## *Purpose*

NOTE:
The KB9202B is shipped with boot loader, Linux kernel, and file system installed. It is not necessary to perform any of the operations listed, here, to use the board.

This document lists steps used to build a Linux-hosted development system for use in developing software applications or kernel device drivers for the KB9202B. The path taken is not necessarily the only way to go – several other approaches could be used.

There is no warranty of any kind supplied with this information. This document is simply a listing of possible steps performed in configuring a Linux-host desktop for use in developing applications on the KB9202B. First, the Operating System is installed - followed by cross compiler tool chain, file system creation, and finally sample application execution.

The reader is strongly encouraged to visit each of the developer sites. The software demonstrated and used here was developed by others and is currently open-source.

Notice that the tools directory is explicitly referenced in several sections. This directory can also be added to the PATH in order to simplify the commands. Throughout this document, the MAC is set with 02:05:05:07:09:14. You must obtain and use a valid MAC. The value shown is for demonstration only.

**Table 1: Table of Contents**

## Operating System Installation

1) Download FC4 (four ISO images) from one of the mirror sites listed at http://fedora.redhat.com/. Burn each image to a CD – use the special "Burn Image" option of your CD burner (e.g., Nero).
2) Install using configuration specified in Appendix A: Fedora Core OS Installation Options.
3) After installation, log in and open terminal (Applications – System Tools – Terminal).

## Cross Compiler and Related Tools Installation

The fastest, simplest, and most current method of building the uClibc libraries and associated tools is buildroot. We strongly recommend running the script on the development system (see instructions below). This freely-available utility downloads necessary updates and patches for the toolset and builds a sample root file system (which will not be used, here). You can get more information about this great tool at http://buildroot.uclibc.org/.

Notice that buildroot constructs the cross compiler toolchain, libraries (uClibc), utilities (BusyBox), and root file system. Only the toolchain and libraries are used in this document, but you are encouraged to explore the other packages.

A pre-built snapshot is available on the distribution CD. While we recommend running the script on the development system, this may not always be possible. In this case, copy the snapshot from the CD:

1) Copy the tarball
   `/ # cp /media/cdrecorder/kb9202b/tools_source/buildroot.tar.gz /`
2) Extract the tarball (this may take a long time)
   `/ # tar –zxvf buildroot.tar.gz`

Alternatively, the following describe the steps used to run the buildroot script:

1) Download the current buildroot script using Subversion
   `/ # svn co svn://uclibc.org/trunk/buildroot`
2) Set the configuration options
   ```
   / # cd buildroot
   /buildroot # make
   ```
3) Set the following parameters:
   ```
   Target Architecture → arm
   Target Architecture Variant →  arm920t
   Toolchain Options →
      Kernel Headers → 2.6.12
      (enable) Build/install c++ compiler and libstdc++
      (disable) Enable ccache support
      (enable) Use software floating point by default
   Exit and save changes
   ```
4) Build the package
   `/buildroot # make`
5) When prompted "`Use BX in function return (USE_BX) [Y/n/?]`", press "n" followed by Enter.
6) Accept the remaining, default options. The next stage of the build procedure takes approximately 35 minutes on a 933 MHz desktop.
7) In the version tested, the build fails with the following error:
   ```
   macro "index" requires 2 arguments, but only 1 given
   ```
8) Update `/buildroot/toolcahin_build_arm_nofpu/gcc-3.4.2/libstdc++-v3/include/ext/rope`
   `/buildroot # gedit toolchain_build_arm_nofpu/gcc-3.4.2/libstdc++-v3/include/ext/rope`
   Change the file as follows:

   ```
   ...
   #include <ext/hash_fun.h>
   ```

```
#undef index

# ifdef __GC
...
```

9) Update `/buildroot/toolcahin_build_arm_nofpu/gcc-3.4.2/libstdc++-v3/include/ext/ropeimpl.h`
`/buildroot # **gedit toolchain_build_arm_nofpu/gcc-3.4.2/libstdc++-v3/include/ext/ropeimpl.h**`
Change the file as follows:

```
...
#include <ext/numeric>

#undef index

namespace __gnu_cxx
...
```

10) Continue with the build procedure
`/buildroot # **make**`

## *Host Sever Configuration*

To utilize advantages of TFTP and NFS functionality, the host system must be configured as a server for each of these protocols. This assumes the network is functional on the host machine. If the `/usr/local/arm` directory does not exist, create it first.

To enable the NFS server, add a NFS share from `Desktop->System Settings->Server Settings->NFS`. Add a share directory for the target.
For example:

```
Directory: /usr/local/arm
Host(s): 192.160.1.79
Permissions: Read/Write
General Options: Sync write operations on request
User Access: Treat remote root user as local root
```

(192.160.1.79 is the IP address of the target)

Enable the services (tftp and nfs) to begin at system initialization in `Desktop->System Settings->Servers->Services`.

Because the default installation enables the firewall, external target access must be explicitly granted on each system initialization (this can be added to the init script):

```
iptables –I INPUT –s 192.160.1.0/24 –-dst 192.160.1.0/24 –j ACCEPT
```

If the rule should be applied on each boot, the rule can be saved with the following command:

```
/sbin/service iptables save
```

After rebooting the system, verify the servers are running (in a terminal window):

```
> netstat –a | grep nfs
       tcp    0      0      *:nfs   *:*     LISTEN
       udp    0      0      *:nfs   *:*

> netstat –a | grep tftp
```

```
udp     0       0       *:tftp  *:*
```

If tftp is not found or the server is not working, the service can be run in standalone mode to help determine the source of the problem:

```
> in.tftpd –l –s /tftpboot
```

## JFFS2 Tools Installation

This section is not required, but provided as a reference in case you want to build another JFFS2 image.

The following describe the steps required to construct a JFFS2 file system from an existing file tree and build the target utility to erase and program MTD devices:

1) Check-out the latest mtd utilities (password = anoncvs)
```
/usr/local/arm # cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs login
/usr/local/arm # cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs co mtd
/usr/local/arm # cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs logout
```
2) Build utilities
```
/usr/local/arm # cd mtd/util
/usr/local/arm/mtd/util # make
```
3) Copy the host-built utilities for later use
```
/usr/local/arm/mtd/util # cp mkfs.jffs2 host.mkfs.jffs2
```
4) Create full file system, if you have one (e.g., at /usr/local/arm/target_fs)
```
/usr/local/arm/mtd/util # ./host.mkfs.jffs2 –-pad=0x4000 –-eraseblock=0x4000 \
–l –-root=../../target_fs -o full_fs.bin
```
5) (Optional) Clean the directory
```
/usr/local/arm/mtd/util # make clean
```
6) (Optional) Rebuild the tools for use on the target
```
/usr/local/arm/mtd/util # make \
CROSS=/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux-
```
7) (Optional) The build fails at mkfs.jffs2.c, but individual utilities can be built as required. This step is executed to build the flash_erase(all) utility for the target.

Step 4 is listed for reference and can be executed after the file system has been constructed (see File System Construction).

## Boot Loader

The KB9202B utilizes a three stage boot loader: processor internal boot loader, EEPROM configuration boot loader, and NOR FLASH u-boot (at 0x10f80000). This approach leverages a standard boot loader interface (u-boot) with a small penalty in boot time. If boot time is a critical parameter, other methods are available to *greatly* reduce time to boot the kernel (fast boot time is only a few seconds to shell).

The following describe steps required to build the u-boot boot loader:

1) Download u-boot (Press Enter when prompted for password)
```
/usr/local/arm # cvs \
–d:pserver:anonymous@u-boot.cvs.sourceforge.net:/cvsroot/u-boot login
/usr/local/arm # cvs –z3 \
–d:pserver:anonymous@u-boot.cvs.sourceforge.net:/cvsroot/u-boot co –P u-boot
```
2) Build
```
/usr/local/arm # cd u-boot
/usr/local/arm/u-boot # make \
CROSS_COMPILE=/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux- kb9202_config
/usr/local/arm/u-boot # make \
CROSS_COMPILE=/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux-
```
3) Copy the image in preparation for download
```
/usr/local/arm/u-boot # cp u-boot.bin /tftpboot/
```
4) Copy the mkimage utility to be used in the next section
```
/usr/local/arm/u-boot # cp tools/mkimage /usr/local/sbin/
```
5) Copy the download utility
```
/usr/local/arm/u-boot # cd ..
```

```
/usr/local/arm # mkdir host_xmodem
/usr/local/arm # cp /media/cdrecorder/kb9202b/host_xmodem/* host_xmodem
/usr/local/arm # cp host_xmodem/download /usr/local/sbin/
```

## Kernel Installation

The following describe steps required to build the Linux kernel (2.6.17):

1) Make a directory for the kernel source
```
/ # mkdir –p /usr/src/arm
/ # cd /usr/src/arm
```
2) Copy the source and patches (from download or distribution CD)
```
/usr/src/arm # cp /media/cdrecorder/kb9202b/tools_source/2.6.17-at91.patch.gz .
/usr/src/arm # cp /media/cdrecorder/kb9202b/tools_source/linux-2.6.17.tar.bz2 .
/usr/src/arm # cp /media/cdrecorder/kb9202b/tools_source/kb9202_nand_patch .
```
3) Extract the kernel source
```
/usr/src/arm # tar –jxvf linux-2.6.17.tar.bz2
/usr/src/arm # rm –f linux-2.6.17.tar.bz2
```
4) Patch the kernel for AT91RM9200
```
/usr/src/arm # gunzip 2.6.17-at91.patch.gz
/usr/src/arm # patch –p1 –d linux-2.6.17 < 2.6.17-at91.patch
/usr/src/arm # patch –p1 –d linux-2.6.17 < kb9202_nand_patch
/usr/src/arm # rm –f kb9202_nand_patch
/usr/src/arm # rm –f 2.6.17-at91.patch
```
5) It is strongly recommended to create a copy of the kernel source
```
/usr/src/arm # mkdir linux-2.6.17.base
/usr/src/arm # cp –dR linux-2.6.17/* linux-2.6.17.base/
```
6) Create a link to the directory
```
/usr/src/arm # ln –s linux-2.6.17 linux
```
7) Copy the default kernel configuration file into the source directory
```
/usr/src/arm # cd linux
/usr/src/arm/linux # cp /media/cdrecorder/kb9202b/config_files/kernel .config
```
8) Build the kernel image
```
/usr/src/arm/linux # make ARCH=arm \
CROSS_COMPILE=/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux- Image
```
9) Create the u-boot compatible image using one of the following (trading speed for size)
   a. Larger size, but faster (default)
```
/usr/src/arm/linux # mkimage –n 'KB9202B 2.6.17 kernel' \
–A arm –O linux –T kernel –C none –a 20008000 \
–e 20008000 –d arch/arm/boot/Image /tftpboot/uImage
```
   b. Smaller size, but slower (approximately twice as long)
```
/usr/src/arm/linux #/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux-objcopy \
–O binary –R .note –R .comment –S vmlinux linux.bin
/usr/src/arm/linux # gzip -9 linux.bin
/usr/src/arm/linux # mkimage –n 'KB9202B 2.6.17 kernel' \
–A arm –O linux –T kernel –C gzip –a 20008000 \
–e 20008000 –d linux.bin.gz /tftpbot/uImage
```

## ramMonitor Execution

A set of utility functions is provided in the 'ramMonitor' program. These utilities are used to erase and program EEPROM, NOR FLASH, and NAND FLASH. The program is installed at 0x10f00000 and can be executed from the bootloader (u-boot) with the command "go 10f00000". See *KB9202 User's Guide* for more information on this utility. With the conversion to u-boot, ramMonitor is only required for crash-recovery (if one of the boot loaders has been corrupted) or to reset the factory default installation.

## BusyBox Utility Construction

The following describes the step taken to build the BusyBox utility functions:

1) Download busybox-1.1.2-tar.bz2 from http://busybox.net/downloads/ and place in `/usr/local/arm`

2) Extract
```
/usr/local/arm # tar –jxvf busybox-1.1.2.tar.bz2
```
3) The file busybox-1.1.2.tar.bz2 is no longer required and can be deleted.
4) Use the predetermined configuration from distribution CD or download
```
/usr/local/arm # cd busybox-1.1.2
/usr/local/arm/busybox-1.1.2 # cp \
/media/cdrecorder/kb9202b/config_files/bb.txt .config
```
5) Prepare for compilation
```
/usr/local/arm/busybox-1.1.2 # make \
CROSS=/buildroot/build_arm_nofpu/staging_dir/bin/arm-linux- menuconfig
```
6) Change as desired, or just leave as is (default)
7) Exit and save changes
8) Compile
```
/usr/local/arm/busybox-1.1.2 # make CROSS=/usr/local/arm/bin/arm-linux-
```
9) Install to `/usr/local/arm/busybox-1.1.2/_install`
```
/usr/local/arm/busybox-1.1.2 # make CROSS=/usr/local/arm/bin/arm-linux- install
```
10) Set the BusyBox executable file with setuid root flags
```
/usr/local/arm/busybox-1.1.2 # chmod 4755 _install/bin/busybox
/usr/local/arm/busybox-1.1.2 # cd ..
```

The default configuration file uses static linkage. This increases the size of the BusyBox executable file, but removes the dependency on libraries installed on the target.

## *File System Construction*

The following describe the steps taken in building the default file system:

1) Copy and extract the hollow target file system tree from distribution or download
```
/usr/local/arm # cp /media/cdrecorder/kb9202b/factory_images/target_fs.tar.gz .
/usr/local/arm # tar –zxvf target_fs.tar.gz
/usr/local/arm # rm –f target_fs.tar.gz .
```
2) Copy utilities
```
/usr/local/arm # cp –Rd busybox-1.1.2/_install/* target_fs/
/usr/local/arm # cp /buildroot/build_arm_nofpu/root/sbin/ldconfig target_fs/sbin
```
3) Edit the network configuration file and select the target IP configuration mode (static or DHCP) following the instructions in the file
```
/usr/local/arm # gedit target_fs/etc/network/interfaces
```
4) Copy shared libraries to file system
```
/usr/local/arm # cp –d /buildroot/build_arm_nofpu/root/lib/* target_fs/lib
```
5) Change ownership of the file tree
```
/usr/local/arm # chown –R 0:0 target_fs/*
```

This file system tree (`/usr/local/arm/target_fs`) can now be used to create the JFFS2 image (see JFFS2 Tools Installation).

## *Mounting a USB Mass Storage Device (keydisk)*

This section is optional, but useful for increasing the storage on the KB9202B.

By default, the power to the USB host port is disabled via a jumper. In order to utilize standard keydisks, this jumper must be moved.

Notice, several warning messages regarding the second USB host port are emitted. This is normal as the second host port is inaccessible on this implementation of the processor.

```
<moved jumper JP1 to power the USB host port>

<inserted the USB keydisk now>

[root@KB9202B:/]
[root@KB9202B:/] usb 1-1: USB disconnect, address 11
```

```
usb 1-1: new full speed USB device using at91_ohci and address 12
usb 1-1: new full speed USB device using at91_ohci and address 13
usb 1-1: configuration #1 chosen from 1 choice
scsi2 : SCSI emulation for USB Mass Storage devices
  Vendor: SanDisk   Model: Cruzer Micro      Rev: 0.3
  Type:   Direct-Access                      ANSI SCSI revision: 02
SCSI device sda: 501759 512-byte hdwr sectors (257 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
SCSI device sda: 501759 512-byte hdwr sectors (257 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
 sda:<7>usb-storage: queuecommand called
 sda1
sd 2:0:0:0: Attached scsi removable disk sda
sd 2:0:0:0: Attached scsi generic sg0 type 0

[root@KB9202B:/] mount -t usbfs none /proc/bus/usb
[root@KB9202B:/] mount /dev/sda1 /mnt/usb
[root@KB9202B:/] df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/mtdblock0           32.0M      2.7M     29.3M   8% /
/dev/sda1               243.7M    114.1M    129.6M  47% /mnt/usb
[root@KB9202B:/] mount
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
none on /proc/bus/usb type usbfs (rw)
/dev/sda1 on /mnt/usb type msdos (rw,fmask=0022,dmask=0022,codepage=cp437)
[root@KB9202B:/]
```

## *Mounting a SD/MMC Device*

This section is optional, but useful for increasing the storage on the KB9202B.

Due to an erratum against the Atmel AT91RM9200 processor, stronger pull-up resistors may be required (R45 and R50). This is somewhat dependent on the system installation and the specific SD/MMC. KB9202B boards with date code F10122006-1.30 and later have the stronger pull-ups installed (22.1k), whereas previous versions utilize 100k.

Inserting a SD/MMC will cause status messages to be displayed on the target. The card file system type can be set with standard tools (e.g. mkfs.ext2, mkfs.ext3, etc.).

```
[root@KB9202B:/] << SD/MMC inserted now >>
[root@KB9202B:/] mmcblk0: mmc0:0001 SDM128 125440KiB
 mmcblk0: p1

[root@KB9202B:/] mount -t ext2 /dev/mmcblk0p1 /mnt/mmc
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
[root@KB9202B:/] mount
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
/dev/mmcblk0p1 on /mnt/mmc type ext2 (rw)
[root@KB9202B:/] df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/mtdblock0           32.0M      2.7M     29.3M   8% /
/dev/mmcblk0p1          118.5M      2.6M    109.7M   2% /mnt/mmc
[root@KB9202B:/]
```

## Booting from SD/MMC Device

To boot from SD/MMC root file system, create the file system (e.g., ext2) on the card and copy the `/usr/local/arm/target_fs` directory onto the card. Then, change the u-boot boot arguments:

```
setenv bootargs console=ttyS0,115200 noinitrd root=/dev/mmcblk0p1 rootfstype=ext2
```

To make this change persistent, save the environment (“saveenv”) in u-boot.

Press the reset switch and watch it boot from the SD/MMC!

## *Restoring the Factory Installation*

These instructions assume the target file system has been created according to the steps provided, above (at `/usr/local/arm/target_fs`). For the tftp transfers listed, the host server must be enabled and the factory image files located at `/tftpboot`. The NFS server must be enabled and configured as stated, above. The process of exiting/entering the terminal is only required for terminal programs which do not support x-modem downloads: e.g., minicom. For other terminals, it is possible to perform the download within the terminal itself. Certain operations take some time - please be patient.

Erasing bad blocks on NAND devices can lead to unpredictable results – which the JFFS2 file system can repair. The entire device is erased in the following section; however, it is strongly recommended to use the `flash_eraseall` command previously generated.

Summary of operations:

1) Open terminal to target (minicom, HyperTerminal, etc.).
2) Short EEPROM jumper JP3 pins 1-2.
3) Press and release the reset button.
4) Verify ‘C’ characters on terminal.
5) Exit terminal.
6) Download stage1.bin using x-modem.
7) Download ramMonitor.bin using x-modem.
8) Open terminal to target.
9) Erase flash (“f e”).
10) Erase NAND (“nand e”).
11) Determine suitable location for download buffer (“x”). Use 0x2000d000.
12) Set the target MAC (“m 2 5 5 7 9 14”).
13) Set the target IP (“ip 192 160 1 79”).
14) Set the server IP (“server_ip 192 160 1 204”).
15) TFTP download EEPROM boot loader. (“tftp 0x2000d000 bootloader.bin”).
16) Verify the TFTP transfer image size (“tftp”).
17) Program the EEPROM boot loader (“eewrite 0 2000d000 <image size>”).
18) TFTP download the flash boot loader (“tftp 2000d000 u-boot.bin”).
19) Verify the TFTP transfer image size (“tftp”).
20) Program the flash boot loader (“f p 10f80000 2000d000 <image size>”).
21) TFTP download ramMonitor (“tftp 2000d000 ramMonitor.bin”).
22) Verify the TFTP transfer image size (“tftp”).
23) Program ramMonitor (“f p 10f00000 2000d000 <image size>”).
24) TFTP download the kernel image (“tftp 2000d000 uImage”).
25) Verify the TFTP transfer image size (“tftp”).
26) Program the kernel (“f p 10000000 2000d000 <image size>”).
27) Press the reset switch.
28) Press a key when u-boot begins the countdown.
29) Set the MAC (“setenv ethaddr 2.5.5.7.9.14”).
30) Set the IP (“setenv ipaddr 192 160 1 79”).
31) Save the environment (“saveenv”).
32) Ping the host (“ping 192.160.1.204”).
33) Adjust the boot arguments (“setenv bootargs console=ttyS0,115200 root=/dev/nfs nfsroot=192.160.1.204:/usr/local/arm/taget_fs ip=192.160.1.79”).
34) Boot the kernel image (“bootm 10000000”).

35) Login as ‘root’, when prompted.
36) TFTP download the file system image (“tftp –l full_fs.bin –r full_fs.bin 192.160.1.204”).
37) Write the image to NAND (“cp full_fs.bin /dev/mtd0”).
38) Reboot (“reboot”).

Detailed listing of operations:

```
~ #cd /usr/local/arm
/usr/local/arm #download /tftpboot/stage1.bin

From: WAIT_FOR_START -> SEND_BLOCK
..................................................
From: SEND_BLOCK -> SEND_EOT
.
Sending EOT
Transfer complete
From: SEND_EOT -> FINISHED

 Transmission complete: size = 7296 (0x    1C80)
/usr/local/arm #download /tftpboot/ramMonitor.bin

From: WAIT_FOR_START -> SEND_BLOCK
................................................................................
................................................................................
................................................................................
.....................................................................
From: SEND_BLOCK -> SEND_EOT
.
Sending EOT
Transfer complete
From: SEND_EOT -> FINISHED

 Transmission complete: size = 40832 (0x    9F80)
/usr/local/arm #minicom


Welcome to minicom 2.00.0

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n
Compiled on Mar  7 2005, 10:29:09.

Press CTRL-A Z for help on special keys



>f e
 Erasing block: 0x10000000
   ( cut )
 Erasing block: 0x10FA0000
 Erasing block: 0x10FC0000
 Erasing block: 0x10FE0000


> nand e
 Erasing block offset: 0x00000000... NAND: ready
 PASS
   ( cut )
 Erasing block offset: 0x01FF4000... NAND: ready
 PASS
 Erasing block offset: 0x01FF8000... NAND: ready
 PASS
 Erasing block offset: 0x01FFC000... NAND: ready
 PASS

>x
 Local buffer available of size: 4194304 bytes (4MB) at address: 0x2000C6D8

 Last x-modem transfer: FAIL (or not initiated)
```

```
>m 2 5 5 7 9 14


>ip 192 160 1 79


>server_ip 192 160 1 204


>tftp 2000d000 bootloader.bin
Starting tftp download of file: bootloader.bin at: 0x2000D000


>tftp
Starting tftp download of file:  at: 0x00000000

 -- Last tftp transfer info --
     address: 0x2000D000
     size: 0x00000748


>eewrite 0 2000d000 760
.............................


>tftp 2000d000 u-boot.bin
Starting tftp download of file: u-boot.bin at: 0x2000D000


>tftp
Starting tftp download of file:  at: 0x00000000

 -- Last tftp transfer info --
     address: 0x2000D000
     size: 0x0001C31C


>f p 10f80000 2000d000 1c31c
.............


>tftp 2000d000 ramMonitor.bin
Starting tftp download of file: ramMonitor.bin at: 0x2000D000


>tftp
Starting tftp download of file:  at: 0x00000000

 -- Last tftp transfer info --
     address: 0x2000D000
     size: 0x00009F68


>f p 10f00000 2000d000 a000
.....


>tftp 2000d000 uImage
Starting tftp download of file: uImage at: 0x2000D000


>tftp
Starting tftp download of file:  at: 0x00000000

 -- Last tftp transfer info --
     address: 0x2000D000
     size: 0x00263D10


>f p 10000000 2000d000 263d10
```

```
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................

>echo "User action = Press reset switch now on KB9202B"

KB9202B(www.kwikbyte.com) v2.5


U-Boot 1.1.4 (Oct  3 2006 - 13:50:59)

DRAM:  64 MB
Flash: 16 MB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
U-Boot> echo "User action = Press a key to prevent continuation of u-boot"
U-Boot> setenv ethaddr 2.5.5.7.9.14
U-Boot> setenv ipaddr 192.160.1.79
U-Boot> saveenv
Saving Environment to EEPROM...
U-Boot> ping 192.160.1.204
host 192.160.1.204 is alive
U-Boot> setenv bootargs console=ttyS0,115200 root=/dev/nfs
nfsroot=192.160.1.204:/usr/local/arm/target_fs ip=192.160.1.79
U-Boot> bootm 10000000

## Booting image at 10000000 ...
   Image Name:   KB9202B 2.6.17 kernel
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2505936 Bytes =  2.4 MB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
OK

Starting kernel ...

Linux version 2.6.17 (root@dev.kwikbyte.com) (gcc version 3.4.2) #12 Wed Oct 4
14:01:34 MST 2006
CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T)
Machine: KB920x
Memory policy: ECC disabled, Data cache writeback
Clocks: CPU 180 MHz, master 60 MHz, main 10.000 MHz
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists
Kernel command line: console=ttyS0,115200 root=/dev/nfs
nfsroot=192.160.1.204:/usr/local/arm/target_fs ip=192.160.1.79
AT91: 96 gpio irqs in 3 banks
PID hash table entries: 512 (order: 9, 2048 bytes)
...
cut
...
IP-Config: Complete:
      device=eth0, addr=192.160.1.79, mask=255.255.255.0, gw=255.255.255.255,
     host=192.160.1.79, domain=, nis-domain=(none),
     bootserver=255.255.255.255, rootserver=192.160.1.204, rootpath=
Looking up port of RPC 100003/2 on 192.160.1.204
Looking up port of RPC 100005/1 on 192.160.1.204
usb 1-2: device descriptor read/64, error -110
usb 1-2: new low speed USB device using at91_ohci and address 4
usb 1-2: device not accepting address 4, error -110
usb 1-2: new low speed USB device using at91_ohci and address 5
usb 1-2: device not accepting address 5, error -110
VFS: Mounted root (nfs filesystem).
```

```
Freeing init memory: 96K
init started:  BusyBox v1.1.2 (2006.10.03-22:35+0000) multi-call binary



   ---------------------------------
   -                               -
   - Sample file system for KB9202B -
   -                               -
   ---------------------------------
KB9202B login: root
Jan  1 00:01:35 login[711]: root login  on `ttyS0'



BusyBox v1.1.2 (2006.10.03-22:35+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

[root@KB9202B:~]
[root@KB9202B:~] cd /
[root@KB9202B:/] ping 192.160.1.204
PING 192.160.1.204 (192.160.1.204): 56 data bytes
84 bytes from 192.160.1.204: icmp_seq=0 ttl=64 time=0.7 ms
84 bytes from 192.160.1.204: icmp_seq=1 ttl=64 time=0.5 ms

--- 192.160.1.204 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.5/0.6/0.7 ms
[root@KB9202B:/] tftp -l full_fs.bin -r full_fs.bin -g 192.160.1.204
[root@KB9202B:/] cp full_fs.bin /dev/mtd0
[root@KB9202B:/] sync
[root@KB9202B:/] rm -f full_fs.bin
[root@KB9202B:/] reboot
The system is going down NOW !!
Sending SIGTERM to all processes.
Please stand by while rebooting the system.
Restarting system.
.

KB9202B(www.kwikbyte.com) v2.5


U-Boot 1.1.4 (Oct  3 2006 - 13:50:59)

DRAM:  64 MB
Flash: 16 MB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
## Booting image at 10000000 ...
   Image Name:   KB9202B 2.6.17 kernel
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2505936 Bytes =  2.4 MB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
OK

Starting kernel ...

Linux version 2.6.17 (root@dev.kwikbyte.com) (gcc version 3.4.2) #12 Wed Oct 4
14:01:34 MST 2006
CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T)
Machine: KB920x
Memory policy: ECC disabled, Data cache writeback
Clocks: CPU 180 MHz, master 60 MHz, main 10.000 MHz
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists
```

```
Kernel command line: console=ttyS0,115200 noinitrd root=/dev/mtdblock0
rootfstype=jffs2 mem=64M
AT91: 96 gpio irqs in 3 banks
PID hash table entries: 512 (order: 9, 2048 bytes)
...
cut
...
eth0: Link now 100-FullDuplex
eth0: AT91 ethernet at 0xfefbc000 int=24 100-FullDuplex (02:05:05:07:09:14)
eth0: Intel LXT971A PHY
NAND device: Manufacturer ID: 0x20, Chip ID: 0x75 (ST Micro NAND 32MiB 3,3V 8-bit)
Scanning device for bad blocks
Creating 1 MTD partitions on "NAND 32MiB 3,3V 8-bit":
0x00000000-0x02000000 : "nand_fs"
at91_ohci at91_ohci: AT91 OHCI
at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
at91_ohci at91_ohci: irq 23, io mem 0x00300000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
MMC: 4 wire bus mode not supported by this driver - using 1 wire
TCP bic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 96K
usb 1-2: new low speed USB device using at91_ohci and address 2
usb 1-2: device descriptor read/64, error -110
usb 1-2: device descriptor read/64, error -110
usb 1-2: new low speed USB device using at91_ohci and address 3
usb 1-2: device descriptor read/64, error -110
usb 1-2: device descriptor read/64, error -110
usb 1-2: new low speed USB device using at91_ohci and address 4
usb 1-2: device not accepting address 4, error -110
usb 1-2: new low speed USB device using at91_ohci and address 5
init started:  BusyBox v1.1.2 (2006.10.03-22:35+0000) multi-call binary
usb 1-2: device not accepting address 5, error -110



   ----------------------------------
   -                                -
   - Sample file system for KB9202B -
   -                                -
   ----------------------------------
KB9202B login: root
Jan  1 00:00:27 login[712]: root login  on `ttyS0'



BusyBox v1.1.2 (2006.10.03-22:35+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

[root@KB9202B:~]
[root@KB9202B:~] df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/mtdblock0           32.0M      2.2M     29.8M   7% /
[root@KB9202B:~] mount
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
[root@KB9202B:~]
```

## *Sample Applications*

The following sample applications are developed on the host. Once the application is built, the executable can be copied to the target file system for stand-alone operation. Each application directory includes a README file with a brief description and instructions. The app can be built in static or dynamic configuration. In the static configuration, the shared libraries are not required to exist on the target. This saves space on the target at the expense of increased executable file size. On first powering the board (only required once), execute 'ldconfig' to resolve shared library mappings.

These examples assume the NFS server is enabled and active.

1) Mount the NFS-hosted drive on the target.
   ```
   [root@KB9202B:/] mount –t nfs –o nolock 192.160.1.204:/usr/local/arm/target_fs \
   /mnt/nfs
   ```
2) Build the application (on the host) using 'make' command within the corresponding directory.
   ```
   /usr/local/arm/target_fs/sampleApps/math/shared # make
   ```
3) Execute the application (on the target).
   ```
   [root@KB9202B:/mnt/nfs/sampleApps/math/shared/] ./hello.out
   ```

## Application using Math Library

See target file system directory `sampleApps/math`.

## Application using C++ and the STL

See target file system directory `sampleApps/cpp`.

## *Appendix A: Fedora Core OS Installation Options*

First, ensure valuable data is copied or backed-up before installing the new OS.

Language Selection = English
Keyboard Configuration = U.S. English
Upgrade Examine (only is previous installation exists) = selected “Install Fedora Core”
Installation Type = Custom
Disk Partitioning Setup = Manually partition with Disk Druid
Disk Setup =

1) Deleted existing partitions
2) Added 20GB partition (ext3) for ‘/’ mount point
3) Added 20GB partition (ext3) for ‘/usr/ mount point
4) Added 3GB partition (swap)

Network Configuration = Enter values according to your network
Firewall Configuration = Enable firewall
Time Zone Selection = Set according to your time zone
Set Root Password = Enter a password – DO NOT FORGET THIS PASSWORD
Package Group Selection (items listed are differences from default selection):
Many of the following options are enabled due to personal preference.

1) Enable KDE and kdeadmin in the submenu
2) Enable Editors and Emacs in the submenu
3) Enable Engineering and Scientific
4) Enable xpdf in Office/Productivity
5) Enable Server Configuration Tools
6) Enable Legacy Network Server and tftp-server in the submenu
7) Enable Development Tools and ddd, expect, and subversion in submenu
8) Enable KDE Software Development
9) Enable Legacy Software Development
10) Enable Eclipse
11) Enable Administration Tools
12) Enable System Tools and festival, iptraf, and net-snmp-utils in submenu

Total size of selected packages is less than 3.3GB.
About to Install = click ‘Next’, verify you have the required CDs, and wait for the installation to complete (about 40 minutes on 900 MHz desktop)