Hacking information from the XZPAD700

This refers to the XZPAD700 (aka zeligpad) ARM based tablet (AL-A13-RT713 pcb) based on an Allwinner A13 SOC but is technically applicable to all Axx SOC's as to my understanding the all boot in the same manner.
If you're just interested in getting started really fast skip to the image builder section.
The image builder has support for almost all Allwinner Axx SOC's. I've thoroughly tested it on mt tablet and I've had success reports on Cuietruck. If you have had success on some other device please report it on the Slackware ARM subforum on LinuxQuestions.

Preface

Although this is a low-end inexpensive device it turns out that the platform is better documented then many other high-end devices with well known brand names. For this platform there is a lot of good documentation on http://linux-sunxi.org/Main_Page but unfortunately there are some common issues with most other ARM based patforms: Patches not making it to mainstream kernel, Drivers left in a incomplete state that that require hard ti find firmware to work properly, Drivers unmaintained that get broken as other back-ports make it into the linux-sunxi kernel tree.

Enough ranting about how ARM linux development lacks organization, let's just gather some information out of this device to help install a standard linux userland (possibly Slackware) on it.

Accessing The Device

Untill more is known on the hardware (revealing where the serial port is) there are 2 basic ways of accessing the device so that you get a command prompt.

1. Android Debug Bridge
2. A Terminal Emulator Application

The former requires installing android-sdk on a pc, activating the developement debug mode on the device, connecting the device to the pc via the usb cable and then launching “adb shell” from the pc. You then gret dropped into a shell prompt on the device.

The latter only requires you to install a terminal emulation software on the device and then using that to snoop around on the device itself. Where possible I prefer using adb because I get frustrated while typing on the virtual keyboard.

In both cases you may need to root the device to get full access to the system but on the XZPAD700 it is not the case as you can just type “su -” and become root right out of the box.

Hardware

Let's have a look at what's in the hardware:

```
root@android:/ # cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
 89 i2c
108 ppp
116 alsa
125 aw_i2c_ts
128 ptm
136 pts
150 cedar_dev
180 usb
188 ttyUSB
189 usb_device
229 ump
230 mali
248 lcd
249 pa_chrdev
250 BaseRemoteCtl
251 ttyGS
252 disp
253 bsg
254 rtc

Block devices:
  1 ramdisk
259 blkext
  7 loop
  8 sd
 11 sr
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
 93 nand
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
254 device-mapper
root@android:/ #
```

It appears the the device has a serial port but it's not where one would expect. This device comes with a smart feature: the MicroSD (uSD) Breakout featuring JTAG and UART connectivity trough the uSD socket but be warned that you may only use one option at the time (ie you cannot have the UART if you use the uSD). More on this later.

USB

This tablet comes with a mini-AB port with true host/slave functionality. According to the devices specifications it should be an OTG port but it's wildly out of spec as it can power up HDU units, internet 3G sticks and other stuff that is typically close to the 500mA limit on standard USB ports. The A13 has another USB port that is dedicated do the internal WiFi card. There's an article on wikipedia that may be useful to those wishing to make their own adapter for the mini-A socket to put the port in HOST mode.

With the stock android on the device I've used successfully all sorts of devices without the use of externally powered hubs, but at time the linux-sunxi kernels make get broken OTG driver. In any case as of 3.4.75 I was able to get the USB Ethernet cards working on the OTG port.

Internal Flash

```
root@android:/ # cat /proc/partitions
major minor  #blocks  name

   7        0       3150 loop0
   7        1      14585 loop1
  93        0      16384 nanda
  93        8      16384 nandb
  93       16      32768 nandc
  93       24     393216 nandd
  93       32    1048576 nande
```

```
 93       40      16384 nandf
 93       48      32768 nandg
 93       56     262144 nandh
 93       64     131072 nandi
 93       72    1981440 nandj
254        0       3150 dm-0
254        1      14584 dm-1
root@android:/ #
```

And here's where they are mounted:

```
root@android:/ # mount
rootfs                /           rootfs rw 0 0
tmpfs                 /dev        tmpfs rw,nosuid,relatime,mode=755 0 0
devpts                /dev/pts    devpts rw,relatime,mode=600,ptmxmode=000 0
0
proc                  /proc       proc rw,relatime 0 0
sysfs                 /sys        sysfs rw,relatime 0 0
none                  /acct       cgroup rw,relatime,cpuacct 0 0
tmpfs                 /mnt/asec   tmpfs rw,relatime,mode=755,gid=1000 0 0
tmpfs                 /mnt/obb    tmpfs rw,relatime,mode=755,gid=1000 0 0
/dev/block/nandd      /system     ext4
rw,nodev,noatime,user_xattr,barrier=0,data=ordered 0 0
/dev/block/nande      /data       ext4
rw,nosuid,nodev,noatime,user_xattr,barrier=0,journal_checksum,data=ordered,n
oauto_da_alloc 0 0
/dev/block/nandh      /cache      ext4
rw,nosuid,nodev,noatime,user_xattr,barrier=0,journal_checksum,data=ordered,n
oauto_da_alloc 0 0
/dev/block/vold/93:72 /mnt/sdcard vfat
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0
702,allow_utime=0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8
,errors=remount-ro 0 0
/dev/block/vold/93:72 /mnt/secure/asec vfat
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0
702,allow_utime=0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8
,errors=remount-ro 0 0
tmpfs                 /mnt/sdcard/.android_secure tmpfs
ro,relatime,size=0k,mode=000 0 0
/dev/block/dm-0       /mnt/asec/com.shootbubble.bubbledexlue-1 vfat
ro,dirsync,nosuid,nodev,relatime,uid=1000,fmask=0222,dmask=0222,codepage=cp4
37,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
/dev/block/dm-1       /mnt/asec/com.wyse.pocketcloudfree-1 vfat
ro,dirsync,nosuid,nodev,relatime,uid=1000,fmask=0222,dmask=0222,codepage=cp4
37,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
root@android:/ #
```

I dumped the nandX flash partitions and had a look at them from my pc:

```
root@darkstar:~/XZPAD700/dumps/nandx_dd# du -ms *
16      nanda
16      nandb
32      nandc
384     nandd
1025    nande
16      nandf
32      nandg
256     nandh
128     nandi
1936    nandj
root@darkstar:~/XZPAD700/dumps/nandx_dd# file *
nanda: x86 boot sector, code offset 0x0, OEM-ID "        ", sectors/cluster
4, root entries 512, Media descriptor 0xf8, sectors/FAT 256, sectors 262144
(volumes > 32 MB) , dos <    4.0 BootSector (0x0), FAT (16 bit)
nandb: data
nandc: data
nandd: Linux rev 1.0 ext4 filesystem data, UUID=bcf4f857-f4ab-df65-
ff57-946fc0f9f25b (needs journal recovery) (extents) (large files)
nande: Linux rev 1.0 ext4 filesystem data, UUID=57f8f4bc-abf4-655f-
bf67-946fc0f9f25b (needs journal recovery) (extents) (large files)
nandf: data
nandg: data
nandh: Linux rev 1.0 ext4 filesystem data, UUID=57f8f4bc-abf4-655f-
bf67-946fc0f9f25b (needs journal recovery) (extents) (large files)
nandi: Linux rev 1.0 ext4 filesystem data, UUID=57f8f4bc-abf4-655f-
bf67-946fc0f9f25b (extents) (large files)
nandj: x86 boot sector, code offset 0x58, OEM-ID "android ", sectors/cluster
8, heads 64, sectors 3955658 (volumes > 32 MB) , FAT (32 bit), sectors/FAT
3856, Backup boot sector 2, serial number 0x4a9d1407, label: "CRANE      "
root@darkstar:~/XZPAD700/dumps/nandx_dd#
```

Touch Screen

Looking at the init scripts I found some evidence of what I suspect is the touch screen that is present on this device:

```
init.sun5i.rc 72:     insmod /system/vendor/modules/ft5x_ts.ko
root@android:/ # lsmod | busybox grep ft5
ft5x_ts 45529 0 - Live 0xbf0a6000
root@android:/ #
```

It's the FocalTech ft5x TouchScreen driver but it's not present in the vanilla kernel (at least not upto and including 3.4.56) probably because the driver code has not yet been accepted for inclusion (see note below), but I've found the driver to be present in both the 3.0 and 3.4 linux-sunxi kernel sources:

```
root@darkstar:/XZPAD700/kernel# find . -type f -name *ft5x*
./linux-sunxi-sunxi-3.4/drivers/input/touchscreen/ft5x_ts.c
./linux-sunxi-sunxi-3.4/drivers/input/touchscreen/ft5x_ts.h
```

```
./linux-sunxi-sunxi-3.0/drivers/input/touchscreen/ft5x_ts.c
./linux-sunxi-sunxi-3.0/drivers/input/touchscreen/ft5x_ts.h
root@darkstar:/XZPAD700/kernel#
```

I've loaded manually the module from my Slackware image and had a partial success with gpm and X but there are issues:

- haven't found any means of calibrating it (ant it looks like it could benefit from a calibration)
- neither gpm nor X seems to work well with this drive/hardware combination
- absence of mouse buttons doesn't help at all

This is my guess to why the ft5x driver has not yet made it into the mainstream kernel: it appears that this driver needs a firmware blob bundled into it to make it functional. This firmware blob depends on the hardware implementation of the platform making it darn hard for an end user to get a working driver for any kernel that is not what came stock with the hardware. Technically the code is GPL, and with that basic code you get some sort of basic but flawed functionality, but without the blob it's pretty much useless on a tablet.

Bootloader

I've found evidence of U-Boot configuration environments on nanda and nandb.

```
U-Boot 2011.09-rc1-dirty (Nov 22 2012 - 14:25:29) Allwinner Technology
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿþÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
bootcmd=nand read 50000000 boot;boota 50000000
bootdelay=1
baudrate=115200
bootdelay=3
bootcmd=run setargs boot_normal
console=ttyS0,115200
nand_root=/dev/nandd
mmc_root=/dev/mmcblk0p4
init=/init
loglevel=8
setargs=setenv bootargs console=${console} root=${nand_root}init=${init}
loglevel=${loglevel}
boot_normal=nand read 50000000 boot; boota 50000000
boot_recovery=nand read 50000000 recovery; boota 50000000
boot_fastboot=fastboot
bootcmd=nand read 50000000 boot;boota 50000000
bootdelay=1
baudrate=115200
bootdelay=3
bootcmd=run setargs boot_normal
console=ttyS0,115200
nand_root=/dev/nandd
```

```
mmc_root=/dev/mmcblk0p4
init=/init
loglevel=8
setargs=setenv bootargs console=${console} root=${nand_root}init=${init}
loglevel=${loglevel}
boot_normal=nand read 50000000 boot; boota 50000000
boot_recovery=nand read 50000000 recovery; boota 50000000
boot_fastboot=fastboot
set_default_env
env_import
mmc_saveenv
nand_saveen
```

System On Chip

```
root@android:/proc # cat /proc/cpuinfo
Processor       : ARMv7 Processor rev 2 (v7l)
BogoMIPS        : 1001.88
Features        : swp half thumb fastmult vfp edsp neon vfpv3
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc08
CPU revision    : 2

Hardware        : sun5i
Revision        : a13b
Serial          : 01c0f06339313030504d4e33162542c4
```

The SOC seems to be a Allwinner A13 (sun5i Cortex-A8) with support for the following optional features:

- NEON,
- VFPv3,
- Thumb-2

and features a single cortex A8 core @ 1ghz and mali400 GPU @320mhz.
There is a lot of very useful information on the linux-sunxi.org main page .

If you have a look at the linux-sunxi.org main page you will find all the information required for customizing this tablet. Here are a few of the useful titles on that site that may be very helpful:

- Binary drivers
- Bootable SD card
- BROM
- BSP
- Buildroot
- CodeSourcery
- Cpufreq
- Display

- FEL/USBBoot
- Fex Guide
- FirstSteps
- GPIO
- How to boot the A10 over the network
- Hwpack
- Initial Ramdisk
- Kernel arguments
- MicroSD Breakout
- Optimizing system performance
- U-Boot
- Wifi

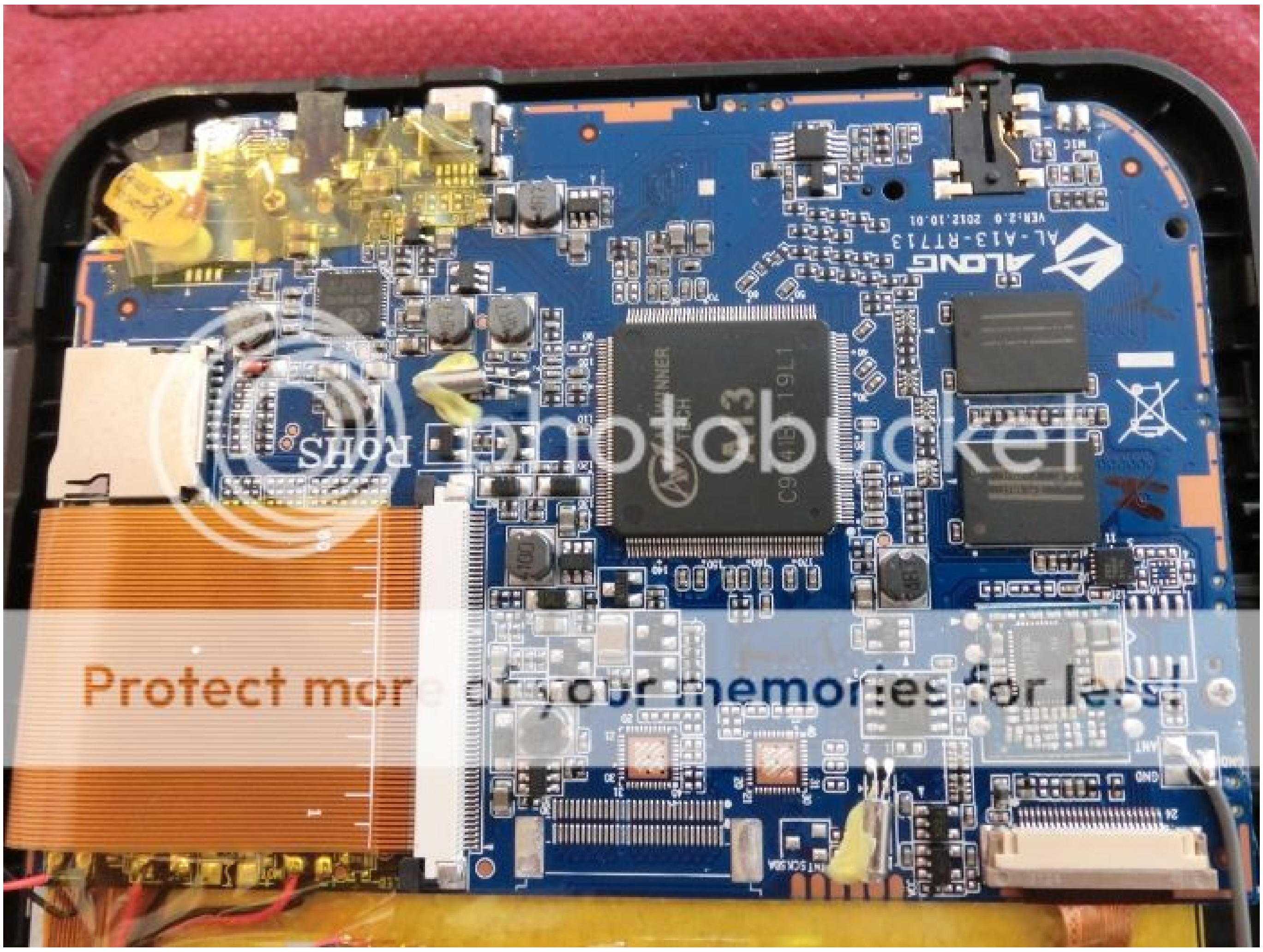
In particular there is an interesting article concerning how the Allwinner SOC based boards boots. This gives a fair idea on the alternatives available for booting custom images besides being an interesting read anyway.

MicroSD Breakout

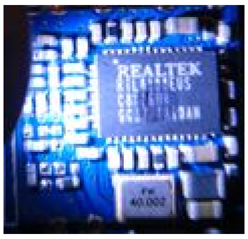
Pin	MMC/MicroSD	JTAG Connection	14-pin ARM JTAG Header	UART Connection	5-pin UART Header
1	Data2	TCK	9	nc	nc
2	CD/Data3	nc	nc	RX	2
3	Cmd	TDO	11	nc	nc
4	VDD	VTG	1,13	VDD	3
5	CLK	nc	nc	TX	1
6	VSS	GND	2,4,6,8,10,14	GND	4, 5
7	Data0	TDI	5	nc	nc
8	Data1	TMS	7	nc	nc
nc	nc	nTRST	3	nc	nc
nc	nc	nRESET	12	nc	nc

If you don't want to tinker you can buy a ready-made UART and JTAG uSD breakout header ready to use on A10/A13 devices from here . But be aware that you may use only one feature at the time, so if you intend to use the uSD slot for storage you will not be able to use the UART.

The Internals

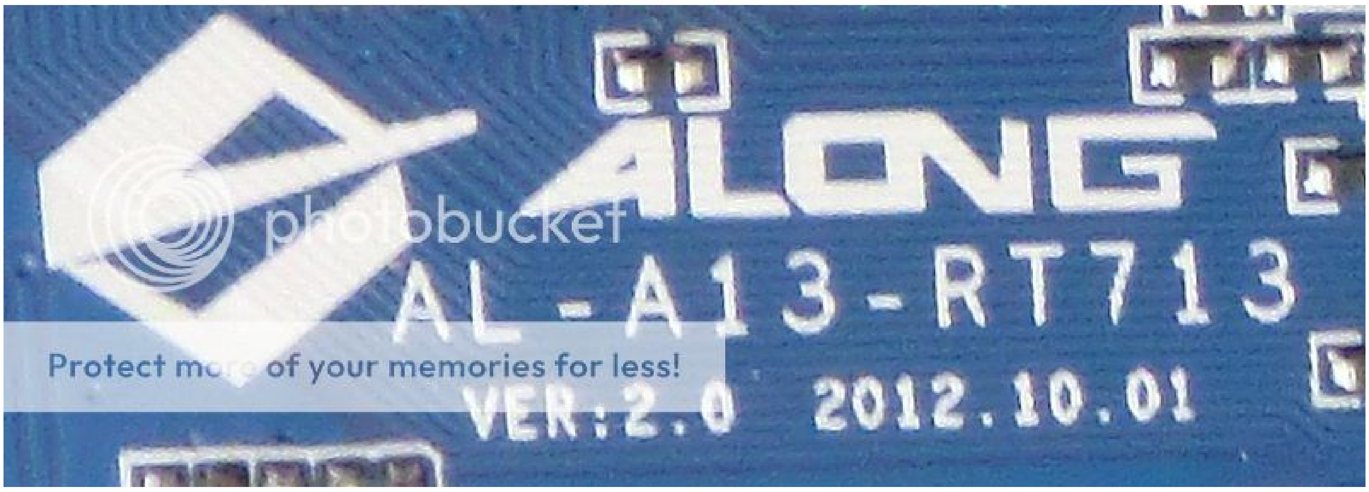


high definition picture

I'm not 100% sure but on this side I think we can see the A13 SOC, the AXP209 lipo battery management chip, the Realteck RTL8188EUS wifi NIC  and 2 DDR ram chips. The internal flash is located under the LCD flect cable . I did a bit of searching with the board label and it appears that this is equivalent to: Allwinner Boxchip A13 Tablet

ALLWINNER
TECH
photobucket
A13
Protect more of your memories for less!
C9041BA 19L1

The SOC



The board label

Booting a Custom Image from MicroSD

See the Wrapping Up section of this article for links to the image builder I'm currently using.

Although a lot of information in this section comes from linux-sunxi.org I'll rearrange it here removing any distro oriented stuff that is not pertinent to Slackware and keeping it as neutral as possible anyway.

This device is based on a AL-A13-RT713 that is as close as you can get to a a13_mid device so we will attempt booting with a setup fro that device. Here's the list of things that will be required:

1. kernel and kernel modules
2. the binary version of FEX config file script.bin
3. sunxi-spl.bin
4. u-boot.bin
5. a root system image containing the userland

1 is available from the A13_mid hardware pack v 3.0 or v 3.4.Alternatively you can try building your own kernel from linux-sunxi source tree (be warned that a lot of patches for the Allwinner SOC have not yet made it to mainstream kernel).
2 needs to be ripped out of the nanda partition on the device itself, failing to do this will result in a kernel panic due to incorrect setup for kernel.
3 and 4 need to bi ripped out of the fedora image for Axx as the ones that come in the hwpack seem to fail booting on this device even with the correct script.bin.

Our userland is going to be one of these because we are Slackware freaks 😉 but you can have anything else you like (prepare a large enough image to hold whatever you prefer).

There are other various things that could get you booting with a little work but I like to keep things as neutral as possible so that if you are not a Slackware freak you can still use most of the information here to boot whatever suits you best.
Jeff Doozan has a rescue system for the A10 SOC systems,there is a thread on Doozan's wiki regarding his rescue system.
BerryBoot also supports booting custom images for the A10 SOC.
Thierry Merle's Flash Kitchen has a working solution for the A10.

Preparing the MicroSD raw image

TheSlackware ARM 14 miniroot can fit in 200Mb so I guess the absolute minimum would be 256Mb uSD but that's not going to be much use on a tablet, I'd reccomend starting off with a 512Mb uSD.
If you want a functional X11, with a minimal window manager like fluxbox and various usefull libraries, a working gcc compiler along with some extra networking stuff you're looking at around 1.6Gb of occupied disc space so I guess that you can do with a 2 commertial Gb uSD. Once you convert the space in to real GB, take away the boot partition space and create the filesystems you'll not be left with much usable space … maybe just enough to download the compressed kernel sources tarball.
Let's try with the just to check everything works.
NB: 512Mb flash devices are 512 commercial Mb and work out to be something like 507379712 bytes or 483.875 Mb
(one Megabite is 2^20 bytes or 1048576 bytes)

Let us begin by creating a raw image in which we will then create the uSD image:

```
root@darkstar:~# dd if=/dev/zero of=512mb_raw.img bs=1024 count=495488
495488+0 records in
495488+0 records out
507379712 bytes (507 MB) copied, 13.4449 s, 37.7 MB/s
root@darkstar:~#
```

Partition the card

Now let's partition it (sfdisk that is on my linux has a bug that forces me to use sectors as units, it would be handier if you can use Mb as units).
The crunch of is is that the firs partition needs to begin 1Mb from the beginning of the device and at least 16Mb big.

```
root@darkstar:~# cat <<EOT | sfdisk -f -uS  512mb_raw.img
2048,32768,c,*
34817,,L
0,0,0
0,0,0
EOT
root@darkstar:~#
```

Older u-boot builds for this platform required the boot partition to be vfat, but this in no longer mandatory with the newer builds and ext2/3 are now valid for boot partition too. You will need a 16Mb boot partition (offset 1Mb from the beginning of the device) and at least one partition for the system.

Formatting the Filesystems

In order to format correctly the filesystems on the raw image we need to use of losetup with offset

and size.
Remember that the first partition begins 1Mb from the beginning of theimage and that the root filesystem will then begin 17Mb from the beginning of the image.

```
root@darkstar:~# losetup -o 1048576 --sizelimit 16777216 /dev/loop0
512mb_raw.img
root@darkstar:~# losetup -o 17826304 --sizelimit  489552896 /dev/loop1
512mb_raw.img
root@darkstar:~# mkfs.vfat /dev/loop0
root@darkstar:~# mkfs.ext4 /dev/loop1
```

The filesystems can now be mounted so that we can deposit the required files in them.

Some thoughts on filesystems designed for ordinary block devices on flash devices:
Unlike flash specific filesystems like jffs2 the linux journaled filesystems ext3 and 4 are designed to work on ordinary block devices, the way blocks are rewritten and the way the journal is used was not conceived to be conservative on the number of rewrites cycles on flash devices. Also the way block rewrites take place does not take into account that the erase block of a flash device is typically 32 times bigger then the block that the fs driver wants to write, resulting in a physical rewrite of much more data then the the block that was requested to be flushed to disk.
Be warned that while most SLC flash devices allow for 100000 rewrites MLC devices typically only get 10000 rewrites before failure. Most low end flash devices are MLC. If you really want journal at least make sure you mount the device with the noatime,data=ordered option or you will literally burn out your uSD card really quickly.
This is why I format most of my root filesystems on flash devices that get managed like ordinary IDE/SATA/SCSI/SAS devices as ext2, but there is a major issue with portable devices: an uncontrolled reset can drop your device into a prompt for root password for maintenance, and if you don't have a keyboard handy you are screwed untill you get hold of one … so I sujjest this device be formatted with a journaled root filesystem.

Preparing the Boot Partition

In the boot partition you need to deposit kernel in a format that uboot will manage (uImage) along with script.bin.
To mount the vfat boot partition we can still use the loop0 device.
the a13_mid_hwpack.tar.xz is the hardware pack that can be downloaded from the link above.

```
root@darkstar:~#  mount /dev/loop0 /mnt/floppy
root@darkstar:~#  tar xf a13_mid_hwpack.tar.xz kernel/script.bin
kernel/uImage -C /mnt/floppy
root@darkstar:~# cat << EOF > /mnt/floppy/boot.cmd
setenv bootargs console=\${console} root=\${root} loglevel=\${loglevel}
\${panicarg} \${extraargs}
fatload mmc 0 0x43000000 script.bin
fatload mmc 0 0x48000000 \${kernel}
watchdog 0
bootm 0x48000000
EOF
root@darkstar:~# cat << EOF > /mnt/floppy/uEnv.txt
console=tty0
```

```
loglevel=5
root=/dev/mmcblk0p2 ro rootwait
extraargs=console=ttyS0,115200 disp.screen0_output_mode=EDID:1280x720p60
hdmi.audio=EDID:0 sunxi_g2d_mem_reserve=0 sunxi_ve_mem_reserve=0
sunxi_fb_mem_reserve=16 sunxi_no_mali_mem_reserve
EOF
root@darkstar:~# umount /mnt/floppy
root@darkstar:~# losetup -d /dev/loop0
```

If you are interested in making an initrd setup (for whatever reason) you might find this an interesting read.

Preparing the root filesystem

In this partition you need to extract a root image. On linux-sunxi.org it is suggested to use an ubuntu-alip image but we are slackware freaks so we will use one of these miniroots. You may actually use any userland that is compatible with the cpu architecture. The sun51 is an CORTEX A8 AMM v7 instruction set so it will likely run any userland available as long as it's compiled in a ABI format compatible with the kernel you put in the boot partition.
Just keep in mind that this device does not have a physical keyboard, not all standard linux include virtual keyboards amongst their packages. You might want to add to your root image xvbkd and have it fire up along with xdm (or whatever other manager you choose to run) so that you can login.

For xdm you can just add a line like this at the end of /etc/X11/xdm/Xsetup_0:
exec /usr/local/bin/xbkbd -xdm -compact -minimizable -geometry 550×185+100+290 &

```
root@darkstar:~# mount /dev/loop1 /mnt/floppy
root@darkstar:~# tar xf a13_mid_hwpack.tar.xz -C /tmp
root@darkstar:~# cp -apr /tmp/rootfs/* /mnt/floppy
oot@darkstar:~# tar xf slack-14.0-miniroot_27Sep12.tar.xz -C /mnt/floppy
root@darkstar:~# cat << EOF > /mnt/floppy/etc/fstab
proc            /proc           proc    defaults        0       0
/dev/mmcblk0p2       /               ext4    errors=remount-
ro,noatime,data=ordered 0 1
EOF
root@darkstar:~# umount /mnt/floppy
root@darkstar:~# losetup -d /dev/loop1
```

Preparing the Bootloader

Unfortunately the bootloader stuff supplied from the hwpack seems to be broken, it's best to rip the bootloader stuff out of the fedora image for the A1* family.

```
root@darkstar:~# losetup /dev/loop2 512mb_raw.img
root@darkstar:~# dd if=/tmp/bootloader/sunxi-spl.bin of=/dev/loop2 bs=1024
seek=8
```

```
20+1 records in
20+1 records out
20992 bytes (21 kB) copied, 0.000304966 s, 68.8 MB/s
root@darkstar:~# dd if=/tmp/bootloader/u-boot.bin of=/dev/loop2 bs=1024
seek=32
211+1 records in
211+1 records out
217036 bytes (217 kB) copied, 0.00115387 s, 188 MB/s
root@darkstar:~# dd if=uEnv-img.bin of=/dev/loop2 bs=1024 seek=544
128+0 records in
128+0 records out
131072 bytes (131 kB) copied, 0.00124961 s, 105 MB/s
root@darkstar:~# sync
root@darkstar:~# losetup -d /dev/loop2
```

If you want to build your own images you might want to have a look at these git repositories:

```
git clone git://github.com/linux-sunxi/u-boot-sunxi.git
git clone git://github.com/linux-sunxi/linux-sunxi.git
git clone git://github.com/linux-sunxi/sunxi-tools.git
git clone git://github.com/linux-sunxi/sunxi-boards.git
```

Wrapping up

You may now write the image to a uSD card with a simple cat redirected to the card's block device. Supposing your card is on /dev/sdg:

```
root@darkstar:~# cat 512mb_raw.img > /dev/sdg
```

Slip the card in the uSD slot and boot your device.

Have a look at the freshly booted device:

```
/dev/mmcblk0p2: clean, 20627/29952 files, 57606/119519 blocks
Remounting root device with read-write enabled.
[    5.0700001] EXT4-fs (mmcblk0p2): warning: checktime reached, running e2fsck is recommended
/dev/mmcblk0p2 on / type ext4 (rw,noatime,errors=remount-ro)
Checking non-root filesystems:
fsck from util-linux 2.21.2
Mounting non-root local filesystems:
nothing was mounted
Using /etc/random-seed to initialize /dev/urandom.
INIT: Entering runlevel: 3
Going multiuser...
Updating shared library links: /sbin/ldconfig
Starting sysklogd daemons: /usr/sbin/syslogd /usr/sbin/klogd -c 3 -x
Triggering udev to write persistent rules to /etc/udev/rules.d/
Triggering udev events: /sbin/udevadm trigger --action=change
Starting RPC portmapper: /sbin/rpc.portmap
Starting RPC NSM (Network Status Monitor): /sbin/rpc.statd
Starting OpenSSH SSH daemon: /usr/sbin/sshd
Starting system message bus: /usr/bin/dbus-uuidgen --ensure ; /usr/bin/dbus-daemon --system
Loading /usr/share/kbd/keymaps/i386/qwerty/uk.map.gz

Welcome to Linux 3.4.43.sun5i+ (tty1)

Slackware login:
```

and here's a video of the device booting (stable kernel this time)

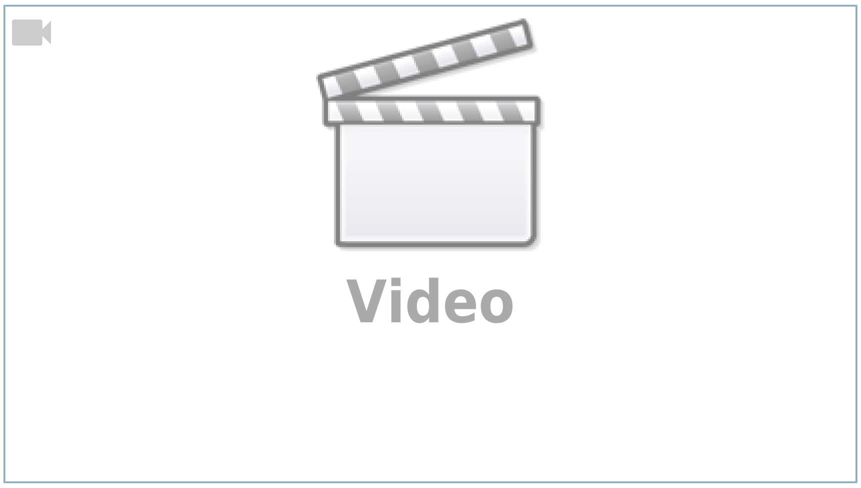


Image Builder

The image builder is for brewing up an arbitrary size SD image to match whatever size SD you have. It is technically possible to use a 256Mb SD for a Slackware ARM miniroot system with little space for a few extra packages, but I advice to start off with no less then 1/2Gb SD card.
You will need to know what type of architecture your Allwinner SOC is (sun4i, sun5i or sun7i) before you start: look this up here. It is also possible to use this image builder with other distribution's root images but it may require a little manual work if the target distribution does not have the modules in /lib/modules/<kernel version>.

Here's a link to the image builder I'm currently using (it has inside all you need to get the Slackware ARM 14.0 or 14.1 miniroot booting on many Allwinner SOC based devices) just download all the files and you're ready to go. The files are also available via http or via rsync.

I've thoroughly tested it on mt tablet and I've had success reports on Cuietruck. If you have had success on some other device please report it on the Slackware ARM subforum on LinuxQuestions.

I'll leave the slightly older version on sourceforge as a tarball for a while. It's not really the right place for it as it does not fit in perfectly with my clashNG project but it's related so I hope sourceforge won't mind having it there for a while.

Booting over Network

There is a documented procedure for booting this tab over network http://linux-sunxi.org/How_to_boot_the_A10_over_the_network. Getting the device to do a normal installation would probably only require having a working kernel for this hardware.

Bootsplash

This device shows 3 splash images/animations while booting:

1. uboot splash image
2. linux kernel logo
3. android boot animation

Changing the kernel logo will require fiddling with the kernel so we'll skip that until I manage extracting it from the internal flash or compile a working version of my own. But it's easy to change the uboot splash and the android boot animation.

U-Boot splash image

You can change the uboot splash image by mounting nanda rw and editing the linux.ini file:

```
busybox mkdir -p /data/local/tmp/mnt/
busybox mount -o rw /dev/block/nanda /data/local/tmp/mnt/
cd  /data/local/tmp/mnt/linux
```

```
busybox vi linux.ini
```

and make the logo_info section look like this

```
[logo_info]
logo_name = c:\linux\slack.bmp
logo_show = 1
```

you then need to create a 800×480 windows bitmap image of whatever you prefer and call it slack.bmp. This should be copied over to the linux directory in the nanda partition and then you can umount the nanda partition. Next time you boot normally from the internal flash uboot should show the custom image before flipping over to the linux kernel logo image and subsequently to the android bootanimation.

Android bootanimation

This device has a bootanimation binary that loads a default image (probably self contained in the binary) if no bootanimation.zip file is found on the system. By inspecting the binary it appears that it looks for such files in the following paths:

```
/system/media/bootanimation-encrypted.zip
/data/local/bootanimation.zip
/system/media/bootanimation.zip
/system/media/boot.mp3
```

I prefer to target /data/local/bootanimation.zip bacause it does not require fiddling remounting as this part is already RW. The zip file needs to something like this in it:

```
root@darkstar:/tmp/my_bootanimation# ls -lR
.:
total 8
-rw-r--r-- 1 drao users   22 Jul  8 14:59 desc.txt
drwxr-xr-x 2 drao users 4096 Jul  8 14:56 part0/

./part0:
total 32
-rw-r--r-- 1 drao users 14124 Jul  8 14:15 001.jpg
-rw-r--r-- 1 drao users 14124 Jul  8 14:25 002.jpg
root@darkstar:/tmp/my_bootanimation# cat desc.txt
480 800 2
p 0 0 part0
root@darkstar:/tmp/my_bootanimation#
```

Where the desc.txt file configures how the animation is to be displayed:

- 480 800 2 means that the images are 800×480 and the animation is showed @ 2 FPS (I've still images so that's fine for me, but it can be 30FPS if you need)

- p 10 0 part0 means that images are loaded from a directory part0 and are looped 10 times (0 would be infinite loop) and each iteration pauses 0 Frames before starting over again

You can then zip it without copmpression:

```
root@darkstar:/tmp/my_bootanimation# zip -r -0 ../my_bootanimation.zip  *
  adding: desc.txt (stored 0%)
  adding: part0/ (stored 0%)
  adding: part0/002.jpg (stored 0%)
  adding: part0/001.jpg (stored 0%)
root@darkstar:/tmp/my_bootanimation#
```

and push it to your device like this

```
 ./adb push /tmp/my_bootanimation.zip /data/local/bootanimation.zip
```

Next time bootanimation binary is executed it will find and load the custom animation. You can execute it manually from a terminal application or from adb shell. If your animation is an infinite loop you can hit CRTL^C to quit execution from the shell that you used to launch it.

Sources

- Originally written by louigi600

howtos, hardware, arm, louigi600