Installing VSIB/VSIC Test Software

Ari Mujunen, amn@kurp.hut.fi
Metsähovi Radio Observatory

19-Sep-2003

Abstract

This document describes the steps required to install the Debian/GNU/Linux
3.0 operating system to support using VSIB data acquisition board test programs.

Contents

1   Introduction

Please find below instructions how to use the two CD-R disks provided to set up
Linux on your nVidia PC.

The first CD-R disk contains a single-disk install of a "mini-Linux" system of De-
bian/GNU/Linux 3.0 ("woody").

The second CD-R disk is a snapshot of our development environment computer
"remus.kurp.hut.fi".

The idea is to install a "plain" Debian Linux 3.0 "woody" on the /dev/hda1 parti-
tion, but at the same time create an alternative boot partition on /dev/hda5. I.e:

/dev/hda1 – bootable mini system, used to re-install /dev/hda5 below

/dev/hda2 – swap partition, used by both bootable systems

(/dev/hda3 – extended partition, holds partitions 5 and above)

/dev/hda5 – the real VSIB development Linux system (expanded from 'remus.tgz'
on CD-R #2)

This dual-boot environment makes it easy to quickly install a "clone copy" of a
given system from a single tar file, like the supplied 'remus.tgz'.

2   Installing "Mini-Linux"

The "mini-Linux" system is installed using standard Debian installation tools. De-
bian has a _very_ extensive installation manual and it is available on the Debian
3.0 Mini-CD sent to you at:

```
mount /cdrom acroread
/cdrom/install/doc/install.en.pdf &
```

(You can also stick the CD-R into a Windows machine and locate the folders 'in-
stall' and 'doc' and double-click on 'install.en.pdf' document.)

The document has lots of background information and you may want to print
all 112 pages, but for this install you'll probably need Acroread pages 53–94 only,
covering Chapters 5–9.

If you haven't used the 'cfdisk' program before to create disk partitions you may
need its manual page which is available in text form at:

```
/cdrom/install/doc/cfdisk.txt
```

'cfdisk' is menu-driven, though, so the only thing to beware is that you can move
between menu selections with left/right arrows only, not up/down. Up/down
is reserved for moving up and down in the list of partitions and free space, and
'Enter' selects a menu choice.

First some hardware checks:

1. You need to ensure that your primary master disk, the system disk /dev/hda is connected. This can be a smaller and slower disk than a 120GB IBM/Maxtor, but such a large disk is ok, of course, too.

2. You need to temporarily connect a CD-ROM to primary slave or secondary master. (Use whichever is cable-wise more convenient.)

    Recently I've tried the so-called "Cable Select" jumpering in IDE drives. Against my expectations, this jumpering scheme seems to work great! I.e. you can jumper all your disks and CD-ROM drives with the setting "Cable Select" and the drives will automatically be masters when at the far end of an IDE cable and slaves when they are connected in the middle ribbon cable connector. All my disks in my Asus nVidia are jumpered with "Cable Select".

3. Set the boot order in Asus/MSI//nVidia BIOS settings to be 1) floppy, 2) CD-ROM, 3) IDE.

4. Boot the "mini Debian" CD-R. Press 'Enter' to the first 'boot:' prompt.

5. Start answering the installation questions. (I'll switch now to the chapter and section numbers of the installation PDF document.)

```
5. Booting the Installation System
5.1. Boot Parameter Arguments - none required
5.2. Booting from a CD-ROM - use this
5.3. Booting from Floppies
5.4. Booting From a Hard Disk
5.5. Booting from TFTP
5.6. Troubleshooting the Install Process
5.7. Introduction to 'dbootstrap'
5.8. "Choose The Language" - 'en', probably :-)
5.9. "Release Notes"
5.10. "Debian GNU/Linux Installation Main Menu"
5.11. "Configure the Keyboard" - qwerty/uk, probably
5.12. Last Chance!
6. Partitioning for Debian
6.1. Deciding on Debian Partitions and Sizes
6.2. The Directory Tree
6.3. PC Disk Limitations
6.4. Recommended Partitioning Scheme
6.5. Device Names in Linux
6.6. Debian Partitioning Programs
- use cfdisk (the default) to create:
/dev/hda1 primary 2000MB type Linux (83, the default)
/dev/hda2 primary 500MB type Linux swap (82)
/dev/hda5 logical 2000MB type Linux (83, the default)
/dev/hda6 logical 2000MB type Linux (83, the default)
6.7. "Initialize and Activate a Swap Partition"
- /dev/hda2, no need to check bad blocks
6.8. "Initialize a Linux Partition"
- /dev/hda1, no need to check bad blocks
- select in "Debian GNU/Linux Installa-
tion Main Menu" again the choice "Initialize a Linux Parti-
tion" /dev/hda5, mount as /s5 - re-
peat for /dev/hda6, mount as /s6
6.9. "Mount a Previously-Initialized Partition" - skipped
6.10. Mounting Partitions Not Supported by 'dbootstrap'
7. Installing the Kernel and Base Operating System
7.1. "Install Kernel and Driver Modules" - from CD-
ROM (the default)
7.2. NFS
7.3. Network
7.4. NFS Root
7.5. "Configure PCMCIA Support" - skipped
```

You should now have a bootable Debian 3.0 system without any frills, X Window system etc. This system is used to extract the full development system snapshot from the second CD-R.

3 Installing a Snapshot of "Full Linux"

The starting point is that your mini-Linux is up and networking works (if you plan to use network) and it has the following partitions:

/dev/hda1 - mini-Linux '/'

/dev/hda2 - swap

/dev/hda5 - full Linux '/s5'

You still need the temporary CD-ROM drive and the mini-Linux has probably set up the /dev/cdrom link to point to the correct /dev/hd{bcd} so that mounting the 2nd "snapshot" CD-R disk is easy (as 'root' user) with:

```
mount /cdrom
```

Now you can simply change to the '/s5' directory (the one which will become '/' of the full Linux) and expand the snapshot of the full system with tar:

```
cd /s5
tar xvzpf /cdrom/remus.tgz
```

4

The names of files will be shown while expanding. When done, you need to change computer name, network settings and similar in the '/s5/...' configuration files.

The '/s5/etc/modules' file lists only the following loadable device driver:

```
---
af_packet
---
```

If you network board is not a 3Com EtherLink III (3c59x, 3c90x) you may need to copy its name from '/etc/modules'. (It could be that your mini-Linux '/etc/modules' doesn't list any network board driver if the kernel recognized the board because it had the driver compiled in. The full Linux will need the name of network board driver because all network drivers were compiled as loadable modules.)

The following is suggested as a good starting point. Edit with 'nano /s5/etc/modules' the file to list the following loadable device driver:

```
---
af_packet
lp
serial
3c59x
---
```

Changing to use your mini-Linux user names and passwords with shadow passwords on:

```
cd /etc
cp -a passwd shadow group gshadow /s5/etc
```

Changing to use your mini-Linux network settings:

```
(still at 'cd /etc')
cp -a /etc/network/interfaces /s5/etc/network/interfaces
cp -a resolv.conf hostname hosts networks networks /s5/etc
```

You will probably want to comment out servering the '/i1' RAID data directory using NFS, at least initially:

```
nano -w /s5/etc/exports
```

(Put a '#' mark in the beginning of the single line.)

You will want to use the ssh host keys of mini-Linux for your computer (and not to use my 'remus' computer keys) so you should replace the following:

```
cd /etc/ssh
cp -av ssh_host_* /s5/etc/ssh
```

It should copy six files, '/etc/ssh/ssh_host{,rsa,dsa}_key{,.pub}'. You can easily re-create them when you have rebooted to full Linux (for instance if you have changed your computer name) by deleting the old files 'rm /etc/ssh/ssh_host_*' and:

```
dpkg-reconfigure ssh
```

5

To change the keyboard layout of full Linux to match the setting you chose for mini-Linux, please copy:

```
cp -a /etc/console/boottime.kmap.gz /s5/etc/console
```

Alternatively, when you have booted to full Linux, you can interactively select the keymap from a set of menus with:

```
dpkg-reconfigure console-data
```

(The "dash" character '-' in the above is the key immediately to left of the right side shift key in the default Finnish keyboard.)

So, for the final trick is to make both the mini-Linux and full Linux to appear in both LILO boot menus, allowing booting either of the two. Your mini-Linux '/etc/fstab' is probably very similar to the following:

```
---/etc/fstab---
remus:~> cat /etc/fstab
# /etc/fstab: static file system information.
# # <file system> <mount point> <type> <options> <dump> <pass>
/dev/hda1 / ext2 defaults,errors=remount-ro 0 1
/dev/hda2 none swap sw   0 0
proc  /proc proc defaults   0 0
/dev/fd0 /floppy auto defaults,user,noauto  0 0
/dev/cdrom /cdrom iso9660 defaults,ro,user,noauto  0 0
# Extra system
/dev/hda5 /s5 ext2 defaults   0 2
---
```

You will want a similar '/etc/fstab' on the full Linux, too, so the best option is to copy yours and edit it:

```
cp /etc/fstab /s5/etc/fstab
nano -w /s5/etc/fstab
```

You need to swap the partitions 'hda1' and 'hda5' so that these lines:

```
< /dev/hda1 / ext2 defaults,errors=remount-ro 0 1
< /dev/hda5 /s5 ext2 defaults 0 2
---
> /dev/hda5 / ext2 defaults,errors=remount-ro 0 1
> /dev/hda1 /s1 ext2 defaults 0 2
```

('nano' editor may auto-wrap these long lines. Simply do the edits on a given line, and finally backspace at the beginning of the wrapped continuation line to get it back to the end of previous line.) So, now when the full system boots, it will mount /dev/hda5 as its '/' file system. To get access to mini-Linux '/' file system files, you will have to create the empty directory for mounting '/s1' under full Linux:

```
mkdir /s5/s1
```

(i.e. when /s5 boots as /, it will have a directory '/s1' which will be used to mount the mini-Linux root/system partition /dev/hda1.)

The /s5/etc/fstab doesn't yet list anything "fixed" about the RAID setup. Also, at this stage, you can rename the following temporarily away:

```
mv /s5/etc/raidtab /s5/etc/raidtab.not-yet
```

This renaming is not absolutely necessary, since the raidtools2 exit gracefully with error messages if RAID0 disks/partitions are not available.

The final step in getting two options in LILO boot menu is to edit '/etc/lilo.conf' file and its counterpart in the full system, '/s5/etc/lilo.conf'. All edits happen at the end of the existing files. The first is to reserve memory for the big physical buffer:

```
---
# Kernel command line options that apply to all installed im-
ages go
# here. See: The 'boot-prompt-HOWO' and 'kernel-
parameters.txt' in
# the Linux kernel 'Documentation' directory.
# # append=""
# Take about 128MB; 90000*1000+2*128kB=22036 4kB pages => 23000
# 144000000 for 18MHz 32 track 2sec:
append="bigphysarea=36000"
---
```

Please ensure that this number is "36000" in both files. This sets aside 36000 4kB pages of real physical memory which can be used by the 'vsib.o' loadable VSIB board device driver as "bigbuf", a large circular buffer for DMAing VSI data directly to PC memory.

The second is to add the 'image=' section for the "other" Linux system in both '/etc/lilo.conf' and '/s5/etc/lilo.conf'. After the fragment:

```
---
# Boot up Linux by default.
# default=Linux
image=/vmlinuz
  label=Linux
  read-only
## initrd=/initrd.img
# restricted
# alias=1
---
```

add another:

```
---
image=/s5/vmlinuz
  root=/dev/hda5
  label=S5Linux
  read-only
---
```

And for '/s5/etc/lilo.conf' add:

```
---
image=/s1/vmlinuz
  root=/dev/hda1
  label=S1Linux
  read-only
---
```

Additionally, in '/s5/etc/lilo.conf' change the line (around line number 30):

```
---
root=/dev/hda1
---
```

to read:

```
---
root=/dev/hda5
---
```

The 'root=' directive works differently depending on its location in /etc/lilo.conf. The first one in the default file (around line 30) appears before any 'image=' bootable system directive. It thus becomes the global default for all subsequent 'image=' sections. So you can have say "Linux", "LinuxOLD", "LinuxSpecial" different bootable kernel images, all sharing the same root file system.

If you put 'root=' *after* an 'image=' directive, the setting only applies when booting that system.

Now update the LILO boot tables for mini-Linux and /dev/hda1:

```
lilo
```

and when you have booted the new full Linux, say 'lilo' again to update the tables to match then active '/etc/lilo.conf' of the new system. The settings of the active '/etc/lilo.conf' when the command 'lilo' is invoked become the new boot settings for the whole hard disk '/dev/hda'. Thus, if you say 'lilo' while in full Linux, the system booted by default becomes the full Linux system (and correspondingly if you say 'lilo' while in mini-Linux).

As both "lilo.conf" files have "their own" system listed in the first 'image=' fragment in the file, that system is booted by default (that is, when no Shift key is pressed while the "LILO" prompt is present on-screen to get the boot menu). The following could be used to name the system you want to boot by default:

```
---
# Boot up Linux by default.
# default=Linux
---
```

As this is commented out, both systems will boot up to the same Linux from which the 'lilo' updating was invoked, i.e. the system mentioned first in the "lilo.conf" file.

To get the "other" Linux you need to press down any of Shift/Control/Alt when you see the 'LILO' prompt during booting. This will present a red arrow-key menu of the available systems and lets you choose 'S5Linux' (or 'S1Linux') for booting.

Now it is time to test the dual-boot:

```
shutdown -r now
```

(Alternatively, you can just simply press ctrl-alt-del.)

When the 'LILO' prompt appears, press Shift to see the LILO boot menu. Select with arrow keys 'S5Linux' and press enter. You should get a system with kernel 2.4.19-ac4. 'cat /proc/bigphysarea' should print something in the style of:

```
---
remus:~# cat /proc/bigphysarea
Big physical area, size 144000 kB free list: used list:
number of blocks: 3 2
size of largest block: 3232 kB 140628 kB
total: 3240 kB 140760 kB
---
```

Now you can finalize the configuration by setting up NTP and mail server.

NTP changes the files '/etc/default/ntp-servers' and '/etc/ntp.conf' with menu-based:

```
dpkg-reconfigure ntp-simple
```

Select a good NTP time server and let the system overwrite '/etc/ntp.conf' when it asks for it. Next reboot will "step" the CMOS time to the value reported by the NTP server and 'ntpd' will keep the time in synch with the server. Timed starts of VSIB software require a NTP server with approximately +/-200msec accuracy compared to GPS time.

Mail handler uses the following files:

```
/etc/exim/exim.conf
/etc/mailname
```

and can be reconfigured with eximconfig:

```
---
remus:~# eximconfig
You already have an exim configuration. Continuing with eximcon-
fig will overwrite it. It will not keep any local modifica-
tions you have made. If that is not your inten-
tion, you should break out now. If you do con-
tinue, then your existing file will be re-
named with .O on the end. [---Press return---]
...
(4) Local delivery only: You are not on a network. Mail for lo-
cal users is delivered.
...
Select a number from 1 to 5, from the list above. En-
ter value (default='1', 'x' to restart): 4
---
```

I think you will probably want no mail server but still need local (system) mails to be delivered, probably to user 'pb'.

The following config files:

```
/etc/emacs21/site-start.d/00debian-vars.elc
/etc/emacs21/site-start.d/42hyperlatex.elc
/etc/xemacs21/site-start.d/00debian-vars.elc
/etc/xemacs21/site-start.d/42hyperlatex.elc
```

do contain the host name 'remus' in them but I don't know how to easily update these ('dpkg-reconfigure emacs21' maybe?) and I don't think having 'remus' there is very dangerous.

Now the full Linux is running; remember to update LILO

```
lilo
```

so that it will default to booting to full and present the option 'S1Linux' in LILO boot menu for returning to mini-Linux for reloading the system.

4 Installing VSIB/VSIC Test Software

The final step in software installation is to expand the 'proj.tgz' snapshot of my '/home/amn/proj' subdirectory structure to your home directory. I would do this when logged in as a normal Linux user (not root) such as 'pb'. (I'm presuming that you created yourself a normal user account in mini-Linux. 'pb' will be used in the remaining text as an example of the Linux user id used to perform VSIB/VSIC tests.)

The '/home' directory of the /dev/hda5 full Linux system is still empty, create a home directory for the VSIB tester user:

```
mkdir /home/pb
chown pb.pb /home/pb
```

and then you can log in (probably in Alt-F2 another virtual console) as 'pb'. Mount[1] the second snapshot CD-R again

```
mount /cdrom
```

and expand the 'proj.tgz' archive:

```
tar xvzf /cdrom/proj.tgz
```

You should get the main development directory 'proj/vsib' where all the current development software is. The only exception is the '/home/pb/proj/vsib/vsib' special device file which can only be created as root and will be skipped by tar when run as 'pb'. The 'root' user has to do:

```
cd /home/pb/proj/vsib
rm vsib
mknod vsib c 254 0
chown root.pb vsib
chmod ug=rw vsib
```

to get:

```
---
remus:~# ls -l /home/pb/proj/vsib/vsib
crw-rw-r-- 1 root pb 254, 0 Nov 2 2001 /home/pb/proj/vsib/vsib
---
```

You can browse the files in the '/home/pb/proj/vsib' directory, the main files are:

vsib.c - Linux device driver

wr.c - write/read VSIB from command line

Makefile - this compiles the others

d32.c - dumps standard input in 32-bit bits

---

[1]The mount can fail if you don't have the CD-ROM drive as primary slave, '/dev/hdb'. In this case replace the symbolic link '/dev/cdrom' by:

```
cd /dev
rm cdrom
ln -s hdx cdrom
```

where 'hdx' is the name of your CD-ROM device, {hdb,hdc,hdd}.

5 Setting up data RAID0 Disk Array

Now it is time to prepare the RAID setup, for now I'd recommend three data disks—RAID can be reformatted relatively easily for two or four disks, if needed.

You probably have to 'shutdown -h now' to swap away the temporary CD-ROM and to connect hdb/hdc/hdd disks. Boot again to 'Linux' (if you remembered to run 'lilo' under the full system; or 'S5Linux' if you forgot :-)

5.1 Creating Partitions on 2–4 Disks

Prepare three equal-sized partitions on hdb/hdc/hdd disks with 'cfdisk':

```
cfdisk /dev/hdb
cfdisk /dev/hdc
cfdisk /dev/hdd
```

For each disk, you need to invoke:

```
[New]
 [Primary]
  largest size available
  [Beginning]
[Type]
 fd (enter the type code as hexadecimal 'fd')
[Write]
 yes
[Quit]
```

to create for every disk one '/dev/hdx1' partition, equal in size on each disk and as large as possible. The type code 'fd' means "Linux auto-detecting raid partition".

Please check the partition sizes with

```
fdisk -l
```

For instance, my current hdb disk lists as follows:

```
---
remus:~# fdisk -l /dev/hdb
Disk /dev/hdb: 255 heads, 63 sectors, 15017 cylin-
ders Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/hdb1 1 14945 120045681 fd Linux raid autodetect
/dev/hdb4 14946 15017 578340 83 Linux
---
```

I created the partition 1 with type code 'fd' to be as large as could fit on the smallest of my disks (hdc is a Maxtor and not IBM...) and created a "filler" /dev/hda4. (This can probably later be used by higher-level software to store plain ASCII "book-keeping" information files.)

If you have three IBMs you can just create three "as large as possible" partitions. Use the "[Type]" menu item to change them to 'fd', RAID autodetect.

If you end up with different-sized partitions, chances are that the motherboard BIOS is doing the LBA translation in a different way for /dev/hd{a,b} (primary)

than for /dev/hd{c,d} (secondary). In this case, instead of accepting the default of "largest size available", please specify a size like "120000M" or "115000M" manually; you should get equal-sized partitions in this case.

When you now have the partitions, please ensure you have the mount point (empty directory) '/i1' for the future mount of raid0 large disk. (This is done only once and is the same regardless if you use 2 or 3 or 4 disks.)

```
mkdir /i1
```

5.2   Edit/Check '/etc/raidtab'

When the partitions have been created you can init the RAID combination disk which becomes '/dev/md0'. Take the /etc/raidtab into use:

```
mv /etc/raidtab.not-yet /etc/raidtab
```

(My standard raidtab should be just fine for three /dev/hd{bcd}1 partitions. You can comment out the third disk or add a fourth with an editor like emacs.)

5.3   Initializing, Starting and Formatting '/dev/md0'

Run

```
mkraid --force /dev/md0
```

to combine the partitions listed in /etc/raidtab into one large '/dev/md0'. (It will ask you again for another longer –force–force–force -style confirmation argument because this wipes out all data in partitions listed in /etc/raidtab for that /dev/md0.)

After RAID creation you can do manually what the boot sequence does automatically in the future:

```
raidstart -a
```

i.e. start all RAID setups in /etc/raidtab. This makes '/dev/md0' available. (It will be automatically made available in subsequent boots to the full Linux system with the '/etc/raidtab' file; so one alternative at this point is to just reboot.)

Format the large RAID disk /dev/md0 with

```
mke2fs -m0 /dev/md0
```

('-m0' leaves 0% for the root user, i.e. you as a normal user like 'pb' can consume all of the disk for test data.)

Ensure that a mount point for the data area has been created:

```
mkdir /i1
```

All right, now all "do once" setup steps have been completed. The steps in section 5.1 "Partitioning" need only to be performed once, when the disk is taken into use for the very first time.

When you want to change the number of disks, you need to un-do 'umount /i1', 'raidstop -a', and then do again all the steps in 5.2 "Editing /etc/raidtab" and 5.3 "Initializing raid0".

5.4 "Per-boot" usage steps

What follows are "after every boot" usage steps. They are currently packaged into one "root-invokable" script.

Get back (as root) to the root home directory '/root':

```
cd
cat ./vsib
```

You will see the "typical" actions needed as root to load the device driver and to mount the RAID disks.

```
---
#!/bin/sh
insmod /home/amn/proj/vsib/vsib.o bigbufsize=144000000
# echo "" > /proc/sys/vm/bdflush
mount /dev/md0 /i1
---
```

'insmod' loads the VSIB board device driver allocating 144000000 bytes for the ring DMA buffer. You should use a number evenly divisible by 1000 (the check is currently missing in 'vsib.c'). That 144MB is enough for 2 seconds of data at 18MHz 32 track mode.

'# echo "" > /proc/sys/vm/bdflush' is a placeholder for the probable disk writing tuning arguments—2.4.19 seems to like to write in _large_ chunks to disk with long intervals between, but it still seems to work at 512Mbit/s. At 576Mbit/s (with parity bits) I've experimented with extra settings, like the following:

```
echo "100 5000 640 2560 150 30000 5000 1884 2" > /proc/sys/vm/bdflush
```

The final 'mount /dev/md0 /i1' just takes the data area into use. You can easily run this after logging in as root in virtual console Alt-F1 by:

```
./vsib
```

It loads the driver and mounts the data disk. If you want the system to do this by default in every boot, the simplest place in Debian (although probably going away gradually) is to put a two-line script into the directory '/etc/rc.boot'. Create the following file:

```
----/etc/rc.boot/vsib---
#!/bin/sh
/root/vsib
---
```

and make it executable with

```
chmod a+x /etc/rc.boot/vsib
```

In this way the script proper is still in 'root' user's home directory where it is easily edited if we need to tune buffer sizes etc.

5.5 Linux Disk Tuning

The default values for disk buffer management in Linux kernel "2.4.19-ac4-bigphysarea" are the following:

```
remus:~/proj/vsib> cat /proc/sys/vm/bdflush
30 500 0 0 500 3000 60 20 0
```

The numbers mean:

1. nfract, default 30% (range 0–100%) - percentage of buffer cache dirty to activate bdflush

    keep relatively low for VSIB

2. ndirty, default 500 blocks (range 1–50000) - maximum number of dirty blocks to write out per wake-cycle

    increase significantly for VSIB

3. dummy2 (old, withdrawn, use 0)

4. dummy3 (old, withdrawn, use 0)

5. interval, default 500 jiffies (0.5s) (range 100–1000000 (1–10000s)) - jiffies delay between kupdate flushes

    keep relatively small for VSIB

6. age_buffer, default 3000 jiffies (30s) (range 0–1000000 (0–10000s)) - time for normal buffer to age before we flush it

    perhaps even zero for VSIB (new dirty buffers immediately to disk)

7. nfract_sync, default 60% (range 0–100%) - percentage of buffer cache dirty to activate bdflush synchronously

8. nfract_stop_bdflush, default 20% (range 0–100%) - percetange of buffer cache dirty to stop bdflush

9. dummy5 (unused, use 0)

So, apparently every 0.5s (parameter 5, "interval") 'kupdate' activates to check "old" (>30s, parameter 6, "age_buffer") dirty buffers and write them out onto disk. 'kupdate' seems to be targeted to take care of "old, scattered" changed disk blocks whereas 'bdflush' below takes care of "mass writes".

On the other hand, the 'bdflush' is activated based on consumption of buffer cache memory. When over 30% (1, nfract) is dirty, wake up 'bdflush' to write 500 blocks (2, ndirty) or if more than 60% (7, nfract_sync) were dirty, write lots of blocks forcing calling process to wait. When again below 20%, stop waking up 'bdflush' ('kupdate' writes out aging buffers).

The first item to change would probably be 2, "ndirty" with a significant increase from 500 to, say, 5000 blocks, perhaps even more.

```
echo "30 5000 0 0 500 3000 60 20 0" > /proc/sys/vm/bdflush
```

5.6 Test Directory

I've been using a test directory called 't', create it as root and "donate" it to 'pb':

```
mkdir /i1/t
chown pb.pb /i1/t
```

So now you can do all tests nicely as 'pb' and need not worry about accidentally messing up something as 'root' in VSIB tests.

Log in in virtual console Alt-F2 as 'pb' and:

```
cd proj/vsib
```

Finally, you can start doing those './wr 90000 1000 1 /i1/t/tref 1000 <vsib' test command lines.

The command line means:

```
---
remus:~/proj/vsib>
./wr ./wr: needs at least blocksize total-
blocks ndirs { path%d blks }
---
```

read and write 90000 bytes at a time (one Mark4 frame /w parity included, 32 tracks)

1000 total # of blocks

using 1 directory/file name template

– the name being '/i1/t/tref'
– and this name will receive 1000 blocks (of 90000 bytes each)

You can put a '%d' printf()-expression in name template and that will be replaced by an incrementing number:

```
./wr 90000 10000 1 /i1/t/tref%04d 1000 <vsib
```

will write files:

```
/i1/t/tref0000
/i1/t/tref0001
/i1/t/tref0002
...
/i1/t/tref0009
```

with 1000 (90000 byte) blocks each until the total of 10000 blocks have been consumed.

For calculating how many blocks to write it is easy to remember that highest-speed tape data is 9000000 bits/s / 22500 bits/frame == 400 frames/s. Thus 32-track (32*9==) 288Mbits/s data is 400 blocks per second (block size being 90000 bytes). A typical recommended file size for this case would be 10 seconds, 4000 blocks, resulting in about 360MB-sized files.

6 Performing VSIB/VSIC Tests

6.1 VSIC Modes

When VSIC JP2 "TEST/MODE" pins 1,3,5,7 are grounded, these four bits select
the operation mode of the VSIC board. The following table lists the selections
currently available.

```
JP2 "TEST/MODE" pin #    7531  -- 1==connect to GND
-----------------------------
alt_VLBA_sampler:       "0000";
VLBA_sampler:           "0001";
alt_MK4_sampler:        "0010"; -- not implemented yet
MK4_sampler:            "0011"; -- not implemented yet
(vacant)                "0100";
(vacant)                "0101";
decrement_testmode:     "0110"; -- decrementing 32-bit word
testmode:               "0111"; -- VSI-H test pattern
MKIV_IOB:               "1000"; -- Mk4 fmt headstack #1 (50-pin)
MKIV_IOB_no_parity:     "1001";
MKIV_WDB:               "1010"; -- Mk4 fmt headstack #2 (40-pin)
MKIV_WDB_no_parity:     "1011";
VLBA_recorder:          "1100";
VLBA_recorder_no_parity:"1101";
S2_sampler              "1110";
milkingmachine:         "1111";
```

The four pins 7-3-5-1 can be though as one hex digit, MSB–>LSB. These bits light
up LEDs in order D2-D3-D4-D5, MSB–>LSB.

6.2 Performing Tests

I'll try to depict the procedure we have used for testing both VSIB and VSIC boards.
The "big picture" is as follows:

1. Get the nVidia PC up and running with those two CD-Rs, details in preced-
   ing sections.

2. Connect a jumper wire between VSIB TEST header (JP2) pin 1 and GND (JP3)
   to enable the Aux VSI connector (J3) to transmit the VSI-H test pattern at
   50MHz clock rate.

3. Connect the Aux VSI connector (J3) to the Primary VSI connector (J2) with
   the VSI cable. ("Loopback connection.")

4. Capture reference data with the './wr' command similar to the following:

   ```
   ./wr 90000 1000 1 /i1/t/tref 1000 <vsib
   ```

5. Provide a +4.5V..+5.0VDC supply to the VSIC power connector J3, GND goes
   to silkscreen +3.3V and +4.5V..+5.0VDC to the label +4.4V. The board requires
   approximately 1.2A when driving one output VSI connector. (Less than 1.5A
   when driving two VSI output connectors simultaneously.) Verify that the
   Xilinx loads its firmware from PROM when the RESET button S1 is pressed:
   LED D6 "NO PGM" should blink briefly and then stay off.

6. Set up VSI test pattern mode "7" at "TEST/MODE" header JP2. Wire-wrap pins 1,3,5 to GND (JP1) and verify that LEDs D1,D2,D3 light up. (Alternatively, you may want to connect a codewheel switch to a short ribbon cable and a 10-pin ribbon cable connector so that the common is a clip wire to GND (JP1) and LSB goes to pin 1 or JP2, next to 3, next to 5, and the MSB goes to 7.) This enables the VSIC to emit the VSI test pattern at 50MHz at both output VSI connectors J1 and J2.

7. Detach the VSI cable from VSIB aux VSI connector and connect it to VSIC VSI1 connector (J1). Capture data:

   ```
   ./wr 90000 1000 1 /i1/t/tvsic1-c1 1000 <vsib
   ```

8. Compare this data file with the reference data:

   ```
   cmp -l /i1/t/tref /i1/t/tvsic1-c1 | less
   ```

9. If discrepancies are found, dump the data file with either hex dump:

   ```
   hexdump /i1/t/tvsic1-c1 | less
   ```

   and/or bit dump

   ```
   ./d32 < /i1/t/tvsic1-c1 | less
   ```

   In bit dump stuck bits are easily identified by scrolling back and forth. The bit order in dump is from VSI connector MSB (b31) to LSB (b0).

10. When 'cmp' delivers the same "tref" results, move the VSI cable to VSI2 connector (J2) and repeat the test:

   ```
   ./wr 90000 1000 1 /i1/t/tvsic1-c2 1000 <vsib
   ```

11. Repeat the test for the second VSIC board you have built.

These steps have verified that the power supply regulators, Xilinx, PROM, XTAL, and the LVDS transceivers are operating correctly. For all this to work you need to setup 'wr.c' to skip every other 32-bit VSI data word as (50MHz*4bytes*8bits)==1600Mbits/s. (This is how 'wr.c' is shipped on the "remus.tgz" CD-R disk.)

The next step is to verify that RS422 inputs work correctly. For this to work correctly you need to set 'wr.c' to setup the VSIB board to capture every VSI 32-bit word. This is performed as follows. Please make sure that your 'wr' executable in 'proj/vsib' directory has been compiled with the following settings within the 'proj/vsib/wr.c' source code.

```
---wr.c---
...
void
start_VSIB(void)
{
  vsib_ioctl(VSIB_SET_MODE,
#if 1                          <---------------- this is one
          // 32-bit Mark5A. */
          (VSIB_MODE_MODE(/*vsib_mode=>*/0)
#endif
#if 0                          <---------------- this is zero
          // 16-bit Mark5A. */
          (VSIB_MODE_MODE(/*vsib_mode=>*/1)
#endif
#if 0                          <---------------- this is zero
```

```
                // 8-bit Mark5A. */
                (VSIB_MODE_MODE(/*vsib_mode=>*/4)
#endif
                | VSIB_MODE_RUN
                | (0 ? VSIB_MODE_GIGABIT : 0)              <---
zero
                | (0 ? VSIB_MODE_EMBED_1PPS_MARKERS : 0)   <---
zero
                | (/*skip_samples=>*/0 & 0x0000ffff))      <---
zero
              );
}  /* start_VSIB */
...
---
```

That is, we will not use VSIB "gigabit mode" (delay aux VSI output 1pps by one clock), will not change TOST data word to '0xffffffff' (the 32-bit VSI word which was captured when 1pps input went high), and "skip_samples" will be zero, so that VSIB does *not* skip any VSI words between saved VSI words.

Your 'wr.c' source code probably contains "/*skip_samples=>*/1" in the above, to allow capturing 50MHz VSI test data coming out of VSIB aux VSI connector when TEST pin 1 is connected to GND. (50MHz*32bits == 1600Mbits/s, way too much for even a short test run. When "/*skip_samples=>*/1", every other word is skipped, 1600/2==800Mbits/s which is ok for nVidia to push into memory; the disks won't sustain it for longer than a few seconds, though.)

When you have finished editing 'wr.c', say

```
make
```

To be absolutely sure you can say 'rm wr' before saying 'make'.

Now you can perform Mk4 formatter tests like follows:

1. Connect Mk4 formatter headstack #2 40-pin outputs ("WDB") to VSIC input connectors, even cable to IN#1 (J4) and odd to IN#2 (J5).

2. Enable a Mk4 mode at headstack #2. You need to set the formatter aux field to enable Mk4 formatter to emit track numbers in frame headers. The following FS commands seem to be able to setup this:

   ```
   form=a,8
   form4=/con 202
   form4=/aux 0x1234 0x5678
   ```

   You can select the data rate by changing the '8' in the above (8–>9 MHz per track, 288Mbits/s total) to, say, '16' for 576Mbit/s output per each 32-track headstack. The separate 'form4=/aux' command is apparently required for the formatter to emit track numbers in headers; the track numbers are set up as a by-product of setting up the fixed auxiliary field.

3. Select VSIC mode "A", that is, GND pins 3 and 7 of "TEST/MODE" header JP2. (LEDs D2 and D4 should be on.)

4. Capture some data:

   ```
   ./wr 90000 1000 1 /i1/t/tmk4h2-c1 1000 <vsib
   ```

5. Bit-dump the data to be able to see track numbers in the dump:

```
./d32 < /i1/t/tmk4h2-c1 | less -N
```

6. As you can see, most of the frame header bits are the same for every track, you'll see rows of '1 1 1 1 ...' and '0 0 0 0 ...'. However, the track-specific track number can be found, MSB first, starting from bit row 39:

```
---
39 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
40 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
41 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
42 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
43 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1
44 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
---
tr#33  31  29  27  25  23  21  19  17  15  13  11  09  07  05  03
       32  30  28  26  24  22  20  18  16  14  12  10  08  06  04  02
```

Again, this part will reveal both stuck bits and shorts to adjacent bits.

7. Repeat the test for another VSIC.

8. You can make the 50-pin - to - 40-pin conversion cables to test the headstack #1 connection. (Wires 41–50 are not connected.) Then if you set VSIC mode "8" (only pin 7 of "TEST/MODE" to GND) you will be able to capture similar data from Mk4 formatter headstack #1 "IOB" connectors.