TS8 PLE firmware specifications

Version: 54d283d 10.02.2016

Introduction

This document gives an overview of PiccoLink Emulation (PLE) firmware specifications. PLE is a default firmware (FW) for TS8 mobile barcode readers and RFID scanners. We also look at software interfaces applicable with TS8 hardware.

Target audience: firmware developers, testers and firmware maintainers.

Firmware description

TS8 platform runs an embedded Linux operating system with a custom root filesystem.

Main components of the firmware are:

- Customised Universal Boot Loader - Das U-Boot
- Das U-Boot environment variables and bootsplash image
- JFFS2 root filesystem
- Linux kernel version 2.6.33.3
- Busybox version 1.6.11
- TS8 hardware drivers, communication module drivers
- Applications and scripts

Flash memory layout

For better flexibility, internal NOR flash is divided into a number of partitions. The complete NOR flash range is 0x00000000 - 0x01000000 (total size 16MB). The table below lists the partitions.

Address Range	Blocks*	Flash Device	Description
0x00000000 - 0x00020000	1:0	/dev/mtd1	Contains U-Boot bootloader
0x00020000 - 0x00040000	1:1	/dev/mtd2	Contains U-Boot configuration block
0x00040000 - 0x00080000	1:2-3	/dev/mtd3	Contains splash graphics in raw format
0x00080000 - 0x00200000	1:4 - 15	/dev/mtd4	Accommodates Linux kernel
0x00200000 - 0x00F60000	1:16 - 122	/dev/mtd5	Contains JFFS2 root file system
0x00F60000 - 0x01000000	1:123 - 127	/dev/mtd6	Contains JFFS2 configuration file system

* The “Blocks” field identifies flash memory blocks that have been allocated for the given partitions, in U-Boot format. The first number identifies the relevant flash chip. The size of each block is 128KB.

Partitions included in firmware files

The table below lists the contents (partitions) of different firmware files.

File name	Bootloader	Configuration block	Splash graphics	Kernel	Root file system	Configuration file system	Description
ts8-ple-fw.pkg	X		X	X	X		Firmware update package
ts8-ple-recovery.bin	X		X	X	X	X	Firmware recovery image
ts8-ple-init.bin	X	X	X	X	X	X	Initialisation image
ts8-ple-kernel-upd.bin				X			Kernel update image
ts8-ple-uboot.bin	X						Bootloader image

Bootloader

U-Boot environment variables

U-Boot variables are stored in U-Boot's configuration block. These variables can be changed from U-Boot's command prompt. Alternatively, the variables can be set from either Linux shell using `usetenv` command or from an application.

NB! Be careful when editing U-Boot variables! Always back up any variable values before you edit them. Errors in some values might render the system unbootable or disable some functionality.

Variables are used to store device-specific information in addition to bootloader configuration. The following table lists variables with their possible values and descriptions.

Variable	Value	Description
baudrate	115200	Console baudrate (in bps)
bootcmd	bootm 0x10080000	Defines a command string that is automatically executed when the initial countdown is not interrupted
bootargs	*	Boot arguments
serial	000001	Device serial number
ft	ple	Firmware type
model	TS8	Device model
hwrev	1	Device mainboard revision
mfdate	1	Device manufacturing time in UNIX time
cmtype	1	Communication module type: 0-no wireless, 1-GPRS, 5-WLAN, 8-Bluetooth
bootdelay	0	Delay bootcmd in seconds
test	0	Device startup mode (see launcher source code for values)
stdin	serial	Standard input
stdout	serial	Standard output
stderr	serial	Standard error

* Boot arguments:

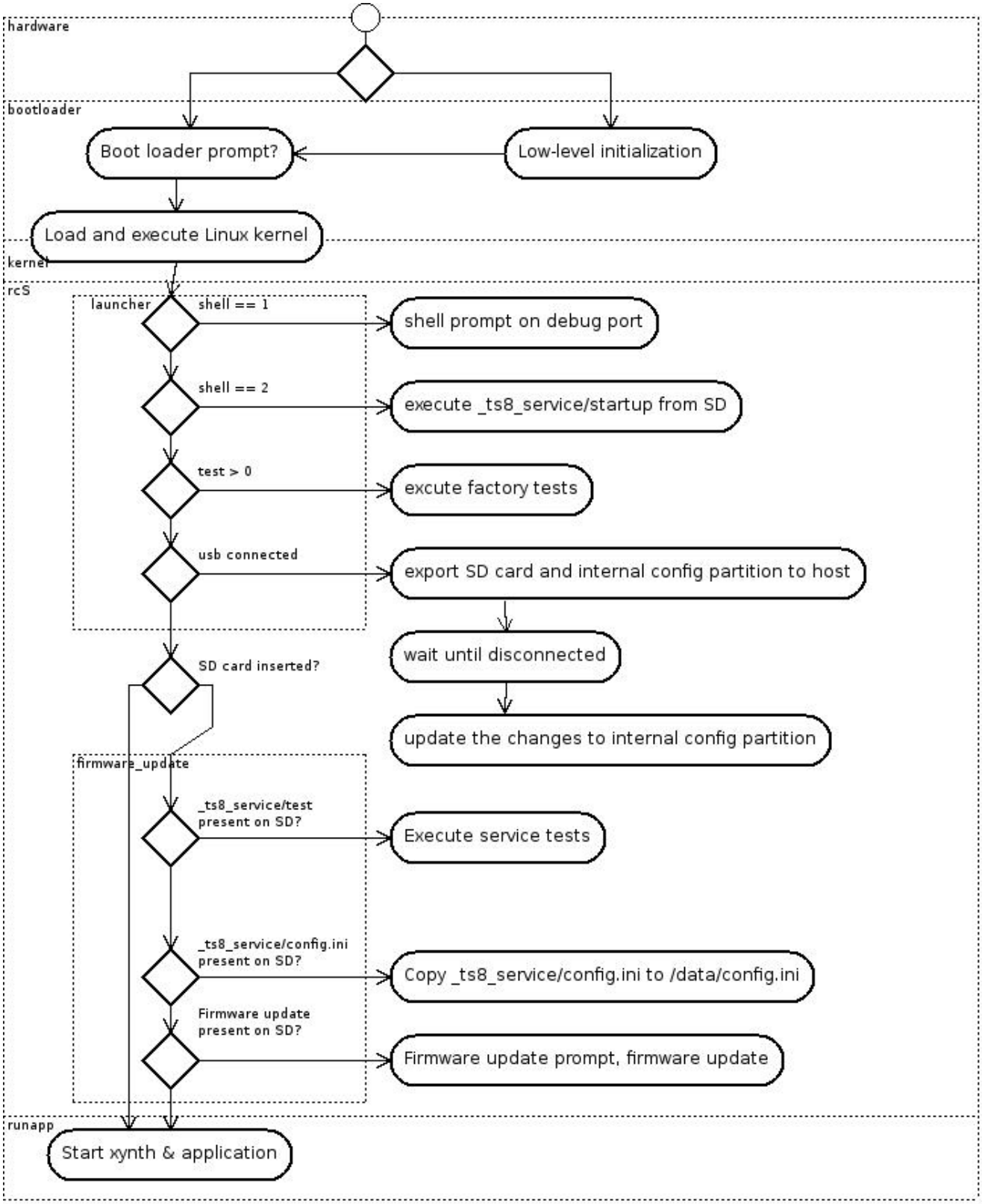
```
mtdparts=physmap-flash.0:16M@0x00000000(flash),128K@0x00000000(u-boot),128K@0x00020000(conf),256K@0x00040000
(splash),1536K@0x00080000(kernel), 13696k@0x00200000(rootfs),640k@0x00F60000(conffs) root=/dev/mtdblock5 ro
rootfstype=jffs2 console=null
```

Splash graphics

Bootsplash is a raw format image, which is displayed during TS8's boot-up.

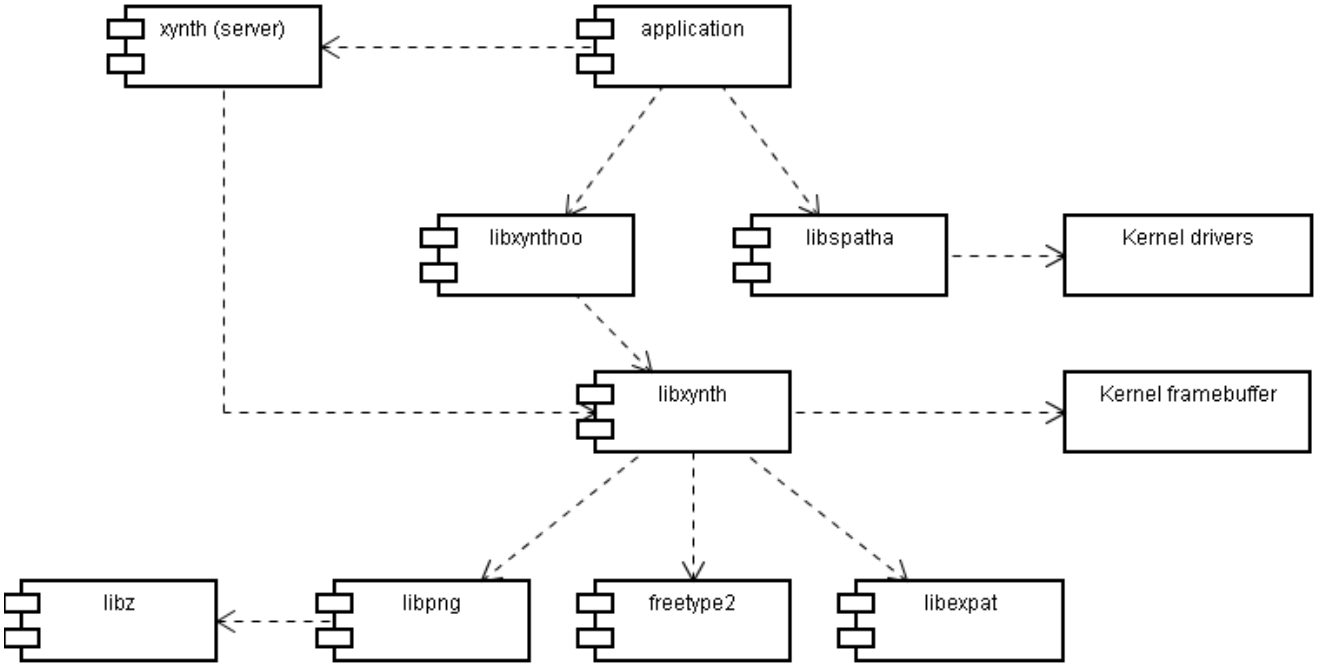
Boot process and options

Boot order is determined by U-Boot variables, any files found on memory card and the state of USB connection. The drawing below illustrates the boot process.



Application Linkage

The diagram below illustrates how the application is linked to corresponding dependencies.



Interfaces and API

Documentation

This section refers to system interfaces and API that are required when developing user applications for TS8.

In order to simplify application development process, we have provided a hardware support library called `libspatha`. Libspatha library encapsulates a number of C++ classes that support most of the hardware interfaces available for TS8 applications.

eXynthoo is a GUI class library that exists as an extension to the eXynth GUI engine. This library is designed for C++ applications that are run from the client side of the GUI engine and is relevant for embedded devices with simple static interfaces.

`libspatha` and `libexynthoo` documentation can be found from FDK repository.

TS8 console access

TS8 comes with a dedicated serial port for development purposes. This can be accessed via `/dev/ttyAT0` device. The port is typically used to run a console with a Busybox shell and allows access to system internals.

Storage Media

NOR flash

NOR flash is available as a MTD device. Use mtdparts command line option to specify partition start addresses and lengths. The flash can be accessed via `/dev/mtdX` as a character device or via `/dev/mtdblockX` as a block device. Block device uses block driver wrapper around an MTD interface (X marks partition's number.). Refer to the Flash Memory Layout section for more information about available flash partitions. We do not recommend using a regular read-write filesystem on a mtdblock device, as its unoptimised access patterns will wear out the flash very fast.

For NOR flashes, new flash chips do not contain bad blocks and the flash blocks will last for a number of writes. In case of a rarely written partition (e.g. bootloader or kernel), the data can reside on the partition itself and no bad block handling is needed.

Usage

For raw flash access without bad block handling use `/dev/mtdX` interfaces. Before writing data onto a flash area, use utility `flash_erase` or `flash_eraseall` to clean up the corresponding area. Flash area can be read and written like any regular character device (e.g. with the dd utility).

We strongly recommend that you use JFFS2 file system for filesystems (especially read-write filesystems). JFFS2 shuffles data blocks and distributes them evenly across the whole partition. This prevents wearing out some of the more frequently written areas of the flash (e.g. FAT table locations).

More information:

- http://www.linux-mtd.infradead.org/faq/general.html
- http://sourceware.org/jffs2/

Micro-SD Card

User space applications can access SD memory card from `/dev/mmcblk0` interface as a regular block device. If a partition table exists on the SD card, the partitions are available as `/dev/mmcblk0pX` (where X indicates partition's number). Libspatha uses `/dev/mmcblk0p1`. SD card can either contain one partition or only it's first partition can be used. If needed, memory cards can be formatted without a partition table (creating a FAT filesystem on /dev/mmcblk0); however, we strongly recommend leaving the card as it is.

Memorycards can be swapped while the device is running. **Bear in mind that it should be unmounted first.** Do not swap the card while the device is in sleep mode, because the device will not detect card change.

Any buffered data from the initial card can be saved onto the new card.

Usage

Mount your card to a mount point using vfat filesystem:

```
# mount -t vfat /dev/mmcblk0p1 /mnt/mmc
```

Do not leave the FAT filesystem mounted in read-write mode when you are not using the filesystem. This can result in data loss, as some changes may be buffered and not written back to the card when the file system is unmounted. We recommend remounting the filesystem as read-only if it is not used for writing:

```
# mount -t vfat /dev/mmcblk0p1 /mnt/mmc -o remount,ro
```

Communication interfaces

TS8-G11 GPRS communication module

The TS8-G11 GPRS communication module incorporates a Telit GE864 GSM/GPRS modem, uses USART1 serial port for data communication and some extra GPIO pins as control lines. In addition, system's power management driver should be used to tristate (high-z) the USART when the device is powered down, otherwise power will leak via USART lines.

For communication, use `/dev/ttyAT1` serial port. This is a full-featured serial port with two-directional data transfer and hardware flow control. Enable the flow control to avoid flooding modem's input buffer with large amounts of data. Modem is usually used at 115200baud/sec baud rate. Configure the serial port, and connect to modem. It will do autobauding and detect the baud rate. On first use after powering up, use AT commands to set the serial port's baud rate on the modem's side to a fixed value. This will prevent autobauding from recalculating baud rate to wrong value.

The modem has 4 output and 1 input GPIO pins (see System Power Management section for control):

- `O_PIN_GPRS_PWR` – Power switch on main board. This can be used to turn off the modem's power and for hard-resetting the modem. The switch is defaulted to ON position (even when the main CPU is powered off) to power the RTC in the modem. This switch is intended only for resetting the modem in case it stops responding. It should not be used for switching off the modem.
- `O_PIN_GPRS_RST` – Modem's RESET input. This pin can be used to generate a reset pulse to the modem.
- `O_PIN_GPRS_ON` – Modem's ON input. Generate a pulse on this pin to power up the modem. Note that the U-Boot bootloader generates a ~1s pulse automatically when the system is started. This is done to save time. The application should first check the I_PIN_GPRS_PWRGD signal state to see if the modem is already powered up.
- `O_PIN_GPRS_DTR` – Modem's DTR input. This is used to control modem's power save state.
- `I_PIN_GPRS_PWRGD` – This input indicates modem's power good state. This can be used to check if the modem is booted up.

Usage

Given below are sample shell commands, which use pwrctrl write interface to control GPIOs. A compiled application should use the IOCTL interface instead. Refer to the System Power Management section for more information.

```
# echo 'o05'>/dev/pwrctrl               # cut power to GPRS modem
# echo 'O05'>/dev/pwrctrl               # re-enable power
# sleep 1
# echo 'O07'>/dev/pwrctrl               # generate GPRS\_ON line pulse
# sleep 1
# echo 'o07'>/dev/pwrctrl
# stty -F /dev/ttyAT1 -echo crtscts     # setup serial port
# cat /dev/ttyAT1 &                     # dump serial port input
# echo -e "AT+CGMR\\r" > /dev/ttyAT1    # send modem version request command
```

More information:

- http://www.telit.com/products/cellular/2g

TS8-W11 WLAN communication module

The TS8-W11 WLAN communication module is built on Ralink RT3070 chipset and uses USB for data communication and signaling. The module is supported by the rt3070sta driver implemented as a loadable kernel module. The driver is supplied with FDK and is built automatically within the firmware build tree. The communication module appears as a conventional wireless network interface ra0 to the operating system.

Additionally, WLAN module can be controlled using the following GPIO pins (see System Power Management section):

- `O_PIN_MPCIE3VEN` – Controls power supply to WLAN communication module. Default: ON
- `O_PIN_WLAN_RADIO_ON` – Enables communication module's radio interface. Default: OFF

Usage:

```
# insmod /lib/modules/wireless/rt3070sta.ko
# ifconfig ra0 up
# iwpriv ra0 set NetworkType=Infra
# iwpriv ra0 set AuthMode=WPA2PSK
# iwpriv ra0 set EncrypType=AES
# iwpriv ra0 set SSID="YourNetwork"
# iwpriv ra0 set WPAPSK="NetworkKey"
# udhcpc -nR -i ra0
```

For more information see `kernel/drivers/wireless/` in the firmware build tree.

USB device interface

USB device port is handled by the Linux USB gadget subsystem. There are various drivers, these are compiled as modules and can be loaded and swapped as needed.

USB mass storage gadget

This gadget can be used to display one or more block devices or image files as USB mass storage devices. These show up as regular mass storage drives on host computer and the host can copy files via USB link. Always unmount the block device before giving it to USB mass storage gadget driver, as the file systems can not be accessed simultaneously from multiple locations.

For example, a SD card can be shared across the USB to create a USB SD card reader:

```
# modprobe at91_udc.ko
# modprobe g_file_storage.ko stall=0 removable=y
# echo '/dev/mmcblk0p1'>/sys/devices/platform/at91_udc/gadget/gadget-lun0/file
```

USB ethernet gadget

This gadget can be used to create an ethernet connection between the host and the device, using USB. Basically, it shows up on the host and the device as an ethernet adapter, acting as there were two ethernet adapters connected to each other with a crossover cable. The virtual ethernet adapter can be bridged on the host's side to allow access to physical network via the host. This network configuration can be used to easily share files between the host and the device. All common network tools are applicable; the device can mount a NFS directory from host and run applications directly from the host's filesystem. The device can also run a GDB server for remote application debugging controlled from the host.

Given below is a sample shell script that configures the usb0 network interface and mounts an NFS share:

```
# modprobe at91_udc.ko
# modprobe g_ether.ko
# ifconfig usb0 192.168.1.2
# mount -t nfs -o nolock,nfsvers=2 192.168.1.2:/home/user/share /mnt/nfs
```

NB! Network adapters' MAC addresses are generated randomly in a range allowed for local use. If you have multiple devices set up into same network (e.g. connected to the same computer or bridged to the same network), ensure that the devices' random number generators are not generating same numbers. In order to do that, set the system time from RTC before loading USB ethernet gadget driver, or allocate MAC addresses manually. Otherwise you may encounter MAC address collisions.

More information:

- http://www.linux-usb.org/gadget/

Bluetooth interface

The Bluetooth chip is connected via external 16550 UART compatible serial port chip, and can be accessed via `/dev/ttyS0` serial port. The chip needs to be initialised on power-up according to its data sheet. Bluetooth uses MAC addresses and the end user needs to provide a unique address for every device during their power-up initialisation.

More information:

- http://www.national.com/pf/LM/LMX9830.html

Scanner interfaces

TS8-RB1R0 barcode scanner

The TS8-RB1R0 reader module incorporates an Opticon 1D barcode laser scanner. The barcode scanner's data connection is provided via USART2 serial port and it uses some GPIO pins for control. In addition, system's power management driver is used to tristate the USART when the device is powered down.

Reader's data port can be accessed via /dev/ttyAT2. The reader is defaulted to 9600 baud, 8N1 . The reader has its own internal configuration flash and all the changes in settings are stored. **NB! Take care that you do not lock yourself out when re-configuring the device!**

The following GPIO pins are used for control (see the System Power Management section):

- `O_PIN_2D_PWR` – Controls the power supply to turn ON/OFF the 2D reader's power. Default: OFF
- `O_PIN_2D_DWNLD_N` – Controls the reader's DWNLD# pin. Usage not documented.
- `O_PIN_2D_AIM` – Controls the reader's AIM/WAKEUP pin.
- `I_PIN_2D_PWR_DWN` – Indicates the reader's PWR_DWN pin state.

To read a barcode, power up the reader (set pins `DWNLD_N` and `AIM` high, then set pin `PWR` high), wait until the reader powers up, issue a continuous trigger command `<ESC>Y0<CR>`, issue a trigger command `<ESC>Z<CR>`, wait for data, issue a de-trigger command `<ESC>Y<CR>`, optionally power down the reader (tristate the serial port and set pins `PWR`, `DWNLD_N`, `AIM` low).

More information:

- http://old.opticon.com/uploads/Manual/Menubook_en.zip

User interfaces

In order to simplify graphical user interface (GUI) development, we provide an optimised version of Xynth GUI engine and a custom GUI layout library called libexynthoo with the TS8 applications.

The items listed below describe a direct access to the mobile data terminal's user interface hardware. Direct access may be desired for smaller utilities, for performance critical applications, or for any other case where the Xynth approach is not applicable.

OLED display

OLED display is provided to application as a frame buffer device at `/dev/fb0`. This application can map frame buffer memory using that interface or use the interface directly to read/write frame buffer. Frame buffer can also be read using common shell commands like dd, or even by redirecting some command returning standard output to frame buffer device. For example: gzipped raw frame buffer images displayed below:

```
# zcat file.gz>/dev/fb0
```

More information:

- http://www.linux-fbdev.org/

Keypad

Keyboard is available as an event device. An application can access the keyboard via `/dev/eventX`. Application should loop through event devices and poll `EVIOCGBIT` command to get event types provided by that event device. The application should find the event device providing `EV_KEY` support.

The following table lists possible key codes.

Key	Description
KEY_0 .. KEY_9	Numeric keys 0..9
KEY_BACKSPACE	DEL key
KEY_ENTER	OK key
KEY_DOT	Key for special characters (located at the lower left corner)
KEY_UP, KEY_DOWN	UP and DOWN arrow keys
KEY_F1 .. KEY_F5	Function keys F1..F5
KEY_F11	Shift key (located at the lower right corner)
KEY_MENU	MENU key
KEY_POWER	Scan / Power key

SCAN-button is also used as power button. When connecting the device to a computer via USB, the device powers on. Due to this, a pulse is generated on SCAN button by rising +5V in USB connector. The application can poll the USB +5V line via system power management driver to filter out button presses caused by plugging in the USB.

More information:

- http://www.linuxjournal.com/article/6429
- `utils/get\_keypress.c` utility in the firmware build tree

Speaker

Speaker is available as an event device at `/dev/input/eventX`. This is a special case of event device, which can control output devices. Poll `EVIOCGBIT` command to find an event device providing `EV\_SND` support. Send an event with value equal to the frequency to start the speaker oscillator. Send frequency 0 to stop the speaker oscillator.

See also `applications/utils/sprktest.c` utility in the firmware build tree

LED indicators

Two LED indicators on top of the keypad are available to user applications. The following GPIO pins are used for control (see the System Power Management section):

- `O_PIN_LED_R` – Enables the red LED indicator
- `O_PIN_LED_G` – Enables the green LED indicator

When the mobile terminal is connected to a host via USB, the LED outputs are driven by battery charger circuitry. The software may then override charger outputs to enable LED indicators.

Ambient light sensor

Relevant data can be read from the on-board `ISL29001` sensor from `/sys/devices/platform/i2c-gpio/i2c-adapter:i2c-0/0-0044/illum1_input`.

System interfaces

CPU power management

Atmel AT91 SoC is capable of switching the system clock to RTT oscillator (32kHz) and stopping fast PLLs and the main oscillator to save power. With all peripherals off and system idle, the current consumption (measured between the battery) is ~80mA, when the system clock is derived from the main oscillator, and ~20mA, when the system clock is switched to RTT oscillator and fast PLLs and the main oscillator are stopped.

To go to the low power mode, write “mem” to /system/power/state.

```
# echo “mem” > /system/power/state
```

The system will wake up when either the RTT (real-time timer, see below) is interrupted or when another configured wake-up source (for example, buttons) is activated. By default, all wake-up sources are enabled, but they can be disabled via sysfs.

System power management

The power manager driver is used to control system's power supplies (via GPIO pins), to adjust keyboard backlight, tristate USART ports and control GPIO ports. The driver can be controlled using either IOCTL interface or write interfaces via `/dev/pwrctrl`.

With IOCTL interface (unsigned long command, unsigned long argument):

IOCTL Command	Argument	Description
PWRCTRL_SET_GPIO_HI	bits 0-7: gpio out id	Set GPIO output pin ID high (pin ID's are defined in pwrctrl.h)
PWRCTRL_SET_GPIO_LO	bits 0-7: gpio out id	Set GPIO output pin ID low
PWRCTRL_GET_GPIO	bits 0-7: gpio out id	Get the GPIO input pin ID state
PWRCTRL_SET_KBDBL	bits 0-7: intensity	Set the keyboard backlight intensity to a defined value
PWRCTRL_FADE_KBDBL_TO	bits 0-7: final intensity	Fade the keyboard backlight to defined final intensity in 25 ms steps
USART_TRISTATE_ON	bits 0-7: usart id	Mux the USART ID pins to high-z state
USART_TRISTATE_OFF	bits 0-7: usart id	Mux the USART ID pins to USART

With write interface (argument xx is in HEX):

Write command	Description
0xx	Set GPIO output pin xx high (pin ID's are defined in pwrctrl.h)
oxx	Set GPIO output pin xx low
Lxx	Set the keyboard backlight intensity to xx
Axx	Fade the keyboard backlight to intensity xx in 25 ms steps
Txx	Mux the usart xx pins to high-z state.
txx	Mux the usart xx pins to USART.

Write interface is intended to simplify the device's control from shell scripts, i.e.:

```
\# echo “AFF” > /dev/pwrctrl
```

Real-time timer

AT91 contains a real-time timer. This timer is from 32768Hz oscillator and it is running even while the system is powered off. The timer counter is incremented at 1Hz and can only be read, not written. The counter is set to 0 when the device is connected to a battery. The RTT is capable of generating an alarm interrupt and waking up the system from suspended state or power-down state.

The real-time timer serves mainly two purposes: keeping track of time while the system is in suspend and waking the system up from its suspended state at a predefined interval.

Access to the RTT timer is provided through standard Linux RTC interface.

Battery voltage

Device's board contains a dedicated chip for reading battery voltage. The value can be read at `/sys/devices/platform/atmel_spi.1/spi1.3/in_input0`.

Temperature sensor

The temperature can be read from the on-board LM75 temperature sensor from `/sys/devices/platform/i2c-gpio/i2c-adapter:i2c-0/0-0048/temp1_input`.

TS8 configuration concept

Partitions and configuration storage process

A TS8, running on PLE firmware, is configured via single configuration file. Depending from the configuration type, additional files might be needed.

Device's configuration file is stored on a dedicated flash partition (`/dev/mtd6`), size 640 kB. This partition is writable by default. Files on this partition can be accessed from the host PC; this, however, is not similar to how the device's memory card is accessed from the host PC via USB storage gadget.

When a USB cable is connected to a switched off device, a "storage device" (a loop device `/dev/loop0`) is created from an empty file (size 640 kB). A volume with MSDOS filesystem is created to the loop device and predetermined files are copied to the partition.

The volume that was created, labeled `TS8-config`, is visible to the host PC as a regular removable storage device.

When the USB cable is disconnected, pre-determined files are copied back to the configuration partition.

This approach enables to control what files are stored in flash, and prevent the host PC from storing excess files, such as trash, indexes and similar.

The process described above is controlled via scripts `connect_usb_storage` and `disconnect_usb_storage` available in root file system.

Configuration partition content

The following files are copied between the configuration partition and a temporary volume `TS8-config`. These files are used by various programs running in the TS8.

- `*.ini` - Ini file format configuration files
- `*.pem *.der *.prv *.key` - Certificates and keys
- `*.conf` - Conf file format configuration files
- `*.txt` - Text files for configuration documentation and identification

In addition, the following information is copied to `TS8-config` file, but not copied back to the file system:

- release_date - Contains firmware build date
- release_version - Contains firmware version

Those are text files, although they do not have relevant extension. These files can be viewed with a text editor.