Ingenic® Libra Development Board for Jz4730

Software Guide

Revision: 1.0
Date: Apr. 2006

北京君正集成电路有限公司
Ingenic Semiconductor Co. Ltd

Ingenic Libra Development Board for Jz4730

Software Guide

Release history

Date	Revision	Change
Apr. 2006	1.0	First release

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenicfor the current documentation and errata.

Ingenic Semiconductor Co. Ltd

E801C, Power Creative Building,
No.1, Shangdi East Road,
Haidian District, Beijing 100085, China
Tel: 86-10-58851003
Fax: 86-10-58851005
Http: //www.ingenic.cn

Content

1 Overview

Libra is the development board for SOC JZ4730. It can help you to try most of integrated functions of JZ4730 and it can help you to develop your own products.

1.1 Hardware

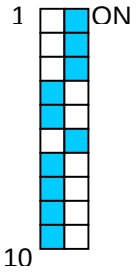
- CPU: JZ4730
- SDRAM: 64MB
- NOR-Flash: Intel TE28F128 (16MB)
- NAND-Flash: SAMSUNG K9F2808U0C (16MB)
- MMC/SD card slot: 1
- LCD: SAMSUNG LTP400WQ (480x272 TFT)
- Audio: AC97 (Philips UCB1400) / I2S (TI TSC2301)
- Camera: OV7111
- PS/2 Keyboard
- USB Host: 2
- USB device: 1
- Network: 10/100M Ethernet
- RS232: 2
- ISO7816: 2
- RTC: Philips PCF8563

1.2 Software

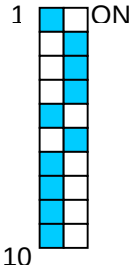
- Boot loader: U-Boot-1.1.3 / NAND-Boot
- OS Kernel: CELinux-040503
- Demo Application: Qtopia-free-1.6.1

1.3 Run Libra

Give a 12V power supply and switch on. It can run, and for 30 second, Qtopia appears on LCD screen. If you want to use PS/2 Keyboard on this board, please attach it before turning it on. A USB mouse can be used in this system. The default boot setting is from NAND-flash. There's a simple LINUX root directory stored in NAND-flash and it contains basic running environment of Qtopia. Qtopia applications are stored in NOR-flash. You can choose booting from NOR-flash by change the state of SW1 as following.



Booting from NAND-Flash



Booting from NOR-Flash

There's a media file stored in directory /opt. You can use file manager of Qtopia to open it. Just double click on it is OK.

2 Development Environment

We recommend using LINUX environment to develop your product software. First of all, the company system administrator setup and run lots of useful services on a LINUX server. These services include: DHCP, NFS and TFTP.

2.1 Install Toolchain

In our provided CD-ROM, you can find the package of toolchain. We also provide toolchain runs on cygwin. Just follow the steps described in CD-ROM, you can install the toolchain easily.

```
$ cd /home
$ mkdir mipstools
$ cd mipstools
$ tar xfj mipseltools-jz-linux-20060308.tar.bz2
$ cd
$ vi /etc/profile
export PATH=$PATH:/home/mipstools/mipseltools-20060308/bin
```

2.2 Install “uudecode”

Please make sure that “uudecode” is installed in you system. It maintained in “shareutils” RPM package. If you use a cygwin develop environment, you might need to down load the source code from Internet and build it yourself.

2.3 Install “mkimage”

If your Linux patch is too old that it cannot generate uImage correctly when you build a uoot image. You might need to copy a generated “mkimage” from “uboot/tools” directory to linux kernel source tree location “arch/mips/uboot”

2.4 Install JTAG Utility

Please refer to the specific document.

2.5 Install NFS-ROOT

NFS service is setting up and runs well. You can decompress a root package from CD-ROM to export directory of NFS service.

3 Boot loader

There is two boot loaders can be used in Libra: one is U-Boot, which is used in NOR-flash booting. Another is NANDBoot, which is used in NAND-flash booting.

U-Boot is original from PPCBoot (for PowerPC). It has powerful functions. It supports serial port downloading, Ethernet downloading etc. It also supports NOR-flash and NAND-flash storage management.

NANDBoot is stored in the first block of NAND-flash. Its size is less than 4096 bytes. The target board will be booting up quickly with it.

3.1 Build U-Boot

First of all, get two packages from the Internet or CD-ROM. These packages are: “u-boot-1.1.3.tar.bz2” and “u-boot-1.1.3-jz-yyyymmdd.patch.gz”. Then patch the code as the following steps:

```
$ tar xfj u-boot-1.1.3.tar.bz2
$ cd u-boot-1.1.3
$ zcat ../u-boot-1.1.3-jz-yyyymmdd.patch.gz | patch –p1
```

Next, using make to configure and build it.

```
$ make libra_config
$ make
```

All u-boot have the default settings built in the binary. You can change this default setting by modifying the file: “u-boot-1.1.3/include/configs/libra.h”.

3.1.1 Use U-Boot

After burning the binary of U-Boot into the NOR-flash. Running the target board, and something will be output on serial port console. Run “help” command, there will be a command list displayed on console. And run “help <COMMAND>” will find the specific usage of <COMMAND>. If you want to download the Linux from the Ethernet, please follow the following steps. (Assume that your LINUX server IP is 10.0.0.1, and some services were running, including DHCP, TFTP and NFS).

```
LIBRA # setenv bootfile uImage
LIBRA # setenv bootargs mem=64M console=ttyS0,115200 ip=bootp
nfsroot=10.0.0.1:/nfsroot/mipsroot rw
LIBRA # setenv ethaddr 00:12:34:56:78:90
LIBRA # bootp
LIBRA # setenv serverip 10.0.0.1
```

```
LIBRA # tftp
LIBRA # bootm
```

You can use “printenv” to display the current settings. There are two commands can be used to setting environment: “setenv” and “askenv”. To save the environment into the flash memory, please use “saveenv” command. And U-BOOT supports lots of other commands; you can refer to the user manual of U-BOOT to learn.

3.1.2 Linux™ Command Line

U-Boot uses environment variable to transfer LINUX command to the kernel. This environment variable is “bootargs”. As we described before, the default value of this variable is placed in the file “u-boot-1.1.3/include/configs/libra.h”

3.1.2.1 Indicate the Memory Size

There are 64 MB SDRAM mounted on Libra, and then the relative command line argument will be:

```
mem=64M
```

If the SDRAM capacity is 16MB, then the 64 in above string will be changed to 16. Not all memory can be used by user applications. Some rooms will be occupied by LINUX kernel and init RAMDISK. If user uses dynamic-link technology, then some space will be occupied by dynamical libraries.

3.1.2.2 System Console

System console will be setting by using ”console=DEV,DEVARGS”. There can be several consoles in a system. The last declared console is the major console; witch will be used as a shell console when init runs shell. System console is used to output some messages from LINUX kernel. If you do not want these messages displayed, please use “quiet”. The following is demos of console setting:

```
console=tty1
console=ttyS0,115200
```

The second line indicates that we use serial port 0, and the baud rate of this port is 115200.

3.1.2.3 IEEE802.3 MAC Address

We can use the following line to tell the kernel the MAC address value is “”00:12:34:56:78:90”.

```
ethaddr=00:12:34:56:78:90
```

3.1.2.4 IP Address

The following lines indicate two method of setting machine IP. Linux supports 3 methods of getting IP address: ARP, BOOTP and DHCP. In fact, DHCP is the extension of BOOTP. We can set the

machine with a static value also (as the second line).

```
ip=bootp
ip=192.168.9.200:::::eth0
```

3.1.2.5 LINUX Root Device

Linux system needs a root file system. And this root file system can be placed on different media. This media is so called root device. The usually used root device is MTD, INITRD and NFSROOT. We provide some root packages on CD-ROM, please choose a suitable one as your working root. But they might not be suitable for you, please organize the suitable one by yourself.

MTD root device

```
root=/dev/mtdblock0 rw
```

NFSROOT root device

```
nfsroot=192.168.9.20:/nfsroot/developroot rw
```

When you using INITRD, you should download the INITRD image into the SDRAM, and then use the following line:

```
initrd=0x00700000,0x008e8dff root=/dev/ram0 rw
```

Of course, we also support building INITRD into the kernel. Then the command line becomes:
root=/dev/ram0 rw

3.2 NANDBoot

NANDBoot is used to initialize the system, and download LINUX kernel image and/or INITRD image from NAND-flash into SDRAM. And then transfers LINUX command line to the kernel. Finally, it will jump to the executive point of the KERNEL.

3.2.1 System Initialization

System initialization includes 3 major steps:

- Init clock and power management unit (CPM). CPM controls the working frequency of the system.
- Init GPIO function pins.
- Init SDRAM controller. Before loading, the SDRAM should be in ready state.

3.2.2 Booting Images

We provide the source code of NANDBoot for Libra. You can change it yourself. One point is booting image. In the source code, we used a 58 to indicate its length. The counting unit of it is “Block”. You can read the datasheet of the NAND-flash to get the block size. Of course, the block size is the

erase size. Another two arguments are start block number and loaded address. In the source code, we performed only one loading image, which is the LINUX kernel. Don't pay more attention on the load address and start block number. You might need to change the value of the image length.

3.2.3 Linux Command Line

NANDBoot performs a LINUX command line with maximum length of 256 bytes. And you can change the content of it to perform your own function.

3.3 How to Use JTAG tool to Program Images

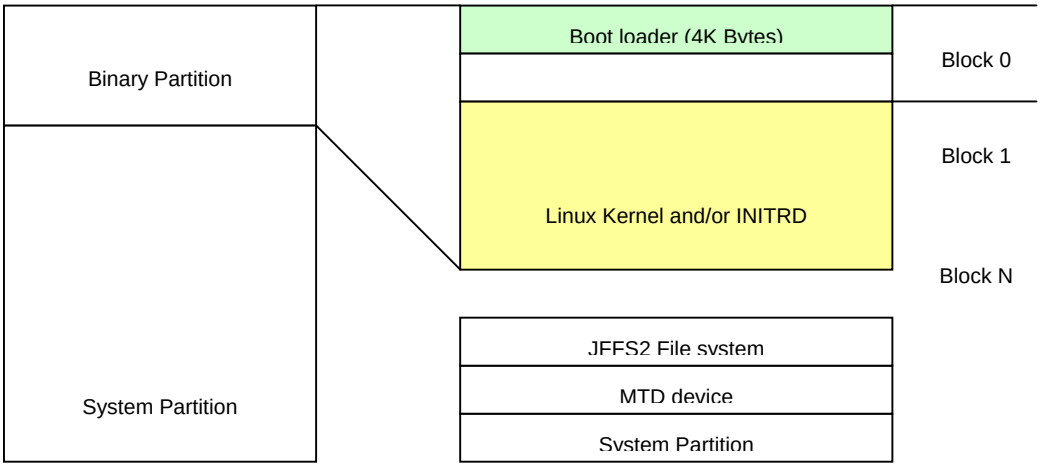
Users can use JTAG tools to burning U-BOOT and NANDBoot into the target board flash memories. You can refer to the specific document to get the details of JTAG tool.

4 NAND-flash Usage

JZ4730 supports a mechanism to boot from NAND-flash directly. Users can place 4K bytes boot code at the very beginning of the first block. When system starts up, this 4K bytes code will be executed first. This 4K bytes code is so called NANDBoot. As we described before, that NANDBoot does some basic initialization and load images stored in NAND-flash into the SDRAM, and then jump to the entry point. How did we manage the whole NAND-flash space? And how did we deal with it? This section will show you our some design details of NAND-flash booting.

4.1 NAND-flash Layout

NAND-flash is separated into two parts: one is binary partition and another is system partition. The binary partition is used to store NANDBoot, LINUX kernel image and/or INITRD image. The system partition is used for LINUX MTD device. And we established a JFFS2 file system on it. The following diagram shows the details.



4.2 Use JTAG Burning NAND-flash

As we described before, our JTAG tool support NAND-flash programming function. And this function can access the whole space of the NAND-flash. So, you should make sure that the operation is only in binary partition. DO NOT perform any operation on system partition; otherwise this might damage the existed JFFS2 file system in it.

4.3 Use “nandutil” Burning NAND-flash

We also support an application runs on target board to program NAND-flash. This application is so called “nandutil”. “nandutil” has the similar function with JTAG tool.

4.3.1 Burning NANDBoot

```
# nandutil –b loader.bin
```

4.3.2 Burning LINUX Image from Block1

```
# nandutil –b zImage –s 1
```

4.3.3 If we had an INITRD and Placed After LINUX Kernel Image (51 blocks)

```
# nandutil –b initrd.bin –s 52
```

4.3.4 Erase 1024 Blocks of NAND-flash from Block0

```
# nandutil –e 1024
```

If there were a second-hand NAND-flash mounted. The normal erase operation will be failure. The reason is that normal erase operation will be ignored on a bad TAG marked block. The second-hand NAND-flash may store some information in the OOB space, and the position might be bad TAG position. This will make the “nandutil” perform a wrong judgment. To avoid this, you can use an additional switch “-f”. This means performing operation by force. But this may lead another problem: the real bad flag might be erased either.

4.4 MTD Device

MTD (Memory Technology Device) is the standard device performed by LINUX. And it is the well-known device for embedded developers. We can establish JFFS2 file system on it. And the following shows the basic steps of set up a JFFS2 file system on MTD device.

Format the MTD partition:

```
# flash_eraseall –j /dev/mtd2
```

Mount the file system and copy the required files

```
# mount –t jffs2 /dev/mtdblock1 /mnt
# cd /mnt
# cp /home/destdir/* . –a
# sync
# cd /
# umount /mnt
```

If your NAND-flash was used before, please use “nandutil” to erase it by force.

```
# nandutil –e 1024 –f
```

5 CE-Linux™ Kernel

CE-Linux™ is developed for Consuming-Electronic Applications. It has a very fast booting speed and it has tiny kernel image size. It also performs power management function.

5.1 Basic Kernel Development Process

5.1.1 Decompress the Source Code and Patch It

```
$ tar xfj celinux-040503.tar.bz2
$ cd celinux-040503
$ zcat ../celinux-jz-yyyymmdd.patch.gz | patch –p1
```

5.1.2 Configure the Kernel

Before compile the kernel, we need to configure it. There are a lot of prepared configurations for you. Please choose one for Libra, and follow the steps shown below.

```
$ make defconfig-libra
$ make oldconfig
```

5.1.3 Build the Kernel

```
$ make dep
$ make zImage OR make uImage
```

“make zImage” will generates 3 kernel images finally.

- Standard kernel with debug information built-in and in ELF format. It is placed at:
  `celinux-040503/vmlinux`
- Compressed kernel in ELF format and it is placed at:
  `celinux-040503/arch/mips/zboot/images/vmlinuz.elf`
- Compressed kernel in binary format and it is placed at:
  `celinux-040503/arch/mips/zboot/images/zImage`

These three kernel images have the different purpose. The first one is used for disassembling and debugging. The second one is used for some bootloader that support ELF downloading. The third one is used for NANDBoot.

“make uImage” will generates 3 kernel images too.

- Standard kernel with debug information built-in and in ELF format. It is placed at:
  `celinux-040503/vmlinux`
- Compressed uImage kernel. It is placed at:
  `celinux-040503/arch/mips/uboot/uImage`
- Normal uImage kernel. It is placed at:

```
celinux-040503/arch/mips/uboot/uImage.uncompressed
```

5.2 CE-Linux Device Driver for Libra

5.2.1 Real-Time Clock (RTC)

Libra has an I2C RTC module: Philips PCF8563. You can use “hwclock” to get or set the current time.

```
Character devices
   [●]y [ ]- [ ]n Philips PCF8563 Real Time Clock (I2C Bus)
```

5.2.2 Smart Card Driver

The smart card driver is designed as a type of data channel. It does not perform ISO7816 protocol stack. But it performs some control interface to user: you can set the extra-guard time; set the F/D parameter; select transfer method (T=0/T=1). We also provide some application to show how to use it. We recommend it that to learn something about ISO7816 before using it.

Smart card device exists as two char device I-node. Its major code is 238, and the minor codes are 32 and 33.

```
Character devices
   [●]y [ ]- [ ]n JzSOC char devices support
 JzSOC char devices support
    [●]y [ ]- [ ]n JzSOC On-chip SmartCard Controller support
```

If there is no any I-node for this device in your root file system, please create it manually.

```
# mknod /dev/scc0 c 238 32
# mknod /dev/scc1 c 238 33
```

5.2.3 MTD Device Driver

We provide two MTD devices in Libra. One is based on NOR-flash. Another is based on NAND-flash. We created JFFS2 file system on both. NOR-flash MTD is “/dev/mtd1” and NAND-flash MTD is “/dev/mtd2”. Specially, “/dev/mtd0” indicates the whole NOR-flash. So, don’t perform any operation on it. Otherwise, your operation will damage u-boot image in it.

```
Memory Technology Devices (MTD)
   [●]y [ ]m [ ]n Memory Technology Device (MTD) support
   [●]y [ ]m [ ]n  MTD partitioning support
   [●]y [ ]m [ ]n  Direct char device access to MTD devices
   [●]y [ ]m [ ]n  Caching block device access to MTD devices
   RAM/ROM/Flash chip drivers
```

```
            [●]y [ ]m [ ]n Detect flash chips by Common Flash Interface
(CFI) Probe
            [●]y [ ]m [ ]n Flash chip driver advanced configuration
options
            [●]y [ ]m [ ]n  Specific CFI Flash geometry selection
            [●]y [ ]m [ ]n  Support 16-bit buswidth
            [●]y [ ]m [ ]n  Support 1-chip flash interleave
            [●]y [ ]m [ ]n Support for Intel/Sharp flash chips
         Mapping drivers for chip access
            [●]y [ ]m [ ]n JZ4730 LIBRA MTD support
         NAND Flash Device Drivers
            [●]y [ ]m [ ]n NAND Device Support
            [●]y [ ]m [ ]n NAND Flash device on JzSOC board
```

To use JFFS2 file system on MTD device is the classical choice in many embedded system. Just follow the settings below:

```
      File systems
         [●]y [ ]m [ ]n Journalling Flash File System v2 (JFFS2) support
         [0           ]  JFFS2 debugging verbosity (0 = quiet, 2 =
noisy)
         [●]y [ ]- [ ]n JFFS2 support for NAND chips
```