CPU Board RD129

English manual V.1.11 – 07/05/2011

@ 2010 ELPA sas - Italy

General Index

1 Hardware

1.1 General Description

The board is a tiny module of size 40x45mm, on the bottom side there are 2 identical 0.8mm pitch SMD connectors, that are used also to fix the board to the carrier board.

Used microcontroller has the following integrated peripherals:

- LCD Controller (STN or TFT)
- Touchscreen controller
- 3 Uarts, 1 has hardware handshaking (RTS/CTS)
- 2 SPI
- 1 I2Cbus
- 1 I2S to drive an external stereo audio codec
- 1 10 bits, 8 inputs ADC (2 inputs are used for touchscreen)
- 1 JTAG (for low-level debug)
- 5 timers, 1 of them is used by Linux
- 1 MMC or SD external memory slot
- 2 USB 1.1 full speed (12Mbps), 1 is a host port and the other is configurable like host or device

The CPU can be clocked at 200, 240, 250 or 266MHz, and communicates with an external SDRAM through a 32-bit bus at half of main clock speed. Depending on the board's version you have, there can be 16MB, 32MB or 64MB of ram and 32MB or 64MB of flash.

1.2 Board's version

On the board is applied a sticker, with the serial number and a 4 characters code identifying the configuration:

The 1st character identifies the quantity of SDRAM: 4 means 16MB, 5 = 32MB, and 6 = 64MB.

The 2nd character identifies the CPU clock speed: A means 200MHz, B = 250MHz, C = 266MHz, D = 240MHz. The SDRAM is always clocked at half of the CPU speed.

The 3rd character is the quantity of Flash: 5 means 32MB, 6 = 64MB.

The 4th character is the temperature range: C is commercial 0..+70C, E is extended -25..+85C, I is industrial -40..+85C

By now, the most popular version is 5D6E (32MB ram, 240MHz, 64MB flash, -25..+85C).

1.3 Handling precautions

The board contains static sensitive devices, so standard procedures for handling ESD-sensitive materials are required.

Special attention is required to insert and remove the board from its supporting baseboard connectors. Please insert/remove the card without power supply applied, and applying force orthogonal to the circuit plane. Don't remove the card by rotating it on one connector, as this can severely damage the card. The 2 connectors should be (un)plugged at the same time.

1.4 List of CPU signals

The board has 2 60 pin connectors, giving a total of 120 contacts. 12 are dedicated for power, 5 for Jtag port, 9 for A/D converter, and 4 for USB ports.

All others 90 signals may be used for integrated peripherals or like generic I/O lines.

In the I/O column there is the I/O processor's port which is connected to the signal. The 6 signals connected to port A can be configured only for output.

All signals work at 3.3V power supply, the only exceptions are signals G8, G9, G10 that are 5V tolerant.

Name	Pos	Peripheral	I/O	Description
VDD	1	Power		3.3V +-5%
VDD	2	Power		3.3V +-5%
VDEN	3	Lcd	C4	Video data enable (TFT) or VM (STN)
TCLK1	4	Timer	G11	Clock input for timers 0 and 1
VCLK	5	Lcd	C1	Video data clock
TCLK0	6	Timer	B4	Clock input for timers 2 and 3
HSYNC	7	Lcd	C2	Video horizontal sync
TOUT3	8	Timer	B3	Timer 3 output
VSYNC	9	Lcd	C3	Video vertical sync
TOUT2	10	Timer	B2	Timer 2 output
VD0	11	Lcd	C8	Video data STN (LSB) or TFT (blue for TFT, LSB)
TOUT1	12	Timer	B1	Timer 1 output
VD1	13	Lcd	C9	Video data STN or TFT (blue for TFT)
TOUT0	14	Timer	B0	Timer 0 output (for STN contrast or TFT luminosity)
VD2	15	Lcd	C10	Video data STN or TFT (blue for TFT)
LCDEN	16	Lcd	G4	LCD enable
VD3	17	Lcd	C11	Video data STN or TFT (blue for TFT)
VLEND	18	Lcd	C0	Video Line End, signal required from some TFT device
VD4	19	Lcd	C12	Video data STN (8 bit only) or TFT (blue for TFT)
SDA	20	I2C	E15	I2Cbus data
VD5	21	Lcd	C13	Video data STN (8 bit only) or TFT (blue for TFT)
SCL	22	I2C	E14	I2Cbus clock
VD6	23	Lcd	C14	Video data STN (8 bit only) or TFT (blue for TFT)
TMS	24	Jtag		TAP Controller Mode Select
VD7	25	Lcd	C15	Video data STN (8 bit only, MSB) or TFT (blue for TFT, MSB)
TDO	26	Jtag		TAP Controller Data Output
VD8	27	Lcd	D0	Video data TFT green (LSB)

Name	Pos	Peripheral	I/O	Description
TDI	28	Jtag		TAP Controller Data Input
VD9	29	Lcd	D1	Video data TFT green
TCK	30	Jtag		TAP Controller Clock
VD10	31	Lcd	D2	Video data TFT green
nTRST	32	Jtag		TAP Controller Reset
VD11	33	Lcd	D3	Video data TFT green
SS1	34	Spi 1	G3	Chip select (only for slave mode)
VD12	35	Lcd	D4	Video data TFT green
SS0	36	Spi 0	G2	Chip select (only for slave mode)
VD13	37	Lcd	D5	Video data TFT green
SDDATA3	38	Mmc/sd	E10	SD data (MSB)
VD14	39	Lcd	D6	Video data TFT green
SDDATA2	40	Mmc/sd	E9	SD data
VD15	41	Lcd	D7	Video data TFT green (MSB)
SDDATA1	42	Mmc/sd	E8	SD data
VD16	43	Lcd	D8	Video data TFT red (LSB)
SDDATA0	44	Mmc/sd	E7	MMC/SD data (LSB)
VD17	45	Lcd	D9	Video data TFT red
SDCMD	46	Mmc/sd	E6	MMC/SD command or answer selector
VD18	47	Lcd	D10	Video data TFT red
SDCLK	48	Mmc/sd	E5	MMC/SD clock
VD19	49	Lcd	D11	Video data TFT red
I2SSDO	50	I2S	E4	Audio codec's output data
VD20	51	Lcd	D12	Video data TFT red
I2SSDI	52	I2S	E3	Audio codec's input data
VD21	53	Lcd	D13	Video data TFT red
CDCLK	54	I2S	E2	Audio codec's system clock
VD22	55	Lcd	D14	Video data TFT red
I2SSCLK	56	I2S	E1	Audio codec's data clock
VD23	57	Lcd	D15	Video data TFT red (MSB)
I2SLRCK	58	I2S	E0	Audio codec's left/right selector
GND	59	Power		0V
GND	60	Power		0V
VDD	61	Power		3.3V +-5%
VDD	62	Power		3.3V +-5%

Name	Pos	Peripheral	I/O	Description
VDD	63	Power		3.3V +-5%
VDD	64	Power		3.3V +-5%
GND	65	Power		0V
GND	66	Power		0V
TXD2	67	Uart 2	H6	UART 2 TX data (used for debug)
TXD1	68	Uart 1	H4	UART 1 TX data
RXD2	69	Uart 2	H7	UART 2 RX data (used for debug)
RXD1	70	Uart 1	H5	UART 1 RX data
GPA12	71	Output signal	A12	Output only
TXD0	72	Uart 0	H2	UART 0 TX data
GPA13	73	Output signal	A13	Output only
RXD0	74	Uart 0	H3	UART 0 RX data
GPA14	75	Output signal	A14	Output only
RTS0	76	Uart 0	H1	UART 0 handshake (output)
GPA15	77	Output signal	A15	Output only
CTS0	78	Uart 0	H0	UART 0 handshake (input)
GPA16	79	Output signal	A16	Output only
INT8	80	Interrupt	G0	Interrupt 8 (standard kernel uses it for SD Card Detect)
INT0	81	Interrupt	F0	Interrupt 0
INT9	82	Interrupt	G1	Interrupt 9 (standard kernel uses it for SD Write Protect)
INT1	83	Interrupt	F1	Interrupt 1
INT16	84	Interrupt	G8	Interrupt 16 (5V tolerant)
INT2	85	Interrupt	F2	Interrupt 2
INT17	86	Interrupt	G9	Interrupt 17 (5V tolerant)
INT3	87	Interrupt	F3	Interrupt 3
INT18	88	Interrupt	G10	Interrupt 18 (5V tolerant)
INT4	89	Interrupt	F4	Interrupt 4
AIN7	90	Adc/touch		Analog input 7
INT5	91	Interrupt	F5	Interrupt 5
AIN6	92	Adc		Analog input 6
INT6	93	Interrupt	F6	Interrupt 6
AIN5	94	Adc/touch		Analog input 5
INT7	95	Interrupt	F7	Interrupt 7
AIN4	96	Adc		Analog input 4
RSTOUT	97	Reset	A21	Reset to external peripherals (output only)

Name	Pos	Peripheral	I/O	Description
AIN3	98	Adc		Analog input 3
USBP0	99	Usb host		Main USB host port (positive)
AIN2	100	Adc		Analog input 2
USBM0	101	Usb host		Main USB host port (negative)
AIN1	102	Adc		Analog input 1
USBP1	103	Usb host/dev.		Aux USB port, host or device (positive)
AIN0	104	Adc		Analog input 0
USBM1	105	Usb host/dev.		Aux USB port, host or device (negative)
VREF	106	Adc		AD converter reference voltage
MISO0	107	Spi 0	E11	Master input, slave output
CLKOUT0	108	Clock gener.	H9	Programmable clock generator
MISO1	109	Spi 1	G5	Master input, slave output
CLKOUT1	110	Clock gener.	H10	Programmable clock generator
MOSI0	111	Spi 0	E12	Master output, slave input
XMON	112	Touchscreen	G12	External gate of touchscreen driving mosfet
MOSI1	113	Spi 1	G6	Master output, slave input
nXPON	114	Touchscreen	G13	External gate of touchscreen driving mosfet
SCLK0	115	Spi 0	E13	Data clock
YMON	116	Touchscreen	G14	External gate of touchscreen driving mosfet
SCLK1	117	Spi 1	G7	Data clock
nYPON	118	Touchscreen	G15	External gate of touchscreen driving mosfet
GND	119	Power		0V
GND	120	Power		0V

By looking at the CPU board from the top side (connectors are on the bottom side) with the crystal toward the bottom, pins are located:

Pin 1 is on the right top of topmost connector

Pin 2 is on the right bottom of topmost connector

Pin 59 is on the left top of topmost connector, near J2 string (this is pin 1 of the topmost connector).

Pin 60 is on the left bottom of topmost connector

Pin 61 is on the right top of bottommost connector

Pin 62 is on the right bottom of bottommost connector

Pin 119 is on the left top of bottommost connector (this is pin 1 of bottommost connector).

Pin 120 is on the left bottom of bottommost connector (near J1 string)

The pitch between the 2 connectors is 30 mm, and are vertically centered around the board 40 mm height.

The connectors' centers are located at 15 mm from the boards' left side and 30 mm from the boards' right side.

Connector's pitch is 0.8 mm, FH series produced by AMP / Tyco .

A compatible connector is produced by FCI.

1.5 RD126 starter kit details

2 kinds of starter kit exists: RD126 and RD126c.

Both starter kits have:

- Main power standard connector, needs only +5Vdc, mechanical compatible with ide hard-disk power connectors
- Reset pushbutton (bigger on RD126c)
- 2 RS232 ports
- 1 SD/MMC connector
- 1 USB host connector
- TFT LCD 240x320 3.5” connector
- TFT LCD 320x240 5.7” connector and additional touch-screen connector
- 3.5mm jack stereo audio output
- 1 Ethernet connector (usb to ethernet converter)
- Real Time Clock chip with 56 bytes ram and battery backup (connected to i2cbus)

Only RD126 has:

- JTAG connector
- STN LCD 320x240 5.7” connector
- 3.5mm jack stereo audio input
- can accomodate high voltage inverter for LCD CCFL backlighting
- eeprom (connected to i2cbus)

Only RD126c has:

- USB device connector
- Additional 2 USB host connectors (it has an internal 4 port usb hub)
- Internal low-voltage inverter and connector for LCD LED backlight
- TFT LCD 640x480 5.7” connector and additional touch-screen connector

1.6 RD126 Starter kit's connectors description

RD126 starter kit needs a single 5Vdc regulated power. Maximum current required is 1A.

The power must be provided on the IDE Hard-Disk style connector. By looking the board with the power connector on the right side, the 2 central pins are connected to ground, the topmost pin is for +5Vdc, the bottom one is not used (usually +12Vdc are provided on this pin by standard PC power supplies).

The 2 DB9 connectors are for serial ports. The one nearest to the ethernet connector is port 2 and is normally used for debug, the other one is port 1. In both of them, only 3 wires are used: pin 2 is

output, pin 3 is input, and pin 5 is gnd. They can be connected to a PC with a straight cable, using only these 3 pins: 2 to 2, 3 to 3 and 5 to 5. On the starter kit there is a female connector, on the PC a male one.

Ethernet 10/100, USB host, USB device and SD/MMC connectors have all standard pinout.

To connect the starter kit to an Ethernet hub you need a straight (or normal) cable, to connect the starter kit directly to a PC you need a cross cable.

RD126 can be equipped with different LCD panels.

1.7 RD156 Evaluation kit's connectors description

RD156 evaluation kit needs a single unregulated power, in the range 8-16Vdc. Maximum current required is 500mA.

It has a QVGA, 3.5” LCD panel on it. There is a touchscreen on it.

On the left side there is the SD card expansion port, on the top side there are the audio mic-in (on the left) and earphones-out (on the right) connectors, on the bottom side there is a reset pushbutton.

On the right side there are, from top to bottom: RS232 debug port, USB device, USB host and power supply connectors.

On the top side there is a 25x2 pins 100 mils pitch strip connector, with all CPU's signal not used in the board.

RD156 lacks an ethernet port, but an ethernet connection can be emulated over the USB device port.

2 Target Software (running on CPU card)

2.1 Included software

Standard production RD129 cards are preloaded with kernel image and a minimal file system.

The preloaded kernel is the main public one, downloadable from site www.kernel.org, patched with a proprietary patch written by ELPA to adapt it to the board's differences.

The flash memory is organized in this way:

- Lower 512KB: Bootstrap loader
- Next 2MB: Compressed Linux kernel
- Next 512KB: Bootloader splash screen
- Remainder: User file system (usually jffs2)

In the user file system there are all files needed by the kernel to boot up. Typical minimum configuration includes busybox and some small configuration files.

To obtain a reduced footprint of flash, we choosed to substitute the standard library libc with the smaller (but compatible) microclibc (www.uclibc.org), and the busybox (www.busybox.org) toolbox.

Customer will have to install its own application software and all needed dynamic libraries. No dynamic libraries are found in the preinstalled file system.

2.2 Bootstrap procedure

At power-up the following operations are performed in sequence (U-boot 1.3.4):

- The CPU loads first 4KB from nand flash memory into an internal static ram cache, and runs it
- This small program initializes pll and sdram controller, loads the main bootloader (called u-boot) from nand flash (next 508KB) into sdram and runs it
- Bootloader searches for a valid .png image from 3rd flash partitions, if it finds it then it initializes accordingly the LCD panel, shows the image on it and activates the backlight.
- If you send a 'U' char on the console debug (3rd serial port), the bootstrap loader stops, displays a prompt, and waits for user commands from console port
- Otherwise, it loads into sdram the compressed kernel image from the 2nd flash partition, expands it at a prefixed address, and runs it
- Kernel starts up and mounts the 4rd flash partition like main file system, of kind nand jffs2
- The kernel runs the script “/linuxrc”
- The script “/linuxrc” tries to mount an external USB flash key, and if it exists it tries to run from it the script file “initrd129.sh”; otherwise it tries to run the user script “/linuxrc.user” from the internal flash. The scripts /linuxrc and /linuxrc.user are user-customizable, like all the files in the file system. We suggest you to customize only linuxrc.user, and keep the original linuxrc. Note: in standard minimal file system, only /linuxrc is present.

2.3 Debug procedure

The simplest mode to communicate with the system is using the RS232 debug port (uart 2).

This port is configured at 115200,n,8,1 without any handshake signal (it uses only TxD and RxD).

By using a terminal emulator (like Windows' Hyperterminal or Linux' minicom) you connect to a user console, that you can use to send standard Linux commands.

Another debug mode can be activated by a telnet session, through ethernet interface. On your target this is possible only if you have an Ethernet interface (anyway for debugging you can use an external Usb to Ethernet converter).

The preloaded Linux kernel has included in it the driver for the usb to ethernet converter that is integrated in the RD126 starter kit.

The kernel image for RD156 has a driver that simulates an ethernet connection using the USB device port, that must be connected to a host system's USB host port using the provided USB cable.

Before using it, you need to assign an IP address.

To make this operation permanent, the suggested procedure is to insert the needed commands in the startup user script, that is executed at the end of kernel startup. You have to type these commands:

- `echo "ifconfig eth0 <ip>" >/linuxrc.user`
- `echo telnetd >>/linuxrc.user`
- `chmod +x /linuxrc.user`
- `sync`

You have to substitute <ip> with the IP number you want to assign to the board (ex.: 192.168.0.47)

In this way, at powerup this batch file will assign the correct IP number, and then will start the telnet server.

You can check the batch file you just created by typing "`cat /linuxrc.user`"

Once configured the board, the serial console can be substituted by one or more telnet sessions.

To connect using telnet from a Windows machine you can use the Hyperterminal program, from a Linux machine you can use the command "`telnet <ip>`".

Anyway the target system will prompt you to ask you the user id "root", and after that the session will be exactly the same like the serial line's one, with the added capability that you can break running programs by sending a <CTRL+C> char.

Note: if you are using emulated usb device port for Ethernet, you have to substitute "eth0" with "usb0".

2.4 Kernel's personalization

The kernel that is installed in the board is configured for a standard set of peripherals. For normal uses, it should be ok.

In the distributed binary image many drivers are activated, that assign peripheral functions to many i/o pins. This can be not acceptable on some target applications, in this case you must remove unused drivers.

Please double-check RD126 and/or RD156 starter kit schematic to understand which i/o pins are used by dedicated peripheral drivers at startup by standard kernel image. Their function can be switched to general purpose i/o after Linux bootup, but during startup their behaving can be unacceptable.

To personalize kernel you need to do the following operations:

- Download the kernel sources from www.kernel.org into the host
- Unzip it in a dedicated directory

- Move on it with "`cd <directoryname>`"
- Patch it with ELPA's patch (available on ELPA web site), using the command "patch -p1 < patchfilename". Please note that every kernel version needs a specific patch file.
- Add the environment variables CROSS_COMPILE=arm-linux- and ARCH=arm (distribution dependent)
- Type command "`make menuconfig`"
- Configure kernel, choosing desired options. The most meaningful are:
  - System Type → S3C2410 Machines → RD129 (must be always selected)
  - Device Drivers → Memory Technology Device (MTD) support → MTD partitioning support → Command line partition table parsing
  - Device Drivers → Memory Technology Device (MTD) support → Direct char device access to MTD devices
  - Device Drivers → Memory Technology Device (MTD) support → Caching block device access to MTD devices
  - Device Drivers → Memory Technology Device (MTD) support → Self-contained MTD device drivers → MTD using block device
  - Device Drivers → Memory Technology Device (MTD) support → NAND Device Support→ NAND Flash support for S3C2410/S3C2440 SoC → S3C2410 NAND Hardware ECC
  - Device Drivers → Memory Technology Device (MTD) support → NAND Device Support→ S3C2410 NAND IDLE clock stop
  - File Systems → Inotify file change notification support → Inotify support for userspace
  - File Systems → Miscellaneous filesystems → Journalling Flash File System v2 (JFFS2) support → JFFS2 write-buffering support
  - File Systems → Pseudo filesystems → /proc file system support → Sysctl support (/proc/sys)
  - File Systems → Pseudo filesystems → sysfs file system support
- Compile it, typing command "`make uImage`"
- Take a long coffe-break, to wait it to compile.
- The generate kernel image file is: arch/arm/boot/uImage.
- Type command "`make headers_install`", to generate in usr/include the kernel includes for user programs (needed only if your application program needs any kernel include file).

The maximum allowable size of compressed Linux kernel is 2MB, that is the flash memory space reserved to store its image. Please keep in mind that nand flash memory like hard-disks can have some bad sectors, so the space available could be something less on some boards.

2.5 Kernel and splash screen upgrade

To update the kernel image, you need the uImage compressed image file.

You can write it on the target in 2 ways: using bootloader or by in-application program.

To write it using bootloader, you need to copy it to an external usb flash dongle (formatted with FAT or FAT32 filesystem).

You have to connect a terminal emulator to the debug serial port (uart 2), power up the target and send immediately an 'U' character. Now, the u-boot prompt should appear on the terminal.

The command to upgrade the kernel is: "`update kernel <imagefilename>`", you can do the same operation at runtime with the command. "`cat imagefilename > /dev/mtdblock1`"

The board can be bootstrapped typing the command "`boot`", or power-cycling it.

To activate the boot-up splash screen you need to provide a .png image with the same exact resolution of lcd panel. It can be maximum 512KB (minus space eventually lost by bad sectors in the dedicated partition).

The image can be loaded at u-boot prompt with command "`update logo <filename>`", or at runtime by copying the .png file to /dev/mtdblock2, with the command: "`cat filename.png > /dev/mtdblock2`"

Important note: some older boards have a bug in the bootloader that can access only first 16MB of USB flash memories. If you experience such a problem please reformat your flash memory before copying the image files in it, in this way the files are located in the first 16MB and also that buggy version can read them.

Important note: to upgrade image partitions from runtime programs standard cat command could fail if nand flash has any bad block in the partition that you want to overwrite. Runtime program should use nandwrite command (that skips bad blocks) to update nand partitions. To update partitions using u-boot update command is always safe, because it correctly skips bad blocks.

2.6 File system replacement

If for some reason the filesystem becomes unusable (can happen if you are experimenting your personalizing startup scripts), you can rescue the situation by restoring into the board a tested empty filesystem.

This procedure is similar to the one to upgrade the kernel, the only difference is that the command is "`update fs <imagefilename>`", where <imagefilename> is the filename of the file system image.

The whole filesystem image is not replaceable runtime by user application.

A tested minimal filesystem image is available in the ELPA web site.

2.7 Nand flash details

Permanent memory in this board is implemented with a single nand flash chip. It contains the bootstrap loader, the kernel image, and the main file system image.

Nand flash is similar to hard-disks: it's big, quite fast and cheap, but it can have some broken parts, called bad blocks.

Like hard-disk it's written in sectors: if I need to write 1 byte, I must write 1 sector. Typical sector size is 512 bytes.

Unlike hard disk, before writing a sector you need to erase it. But due to hardware limitation, minimal erase size (called erase block) is bigger than sector size. Typical erase block size is 16KB.

Chip manufacturers grant that at origin at least 98% of the blocks are good, and good blocks are granted for minimum 100000 rewrites.

Linux file system knows all about this, and manages it accordingly and transparently to the user.

To rise performances, all the write operations to the nand flash are cached. After a write, before

powering down the system you have to wait 5 seconds or execute a "`sync`" command.

To help user programs, the standard linuxrc startup script installs ram-disks in some file system subdirectories. After booting, all accesses to /mnt, /dev and /tmp files are routed to ram-disks.

To avoid excessive flash wear, if user programs need to store temporary informations they should put them under the /tmp directory, that is shadowed with a ram-disk. All write operations to the flash are much slower, and flash' lifetime is very big, but not infinite.

2.8 Peripheral's registration

To install and remove dinamically the peripheral's drivers, linux uses a procedure called hotplug. When a peripheral (tipically USB) is inserted or removed from the system the kernel calls an user program.

This program, examining environment variables, calling arguments and virtual files in "/sys" directory, must execute the operations needed to install/remove the peripherals.

Normally this task is performed by "mdev", a daemon that is installed by startup script that is called by the system when peripherals are plugged or removed. It's contained in busybox.

By examining some rules files (located in /etc/mdev.conf), mdev dinamically creates/destroys device nodes in directory /dev.

Note: to avoid loss of data, before extracting these memories is better to type the commands "`sync`" and "`umount <subdirectory>`".

2.9 Main file system

The main file system, JFFS2 type, is in the 4th partition of internal NAND flash memory.

At startup the filesystem is mounted in the root directory.

The main subdirectories are:

- /mnt contains the filesystems of the plugged removable devices (ex.: /usb0 and /mmc). At startup it's shadowed by a ram-disk.
- /dev contains device nodes that allow to use peripherals. At startup is shadowed by a ram-disk.
- /var contains variable datas, is shadowed by ram-disk.
- /lib contains dynamic libraries, with extension ".so"
- /lib/firmware contains peripheral's dedicated firmwares. For copyright reasons they are not embedded into the driver's code, that usually is open-source.
- /lib/modules contains additional optionals drivers that can be dinamically installed and removed after kernel startup by the commands modprobe, insmod and rmmod. These drivers are named modules and have extension ".ko" (Kernel Object).
- /etc contains various configuration files.
- /sys contains virtual files, containing various infos about operating system's and connected peripherals' status. Some of these can be written, to modify some parameters.
- /proc is older version of /sys, it's similar, but it has been replacing by /sys
- /linuxrc is the script file that is executed at startup (similar to MSDOS' autoexec.bat).
- /bin and /usr/bin contains user's commands
- /sbin and /usr/sbin contains supervisor's commands

– /tmp is shadowed by a ram-disk. It's normally used to store temporary informations.

All the commands in /bin, /sbin, /usr/bin and /usr/sbin are only links to program /bin/busybox, that is a single program that contains all the command's functionality. This allows to reduce the filesystem's footprint (look at www.busybox.org).

2.10 Internal peripheral's drivers

In linux peripheral's drivers can be compilated together the kernel or can be loaded after kernel's startup. In this case they are named modules, have ".ko" extension and are installed with command "`insmod`" or "`modprobe`", removed with "`rmmod`" and listed with "`lsmod`". Usually they are stored into the subdirectory "/lib/modules".

Included in the kernel there are drivers for all internal peripherals.

In Linux every device is identified by 2 numbers, named Major and Minor.

To communicate with a driver you need an access node in the filesystem. Usually these nodes are automatically created at runtime by mdev program, and stored in the "/dev" subdirectory; you can list them with the command "`ls -l /dev`". The command to manually create them is "`mknod`".

User programs can use these drivers using standard api calls "open", "close", "read", "write", "ioctl", passing the node name as parameter to "open" call.

Some drivers have a virtual files' interface: by reading or writing some files (usually in subdirectory "/sys") you can read and write some parameters, or perform special operations.

2.10.1 Asynchronous serial ports

To compile this driver in the kernel, you have to select in menuconfig "Device Drivers → Character Devices → Serial drivers → Samsung S3C2410/S3C2440/S3C2442/S3C2412 Serial port support".

RD129 has 3 uarts, named: /dev/ttySAC0, /dev/ttySAC1 and /dev/ttySAC2.

To enable console terminal on it, you have also to enable "Support for console on S3C2410 serial port". Normally this driver is enabled, but to use ttySAC0 and7or ttySAC1 you must switch the related pin (H2 and H3 for ttySAC0, H4 and H5 for ttySAC1) functionality from general purpose i/o to peripheral dedicated (look further at chapther "General purpose I/O driver").

/dev/ttySAC2 is usually used for debug, at powerup it allows user to perform low-level operations interacting with bootloader and once the kernel is started it becomes a virtual console. On the starter kit RD126, it's located near the Ethernet connector. The pinout is standard, on a DB-9 connector.

On the RD156 evaluation kit, it's the only DB-9 connector.

Because ttySAC2 is used for user interface also by bootloader, we suggest to use this port in target application only for debugging.

Unfortunately the used processor cannot generate an end of transmission interrupt, so if you need to interface with RS485 devices you need to generate the Tx-enable signal with additional timing hardware.

2.10.2 Watchdog

To compile this driver in the kernel, you have to select in menuconfig "Device Drivers → Character Devices → Watchdog Timer Support → SS3C2410 Watchdog".

Device Drivers → Input Device Support → Touchscreen interface

Watchdog timer is activated opening device /dev/watchdog. Since that, you have to perform a write to the device every 15 seconds or faster, otherwise the timer trips and resets the board.

To disable the timer you must write a 'V' character and then close the device.

The trip time of 15 seconds is configurable using the dedicated ioctl call WDIOC_SETTIMEOUT.

To make your application safer, you can configure the kernel to start the watchdog timer at kernel startup.

2.10.3 IICbus controller

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → I2C Support → I2C Hardware Bus support → S3C2410 I2C Driver”.

Usually it's not used directly by the user, but by other peripheral-specific drivers. Anyway, it's possible to perform single transfers on it using ioctl function at device /dev/i2c-0.

2.10.4 USB Host

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → USB Support → OHCI HCD support”.

You can then configure “Samsung S3C2410 USB Host ports” to 1 Host and 1 Device or 2 Host ports. If you choose “Autoselect”, if GPD0 at bootstrap is strapped down it will be configured to 1H+1D, othervise 2 Hosts. Standard kernels are configured in autoselect mode.

This driver is not called directly by the user, only by other peripheral-specific drivers.

2.10.5 USB Flash memories

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → USB Support → Support for Host-side USB”, “Device Drivers → USB Support → USB Mass Storage support”, “Device Drivers → SCSI device support → SCSI device support” and “Device Drivers → SCSI device support → SCSI disk support”.

To acces dos/fat filesystems, you have also to enable “File systems → Partition Types → Advanced partition selection → PC BIOS (MSDOS partition tables) support”, “File systems → Partition Types → Advanced partition selection → Windows Logical Disk Manager (Dynamic Disk) support”, “File systems → DOS/FAT/NT Filesystems → MSDOS fs support” and/or “File systems → DOS/FAT/NT Filesystems → VFAT (Windows-95) fs support”

When some usb peripherals are plugged or unplugged the kernel calls the mdev daemon that generates and destroys the device nodes in the /dev subdirectory, and mounts or unmounts the filesystems in the directories /mnt/usb0, /mnt/usb1, etc. The mounted filesystem will be accessed in that subdirectories.

After a write, to avoid loss of datas is preferable to execute the “umount” command before unplugging the memory card.

2.10.6 USB device

The 2nd USB port can be configure like host or device port.

In RD126a starter kit it is configured likeh ost.

In RD126C and RD156 evaluation kit it's configured like device port, and emulates an ethernet port.

If you connect it to an host port of a linux host system, it will recognize it like a virtual ethernet port, named “usb0”. Also on the target you will operate with an “usb0” ethernet port, equivalent to a standard “eth0” classic port. The only difference related to a real ethernet connection is speed.

2.10.7 MMC/SD memories

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → MMC/SD/SDIO card support → Samsung S3C SD/MMC Card Interface support”.

If you choose to use this driver, it uses gpio pins: E5, E6, E7, E8, E9, E10, G0 and G1,

The MMC and SD flash memories are handled by the mdev daemon like the flash usb memories, the filesystems are automatically mounted in the subdirectory /dev/mmc.

After a write, to avoid loss of data is preferable to execute the “umount” command before unplugging the memory card, that flushes all buffered write operations.

If you wish to use micro-sd cards, there are some hardware and software differences:

- you must connect SDDAT3 and SDCD together (microsd wants both these signals on the same pin)
- you must add one 100Kohm resistor between SDCD and GND

If you are using microSD, you must select “Device Drivers → MMC/SD/SDIO card support → MicroSD on RD129

2.10.8 LCD screen

To compile this driver in the kernel, you have to select in menuconfig “Graphics support → Support for frame buffer devices” and “Graphics support → S3C2410 LCD framebuffer support”.

This driver manages different kinds of TFT LCD screens.

It implements a framebuffer; you can access it opening the node /dev/fb0, and mmap it. The video memory format can be 8 (256 colors in a palette of 16M) or 16 (65K colors) bits per pixel.

Many standard resolutions are supported, from QVGA WSVGA. All these are tested and selectable with command fbset. You can select supported resolutions enabling them in “Device Drivers → Graphics support → Support for frame buffer devices”.

Other resolutions and formats are easy to add, please contact ELPA.

Note: Ampire QVGA displays need a “dummy” line, to select that resolution you have to specify 320x241.

Note: QVGA Powertip display used on evaluation kit RD156 needs an initialization sequence on its SPI port, if you use this display you must enable “3.5' POWERTIP DISPLAY”

Because the LCD controller's memory is shared with CPU, the greater the resolution and the colors' depth, the slower the CPU throughput. Memory bandwidth usage is also proportional to LCD refresh rate.

2.10.9 LCD backlight

Attention: This changed from older kernel versions

To compile this driver in the kernel, you have to select in menuconfig “System Type → PWM device support” and “Device Drivers → Graphics support → Backlight & LCD device support → Lowlevel Backlight controls → Generic PWM based Backlight Driver”.

In the same page you can also change the pwm frequency.

The user program can change the backlight intensity from 0 to 100% by writing the desired value in ascii into the virtual file /sys/class/backlight/pwm-backlight/brightness.

This driver operates on Timer 0 pwm output.

2.10.10 SPI interfaces

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → SPI support → Samsung S3C24XX series SPI”.

From kernel 2.6.32, ELPA added an alternative driver for SPI that uses DMA. You can enable it in the same menu page in place of traditional one. For monodirectional transfers it's much faster than traditional one, that generates one interrupt every byte transferred.

This driver manages only master mode. The standard ELPA kernel configures 1 user device connected to SPI0, with chip select signal driven on GPA16 pin, and 2 devices connected to SPI1: 1 user device with CS on GPA14 and 1 experimental Ethernet interface with CS on GPA15 and interrupt on GPG10.

If you want to manage different devices connected to the same SPI port (using separated CS signals, one for each device) you have to customize the mach-rd129.c kernel source file and recompile the kernel.

This is a low level driver, you can use it in 2 ways:

1) Binding a device specific driver (kernel customization required)
2) Directly accessing SPI device. You can read and write to the /dev/spidevX.X device, or execute simultaneous read/write transfer with a dedicated ioctl call.

Note: If you want to direct access spi driver, you must also enable in menuconfig “User mode SPI device driver support”.

2.10.11 Touch-Screen / ADC

To compile the internal ADC driver in the kernel, you have to select in menuconfig “System Type → ADC common driver support”, “Device Drivers → Hardware Monitoring support → S3C24XX/S3C64XX Inbuilt ADC” and “Device Drivers → Hardware Monitoring support → Include raw channel attributes in sysfs“.

If you want to use also touchscreen, you need to select also “Device Drivers → Input device support → Event interface” and “ Device Drivers → Input device support → Touchscreens → Samsung S3C2410 touchscreen input driver”

Touchscreen driver can be accessed by reading from standard input device /dev/input/event0.

Adc channels can be acquired by reading the virtual files in “/sys/devices/platform/s3c24xx-adc/s3c-hwmon”: adcX_raw gives conversion in unit (0..1023), and inX_input gives conversion in millivolt (0..3300).

Please note that if you enable the touch-screen adc channels 5 and 7 are dedicated to it.

Note: to use touchscreen device, you need to wire 2 small double mosfets and 2 RC filters on your application board, and provide a stable 3.3V reference voltage to the ADC converter. For details please refer to the RD126 or RD156 starter kit schematic.

2.10.12 General purpose I/O driver

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → Userspace I/O drivers → S3C2410 general purpose digital I/O driver”.

This driver gives user access to all the microcontroller's general purpose I/O pins.

All operations are performed by reading/writing virtual ascii files located into directory /sys/devices/platform/s3c2410-gpio/

There are different groups of these files:

- Pin status files, with same name of pin (ex: g0), can be used to read or write the pin status (0 or 1)
- Pin configuration files, with name of pin followed by suffix -cfg, can be used to read or change the pin mode, that can be: input, output (not allowed on all pins), fn1 or fn2. fn1 is the main peripheral function of pin, fn2 is the auxiliary function (only some pins have it). When programmed as fn1 or fn2, when you read it you get the function name (ex: if you write “fn1” into file “b0-cfg”, when you read it you get “tout0”. You could also write directly “tout0” in that file).
- Pullup configuration file, with name of pin followed by suffix -pup, can be used to read or change the pullup status (1=on, 0=off).
- External interrupt configuration file, named “eintX-cfg”. It reflects the external interrupt mode. It can have the values “rising”, “falling” or “both” and selects which signal's edge triggers the interrupt. These files appear only when relative pins are configured for external interrupt function.
- External interrupt wait file, named “eintX”. When a program tries to read this file, it returns only when that external interrupt has been triggered. These files appear only when relative pins are configured for external interrupt function.
- Port mask file, named “a-msk”, “b-msk”, and so on. They contain a decimal mask used in writing a port. Only bits active in this mask will be affected when writing in the port virtual file.
- Port virtual file, named “a”, “b”, and so on. They allow to read or write a whole port with a single operation. When writing, only bits active in the port mask file will be affected. The value is in decimal notation.

Because all these virtual files are in ascii, you can experiment with them by using simple console command “cat <filename>” (to read them) end “echo <value> > <filename>”(to write them). Please note that if you try to read a “eintX” file the call doesn't return until that interrupt triggers, so it's better to use this functionality only in multithreaded programs.

2.10.13 Clock driver

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → Misc devices → User mode clock driver”.

This driver gives user access to internal clocks.

All operations are performed by reading/writing virtual ascii files located into directory /sys/class/clock/

In this directory there is one subdirectory for each clock present in the system, with the clock's name.

In each subdirectory there are 2 virtual files:

- parent: contains the clock' parent's name.
- rate: contains the clock's rate.

With this driver you can program external clock generators and muxes:

clkoutX parent can be: mpll, upll, fclk, hclk, pclk, dclkX

dclkX parent can be: pclk or upll. Its rate is configurable (within certain limits).

Please note that to effectively make clocks to exit from the chip, you have also to program the relative I/O pins to their peripheral function. In particular, to make clkout0 exit from pin H9 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/h9-cfg” and to make clkout1 exit from

pin H10 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/h10-cfg”.

2.10.14 Input keys

Linux provides a generic driver to allow single pushbutton keys wired to interrupt capable input pins to generate standard input events, and so be understood by general purpose user programs.

To activate this feature, you need to select in menuconfig “Device Drivers → Input device support → Keyboards → GPIO Buttons”, and to customize the structure gpio_keys_button in the mach_rd129.c kernel source file.

This driver doesn't manage matrix keyboards.

2.10.15 Timers driver

To compile this driver in the kernel, you have to select in menuconfig “Device Drivers → Misc devices → User mode timer driver”.

This driver gives user access to internal timers, that can be used in 2 ways: to generate an external signal or to wait for a defined time.

All operations are performed by reading/writing virtual ascii files located into directory /sys/devices/platform/um-timer.X/, where X ranges from 0 or 1 to 3. Timer 4 is used exclusively by Linux' scheduler.

In this directory there are 3 virtual files:

- frequency: sets the frequency (in Hz) you want to generate on dedicated pin. When you write to this file, the duty-cycle is setted to 50%.
- duty: sets the duty-cycle of the signal you want to generate on dedicated pin. Range is from 0 to 1000.
- wait: when written it programs the time required in microseconds and starts the timer, when read waits until the programmed time expires.

Note 1: Frequency and duty cannot have any value, you may want to read them after writing to check the real value you programmed in.

Note 2: To effectively make clocks to exit from the chip, you have also to program the relative I/O pins to their peripheral function. In particular, to make timer0 exit from pin B0 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/b0-cfg”, to make timer1 exit from pin B1 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/b1-cfg”, to make timer2 exit from pin B2 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/b2-cfg” and to make timer3 exit from pin B3 you have to write “fn1” in “/sys/devices/platform/s3c2410-gpio/b3-cfg”.

Note 3: Some timer may not be available, if already used by other drivers like lcd backlight.

Note 4: Obtainable values can be not the same for all timers, due to timer's prescalers limits.

2.11 Additional devices in the RD126 / RD156 starter kits

In the starter kits there are power supplies, interfaces and connector, and some additional peripheral that are supported by kernel drivers.

2.11.1 RTC (Real Time Clock)

In the preloaded kernel is included the driver for the RTC chip Maxim DS1338, that is mounted in the starter kit and wired to the I2C bus.

To compile its driver in the kernel, you have to select in menuconfig “Device Drivers → Real Time Clock → Dallas/Maxim DS1307/37/38/39/40, ST M41T00”.

If you want linux at startup to read the current date and time from the RTC chip in the system clock, you have also to enable “Device Drivers → Real Time Clock → Set system time from RTC on startup and resume”, “Device Drivers → Real Time Clock → RTC used to set the system time = rtc0”, “Device Drivers → Real Time Clock → /sys/class/rtc/rtcN (sysfs)”, “Device Drivers → Real Time Clock → /proc/driver/rtc (procfs for rtc0)” and “Device Drivers → Real Time Clock → /dev/rtcN (character devices)”.

To set current system date and time you must use the command “date” with appropriate parameters, or call the settimeofday standard library function.

To program the system clock into the RTC chip you must use the command “hwclock -w”, or call a dedicated ioctl on /dev/rtc0.

This chip contains also 56 bytes of battery-backed non volatile ram. This memory can be accessed through the virtual file /sys/class/rtc/rtc0/device/nvram.

2.11.2 Ethernet interface

In the RD126 starter kit there is a USB → Ethernet 10/100 converter, that uses a Realtek RTL8150 chipset. When this chipset's driver is included in the preloaded kernel, at powerup the standard ethernet device “eth0” will be available, that will be managed in the standard Linux's way.

To compile its driver in the kernel, you have to select in menuconfig “Device Drivers → Network device support → USB Network Adapters → USB RTL8150 based ethernet device support”.

Another interface tested with RD129 is the Microchip ENC28J60 SPI to Ethernet converter. If you are interested in implementing this solution in your application board ELPA can provide you (for free, as usual) the schematic and all the info you need, including an expansion board for RD126. Please note that the ELPA kernel patch modifies the standard 28J60 kernel driver, because to work properly it needs to use DMA. This solution is cheaper and doesn't use an usb port, but implements only 10Mb Ethernet.

You then have to select all the “Network → Networking options →” you need.

To assign an IP number, you must use the command “ifconfig eth0 <ip>”.

After configuring the IP, if you run the command “telnetd”, you activate a telnet server that allows remote clients to connect and interact with the system.

2.11.3 Audio codec

In the RD126 starter kits there is a PCM3006 stereo audio codec, connected to CPU through a I2S bus.

The I2S port is designed to transfer a continuous stereo audio stream, in both directions. In practical applications the transfers are monodirectional.

In the older RD126 starter kits both input and output connectors are wired, in the new ones RD126C only output connector but we added a small audio amplifier to directly connect an earphone.

Sound management is fairly powerful but complex, users normally interact usign the alsa-lib sound library. This library requires some small ascii configuration files in directory /usr/share/alsa (please refer to fs-demo file system)

For experimenting alsa-utils are available, there is a program called aplay that plays a .wav audio file.

QT has some classes to generate sounds, that uses OSS api.

The codec used in the starter kit is a very basic one: it has only one stereo 16 bits DAC and one stereo ADC, without any programmable capabilities. It has fixed levels, no analog multiplexers, no earphone amplifier. It support rates between 8000 and 48000 samples per second.

Moreover, it can work only in slave mode (clock must be externally generated). Because the CPU has no dedicated PLL for them, this means that the rates produced are slightly different from the requested ones.

In the RD156 evaluation kit there is a Wolfson WM8731 codec, its driver must be enabled instead of PCM3006 one). It's cheap and powerful, doesn't requires dedicated crystal, and supports sample rates of 8, 32, 44.1, 48, 88.2 and 96KHz.

Please note that S3C2410 CPU doesn't have an AC97 audio interface, so only stereo output is possible.

If you select this driver, it uses GPIO pins E0, E1, E2, E3 and E4 (E2 is not required for WM8731, that uses clock generator Clkout1 on pin H10 for system clock).

2.11.4 I2Cbus Eeprom (24xx)

This driver allows read/write access to a single 24xx device wired on the I2Cbus.

To avoid false aliasing, only 1 device is recognized, with wired address = 0.

By default the driver expects a 24c64 device, if you wish to use a different device you have to personalize kernel, compile it like a module and install it passing appropriate parameter.

Eeprom content can be read/written accessing the virtual file "/sys/devices/platform/s3c2410-i2c/i2c-adapter:i2c-0/0-0050/data".

This device is mounted only on older starter kits.

2.11.5 CAN Interface

RD129 has no CAN bus port, but a very cheap solution is available, based on Microchip MCP2515 SPI to CAN bus converter.

ELPA can provide you with documentation and schematic to add this chip to your application board. Its driver must be compiled into the kernel, and you can take advantage of the can bus port fully integrated with linux kernel.

The standard way to access can bus in Linux is to use it like a network port. All is explained in the linux documentation file linux/Documentation/networking/can.txt

To set up the can interface and program the bitrate the "ip" command is needed. Because can bus is integrated into linux from a short time, the "ip" command provided by busybox cannot manage can port commands, nor the ip command provided by buildroot 2010.08. Only recent version of ip command can manage the can bus. To configure can bus interface you'll need the ip command provided by buildroot 2011.02.

2.12 Busybox

All system commands are implemented in a single executable called busybox. All commands are links to the same big executable.

The busybox program can be configured on the host with the command `make menuconfig`, an compiled with the command `make`, then, the generated busybox executable must be copied on the target in the /bin directory.

A standard configuration is downloadable from the elpa web site, you need to copy the .config file

into the busybox directory.

The links are installed in the directories /bin, /sbin, /usr/bin and /usr/sbin. To autoinstall them, run on the target the command: `busybox –install -s`

Note: before copying the new busybox command on the target, remember to give it execution permissions, with the command: `chmod +x busybox`

2.13 Graphic libraries

The system is delivered with only linux kernel preinstalled, including framebuffer driver.

The framebuffer driver gives low-level access to the user for graphic memory.

If you open device “/dev/fb0” and call “mmap” you obtain the framebuffer's base address, that is the start address of the memory region that is continuously transmitted to the LCD display.

The memory's format depends by the display's model, by the resolution and the colors' format. The format can be from monochromatic (1 bit per pixel) up to full-color (24 bits per pixel), the memory footprint changes depending on it.

Usually the graphic library take account of all this.

There are a lot of graphic libraries usable, some simpler, some more complex, some free, come commercial.

In the standard PC Linux distributions usually is used X-window, that is free but quite big and heavy.

For embedded applications with RD129 we suggest to use the QT library, that has a lot of advantages:

- Well maintained and documented (by Nokia)
- Multiplatform: the same application can be developed and runs on Linux embedded, Linux, Windows, and Mac
- Not only a GUI: It virtualizes files, directory and network access
- Can use scalable or fixed fonts.
- Supports full utf-8 characters set, for internazionalization.
- Monolithic (it doesn't need other libraries).
- Dedicated IDE (QTCreator) available for free.

Another option is fltk, a simpler light-weight layered alternative. In this case you need:

- nano-X (www.microwindows.org), that is quite simple, free and uses small memory. It's available with 2 different application interfaces, one similar to Windows (called microwindows) and one similar to X-window (called nano-X).
- nxlib is a small library that makes nano-X appear like a X substitute
- fltk (www.fltk.org) is a nice user interface library that has also a graphic user interface editor that can run both on Windows and Linux hosts.
- freetype (www.freetype.org) is a font-rendering library that can render any font, fixed or scalable.

Note : Caused by a nano-X bug, in screen rotated mode images and scalable fonts are not working.

All these librariese are integrated, working and free. They are used by the graphic demos preinstalled in the boards sold with the starter kit.

3 Host software

3.1 Development system installation

To compile programs running on the RD129, many solutions are possible.

We develop the applications on a host development system, that is a PC running the Linux operating system.

You can choose the distribution you prefer; we actually use Kubuntu, but any distribution is acceptable.

After installing Linux on the host system, you have to install on it the cross-compiler toolchain.

To download, generate and install all the tools you need you may need to install the following packages: ncurses-dev, g++, bison, flex, texinfo, gettext, patch, wget, cpio, python, ncftp, subversion, uboot-mkimage, mtd-utils, minicom, telnet. On Kubuntu, this can be done with the command “sudo apt-get install <packagenames>”

3.2 Cross-compiler toolchain generation using buildroot

To compile C or C++ programs, you need to have the standard gcc cross-compiler.

We suggest to generate the cross-compilation toolchain using the buildroot utility. To install it, you have to:

- download it from http://buildroot.uclibc.org and expands its files in a directory. The last supported version is indicated on ELPA's web site.
- download from ELPA's web site the preconfigured .config file and copy it into the buildroot's directory.
- “make menuconfig” and then exit
- ”make”
- if “make” exits with an error, run “make” another time.

This starts a long process for your PC to dowmload, configure, build and install all the packages you need to cross-compile and debug applications for the RD129 CPU.

After generation completes, you have to add to your PATH the subdirectory output/staging/usr/bin under your buildroot path, that contains all the cross-compilation tools. On Kubuntu, you can simply add it to the /etc/environment script (you can edit it using the command “sudo kate /etc/environment”), by adding your search path (separated by :) to the statement PATH=”...

On Kubuntu, a problem can arise because commands executed with “sudo” uses a different PATH. This can be solved by adding the line “Defaults exempt_group=<groupname>” to the /etc/sudoers configuration file. Please note that this is a very risky operation, that must be performed using the command “sudo visudo”. If you correctly patch this problem, “env” and “sudo env” should report the same path.

Buildroot generates a toolchain that uses uClibc instead of glibc, a smaller glibc compatible version optimized for embedded systems.

If you link your applications using dynamic libraries, remember to copy them from output/target/lib and output/target/usr/lib buildroot's subdirectories to the /lib directory in your target. If your project needs other dynamic user libraries, you should copy them from /usr/local/lib. Dynamic libraries in Linux have suffix .so

Note: the .config that ELPA provides configures buildroot to produce the arm cross compiler, cross debugger, uClibc (a compact version of the runtime C library), QT (graphical user interface library), and busybox (standard commands toolbox).

3.3 Programming workflow

At this point you can try to write, compile and debug your application:

- write your application in C or C++ using any editor you prefer (ex.: kedit or kwrite)
- compile and link it using the `arm-linux-gcc` or `arm-linux-g++` command
- transfer it on the target a stripped copy of the executable
- make it runnable on the target, typing the command "`chmod +x <filename>`"
- run it on the target, or cross-debug it

3.4 Programs' compilation

To compile C or C++ programs, you need to use the standard gcc cross-compiler.

The C cross-compiler's name is "arm-linux-gcc". The C++ cross-compiler's name is "arm-linux-g++".

Depending on its arguments, the compiler compiles and/or links the requested files up to generate the final executable, that can have or not have the debug symbols.

To generate a program that uses additional external libraries, when running the compiler you have to add arguments "-I <includedirs>", "-L <librarypath>" and "-l <libraryname>". Please note that <libraryname> must be without "lib" prefix and without ".a" or ".so" suffix.

If you want to generate an application linked with static libraries, you have to give compiler also the option "-static". In this way, compiler will link with ".a" libraries.

Otherwise the compiler will suppose to link with dynamic libraries. In Linux dynamic libraries have extension ".so", and are usually placed in the target in the "/lib" directory.

To compile complex programs is better to put all needed commands and parameters in a file called "Makefile", that can be called by program "make".

One of the Makefile's goals is to resolve dependencies, to reprocess only the files that depend by updated ones.

If you use the QT Creator IDE, all these operations are performed graphically by the IDE (please refer to QT creator's documentation).

3.5 Program's transfer on the target

To transfer programs and files from/to host and target, several solutions are possible:

3.5.1 FTP server on Target

One solution is to run a ftp server on the target, and to transfer files issuing ftp client command on the host PC. This is good to automate file copying, because if you install the packet "ncftp" on your host you can add batch ftp transfer commands to your IDE post-build commands. On Kubuntu, you can install it with the command:

```
sudo apt-get install ncftp
```

To configure your target, first of all you need to have the commands inetd and ftpd available on

your target. If you haven't, please add them to your target's busybox.

You need also to have the following line into the /etc/inet.d file on your target:

```
21 stream tcp nowait root ftpd ftpd -w /tmp
```

If you haven't it already in the target's file system, you can generate it by typing the command:

- `echo "21 stream tcp nowait root ftpd ftpd -w /tmp" >/etc/inetd.conf`

You need also to run the inetd program at boot, you can insert it in the starting script by typing:

- `echo inetd >>/linuxrc.user`

Note: you need to do all these operations once, the configuration files you generate are stored in flash and are started at the next power-up.

To transfer a file from the host PC to the /tmp directory on the target, you must issue on the host the command:

```
ncftpput <targetip> . <filename>
```

Where <targetip> is the ip of the target and filename the file you have to transfer. Example:

```
ncftpput 192.168.0.47 . test
```

Note: Before running the transferred program you need to give it execution permission, with the command: `chmod +x <filename>`

3.5.2 Network drive mount

Another way is to mount a network drive, that you previously shared from your system. In this way, the host's shared directory is always seen in the /mnt path as a local one.

To use this method you must share a directory in your host PC, and mount at target startup's the shared directory. To do this, your kernel must be compiled with NFS or CIFS enabled, and you must insert the appropriate mount command in the startup script.

3.5.3 FTP server on host PC

Another solution is to run an ftp server on the host PC, and transfer the files issuing ftp client commands on the target.

For this, you need to install on your host system a ftp server, that uses a dedicated host's directory to store the files that are exchanged with the target system. One of the most common ones is proftpd. This can change between distributions, please refer to application's specific help.

To get a file from this directory to the target system, you have to type on the target the command "ftpget <ip> <destfilename> <sourcefilename>", in which <ip> is the host's ip number, <destfilename> is the name of the file you want to create on the target, and <sourcefilename> the name of the file in the host's directory you want to get.

To make things easier, I suggest to create a small script by typing these commands on the target:

- `echo "ftpget <ip> ./\$1 \$1" >/bin/g`
- `echo "chmod 777 \$1" >>/bin/g`
- `chmod 777 /bin/g`
- `sync`

You can check the batch file you just created by typing "`cat /bin/g`"

In this way, to get a file from the host system to the current directory you have to type only "`g <filename>`".

Note: If your ftp server needs user identification, you may have to change the first line to:

```
echo "ftpget -u <username> -p <password> <ip> ./\$1 \$1" >/bin/g
```

The similar command to move files from the target to the host's ftp directory is “ftpput”.

Note: This method requires you issue transfer commands on the target.

3.5.4 Using external USB flash memory

Another way can be used by copying the file(s) on a external USB flash memory (if you are using a Linux host remember to unmount the flash key before removing it), plug the memory into the target and copy the required files from the automounting directory “/mnt/usb0” to the desired position. Normally this method is only used when you have no ethernet connection to your target, because it requires to plug and unplug the usb memory twice at each transfer.

3.6 Program's debug

The standard gnu toolchain's debug program is named arm-linux-gdb, and is commandline. Usually is ran by a graphical interface (ddd, insight or others) or by an IDE (Integrated Development Environment).

To debug a program you need to:

- Compile it with debug symbols (gcc's option -g3)
- Transfer a symbol-stripped copy of executable to the target (to eliminate debug symbols from this file, using the command “`arm-linux-strip execname`” before transferring file into the target)
- Run on the target the command “`gdbserver <ip>:<port> <filename>`” in which <ip> is the host IP number and <port> an unused port's number (ex.: 1234)
- Start the debug session on QT creator

Note1: you need to copy output/target/usr/bin/gdbserver from your buildroot directory on the host to /usr/bin/gdbserver on your target, and the libraries: libc.so.0, libthread_db.so.1, ld-uClibc.so.0 from buildroot/output/target/lib[1] to the /lib directory on the target (attention to the links !).

Note2: don't use newer versions of arm-linux-gdb, they have problems with arm target. We use arm-linux-gdb 6.7.1.

3.7 Integrated Development Environment

All explained operations to compile and debug a program can be speeded up using an Integrated Development Environment (IDE).

IDE coordinate editor, makefile generator, compiler, linker and debugger in the same graphical environment, allowing an easy applications' developing like modern Windows' tools.

In the web there are many, some free, some commercial.

QT libraries have its own IDE, named QT-creator. It's very powerful, well documented and free.

Another tested solution is the low-cost commercial tool Visual Slick Edit www.slickedit.com. On this web site you can download a 15-days trial version.

Many others are possible (Eclipse, etc.), but are untested by ELPA.

3.8 Working with QT

QT is a very powerful multiplatform opensource library, that implements all you need to write applications with full Graphical User Interface.

1 It was buildroot/output/staging/lib on older versions of buildroot, upto 2010.

To use QT, you must download and install from web site http://qt.nokia.com/products the full QT SDK, for the developing platform you prefer.

You have also to generate the QT libraries for embedded Linux. This is automatically done at the end of buildroot process, if you generate buildroot using ELPA's .config file.

At the end of the buildroot compilation you should find in buildroot/output/target/lib and buildroot/output/target/usr/lib (in your buildroot directory) all the dynamic library files you need to copy in the /lib directory on your target (your application probably will not need all of these).

Also the files contained in the “buildroot/output/target/usr/lib/fonts” subdir are needed by Qt.

To run your QT applications, call them using this small script:

```
export QWS_MOUSE_PROTO=LinuxInput
export QWS_DISPLAY=:mmWidth=72
$1 -qws
```

To run it rotating the screen, you need to specify:

```
export QWS_MOUSE_PROTO=LinuxInput
export QWS_DISPLAY=Transformed:Rot90:mmWidth=72
$1 -qws
```

Note: the 72 mm value must be adjusted to your display effective viewing area width.

Please note that a copy of all the files needed to run QT applications, together with autostarting graphical demos, is available on ELPA web site. These files are preloaded on the CPU distributed with starter kits.

QtCreator is the main free IDE to develop your application using QT. Please refer to its documentation, that is fully exhaustive.

To configure QtCreator to generate code for the target system, you have to:

- Run QtCreator and load or create a design
- On the left side, click on “Projects” selector
- On the top side, select “Build Settings”
- Create a new build configuration
- Click on the “Manage” button near the “Qt Version” listbox
- Click on the “+” button to add a new target
- Type a new name (e.g.: “Rd129”) in the “Version name” edit box
- Browse the “qmake location” to buildroot/output/staging/usr/bin/qmake
- Click on the Debugging helper “Rebuild” button
- That's all: in the “Build Settings” window now you can choose the Rd129 target, and QtCreator will use the buildroot generated arm cross-compiler instead of native one, and link with Qt embedded linux libraries.
- With ”Add build steps” you can customize QtCreator to automatize operations like stripping the target image and transfering it to the target

To debug a program with Qt, you must copy gdbserver and required dlls into the target (see chapter “Program's debug”), copy a stripped-down version of your compiled application into the target, and run it using this small script:

```
export QWS_MOUSE_PROTO=LinuxInput
export QWS_DISPLAY=:mmWidth=162
gdbserver 192.168.0.79:1234 $1 -qws
```

Note1: the 1st gdbserver's parameter is the host PC IP address and used port

To start debugging, select the menu command “Debug → Start Debugging → Start and Attach to Remote Application”. A Dialog will open, in which you must put:

- Debugger: browse it to buildroot/output/staging/usr/bin/arm-linux-gdb
- Local Executable: browse it to the unstripped version of your application's local copy
- Host and port: type here the target IP address and port (e.g.: 192.168.0.47:1234)
- Architecture: select “arm”

When you press Ok, the cross-debug section will start.

Note2: Be sure to compile your project with “CONFIG+=debug” in the qmake parameters

3.8.1 Loading QT demos on RD129

To test QT demos, you have to:

- Download from ELPA web site and put on a USB flash key: linux kernel (you have to choose the binary image tailored for your starter kit), standard file system, qt file system.
- Power-on starter kit with USB flash inserted, and press 'U' key on the serial debug terminal within 1 second. You should obtain bootloader's prompt.
- Type “update kernel uImage-rd126-2.6.32” (or “update kernel uImage-rd156-2.6.32”)
- Type “update fs”
- Type “boot”
- When you get linux prompt, Type “cd \”
- Type “tar -xjf /mnt/usb0/fs-qt.tar.bz2” and wait for completion (about 2 min.)
- If you have RD126, add to /linuxrc.user the appropriate fbset command
- If you have RD126, modify in /usr/bin/qtrun and /usr/bin/qtrot the mmWidth= value with the width in mm of the lcd panel's visible area.
- Reset system, or run /linuxrc.user

3.9 Working with FLTK

On the ELPA web site there is an archive containing all the additional include files and libraries (both static and dynamic) needed to work with fltk. Your project should refer to /usr/local/include for include files and to /usr/local/lib for static libraries.

To run them, you have to make the nano-X server running with the command: “`nano-X -p & nxcal -d /etc/nxcal.dat`” (once at startup), and then run your program.

Nxcal is the nano-X touchscreen calibration manager. When run, it reads the calibration data file, sets up nano-X calibration and then exits. If it cannot find that file, it performs a touchscreen calibration by asking the user to press some crosses on the screen, and then generates the calibration data file.

To use truetype fonts, you need to put the font files in the /lib/X11 directory in the target filesystem. In the same directory there must be also a “fonts.dir” text file describing the fonts associations (I suggest to start with the one in the nxlib sources).

FLTK has its own graphical resource editor named fluid, but unlike QT it needs an external IDE.

You can experiment with FLTK by unpacking in the target the precompiled FLTK demos available

on ELPA web site.

3.10 Compiling free libraries and applications (obsolete)

Here follows the instructions to recompile some of the freely available and usable libraries and applications downloaded from internet.

All of these presume you downloaded the appropriate source tarball, you expanded it into a dedicated directory and cd to chat directory.

To recompile kernel image (elpa patch file needed):

- `patch -p1 <patch-linux-2.6.xx.xx.elpa`
- `make menuconfig (if you want to configure kernel options)`
- `make headers_install`
- `make`

The compiled file image is located in arch/arm/boot/uImage

To recompile busybox 1.17.3:

- `copy in the root directory the provided ".config" file`
- `make menuconfig (if you want to customize it)`
- `make`

To recompile zlib 1.2.5:

- `CC=arm-linux-gcc ./configure`
- `make`
- `sudo make install`

To recompile libpng 1.4.4:

- `./configure –host=arm-linux CFLAGS=-I/usr/local/include`
- `make`
- `sudo make install`

To recompile giflib 4.1.6:

- `./configure –host=arm-linux`
- `make`
- `sudo make install`

To recompile tiff 3.9.4:

- `./configure –host=arm-linux`
- `make`
- `sudo make install`

To recompile libjpeg 8b:

- `./configure –host=arm-linux`
- `make`
- `sudo make install`

To recompile freetype 2.4.2:

- `./configure –host=arm-linux`
- `make`
- `sudo make install`

To recompile nano-X, nxlib and fltk please refer to the README file in the patch archive on the ELPA web site.

4 Warranty and disclaimer

RD129 cards have a warranty for 1 year from the date of purchase. In the case of board's fault not caused by a wrong use ELPA will provide for free a replacement card. This warranty does not include shipping costs.

Excessive nand flash use above 100000 erase-write cycles can gradually cause reduction of the usable space. This is normal, and is not covered by this warranty. Application programs should not write continuously in internal flash. If such request is needed, please consider using an external fram or cmos ram that have no limit on number of writes.

In any case, ELPA will not be responsible for any loss of data or damage caused by a board's fault.

The included software: Linux operating system, u-boot, busybox, uclibc, nano-X, nxlib, fltk, freetype, QT and all free softwares are in general without any warranty.

ELPA declares that RD129 cards with S/N >= 00097 are assembled with a RoHS process using all RoHS parts.

5 Updates history

17 jan 2008: Kernel 2.6.23.13:

- Changed filesystem structure with ramdisks in /mnt, /dev, /tmp
- Replaced hotplug with udev
- Integrated rd129.ko module into kernel.
- Added digital i/o and internal clocks driver
- Updated web pages with all softwares' links
- Updated rd126 demos: added fltk demo, fbvnc and mplayer

24 feb 2008: Kernel 2.6.24.2:

- h3600_tsraw driver disappeared, patched microwindows to interface directly with /dev/input/event0
- Now using nxcal to manage touchscreen calibration. Removed microwindows mine demo
- Updated busybox to 1.9.1
- Moved lcd autorecognition from kernel to demos' startup script
- Tested 800x480 lcd
- Added “samples” and “trigger” virtual files' parameters to adc driver
- Separated clock driver from gpio driver
- Added internal timers driver
- Added wiring description

04 may 2008: Kernel 2.6.25.1:

- Updated busybox to 1.10.1
- Audio driver for PCM3006 codec now working
- Compiled alsa-lib and alsa-utils for audio management
- Updated demo program to play .wav audio files (calling aplay)
- Updated microwindows to manage touchscreen and keyboard input events
- Activated and tested SPI device driver
- Clock driver: changed directory name from /sys/devices/platform/s3c2410-clk.0/ to /sys/devices/platform/clocks
- Gpio driver: changed directory name from /sys/devices/platform/s3c2410-gpio.0/ to /sys/devices/platform/s3c2410-gpio
- RTC: changed virtual file to acces nvram from to /sys/devices/platform/s3c2410-i2c/i2c-adapter/i2c-0/0-0068/nvram to /sys/class/i2c-adapter/i2c-0/0-0068/nvram
- Gpio driver: added capability to read or write a whole port with a single file access.
- Activated keyboard emulation driver using interrupt capable input pins.

29/08/2008: Kernel 2.6.26:

- Updated busybox to 1.11.2
- Solved nxlib problems about font's background color

- Enable truetype fonts in nano-X
- Added LM95241 temperature sensor driver
- Enabled SPI and LM75 sensor driver

15/11/09: U-boot 1.3.4:

- Changed flash partitioning
- Added boot-up splash screen

15/02/09: Kernel 2.6.28.5:

- Moved clock virtual files into /sys/class/clocks
- Tested RD126 with WM8731 codec driver

13/08/09: Kernel 2.6.30.4.elpa:

- Using “official” adc device, this reflects in changing virtual file names path from /sys/devices/platform/s3c2410-adc to /sys/devices/platform/s3c24xx-adc
- Using “official” driver for 24cXX I2Cbus eeproms.

26/12/09: Kernel 2.6.32.elpa:

- Using “official” adc driver, this reflects in changing virtual file names path from /sys/devices/platform/s3c2410-adc to /sys/class/hwmon/hwmon0/device/ and virtual filenames from adcX to adcX_raw (range 0..1023) and inX_input (range 0..3300)

26/12/09: QT embedded

- Added support for QT 4.6 embedded library
- Buildroot supported instead of ELDK.
- Replaced udev with mdev, included in busybox

05/01/09: QT examples

- Added instructions about loading demos into board

28/10/10: Major revision

02/04/11: Qt Creator, Buildroot

- Added instructions about Qt Creator's configuration and debug
- Updates for buildroot 02-2011

6 FAQ / Howto

When I run my program it exits with error “not found”. What does it mean ?

The 3 more common causes are:

1. You don't have exec permission. Give it with “chmod +x programname”.
2. If you are in the same directory and you didn't specified a path, run it with “./programname”
3. Some dlls are missing (see next question).

How can I know which dlls my program needs ?

On the host system, type “arm-linux-readelf -d executablename”

Compiled program is very big. How can I strip debug symbols from executable ?

On the host system, type “arm-linux-strip executablename”

When I run a program I lose console prompt until program ends, how can I get control again ?

Run program in a separated thread, following it with a “&”. Example: “programname &”

How can I terminate a program ?

killall execname

How can I add / change environment variables in host system ?

It depends by the Linux distribution you are using. On Fedora, they are stored in /etc/profile; on Ubuntu, they are in /etc/environment. Anyway, you must have root permissions to modify these files. In Fedora you can add a directory to the PATH using the macro “pathmunge”. You need to rebootstrap after modifying it.

Why isn't my batch file working ?

The 2 more common causes are:

1) The file has no execution permission (give it with chmod +x batchfilename)
2) You edited the file with a Windows editor, that adds <CR> characters after <LF>. Linux scripts must NOT have <CR> characters.

I'm using Ubuntu. How can I add an executable directory to PATH ?

- Add it into PATH= statement into /etc/environment, separated by :
- Add this line at the end of /root/.profile:

  export PATH=”$PATH:dirname_to_add”

For any additional question, please contact elpa.rizzo@gmail.com