$ make xconfig
```

or

```
$ make gconfig
```

Once the desired configuration has been applied to the Linux kernel, it is possible to build the Linux ELF with the commands:

- full configuration

  ```
  $ make vmlinux
  ```

- native configuration

  ```
  $ make elfImage
  ```

Note that it assumes that the environment variable ARCH is set to arm, otherwise it is possible to use the following command:

```
$ make ARCH=arm vmlinux
```

If something went wrong it is possible to make the changes and rebuild with the command:

```
$ make vmlinux
```

Sometime it is necessary to force the rebuild of everything, and then in such case it is better to execute the following command before to rebuild the kernel:

```
$ make clean
```

To change the board the kernel is built for it is necessary to delete everything using the following command:

```
$ make distclean
```

Once everything is built correctly the resulting Linux kernel file `vmlinux` can be found in the root folder of the kernel tree, while the file elfImage is in the folder:

```
<linux src>/arch/arm/boot.
```

5.1.4 Pack Linux kernel into boot image

The building of the firmware image is slightly different according to the configuration: full or native.

Full configuration

Once the Linux kernel with the OKL4 Microvisor support is ready, it is possible to pack it into an *.elf file together with the OKL4 Microvisor and the system software and to generate a new firmware image.

This can be done using the generation scripts provided in the system SDK. (See *Section 5.2 on page 29* for details).

Note that due to the presence of mutual exclusive peripherals and multi-mode peripherals within the Linux kernel it is necessary to specify a configuration within the cmdline ATAG.

Table 4 specifies the supported ATAG to implement the pad muxing and console options:

Table 4. XML command line options for padmux configuration

Peripherals	Values
CLCD	– clcd=on:24bpp, if CLCD is enabled with 24 bpp – clcd=on:18bpp, if CLCD is enabled with 18 bpp – clcd=off,if CLCD is disabled
PCIe bridge	– pcie=on:rc:1, if PCIe is configured as a root complex with the MiPHY™ clock generated by the pll2 input clock – pcie=on:rc:2, if PCIe is configured as a root complex with the MiPHY clock generated by the qfs4 input clock – pcie=on:rc:3, if PCIe is configured as a root complex with the MiPHY clock generated by the external clock – pcie=on:ep:1, if PCIe is configured as an endpoint with the MiPHY clock generated by the pll2 input clock – pcie=on:ep:2, if PCIe is configured as an endpoint with the MiPHY clock generated by the qfs4 input clock – pcie=on:ep:3, if PCIe is configured as an endpoint with the MiPHY clock generated by the external clock – pcie=off, if PCIe is disabled
USB controller	– usb=on:device, if the USB is activated as a gadget – usb=on:host, if the USB is activated as a host – usb=off if the USB is not configured

Table 4. XML command line options for padmux configuration (continued)

Peripherals	Values
Ethernet network controller	– eth=<on, off, rtos>:<primary, secondary>:<1,…,3>:<Mac Address> Some examples: – eth=on:primary:1:00:80:40:AE:20:98, if the device driver is configured on low GPIOs groups, with the pll2 input clock as the PHY clock root, and with a default MAC address – eth=on:secondary:2:00:80:40:AE:20:98, if the device driver is configured on high GPIOs groups, with the qfs4 input clock as the PHY clock root, and with a default MAC address – eth=on:primary:3:00:80:40:AE:20:98, if the device driver is configured on low GPIOs groups, with the external clock as the PHY clock root, and with a default MAC address – eth=off, if Ethernet is disabled – eth=rtos:primary if the device driver is configured on high GPIOs groups and assigned to system FW
I^2C controller	– i2c=on, if the I^2C interface is enabled and configured – i2c=off, if the I^2C interface is disabled
Synchronous serial port	-ssp=on:<24,…,39>, if the SPI interface is enabled and configured with a fixed CHIP-SELECT line ssp=off, if the SPI interface is disabled The default value for the number of the GPIO used by the OK Linux to reserve the SPI CHIP-SELECT line is 39. For STreamPLug OK Linux GPIOs the numbers reserved start from 24 to 39.
UART port 1	– uart1=on:primary, UART1 enabled on the GPIO primary group – uart1=on:secondary, UART1 enabled on the GPIO secondary group – uart1=rtos:primary, UART1 enabled on the primary group by system FW
UART port 2	– uart2=on:primary, UART2 enabled on the GPIO primary group – uart2=on:secondary, UART2 enabled on the GPIO secondary group – uart2=rtos:secondary, UART2 enabled on the secondary group by system FW
CAN network controller	– can=on:primary, if the device driver is configured on the low GPIOs group – can=on:secondary, if the device driver is configured on the high GPIOs group – can=off, if the device driver is disabled
FIrDA	– firda=on:1, if it supports only the SIR mode – firda=on:2, if it supports only SIR and MIR modes – firda=on:3, if it supports all SIR, MIR and FIR modes – firda=off, if the device driver is disabled
FSMC	– fsmc=on:<nand,sram,nor><0,1> Some examples: – fsmc=on:nand0, if the FSMC controller is enabled and configured for NAND Flash memory devices with 8-bit data width – fsmc=on:nand1, if the FSMC controller is enabled and configured for NAND Flash memory devices with 16-bit data width – fsmc=on:nor1, if the FSMC controller is enabled and configured for parallel NOR Flash memory devices with 16-bit data width – fsmc=off, if the FSMC is disabled *Note: By default, the system firmware performs the setup and adds the option "initdone" when passing the ATAG to the Linux if running in the full configuration.*

Table 4. XML command line options for padmux configuration (continued)

Peripherals	Values
SATA	– sata=on:1, if the SATA is enabled and configured with the MiPHY clock generated by the pll2 input clock – sata=on:2, if the SATA is enabled and with the MiPHY clock generated by the qfs4 input clock – sata=on:3, if the SATA is enabled and with the MiPHY clock generated by the external clock – sata=off, if the SATA is disabled
SPORT	– sport=on, if the SPORT is enabled – sport=off, if the SPORT is disabled
TS	– ts=on, if the TS is enabled – ts=off, if the TS is disabled
AS (ARK) GPIO	– ark_gpio=<on,off>:<nnnnnn, n=0,1,2> Some examples: – ark_gpio=on:110000, if the ARK_GPIO device driver is enabled with only the GPIOs group A and B enabled – ark_gpio=on:011221, if the ARK_GPIO device driver is enabled with ARK GPIOs groups A disabled, B enabled, C enabled on GPIO_GROUP 04, D enabled on GPIO_GROUP 10, E enabled on GPIO_GROUP 18 and the group F enabled on GPIO_GROUP 13 – ark_gpio=off, if the ARK_GPIO is disabled
GP (ARM) GPIO	– arm_gpio1=on, if the ARM GPIO group 1 is enabled – arm_gpio2=on, if the ARM GPIO group 2 is enabled For the native Linux, they should be always on and they are set by default cmdline.
Linux CONSOLE	– console=none if the Linux console is suppressed – console=<tty device>,<tty configuration> – console=ttyAMA0, 115200n8 , if the Linux console on ttyAMA0 with configuration as in *Section 2.1: Host PC requirements on page 5* (default) – console=ttyAMA1, 115200n8 , if the Linux console on ttyAMA1 (valid only if both UARTs to Linux) with configuration as in *Section 2.1*.
SYSTEM (RTOS) CONSOLE	– rtosconsole=<uart port>, <uart configuration> – rtosconsole=uart1, 115200n81 – rtosconsole=uart2, 115200n81 (default) If not present no console is available on UARTs. In this case, the stpconsole application example can be used to access the RTOS console from the Linux.

Native configuration

If the native Linux configuration is chosen, the Linux kernel image to be used is the binary “elfImage” that provides the physical entry point, while vmlinux still has the virtual one.

In order to generate the proper cmdline ATAG (see *Table 4*) a small loader has to be compiled: smartboot. Then, the two *.elf images of the Linux and the loader have to be packed together, this can be done using the generation scripts provided in the system SDK and the build example that comes with the smartboot. (See *Section 5.2* for details).

In order to generate the proper ATAGs, the smartboot build supports several make variables. The complete list of these options can be displayed by typing the following command from the folder of the smartboot sources:

```
$ make help
Usage: make OPTION=VALUE...
Options:
COMMAND_LINE:     the command line passed to linux kernel
SDRAM_SIZE:       specify the size of installed DDR module
CLCD_MODE:        select padmux mode for clcd (0 to disable, 1 to enable
                  24bpp mode; 2 to enable 18 bpp mode)
SATA_MODE:        select padmux mode for sata (0 to disable, 1 to enable)
SATA_CLK_OPT:     select miphy clock source for sata (1 for PLL2, 2 for QFS4,
                  3 for External)
PCIE_MODE:        select padmux mode for pcie (0 to disable, 1 for mode 'rc',
                  2 for mode 'ep')
PCIE_CLK_OPT:     select miphy clock source for pcie (1 for 'PLL2', 2 for
                  'QFS4', 3 for 'External')
USB_MODE:         select padmux mode for usb (0 to disable, 1 for mode
                  'device', 2 for mode 'host')
ETH_MODE:         select padmux mode for eth (0 to disable, 1 for mode
                  'primary', 2 for mode 'secondary')
ETH_MODE_OPTIONS: MAC address of adapter
ETH_CLK_OPT:       select phy clock source for eth (1 for 'PLL2', 2 for
                   'QFS4', 3 for 'External')
I2C_MODE:          select padmux mode for i2c (0 to disable, 1 to enable)
SSP_MODE:          select padmux mode for ssp (0 to disable, 1 to enable)
SSP_MODE_OPTIONS: the number [24-39] of GPIO pin used as Chip Select by ssp
UART1_MODE:        select padmux mode for uart1 (0 to disable, 1 for mode
                   'primary', 2 for mode 'secondary')
UART2_MODE:        select padmux mode for uart2 (0 to disable, 1 for mode
                   'primary', 2 for mode 'secondary')
CAN_MODE:          select padmux mode for can (0 to disable, 1 for mode
                   'primary', 2 for mode 'secondary')
FIRDA_MODE:        select padmux mode for firda (0 to disable, 1 to enable)
FIRDA_MODE_OPTIONS:     set firda mode option: (0 for 'SIR only', 1 for 'SIR
                  and MIR', 2 for 'SIR and FIR', 3 for 'SIR, MIR and FIR')
FSMC_MODE:         select padmux mode for fsmc  (0 to disable, 1 for mode
                   'nor', 2 for mode 'sram', 3 for mode 'nand')
FSMC_MODE_OPTIONS:      set fsmc memory data width: (0 for '8 bits', 1 for
                   '16 bits')
SPORT_MODE:              select padmux mode for sport (0 to disable, 1 to
                  enable)
TS_MODE:                select padmux mode for ts (0 to disable, 1 to enable)
ARK_GPIO_MODE:           select padmux mode for ark gpio (0 to disable, 1 to
                  enable)
ARK_GPIO_A_MODE_OPTIONS:    select ark gpio group a (0 to disable, 1 to
                  enable)
ARK_GPIO_B_MODE_OPTIONS:    select ark gpio group b (0 to disable, 1 to
                  enable)
```

```
ARK_GPIO_C_MODE_OPTIONS:    select ark gpio group c (0 to disable, 1 muxed
                  on gpio04, 2 muxed on gpio09)
ARK_GPIO_D_MODE_OPTIONS:    select ark gpio group d (0 to disable, 1 muxed
                  on gpio05, 2 muxed on gpio10)
ARK_GPIO_E_MODE_OPTIONS:    select ark gpio group e (0 to disable, 1 muxed
                  on gpio12, 2 muxed on gpio18)
ARK_GPIO_F_MODE_OPTIONS:    select ark gpio group f (0 to disable, 1 muxed
                  on gpio13, 2 muxed on gpio19)
```

It's important to notice that the mtd partitions table has to be provided via cmdline.

After that, the two binaries can be packed into a firmware image loadable via UART. A build example is provided to support this operation. It is based on a Makefile. The following steps are needed to run the example:

- Define the environment variable: `STP_SDK_DIR =<path to the SDK>`
- Copy both binaries in the “src” folder and create the destination folders: “work” and “images”
- Select the proper DDR and debug configuration by modifying the Makefile or providing the proper variables DDR_CONFIG and DBG_CONFIG when invoking “make images”.

For a detailed reference of DDR configurations, image generation and loading tools please see the UM2004 - “STreamPlug ST2100 SDK and quick start guide” user manual.

5.1.5 Updated root filesystem with Linux kernel modules

The modules can be built using the command:

```
$ make modules
```

The Linux modules can be installed in the template filesystem of the buildroot using the following command:

```
$ make INSTALL_MOD_PATH=<path_to_buildroot>/package/customize/source
modules_install
```

Then the root filesystem must be rebuilt and reflashed on the STreamPlug board.

Building modules from wireless backport

In order to build and install the kernel modules from the wireless backport the following environment variables must be defined:

```
$ export KLIB_BUILD= <path_to_linux_src>
$ export KLIB = <path_to_buildroot>/package/customize/source
```

After that, the commands to be used to configure, build and install the backport are:

```
$ make defconfig-<selected configuration>
$ make
$ make install
```

5.2 Building and loading firmware image

Several steps are needed to build a new image that can be loaded on the Flash. This guide explains how to build the Linux kernel (vmlinux file), and the root filesystem. See the UM2004 - “STreamPlug ST2100 SDK and quick start guide” user manual for more information and details on building images and loading firmware.

6 Glossary

The list of abbreviations used in this STreamPlug getting started guide is listed in *Table 5*.

Table 5. List of abbreviations

Term	Definition
AMBA	Advanced microcontroller bus architecture
API	Application programming interface
ARM	Advanced RISC machine
AVB	Audio Video Bridging
BSP	Board support package
C3	Channel controller coprocessor
CAN	Controller area network
CPU	Central processing unit
DDR	Double data rate (RAM)
DDR	Double data rate SDRAM
DHCP	Dynamic host configuration protocol
DMA	Direct memory access
DWC	Designware cores
EEPROM	Electrically erasable programmable read only memory
EP	PCIe endpoint device
FAT	File allocation table
FS	Filesystem
FSMC	Flexible static memory controller
FTP	File transfer protocol
FW	Firmware
GCC	GNU compiler collection
GPIO	General purpose input/output signal
GPL	General public license (GNU)
I^2C	Inter-integrated circuit
I^2S	Inter-IC sound
IDE	Integrated development environment
IP	Internet protocol
IPs	Intellectual Properties
JPEG	Joint photographic experts group
JTAG	Joint test action group
KSP	Kernel support packages provided by OKL Microvisor

Table 5. List of abbreviations (continued)

Term	Definition
LAN	Local area network
LGPL	Lesser GPL
LSP	Linux support package
MAC	Media access control
MFIO	Multifunction input/output signal
MTD	Memory technology device
NFS	Network filesystem
OKL	Open Kernel Lab
OOB	Out-of-band
OS	Operating system
PC	Personal computer
PCIe	Peripheral component interconnect express
RAM	Random access memory
RC	PCIe root complex device
RevMII	Reverse media independent interface
RPM	RPM package manager
RTC	Real-time clock
SATA	Serial advanced technology attachment
SCP	Secure copy Linux command
SDK	Software development kit
SDRAM	Synchronous dynamic random access memory
SMI	Serial Management Interface
SoC	System-on-chip
SPI	Serial Peripheral Interface bus
SPORT	Serial port
SRAM	Static RAM
TAG	Tagged List
TFTP	Trivial file transfer protocol
UART	Universal asynchronous receiver transmitter
USB	Universal serial bus
VIC	Vectored interrupt controller

7 Revision history

Table 6. Document revision history

Date	Revision	Changes
02-Feb-2016	1	Initial release.