Quick start

Contents

Figures and Tables

1. Building Development Environment

Before you start, please check the NuWa package to ensure all components are present. The NuWa 470 contains:

- A NuWa 470 platform
- 5V DC power supply
- DVD-ROM containing original sources with Nuwa cross toolchain and documentation

1.1.Development Environment

First of all, you should have a development environment appears as in the diagram below:

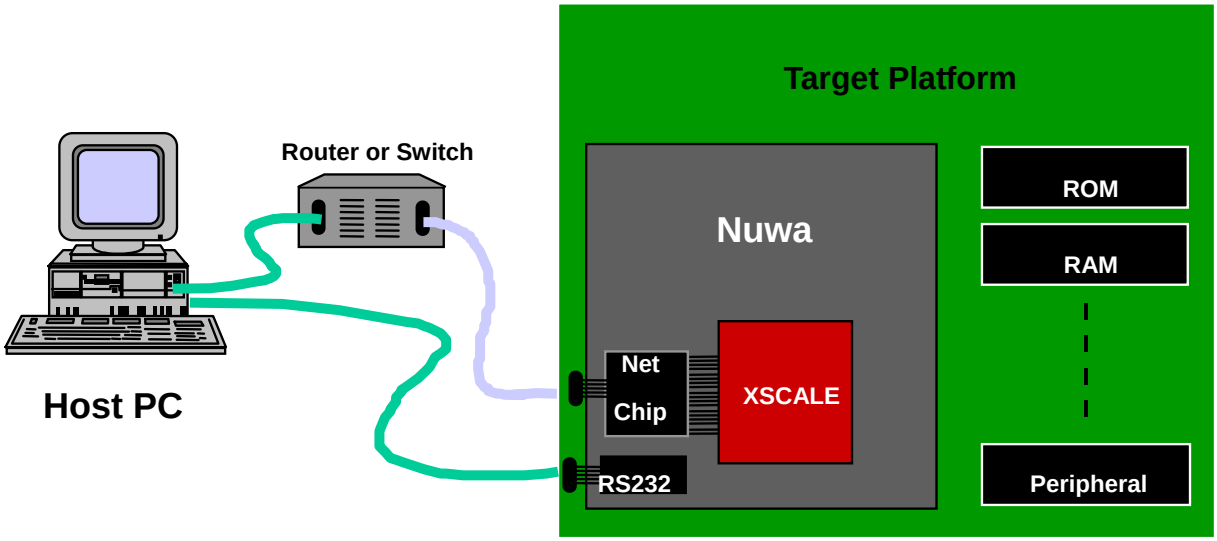

Figure 1. Development Environment

1.2. File list on CDROM

Now we explain each directory on CD ROM very simply.

- /app Application Software
- /compiler Cross compiler for target board
- /kernel Linux kernel for target board
- /rootfs Root file system for target board

- /rpm RPM for target board
- /u-boot Boot loader for target board
- /tools Tools software for PC
- /drivers/ts Touch screen driver for X Windows

2. System Setting

2.1.Setting Console

Connect a serial cable between your PC and the NuWa serial port. Start a terminal emulator on the PC and set it to 115200 baud, 8 bit, no parity and no flow control.

Figure 2. Setting up com port parameter

If you use Desktop Linux to download file to target, you have to know minicom usage first. Desktop Linux has minicom program for serial communication. It is used for command prompt of uboot or shell prompt of embedded linux.

Set up the values before using minicom program.

Select "Serial port setup" item.
Push "A" key for setting "Serial Device", then write serial port which is connected to target board. (If using COM1, write /dev/ttyS0, if COM2, write /dev/ttyS1.)

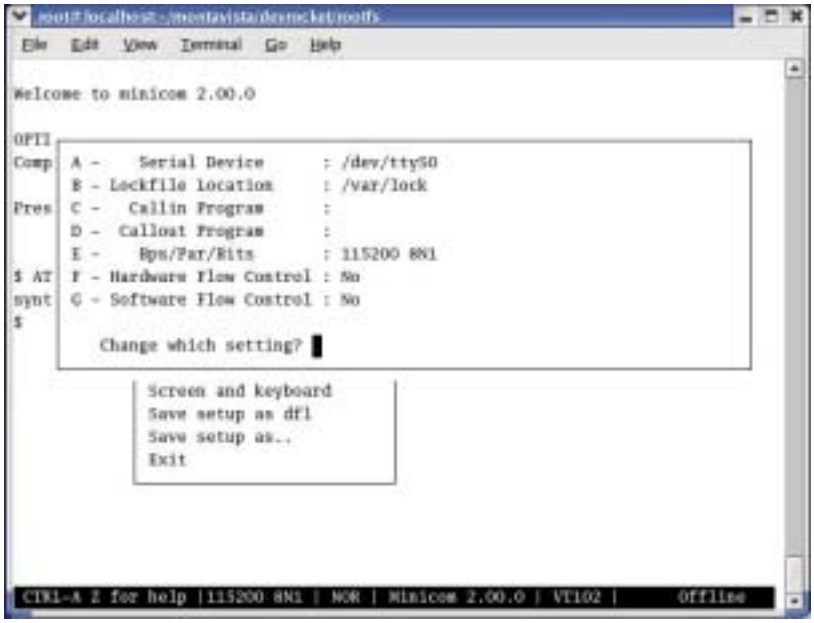

Figure 3. Setting up minicom

3. Boot Linux

On every board RESET or power up, do not press any key on keyboard. You should see the following message on your terminal emulator:

```
U-Boot 1.1.1 (Dec  8 2004 - 17:38:28)        <----------------------    Boot Loader Start

U-Boot code: A3080000 -> A3099988  BSS: -> A309DF88
RAM Configuration:
Bank #0: a0000000 64 MB
Bank #1: a4000000  0 kB
Bank #2: a8000000  0 kB
Bank #3: ac000000  0 kB
Flash: 32 MB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
## Booting image at 00040000 ...
   Image Name:   name                <------------ move kernel to SDRAM
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    640849 Bytes = 625.8 kB
   Load Address: a0008000
   Entry Point:  a0008000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...                      <--------------- Kernel Running

Linux version 2.4.20 (root@localhost.localdomain) (gcc version 3.3.1 (MontaVista
 3.3.1-3.0.10.0300532 2003-12-24)) #366 Wed Jan 19 16:20:07 CST 2005
CPU: XScale-PXA255 [69052d06] revision 6 (ARMv5TE)
```

```
CPU: D undefined 5 cache
CPU: I cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
CPU: D cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
Machine: ICPDAS SBC Platform
Ignoring unrecognised tag 0x00000000
Memory clock: 99.53MHz (*27)
Run Mode clock: 398.13MHz (*4)
Turbo Mode clock: 398.13MHz (*1.0, inactive)
On node 0 totalpages: 16384
zone(0): 16384 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/mtdblock2 console=ttyS0,115200
Calibrating delay loop... 397.31 BogoMIPS
Memory: 64MB = 64MB total
Memory: 63312KB available (1200K code, 232K data, 44K init)
XScale Cache/TLB Locking Copyright(c) 2001 MontaVista Software, Inc.
Dentry cache hash table entries: 8192 (order: 4, 65536 bytes)
Inode cache hash table entries: 4096 (order: 3, 32768 bytes)
Mount-cache hash table entries: 1024 (order: 1, 8192 bytes)
Buffer-cache hash table entries: 4096 (order: 2, 16384 bytes)
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
 Version ID = 0
LSP Revision 1
Starting kswapd
Disabling the Out Of Memory Killer
JFFS2 version 2.1. (C) 2001, 2002 Red Hat, Inc., designed by Axis Communications
 AB.
Serial driver version 5.05c (2001-07-08) with MANY_PORTS enabled
ttyS00 at 0xf8100000 (irq = 15) is a XSCALE UART
ttyS01 at 0xf8200000 (irq = 14) is a XSCALE UART
ttyS02 at 0xf8700000 (irq = 13) is a XSCALE UART
ttyS03 at 0xf4200000 (irq = 112) is a 16450
```

ttyS04 at 0xf4300000 (irq = 113) is a 16450
ttyS05 at 0xf4400000 (irq = 114) is a 16450
ttyS06 at 0xf4500000 (irq = 115) is a 16450
ttyS07 at 0xf4600000 (irq = 116) is a 16450
ttyS08 at 0xf4700000 (irq = 117) is a 16450
ttyS09 at 0xf4800000 (irq = 118) is a 16450
ttyS10 at 0xf4900000 (irq = 119) is a 16450
SA1100 Real Time Clock driver v1.02
SA1100/PXA Watchdog Timer: timer margin 60 sec
eth0: DM9000 9000-a46 at 0xf1000300, 00:e0:60:00:00:a8, IRQ 108.
eth1: DM9000 9000-a46 at 0xf1100300, 00:e0:60:00:00:58, IRQ 109.
SCSI subsystem driver Revision: 1.00
ac97_codec: AC97 Audio codec, id: NSC72(National Semiconductor LM4548A)
Probing ICPDAS SYSTEM Flash at physical address 0x00000000 (32-bit buswidth)
cfi_cmdset_0001: Erase suspend on write enabled
Using buffer write method
RedBoot partition parsing not available
cmdlinepart partition parsing not available
Probing ICPDAS DATA Flash at physical address 0x04000000 (16-bit buswidth)
cfi_cmdset_0001: Erase suspend on write enabled
Using buffer write method
Probing ICPDAS DATA SRAM at physical address 0x08000000 (32-bit buswidth)
Using static partitions on ICPDAS SYSTEM Flash
Creating 3 MTD partitions on "ICPDAS SYSTEM Flash":
0x00000000-0x00040000 : "U-BOOT"
0x00040000-0x00140000 : "KERNEL"
0x00140000-0x02000000 : "JIFF2 RFS"
Registering ICPDAS DATA Flash as whole device
Registering ICPDAS DATA SRAM as whole device
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
hc_isp116x.c: USB starting
hc_isp116x.c: USB ISP116x at f4100000/0 IRQ 104 Rev. 10 ChipID: 6122
usb.c: new USB bus registered, assigned bus number 1
USB HC dev alloc 384 bytes
Product: USB ISP116x Root Hub

```
SerialNumber: 0
hub.c: USB hub found
hub.c: 2 ports detected
usbdcore: usbdcore 0.1 034 2002-06-12 20:00 (dbg="")
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 4096 bind 8192)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
VFS: Mounted root (jffs2 filesystem) readonly.      <---------- Mount Root file
system
Freeing init memory: 44K
INIT: version 2.78 booting
INIT: Entering runlevel: 3


PXA Linux Preview Kit
Kernel 2.4.20 on an armv5tel
Linux login: root                          <------------ type ‘root’
login[51]: root login   on `ttyS0'

[root@Linux root]#                              <------------- Bash Running
```

3.1. Using the Ethernet

As you are rebooting, watch the console for error messages. Use the ping command to test your connectivity. Try to ping the gateway machine IP address first. This will test local connectivity on the LAN. If you cannot ping the gateway, you are not going to be able to connect to the Internet. If you can ping the gateway, try pinging a known host on the Internet. For example, ping www.yahoo.com will test both the ability to do a DNS lookup from your name server as well as your ability to connect to the Internet

3.2. Using the Audio

Change to folder /home/sound
Then you can play an mp3 file by specifying its name:
#./mp3player moon.mp3
4.3 Using the USB Host
In the bash shell, you should be able to plug a USB mouse into the USB slot on the taget board and receive input. To verify that the device is working, you can examine the input through the event interface device. First create the following character device (if it does not already exist):
mknod /dev/input/event0 c 13 64

A program, evtest, is provided to read from this device file. Run it with the following command:
evtest /dev/input/event0
As you use move the mouse (for instance), it should produce the following type of output:

Event: time 946695141.507730, type 2 (Relative), code 0 (X), value –1
Event: time 946695141.507734, type 2 (Relative), code 1 (Y), value –1
...

3.3. Using the Display

Change to folder /home/fbv
Then you can display a picture file by specifying its name:
#./fbv 6.jpg

4.5 Using the PCMCIA & CF Card
You should be able to insert a CF IDE Card (FAT32 format) into the Compact Flash slot on the taget board . Then you should type those command below
cardmgr
#mount –t vfat /dev/hda1 /mnt

Finally, you can see the files in the folder “/mnt”.

3.4. SRAM

We use the mke2fs command to create a standard EXT2 Linux filesystem and to read and write access on the SRAM device .

```
#mke2fs /dev/mtdblock5
        mke2fs 1.19, 13-Jul-2000 for EXT2 FS 0.5b, 95/08/09
        Filesystem label=
        OS type: Linux
        Block size=1024 (log=0)
        Fragment size=1024 (log=0)
        128 inodes, 1024 blocks
        0 blocks (0.00%) reserved for the super user
        First data block=1
        1 block group
        8192 blocks per group, 8192 fragments per group
        128 inodes per group

        Writing inode tables: done
        Writing superblocks and filesystem accounting information: done
```

To add files and folders, first mount the file system as ext2.

```
#mount –t ext2 /dev/mtdblock5 /mnt
```

4. CAN Bus and PC104 Bus Application

The Nuwa Family support CAN (Controller Area Network) and PC104 Bus. There is a sample environment appears as in the diagram below:

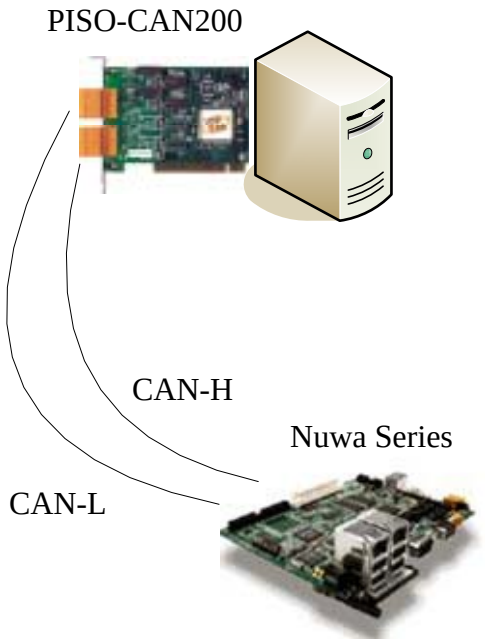

Figure 4. Can wiring connection

First create the following character device (if it does not already exist):

```
# mknod /dev/can c 120 0
```

To verify that the device is working, you can check the status of the can bus with cat /proc/can.

The can driver provide 2 sample file operations. Cansend is the example is designed to send out the CAN message and canmon is designed to receive the CAN message.

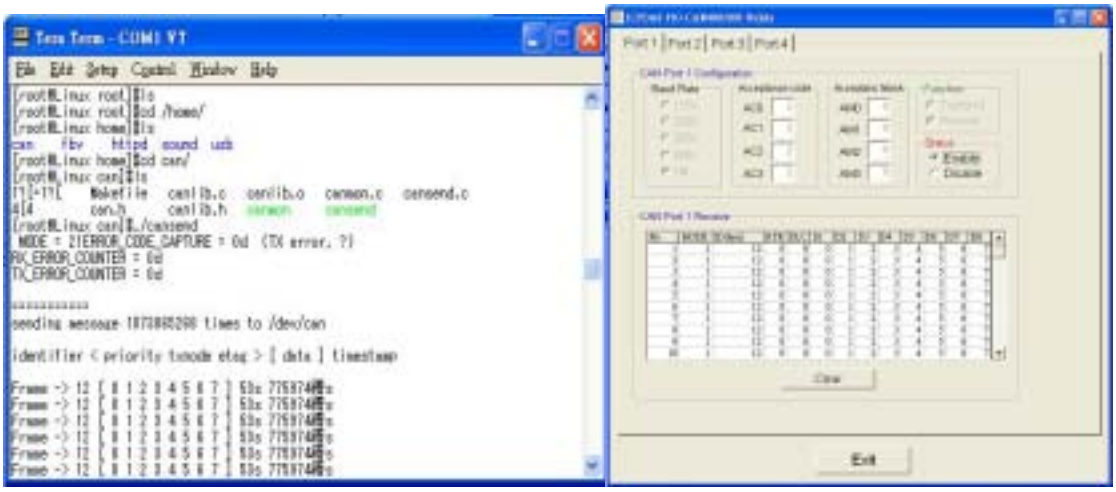

Figure 5. Cansend

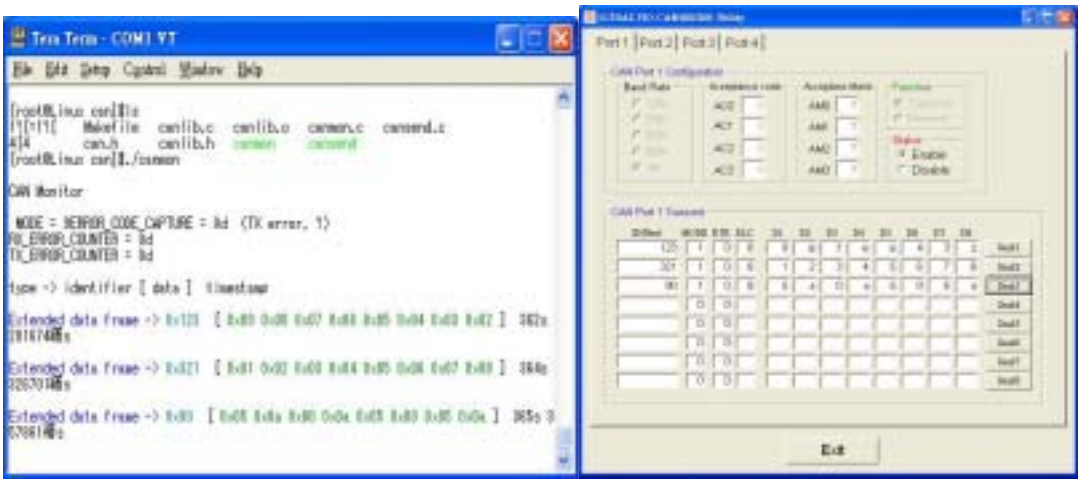

Figure 6. Canmon

The NuWa-430 support pc104 bus. These sample can be to apply to the 8 bit ISA card, the 16 bit ISA card, the 8 bit PC-104 card, and the 16 bit PC-104 card.

To reference the driver, you must have loadable module support enabled in your kernel. If you have kernel running, and do a "make install" as above, kernel should load the module on demand for you. To load the module manually, without kernel, type "insmod kito.o". (On most systems, "insmod kito" alone will do after you have done a "make install". Use the complete path if "insmod kito" alone comes up with a "file not found" error.) To unload the module manually, type "rmmod kito".

Here are a few things you may need to edit in "kito.h" for your specific

installation:

- "#define KITO_IRQ". Make sure this matches the IRQ Number on your card. Make sure it doesn't conflict with other cards.
- "#define KITO_IOADDR" Make sure this matches the value set via dip-switches on your card. Make sure it doesn't conflict with other cards.
- "#define KITO_MEMADDR" Use this to define where in memory you want the card to map in. 0x0000 is the most common default. The driver will softset the card to the value specified here when the driver is inserted via insmod or by kernel. Make sure this value doesn't conflict with a memory block used by another device.

After you have compiled the driver and inserted it into the kernel as a module, you are ready to run the utility applications in this package and talk to your pc104.c from Linux. The library just is a sample application to show the number on 7-Seg LED, but you'll have to write the applications yourself.

5. The I8K Module SDK

In this section, we will focus on examples for the description of and application of the functions found in the Libi8k.a. The Libi8k.a functions can be clarified into 3 groups which are listed in Fig. 19.

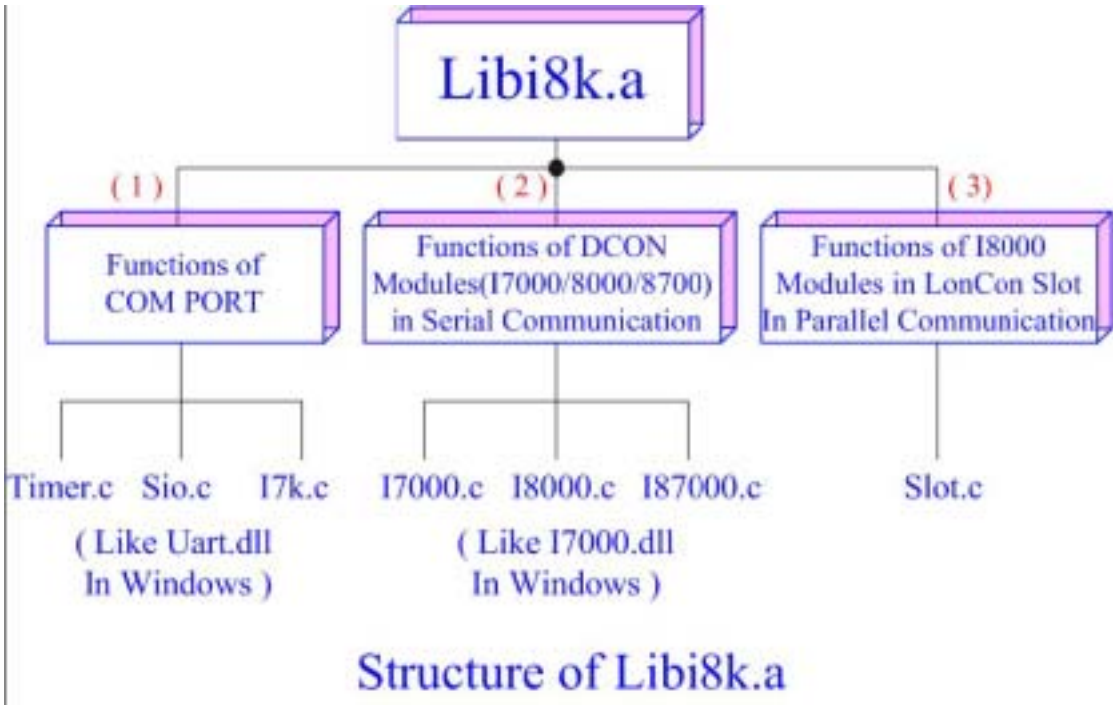

Figure 7. structure of Libi8k.a

Functions (1) and (2) in the Libi8k.a are the same as with the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000/I-8000/I-87000 in serial communication). You can refer to the DCON.DLL Driver manual which includes the functions on how to use DCON modules. The DCON.DLL Driver has already been wrapped into the Libi8k.a. Functions (3) of the Libi8k.a consist of the most important functions as they are specially designed for I-8000 modules in the LinCon-8000 slots. They are different from functions (1) and (2) because the communication of I-8000 modules in the LinCon-8000 slots are parallel and not serial. Therefore ICPDAS rewrote I8000.c to Slot.c especially for I-8000

modules in the LinCon-8000 slots. Here we will introduce all the funcitions for slot.c and they can be divided into eight parts for ease of use.

1. System Information Functions;

1. System Information Functions

2. Digital Input/Output Functions
3. Watch Dog Timer Functions
4. EEPROM Read/Write Functions
5. Analog Input Functions
6. Analog Output Functions
7. 3-axis Encoder Functions
8. 2-axis Stepper/Servo Functions

The functions in the Libi8k.a are specially designed for LinCon-8000. Users can easily find the functions they need for their applications from the descriptions in Lincon manualand in the demo programs developed by ICPDAS.