Getting Slackware ARM on the Toshiba AC100 (also know as the Dynabook AZ)

When all this started off I was using ARMedslack 13.37 but as of version 14 the official Slackware ARM port changed name to Slackware ARM. Any reference to ARMedslack in this short tutorial refers to the Slackware ARM official port.

Preface

The Slackware ARM userland can run on almost any ARM based hardware out there and the Toshiba AC100 is no exception to that rule of thumb. The only impediment to going straight ahead is the kernel. I like to work around the problem by borrowing the kernel from some other distribution that has support for the AC100. currently you have 3 options to get reasonable support for the hardware:

- 2.6.38 kernel patched up for AC100
- 3.0 kernel patched up for AC100
- 3.15+ mainstream kernel (untested yet)

The 3.1 kernel development got broken on the way and to my understanding is unmaintained.

There are several places you can get the binary kernel images or the sources to compile them I generally look for Ubuntu first. Initially I used the Ubuntu kernel version 2.6.38 but over time I upgraded to version 3.0.27.

Hardware information

Here is some of the information I fetched on my AC100 before going into reflashing it with a different OS

The original partition layout

These values are relative to my model (10G), other models (even if with the same size EMMC) may have a different layout. You should double check how your EMMC is layed out before you attempt flashing anything. nvflash –getpartitiontable

Offsets show below are absolute … while offsets seen in tegrapart (from the kernel boot args) seem off from these absolute values by -2048 (-0x800).

```
part #  name    star        sector     size         description
remarks
nvflash   sector
2       BCT     0       15363     (3 MB)          boot config table
```

```
3       PT      1536        256     (512 kB)        partition table
4       EBT     1792        1024     (2 MB)          bootloader
5       SOS     2816        2560     (5 MB)          recovery partition
mmcblk0p1
6       LNX     5376        4096     (8 MB)          linux kernel and initrd
mmcblk0p2
7       MBR     9472        512     (1 MB)          master boot record
8       APP     9984        153600     (300 MB)        applications
(/system)        mmcblk0p3
9       CAC     163584      204800     (400 MB)        cache (/cache)
mmcblk0p4
10      MSC     368384      1024     (2 MB)          misc (/misc)
mmcblk0p5
11      EM1     369408      256     (512 kB)
12      UBA     369664      622320     (1215.5 MB)     user data (/data)
mmcblk0p6
13      EM2     1001984     256     (512 kB)
14      UDB     1002240     6837504 (13354.5 )      user data (/storage)
mmcblk0p7
```

All partitions have sectors of 2048 bytes and id 18.

The tegrapart format is a coma separated list of items like this :

```
<partition name>:<hex start sector>:<hex partition size in sectors>:<hex
sector size in bytes>,...
```

that needs to be read out from /proc/cmdline while the AC100 is running the stock OS or needs to be manually created based on the info from the output of tegrapart (the latter is tricky as it is undocumented).

Booting Custom Images

In order to boot and run the Slackwate userland you need to prepare a bootable image for aboot. Threre is 2 possible ways of doing that:

- via the rescue boot image
- via the normal boot image

Even if you manage to wreck both you can still rescue your device using nvflash but if you make backups and only flash one partition at a time you can use the other to put things right. You should keep the backups safe for any future use and/or restoring back the original OS. I tried flashing the rescue image first and then I use that to flash the rest … but you can keep using nvflash for both boot images and even for the root filesystem if you like the idea. There is a new rescue image that can be run from ram without actually flashing anything into the rescue partition, but it is handy to have a rescue partition that can be used to assist fixing the OS you are going to install (as the stock one is

useless on anything but the stock OS).

Requirements

What you'll need:

- kernel image for AC110 in zImage along with the kernel modules and firmware (here is the one I use)
- the boot and rescue images (these could be ok for a ac100-10G model but may require changing the kernel arguments on other models)
- abootimg utility for manipulating the images (git clone –depth 1 https://gitorious.org/ac100/abootimg.git)
- tegra-linux utility for doing the nvflash stuff (available from Toshiba at the time of writing)
- Slackware ARM miniroot tarball from http://ftp.slackware.org.uk/slackwarearm/slackwarearm-devtools/minirootfs/roots/
- A copy of the content of /proc/cmdline before you change kernel

Backing Up

Before you actually go ahead it's a good idea to make a backup of the original rescue and boot images. It won't hurt if you also do this for all the partitions … I'll leave that up to you. Start your device into recovery mode (POWER + ESC + CTRL) and then from the pc where you have the tegra-linux utils run the following com mands:

```
nvflash --bl harmony/fastboot.bin --sync
nvflash -r --read 5 original_part05_rescue.img
nvflash -r --read 6 original_part6_boot.img
```

Creating An Initrd

Although it is technically possible to boot without using an initrd I've not had the time to experiment that. If all you need is simple boot then the initrd only really needs busybox and a very simple script that mounts root and then switches root.

I make a slight difference between the rescue init script and the boot init script, making the former always drop you into a shell prompt from the initrd system and the latter always attempt to switch root.

This is what my boot init script looks like:

```
#!/bin/sh
exec 3>&1
mount -t sysfs -o nodev,noexec,nosuid none /sys
mount -t proc -o nodev,noexec,nosuid none /proc
sysctl -w kernel.printk="4 4 1 7" >/dev/null 2>&1
mount -t tmpfs -o mode=0755 none /dev
```

```
mknod -m 0600 /dev/console c 5 1
mknod /dev/null c 1 3
mkdir /dev/pts
mount -t devpts -o noexec,nosuid,gid=5,mode=0620 none /dev/pts
mdev -s
sleep 5
echo 0x0100 > /proc/sys/kernel/real-root-dev
mount -o ro /dev/mmcblk0p7 /root
if [ ! -r /root/sbin/init ]
then
  /bin/sh
fi
umount /proc
umount /sys
umount /dev/pts
exec switch_root /root /sbin/init 3
```

If you wish to alter the init scripts you can extract them from the rescue and boot images: first use abootimg to extract the initrd.img then use zcat | cpio -idm to extract the content from the initrd image. Edit them to suit your desired setup (in my case root is on the internal EMMC biggest partition), load any modules required for switching root and then repackage the images.

Repackaging The Boot Images For Aboot

You need to make sure the arguments passed to the kernel via cmdline are correct for your device so you need to disassemble the boot images anyway to check out the bootimg.cfg. Do the in a separate folder for each image:

```
abootimg -x <boot image>
```

If you made changes to the initrd or changed the kernel you need to repackage the boot images before going any further. From your initrd tree repackage the initrd image:

```
find . | cpio -o -H newc |gzip -9c > ../initrd.img
```

Edit the cmdline in the bootimg.cfg with the correct arguments that you previously read out from your device and add to them “root=<your root device> rootwait ro”. Failing to do this can result in unusable images.

Then repackage the image for aboot:

```
abootimg -create rescue_new.img -k zImage -f bootimg.cfg -r initrd.img
```

You are now theoretically ready to try your rescue image without actually flashing it.

Restart your device into recovery mode (POWER + ESC + CTRL) an load the rescue image into ram. I've not yet got round to prepare a ram rescue image (requires 3.8+ kernel, u-boot and other details

prepended to the kernel and initrd images) but over at ac100.grandou.net they have it sorted out, so for the mean time you can try their ram loading sos-uboot-r5-alpha.bin image ... other wise skip this step and go to the one below. If you want to read more about sos-uboot-r5-alpha you can do so here.

```
nvflash --bl sos-uboot-r5-alpha.bin --sync
```

Using traditional rescue system flashed to rescue partition.

```
nvflash --bl harmony/fastboot.bin --sync
nvflash -r --download 5 rescue_new.img
```

And then boot into rescue (POWER + HOME) then select 1.

If the rescue image works fine you can use it to flash the rest. On my device the rescue partition is /dev/mmcblk0p1 and the normal boot partition is /dev/mmcblk0p2. You will use dd for the rescue and boot images and mk?fs/tar for the root filesystem. Your images need to be on SD or usb mas storage so that you can read them from the rescue system.

Flashing From The Rescue Initrd

I forgot to mention: you need to put the Slackware ARM miniroot tarball and the boor image(s) on SD or USB so that you can flash them from rge rescue image. Once you're on the shell prompt from the rescue image mount the external media with the images

```
mkdir /mnt
mount /dev/<device> /mnt
```

Double check the partition layout on your dvice, it may differ from mine.

If you are running rescue from ram without having flashed it to EMMC

```
dd if=/mnt/rescue.img of=/dev/mmcblk0p1 bs=2048
```

If you are here you know the rescue image works so it's safe to flash the boot image too:

```
dd if=/mnt/boot.img of=/dev/mmcblk0p2 bs=2048
```

Fprmat the root filesystem (yoyr's may be on /dev/mmcblk1p? or /dev/sda?)

```
mke2fs -t ext4 -j -m 1 -b 4096 -i 16384 -l root /dev/mmcblk0p7
```

Mount it so you can extract the tarball:

```
mkdir /root_fs
mount /dev/mmcblk0p7 /root_fs
tar xpf /mnt/slackwarearm<version>.tgz -C /root_fs/
```

Make sure you set up fstab on the root partition to match your system. Try chroot to it to check all in

in place:

```
chroot /root_fs
```

Reboot and enjoy.

Handy Workarounds

Here are some little issues I worked around that other people might find handy

Audio With The Newer Kernel

Finally the speakers are working but in order to get any audio out of the AC100 with 3.0 kernels you haveto do a few things:

1. use alsamixer to unmute everything (or at least selectively what you need)
2. run “alsaucm -c tegraalc5632 reset” to reset the audio hardware
3. use “alsactl store” to store the settings so that you can restore them at boot time

You can then add this to rc.local

```
/usr/bin/alsaucm -c tegraalc5632 reset
/usr/sbin/alsactl restore
```

MPlayer

Mplayer from armedslack has unsuitable defaults for the AC100 so if you want video and audio playback go and edit /etc/mplayer/mplayer.conf and set these two lines:

```
vo=x11
ao=alsa
```

pppd

pppd does not correctly setup /etc/resolve.conf … I worked around this by writing very simple ip-up and ip-down to be put in /etc/ppp/

/etc/ppp/ip-up:

```
#!/bin/bash
```

```
if [ $USEPEERDNS -eq 1 ]
then
cp /etc/resolv.conf /etc/org_resolv.conf
> /etc/resolv.conf
[ "$DNS1" != "" ] && echo "nameserver $DNS1" >> /etc/resolv.conf
[ "$DNS2" != "" ] && echo "nameserver $DNS2" >> /etc/resolv.conf
[ -s /etc/resolv.conf ] && cat /etc/ppp/resolv.conf > /etc/resolv.conf
fi
```

/etc/ppp/ip-down:

```
#!/bin/bash
[ -s /etc/org_resolv.conf ] && cat /etc/org_resolv.conf > /etc/resolv.conf
```

Battery Status

Acpi is not working right and apm seems useless so I read the battery charge state directly from /sys/class/power_supply/battery/charge_now but the format is not directly human readable so I wrote something that calculates is as a percentage:

```
echo " $(echo "($(cat /sys/class/power_supply/battery/charge_now) * 100) /
$(cat /sys/class/power_supply/battery/charge_full_design)" | bc) %"
```

You will get an error if you run this when the AC100 is on AC power without the battery.

Adobe Flash plugin

Although you cannot download it from adobe, Android has the blasted libflasplayer.so library but even if you hack it out of some apk you will run into trouble with incompatible libraries dew to the different instruction sets used in android and ARMedslack. In order to get some basic flash support in my ARMedslack AC100 I used swfdec: it will not play youtube videos as you get told that you need to update but it's better then nothing. I've been reported that the android libflasplayer.so works on Slackware ARM 14.

Alternative Way To Boot Linux The First Time

If I remember correctly I think this is how I first got my custom image from my linux PC to the AC100 with the linux4tegra utilities. Start in Recovery mode

```
$ nvflash --bl fastboot.stock.bin --go
```

Backup your target partition

```
$ nvflash --resume --read 5 part05.img
```

Erase the kernel partition

```
$ nvflash --resume --format_partition 5
```

The system should go automatically to fastboot mode

Flash a kernel and a ramdisk

```
$ fastboot flash:raw boot zImage init.img
```

Building Newer 3.0 Kernel

This is work in progress, it boots and works but it's not been tested extensively like 2.6.38

There has been a lot of work done on mainstream kernel from 3.8 on, but I've not had the time to check out if it's functional enough to be used on the AC100 as a netbook and besides that there are also other issues that are holding me back on doing the testing on mainstream kernel:

- I still have not wired up the uart for debugging things if the frame buffer console ceases to function as expected
- mainstream kernels do not support tegra part so work needs to be done to migrate on-board emmc to gpt partition
- at some point also fdt will need attention too

According to ac100.grangow.net form 3.15 on there is full support on mainstream kernel.

Native kernel build (in order to compile this native 512Mb are no longer sufficient so you need to swap on something {SD,usb storage or internal emmc}: an extra 512Mb swap will suffice). It's technically possible to work form marvin's kernel git but I've not been able to configure that kernel to boot correctly ... it hangs while attempting to read RTC, but it's probably just something wrong that I did in the configuration. #git clone –depth 1 -b chromeos-ac100-3.0 https://gitorious.org/~marvin24/ac100/marvin24s-kernel.git

```
wget
https://launchpad.net/ubuntu/+archive/primary/+files/linux-ac100_3.0.27-1.1.
tar.gz
tar xf linux-ac100_3.0.27-1.1.tar.gz
cd linux-ac100-2.6.38 #yeah the mane of the directory is misleading !
make paz00_defconfig
make zImage modules firmware INSTALL_MOD_STRIP=1
make firmware_install
make modules_install INSTALL_MOD_STRIP=1
```

Prepare The Image For Booting

Once you've the modules in place you need to package the kenel in aboot format … I previously did this with an old tool … there's now a standard tool: “abootimg”

```
git clone --depth 1 https://gitorious.org/ac100/abootimg.git
cd abootimg
make
make install
```

There are 2 possibilities for booting with the stock boot loader:

1. form the normal boot partition (/dev/mmcblk0p2 on my device)
2. from the rescue boot partition (/dev/mmcblk0p1 but only works on android 2.1 systems)

You can choose to leave the normal boot partition alone and target the rescue partition or vice versa but don't mess with both until you have transitioned to a functional system.

The rescue partition is only 5Mb while the normal boot is 8Mb depending on how bid your image is you might be forced to use the ordinary boot … but conceptually the image creation is the same.

Dump your target partition with whatever tool is most convenient: dd if you have a linux system or rooted busybox on the AC100 or nvflash for non rooted devices. then examine the image with “abootimg -i”

```
root@ac100:~# abootimg -i /dev/mmcblk0p2

Android Boot Image Info:

* file name = /dev/mmcblk0p2 [block device]

* image size = 8388608 bytes (8.00 MB)
  page size  = 2048 bytes

* Boot Name = "Ubuntu Boot Img"

* kernel size       = 2628788 bytes (2.51 MB)
  ramdisk size      = 2219793 bytes (2.12 MB)

* load addresses:
  kernel:       0x10008000
  ramdisk:      0x15000000
  tags:         0x10000100

* empty cmdline
* id = 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000

root@ac100:~#
```

You also need to save the original command line passed to the kernel because it's not stored in the original image.

```
root@ac100:~# cat /proc/cmdline
mem=448M@0M lp0_vec=0x2000@0x1c29e000 tegraboot=sdmmc
tegrapart=recovery:300:a00:800,boot:d00:1000:800,mbr:1d00:200:800,system:1f0
0:25800:800,cache:27700:32000:800,misc:59700:400:800,userdata:59c00:9a600:80
0
root@ac100:~#
```

When you repackage up the image you will want to add to those arguments: “root=<root device> rootwait ro” where <root device> will be wherever your root filesystem will be (/dev/mmcblk0p7 for internal EMMC, /dev/mmcblk1p1 if root will be on the SD's first partition or /dev/sda1 if on a usb stic's first partition).

The first time I repackaged up an image I did not know much about aboot so I used a custom inirtd to leave the boot process as close as possible to the original … but it is possible to boot without initrd I just have not had time to play with it. It your original image is from android you may need to create your own initrd … the kernel has all that's required to boot so the initrd needs only do vary basic stuff (all that you really need is busybox if all you want is just booting).

First extract the contents of the original image by running “abootimg -x <image_file_name>” This will extract the config file, the kernel and initrd (bootimg.cgf, zImage and initrd.img respectively)

Modify the bootimg.cfg to look something like this:

```
root@ac100:/boot# cat bootimg.cfg
bootsize = 0x800000
pagesize = 0x800
kerneladdr = 0x10008000
ramdiskaddr = 0x15000000
secondaddr = 0x10f00000
tagsaddr = 0x10000100
name = Slackware Boot Img
cmdline = mem=448M@0M lp0_vec=0x2000@0x1c29e000 tegraboot=sdmmc
tegrapart=recovery:300:a00:800,boot:d00:1000:800,mbr:1d00:200:800,system:1f0
0:25800:800,cache:27700:32000:800,misc:59700:400:800,userdata:59c00:9a600:80
0 root=/dev/mmcblk0p7 rootwait ro
root@ac100:/boot#
```

If the original image is an android image you need to create your own initrd. Replace the zImage with whatever you compiled and repackage it up with “abootimg –create slack.img -k zImage -f bootimg.cfg -r initrd.img”

Flash the new boot image back to the partition you decided ti target (with whatever method was used to extract it in the first place) and reboot.

Contacts

If you have any issues or suggestions about this short tutorial you are welcome to contact me (louigi600 (at) yahoo (dot) it) or even contribute yourself directly on this wiki. I apologize in advance for any vital information that may have been left out when tosh-ac100.wetpaint.com was blocked and this how-to was moved here.

Sources

- Originally written by louigi600

howtos, hardware, arm, author louigi600