Hogeschool Utrecht Linux Kennisgroep

Customizing Sweex LB000021

Niek Linnenbank

June 11, 2008

$Id: Customizing_Sweex_LB000021.tex 94 2008-06-10 21:09:02Z svn $

Contents

1 Introduction

This document describes how you can tweak your Sweex LB000021 router to fit your own needs. The intended audience are people working with embedded systems, Linux and open source in general.

Suggestions, questions and critics are more than welcome on <*niek.linnenbank@planet.nl*>.I would especially like to thank Adrie van Doesburg for teaching me about embedded devices, and everyone else for reading this document.

Niek Linnenbank

2 Sweex LB000021

2.1 Hardware

Example 1 /proc/cpuinfo

```
system type             : ADM5120 Demo Board
processor               : 0
cpu model               : MIPS 4Kc V0.11
BogoMIPS                : 174.48
wait instruction        : yes
microsecond timers      : yes
tlb_entries             : 16
extra interrupt vector  : yes
hardware watchpoint     : yes
VCED exceptions         : not available
VCEI exceptions         : not available
```

Example 2 /proc/meminfo

```
        total:    used:    free:  shared: buffers:  cached:
Mem:  15085568  7172096  7913472        0    36864  6049792
Swap:        0        0        0
MemTotal:        14732 kB
MemFree:          7728 kB
MemShared:           0 kB
Buffers:            36 kB
Cached:           5908 kB
SwapCached:          0 kB
Active:            428 kB
Inactive:         5704 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:        14732 kB
LowFree:          7728 kB
SwapTotal:           0 kB
SwapFree:            0 kB
```

Also see *tweakers.net* [21] and *linux-mips.org* [17] for more information.

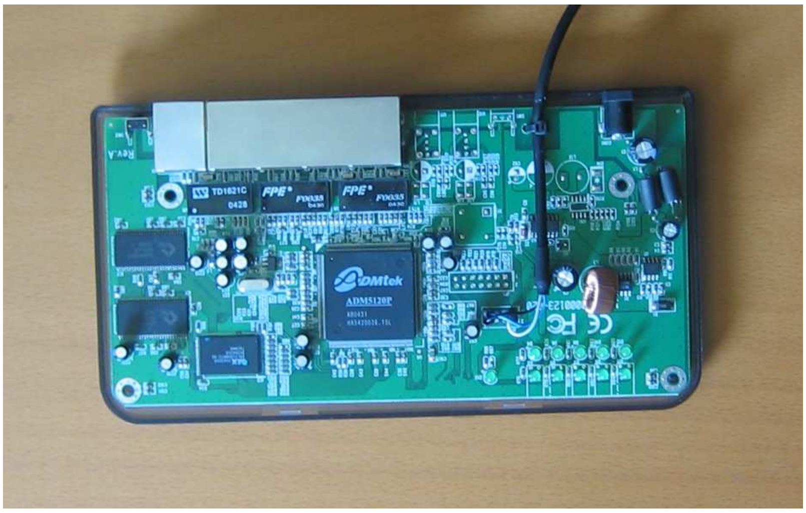


Figure 1: Sweex LB000021 from the inside.

2.2 Software

The LB000021 runs per default EdiMax's 2.4 Linux kernel and firmware. The latest source code and cross compiler toolchain should be available on *www.edimax.com* [7].

2.3 Soldering and connecting serial cable

If you want to do anything usefull on the LB000021, you need a serial console. This requires soldering a 3.3V serial cable to the LB000021 using the following scheme [5]:

JP2

2	4	6	8
+3v3	n/c	n/c	Gnd
RxD	n/c	n/c	TxD
1	3	5	7

2.3.1 Minicom settings

To use the serial console of the LB000021 with **minicom(1)** [13], change *Serial port setup > Serial Device* to point to the correct serial device on your system. Modify *Serial port setup > Bps/Par/Bits* to '115200 8N1' and *Serial port setup > Hardware Flow Control* to 'No'.

2.3.2 Tip settings

On FreeBSD [10], you can also use **tip(1)** to listen on the serial port:

Example 3 Using **tip(1)** on FreeBSD.

```
# tip -115200 sio0
```

3 Cross compiling for MIPS

In order to build our own kernel and user programs, we need to have a cross compiling toolchain. A cross compiler is a special kind of compiler which allows you to compile programs for another architecture, such as MIPS.

Depending on the operating system you use, you can choose several ways to install a cross compiler on your system.

3.1 EdiMax's toolchain

Perhaps the easiest way is to use EdiMax's [7] precompiled cross compiler toolchain. Just extract the ZIP archive, extract *6114-tool-chain.tgz* and move the extracted contents to */export*:

Example 4 Installing EdiMax's toolchain.

```
# cd /usr/src
# unzip Linux-SC.zip
# cd Linux-SC
# tar zxf 6114-tool-chain.tgz
# mv export /export
```

Note that you will have to modify your *PATH* environment variable to include /export/tools/bin and /export/tools/mipsel-linux-uclibc/bin. You can set *PATH* permanently in the system wide */etc/profile* file:

Example 5 Configuring PATH for EdiMax's toolchain.

```
# echo 'export PATH=$PATH:/export/tools/bin' >> /etc/profile
# echo 'export PATH=$PATH:/export/tools/mipsel-linux-uclibc/bin' \
> >> /etc/profile
```

To see if the cross compiler works, try it out with:

Example 6 Building MIPS programs with EdiMax's toolchain.

```
# mipsel-linux-gcc -o myprog myprog.c
```

3.2 Gentoo

With Gentoo Linux you can use crossdev [16] to automatically build and install a cross compiler for any architecture, operating system and C library. You can install crossdev [16] like any other package with **emerge(1)**:

Example 7 Installing crossdev with **emerge(1)**

```
# emerge crossdev
```

Now use crossdev [16] to automatically build and install a cross compiler for MIPS little endian, using uClibc [1]:

Example 8 Build a cross compiler for MIPS little endian, using uClibc

```
# crossdev --stable -t mipsel-unknown-linux-uclibc
```

UClibc [1] is a C library especially designed for embedded systems with limited resources. It is mostly compatible with GNU's C Library, and it's often used to statically link **busybox(1)** [23]. After crossdev [16] finishes, you can compile MIPS programs using **gcc(1)** as usual:

Example 9 Building MIPS programs with crossdev's **gcc(1)**

```
# mipsel-unknown-linux-uclibc-gcc -o myprog myprog.c
```

We can now use this cross compiler to build programs for the LB000021. See the Gentoo Embedded Project [11] for a complete manual about cross compiling under Gentoo.

3.3 Debian

Embedded Debian [18] offers cross compiling toolchains for several architectures, including MIPS little endian. The easiest way is to install a binary cross compiler package from emdebian.org. First add the emdebian repository to your */etc/apt/sources.list*:

Example 10 /etc/apt/sources.list entry for emdebian

```
deb http://www.emdebian.org/debian/ stable main
```

Then run **apt-get(8)** to update your local cache and install a cross compiler:

Example 11 Installing a MIPS little endian cross compiler with **apt-get(8)**

```
# apt-get update
# apt-get install gcc-4.1-mipsel-linux-gnu
```

When **apt-get(8)** has finished installing your cross compiler, run it with:

Example 12 Building MIPS programs with emdebian's **gcc(1)** compiler.

```
# mipsel-linux-gnu-gcc -o myprog myprog.c
```

3.4 Cross Linux From Scratch

If you are interested in learning the details of cross compiling, *Cross Linux From Scratch* [4] is certainly a good start. See the *Cross Linux From Scratch MIPS Book* [14] to learn how to hand compile a working MIPS cross compiler.

4 Building a mini Linux system

In theory, it is possible to run any random program you wish on the LB000021, as long as it compiles and runs under MIPS. The LB000021 has limited RAM memory and flash space, so it is important to keep the system as small as possible.

Personally, I think Gentoo [19] is perfect for embedded system, as it encourages freedom in customizing your system the way you want it to be. Tiny Gentoo [12] and the *Embedded Gentoo Handbook* [11] can be used as reference when building gentoo for embedded systems such as the LB000021.

4.1 Building the filesystem

4.1.1 Creating standard directories.

A normal Linux installation in a desktop or server environment needs standard directories and files such as */bin*, */lib* and */etc* to be functional. This is mostly also true for embedded Linux systems, so let's create a very basic filesystem layout which we will use for the LB000021:

Example 13 Creating standard directories.

```
# mkdir /usr/src/tinygentoo
# chdir /usr/src/tinygentoo
# mkdir dev etc lib proc root usr var tmp
# chmod 1777 tmp
# chmod 700 root
```

4.1.2 Adding basic system files.

Now we add *passwd*, *shadow*, *resolv.conf* and *nsswitch.conf* to */etc*. You can add more users and modify passwords if you wish:

Example 14 Filling the basic /etc files.

```
# cd /usr/src/tinygentoo
# echo 'root:x:0:0:root:/root:/bin/sh' > etc/passwd
# echo 'root:*:9797:0:::::' > etc/shadow
# echo 'root::0:root' > etc/group
# echo 'nameserver 192.168.1.1' > /etc/resolv.conf
# cat > etc/nsswitch.conf
passwd:   files
shadow:   files
group:    files
hosts:    files dns
networks: files dns
# chmod 600 etc/shadow
```

4.1.3 Setting up device files

To do anything usefull, we must give our Linux system device files:

Example 15 Creating device files.

```
# cd /usr/src/tinygentoo/dev
# mknod console c 5 1
# mknod led0 c 166 0
# mknod mtd b 31 0
# mknod null c 1 3
# mknod ptmx c 5 2
# mknod ptyp0 c 2 0
# mknod ptyp1 c 2 1
# mknod ptyp2 c 2 2
# mknod ptyp3 c 2 3
# mknod ptyp4 c 2 4
# mknod random c 1 8
# mknod tty c 5 0
# mknod ttyS0 c 4 64
# mknod ttyS1 c 4 65
# mknod ttyp0 c 3 0
# mknod ttyp1 c 3 1
# mknod ttyp2 c 3 2
# mknod ttyp3 c 3 3
# mknod urandom c 1 9
# mkdir pts
# chmod 777 null
# chmod 666 random ptmx
```

4.2 Programs

A Linux system without programs is quite useless, let's build some basic system utilities we can use to interact with the system.

4.2.1 Busybox

Having one statically linked program serving all system commands saves space, and **busybox(1)** [23] does just that. It allows you to configure which applets you want to be compiled and linked into the final executable:

Example 16 Downloading, configuring, compiling and installing Buxybox.

```
# cd /usr/src
# wget http://www.busybox.net/downloads/busybox-1.10.3.tar.gz
# tar xf busybox-1.10.3.tar.gz
# cd busybox-1.10.3
# make menuconfig
# make CROSS_COMPILE=mipsel-uclibc-
# make CONFIG_PREFIX=/usr/src/tinygentoo install
```

Make sure you choose *Busybox Settings > Build Options > Build Busybox as a static binary* so busybox will be statically linked with uClibc.

4.2.2 Dropbear SSH Daemon

A neat feature is to have a small SSH daemon on your LB000021, such as Dropbear [15]. Installation and compilation is pretty straight forward, but you will need to configure Dropbear in order to remotely login:

Example 17 Downloading, compiling and installing Dropbear.

```
# cd /usr/src
# wget http://matt.ucc.asn.au/dropbear/releases/dropbear-0.51.tar.gz
# tar zxf dropbear-0.51.tar.gz
# cd dropbear-0.51
# ./configure --disable-syslog --disable-utmp --disable-utmpx \
> --disable-wtmp --disable-wtmpx --disable-loginfunc \
> --disable-pututline --disable-pututxline --disable-pam \
> --disable-zlib --disable-lastlog \
> --host=mipsel-unknown-linux-uclibc LDFLAGS=-static
# make clean
# make CC=mipsel-uclibc-gcc
# mipsel-uclibc-strip dropbear
# cp dropbear /usr/src/tinygentoo/sbin/
```

As you see, Dropbear uses GNU Autoconf [9] which makes things a lot easier. To use Dropbear, we have to generate two keys. We cannot use the **dropbearkey(8)** program we just compiled, as it only runs on MIPS, so we recompile for our own architecture to run it:

Example 18 Configuring Dropbear with **dropbearkey(8)**.

```
# make clean
# make dropbearkey
# ./dropbearkey -t rsa -f /usr/src/tinygentoo/etc/rsa.key
# ./dropbearkey -t dss -f /usr/src/tinygentoo/etc/dss.key
```

4.3 Writing a /sbin/init

Now we have a basic Linux system, it is time to write a */sbin/init* so it can boot. It is possible to use Busybox's [23] **init(8)**, but as our system is so small, we will just use a shell script:

Example 19 Setting up a minimal **init(8)**.

```
# cat > sbin/init
#!/bin/sh
echo "Mounting filesystems..."
mount -t proc proc /proc
mount -t devpts devpts /dev/pts

echo "Bringing up network..."
ifconfig eth0 up
route add default gw 192.168.1.1 dev eth0

echo "Applying local settings..."
hostname -F /etc/hostname
/usr/sbin/rdate -s ntp.xs4all.nl

echo "Starting services..."
/sbin/dropbear -r /etc/rsa.key -d /etc/dss.key

while true ; do
    echo "(Re)launching shell..."
    /bin/sh --login
done
# chmod 755 sbin/init
# ln -s sbin/init linuxrc
# ln -s sbin/init init
```

Ofcourse you can write any other commands you wish to */sbin/init*, or add fancy terminal colors.

4.4 Building the ramdisk

In order to test and later on boot the mini Linux system, we should make a ramdisk. A ramdisk is just a compressed file containing a filesystem with a Linux system. During development it is comfortable to automate this process with a simple *Makefile*. Put *Makefile* in */usr/src* and run **make(1)** to generate ramdisks:

Example 20 Makefile for generating ramdisks

```
DIR=tinygentoo

all: ext2 cramfs tar

ext2: clean
	dd if=/dev/zero of=$(DIR).ext2 bs=1M count=5
	mke2fs -F $(DIR).ext2
	umount tmp.ext2 || exit 0
	rmdir tmp.ext2 || exit 0
	mkdir tmp.ext2
	mount -o loop $(DIR).ext2 tmp.ext2
	cp -Rp $(DIR)/* tmp.ext2
	umount tmp.ext2
	rmdir tmp.ext2
	bzip2 $(DIR).ext2
	ls -alh $(DIR).ext2.bz2

cramfs: clean
	mkcramfs $(DIR) -o $(DIR).cramfs
	bzip2 $(DIR).cramfs
	ls -alh $(DIR).cramfs.bz2

tar: clean
	tar zcf $(DIR).tar.gz $(DIR)
	ls -alh $(DIR).tar.gz

clean:
	rm -f $(DIR).ext2.bz2 $(DIR).bz2 $(DIR).cramfs
	rm -f $(DIR).cramfs.bz2 $(DIR).tar.gz
	rmdir $(DIR).ext2 $(DIR).cramfs || exit 0
```

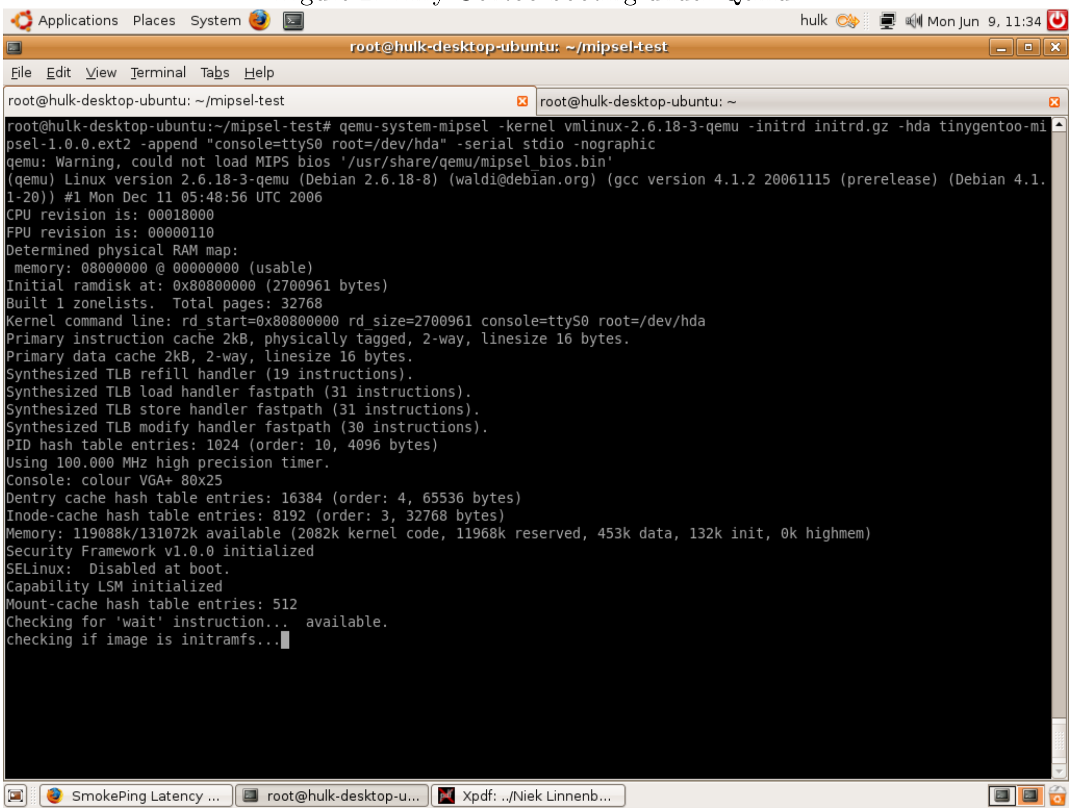
4.5 Testing in Qemu

To test the ramdisk, Qemu [2] is very handy. Just download the MIPS kernels from the Qemu [2] project page and use your ramdisk as *-hda*:

Example 21 Testing a ramdisk under Qemu

```
# wget http://bellard.org/qemu/mipsel-test-0.2.tar.gz
# tar zxf mipsel-test-0.2.tar.gz
# cd mipsel-test
# qemu-system-mipsel -kernel vmlinux-2.6.18-3-qemu -initrd initrd.gz \
> -hda /usr/src/ramdisk.bz2 -append "console=ttyS0 root=/dev/hda" \
> -serial stdio -nographic
```

Figure 2: Tiny Gentoo booting under Qemu.



5 Cooking a Linux kernel

To run your own Linux on the LB000021 you'll have to compile a kernel using a MIPS little endian toolchain. Using the EdiLinux 2.4 kernel sources, we can configure, build and run a suitable kernel for the LB000021. First extract all source code:

Example 22 Extract EdiLinux ZIP archive

```
# cd /usr/src
# unzip Linux-SC.zip
```

5.1 Configuration

In the *Linux-SC/EdiLinux* directory, there is a Linux 2.4.18 kernel included, which can be configured and compiled with GNU Make. Before you happily start to build a new fresh fat kernel for the LB000021, keep in mind that **the total size of the final image cannot exceed 1964K**. The EdiMax bootloader fails when attempting to upload an imager larger than 15872 sectors / 1964K, which is strange, as the RAM size is 16M. One possible reason could be, that the bootloader restricts images to the size of the onboard flash memory.

Before starting to configure your kernel, make sure you have a clean environment with the 'mrproper' GNU Make target. Then use the default configuration as a start. You can configure the kernel in textmode or graphical (X11) mode. Personally, I prefer using the ncurses interface (menuconfig):

Example 23 Clean and configure the kernel.

```
# make mrproper
# cp arch/mips/defconfig .config
# make oldconfig
# make menuconfig ARCH=mips
```

From this point, it's up to you to decide what functionality you want in the LB000021. For example, it's possible to include netfilter and IPtable modules to build a firewall. After the configuration, it is time to compile the kernel.

5.2 Compilation

Cross compiling the Linux kernel is easy. With the *CROSS_COMPILE* argument we can specify a prefix which the Makefile will use before all binutils and gcc commands. For example, if our cross compiler is *mipsel-linux-gcc*, we can compile a working *vmlinuz* using:

Example 24 Compiling the kernel.

```
# make CROSS_COMPILE=mipsel-linux- vmlinux
```

Depending on your machine, this could take a while. Once the compilation is complete, you should see a file named *vmlinuz*, which is a compressed Linux kernel suitable for the LB000021.

5.3 Running the kernel

5.3.1 Accessing the bootloader

In order to run our own kernel, we must gain access to the bootloader. Via the bootloader, you can choose two ways to run your own code on the LB000021. One way is to permanently flash a kernel image and reboot. A more friendly and lower risk method is to upload a *vmlinuz* to SDRAM and execute it directly, without touching flash.

To access the bootloader, listen on the serial console with your favorite program and power on the LB000021. Immediately press three times the spacebar when you turn on the router, and the bootloader should give you a prompt:

Example 25 LB000021's bootloader prompt.

```
ADM5120 Boot:



Linux Loader Menu
====================
(a) Download vmlinuz to flash ...
(b) Download vmlinuz to sdram (for debug) ...
(c) Exit

Please enter your key :
```

You will have to be quick with pressing the spacebar, otherwise the kernel on flash will boot. When developing, this process might get frustrating. With a small python script called *adm_tx.py* [3] we can fully automate downloading, running and even flashing linux on the LB000021.

To use *adm_tx.py* [3], you need the python pexpect module. If you run Gentoo [19], install *dev-python/pexpect* with **emerge(1)**. On Debian/Ubuntu, use **apt-get(8)** to install *python-pexpect*.

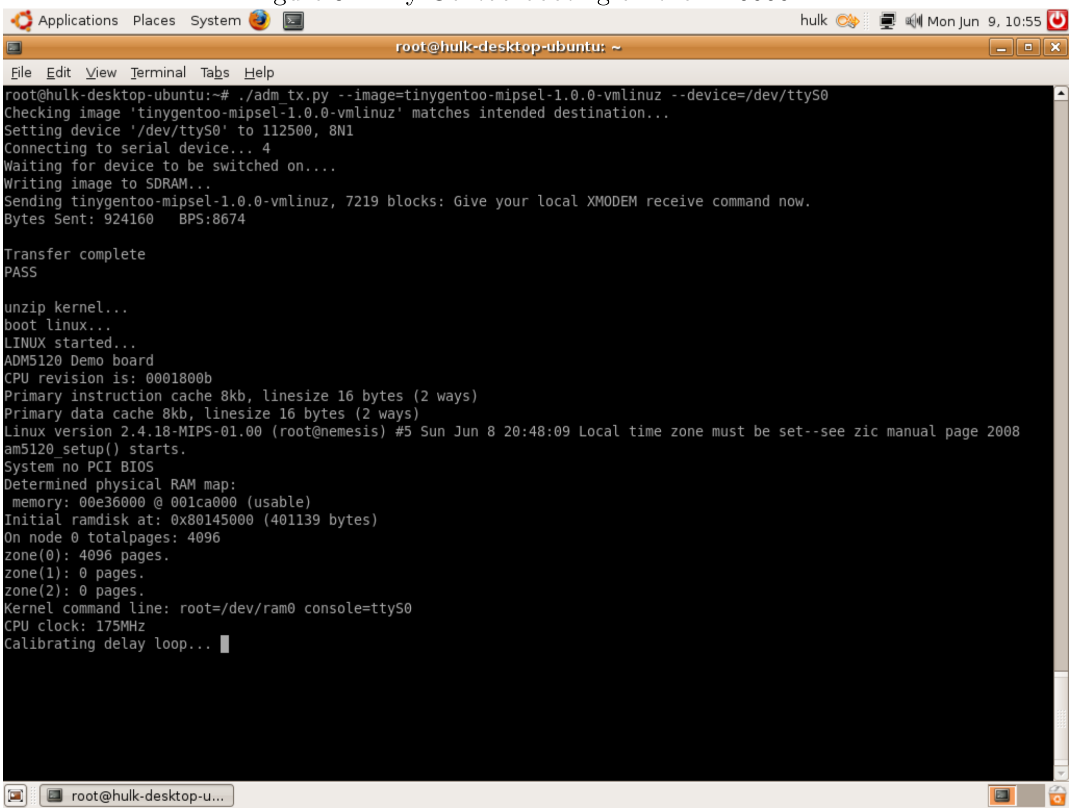
5.3.2 Downloading to SDRAM

Downloading the kernel to SDRAM is the safest way to try out self compiled kernels. Use *adm_tx.py* [3] to start the upload (takes long):

Example 26 Downloading Linux to SDRAM.

```
# ./adm_tx.py --device=/dev/ttyS0 --image=vmlinuz
```

Figure 3: Tiny Gentoo booting on the LB000021.



5.3.3 Permanent flashing

With *adm_tx.py* we can also permanently flash Linux on the LB000021. Note that this action is irreversable, so you should be very sure the kernel works when you attempt flashing:

Example 27 Flashing Linux on the LB000021.

```
# ./adm_tx.py --device=/dev/ttyS0 --burn --image=vmlinuz
```

6 Other tweaks

6.1 Writing MIPS assembly programs

Running your own kernel with cross compiling programs is one thing, but to really understand how the LB000021 works, learning MIPS assembler is essential. As with all programming languages, try to write *"Hello, World"* first:

Example 28 "Hello, World!" in MIPS assembly.

```
#
# Hello World in MIPS assembly
#

.data
        msg:
.ascii  "Hello, World!\n\000"

.text
.align  2
.globl  __start
.type   __start,@function
.ent    __start

__start:

li      $2, 4004        # write(2)
li      $4, 1           # stdout
la      $5, msg         # address of hello world
li      $6 ,14          # Length of hello world
syscall

li      $2, 4001        # exit(2)
li      $4, 0           # zero exit status
        syscall

.end    __start
```

MIPS has a total of 32 registers, and like x86 it has instructions for arithmetics, comparisions, jumps and data operations. Also see *MIPS Assembly Language Programming* by Daniel J. Ellard [8], Wikipedia [25] or WikiBooks [24] for more details about the MIPS architecture. To test the program we can use an emulator such as Qemu [2]:

Example 29 Assembling, linking and running "Hello, World!".

```
# mipsel-linux-as -o helloworld.o helloworld.s
# mipsel-linux-ld -o helloworld helloworld.o
# qemu-mipsel ./helloworld
```

6.2 Controlling leds

A nifty feature is to control the onboard leds of the LB000021. Using */dev/led0* you can control the led by using **echo(1)** on it:

Example 30 Controlling the leds

```
# echo 'LED ON' > /dev/led0
# echo 'LED OFF' > /dev/led0
```

6.3 Soldering USB connectors

See Jeroen Domburg's LB000021 project page [6] for a complete HOWTO on soldering an USB connector to the LB000021.

6.4 OpenWRT

The LB000021 (i.e. ADM5120) is fully supported by OpenWRT [22]. See *openwrt.org* [22] for more information.

6.5 NetBSD

NetBSD can also run on the ADM5120 chipset. Take a look at the *sourceforge* [20] page for details.

References

[1] Erik Andersen, *uclibc home page*, http://www.uclibc.org, June 2008.

[2] Fabrice Bellard, *Qemu emulator*, http://bellard.org/qemu/, June 2008.

[3] biffer@yahoo.co.uk, *adm_tx.py*, http://www.biffer.talktalk.net/sweex/clock/adm_tx.py, June 2008.

[4] cross lfs.org, *Cross linux from scratch*, http://trac.cross-lfs.org/, June 2008.

[5] dbzoo.com, *Sweex router hacking*, http://www.dbzoo.com/wiki/sweex/sweex, February 2008.

[6] Jeroen Domburg, *Jeroen domburg's lb000021 project page*, http://meuk.spritesserver.nl/projects/lb000021/, June 2008.

[7] EdiMax, *Edimax home page*, http://www.edimax.com/, June 2008.

[8] Daniel J. Ellard, *Mips assembly language programming*, June 2008.

[9] Free Software Foundation, *Gnu autoconf*, http://www.gnu.org/software/autoconf/, June 2008.

[10] FreeBSD Foundation, *Freebsd project*, http://www.freebsd.org, June 2008.

[11] Embedded Gentoo, *Gentoo embedded handbook*, http://www.gentoo.org/proj/en/base/embedded/handbook/?pa June 2008.

[12] gentoo wiki.com, *Tinygentoo*, http://gentoo-wiki.com/TinyGentoo, May 2008.

[13] Martin Godish, *Minicom freshmeat.org project page*, http://freshmeat.net/projects/minicom/, June 2008.

[14] Ryan Oliver Jim Gifford, *Cross linux from scratch version 1.0.0-mips*, http://cross-lfs.org/files/BOOK/1.0.0/CLFS-1.0.0-mips.pdf, 2006.

[15] Matt Johnston, *Dropbear ssh daemon*, http://matt.ucc.asn.au/dropbear/dropbear.html, June 2008.

[16] Joshua Kinard, *Gentoo crossdev*, http://www.gentoo.org/, June 2008.

[17] Linux-mips.org, *Linux mips wiki: Adm5120*, http://www.linux-mips.org/wiki/ADMtek, June 2008.

[18] EmDebian Project, *Emdebian project*, http://www.emdebian.org, June 2008.

[19] Gentoo Project, *Gentoo project page*, www.gentoo.org, June 2008.

[20] NetBSD ADM5120 Project, *Netbsd adm5120 project*, http://linux-adm5120.sourceforge.net/netbsd/, June 2008.

[21] Sprite_tm, *Routerhacking*, http://gathering.tweakers.net/forum/list_messages/984121/0, June 2008.

[22] OpenWRT Team, *Openwrt project*, http://www.openwrt.org, June 2008.

[23] Denys Vlasenko, *Busybox*, http://www.busybox.net, June 2008.

[24] Wikibooks.org, *Mips assembly*, http://en.wikibooks.org/wiki/Programming:MIPS_assembly, June 2008.

[25] Wikipedia.org, *Mips architecture*, http://en.wikipedia.org/wiki/MIPS_architecture, June 2008.