Bootable SD card

Contents

Introduction

This page describes how to create a bootable SD card. Depending on how the SD card is connected, the location to write data to can be different. Throughout this document `${card}` refers to the SD card and `${p}` to the partition if any. If the SD card is connected via a USB adapter, linux will know it for example as `/dev/sdb` (with `/dev/sda` being a boot drive). Please notice that this device can be different based on numerous factors, so when not sure, check the last few lines of dmesg after plugging in the device (`dmesg | tail`). If connected via a SD slot on a device, linux will know it as `/dev/mmcblk0` (or `mmcblk1`, `mmcblk2` depending on which mmc slot is used).

Data is either stored raw on the SD card or in a partition. If `${p}` is used then the appropiate partition should be used. Also this differs for USB adapters or mmc controllers. When using an USB adapter, `${p}` will be 1, 2, 3 etc so the resulting device is `/dev/sdb1`. Using an mmc controller, this would be p1, p2, p3 etc so the resulting device is `/dev/mmcblk0p1`.

To summarize: `${card}` and `${card}${p}1` mean `/dev/sdb` and `/dev/sdb1` on a USB connected SD card, and `/dev/mmcblk0`, `/dev/mmcblk0p1` on an mmc controller connected device.

⚠ If the SD card is connected in another way, the device nodes can change to be even different, take this into account.

SD Card Layout

A default U-Boot build for an Allwinner based board uses the following layout on (micro-)SD cards or eMMC storage (from v2018.05 or newer):

start	sector	size	usage
0KB	0	8KB	Unused, available for an MBR or (limited) GPT partition table
8KB	16	32KB	Initial SPL loader
40KB	80	-	U-Boot proper

Typically partitions start at 1MB (which is the default setting of most partitioning tools), but there is no hard requirement for this, so U-Boot can grow bigger than 984KB, if needed.

The 8KB offset is dictated by the BROM, it will check for a valid eGON/TOC0 header at this location. The 40KB offset for U-Boot proper is the default U-Boot setting and can be changed at build time using the `CONFIG_SYS_MMCSD_RAW_MODE_U_BOOT_SECTOR` configuration variable.

Newer SoCs (tested on H2+, A64, H5, H6, T113) can also load the SPL from sector 256 (128KB) of an SD card or eMMC, if no valid eGON/TOC0 signature is found at 8KB (BROM boot order). The U-Boot proper offset needs to be adjusted accordingly in this case. U-boot patch (https://groups.google.com/forum/#!topic/linux-sunxi/BW7BKGjSTtM) more details (https://groups.google.com/forum/#!topic/linux-sunxi/MaiijyaAFjk)

Mainline U-Boot used to have a more complex, fixed layout for the SD card/eMMC sectors in the first Megabyte:

Legacy SD card layout

start	sector	size	usage
0KB	0	8KB	Unused, available for MBR (partition table etc.)
8KB	16	32KB	Initial SPL loader
40KB	80	504KB	U-Boot
544KB	1088	128KB	environment
672KB	1344	128KB	Falcon mode boot params
800KB	1600	-	Falcon mode kernel start
1024KB	2048	-	Free for partitions

As the feature set of U-Boot proper grew over time, this proved to be too restricting, as we completely filled the area before the environment and started to corrupt it. To avoid future issues, it was decided to move the default location for the environment to a FAT partition, which is more flexible and has no real size limits.

Identify the card

First identify the device of the card and export it as `${card}`. The commands
`cat /proc/partitions`
or
`blkid -c /dev/null`
can help with finding available/correct partition names.

- If the SD card is connected via USB and is sdX (replace X for a correct letter)

```
export card=/dev/sdX
export p=""
```

- If the SD card is connected via mmc and is mmcblk0

```
export card=/dev/mmcblk0
export p=p
```

Cleaning

To be on safe side erase the first part of your SD Card (also clears the partition table).

```
dd if=/dev/zero of=${card} bs=1M count=1
```

If you wish to keep the partition table, run:

```
dd if=/dev/zero of=${card} bs=1k count=1023 seek=1
```

Bootloader

You will need to write the *u-boot-sunxi-with-spl.bin* to the sd-card. If you don't have this file yet, refer to the "compilation" section of mainline or legacy U-Boot.

```
dd if=u-boot-sunxi-with-spl.bin of=${card} bs=1024 seek=8
```

To update the bootloader from the U-Boot prompt itself:

```
mw.b 0x48000000 0x00 0x100000                # Zero buffer
tftp 0x48000000 u-boot-sunxi-with-spl.bin    # Or use load to read from MMC or SCSI etc
mmc erase 0x10 0x400                         # Erase the MMC region containing U-Boot, do not reset at this
point!
mmc write 0x48000000 0x10 0x400              # Write updated U-Boot
```

If using U-Boot v2013.07 or earlier then the offsets, and therefore procedure, are slightly different:

Note: if bootloader was generated by Buildroot (tested on 2015.02), this is the case.

```
dd if=spl/sunxi-spl.bin of=${card} bs=1024 seek=8
dd if=u-boot.bin of=${card} bs=1024 seek=32
```

Partitioning

With recent U-Boot it's fine to use ext2/ext4 as boot partition, and other filesystems in the root partition too.

With separate boot partition

Partition the card with a 16MB boot partition starting at 1MB, and the rest as root partition

```
blockdev --rereadpt ${card}
cat <<EOT | sfdisk ${card}
1M,16M,c
,,L
EOT
```

You should now be able to create the actual filesystems:

```
mkfs.vfat ${card}${p}1
mkfs.ext4 ${card}${p}2
```

```
cardroot=${card}${p}2
```

Boot Partition

```
mount ${card}${p}1 /mnt/
cp linux-sunxi/arch/arm/boot/uImage /mnt/
cp sunxi-boards/sys_config/a10/script.bin /mnt/
umount /mnt/
```

Single partition

```
blockdev --rereadpt ${card}
sfdisk  ${card}
cat <<EOT | sfdisk ${card}
1M,,L
EOT
```

```
mkfs.ext4 ${card}${p}1
```

```
cardroot=${card}${p}1
```

Boot Partition

```
mount ${card}${p}1 /mnt/
mkdir /mnt/boot
```

```
cp linux-sunxi/arch/arm/boot/uImage /mnt/boot
cp sunxi-boards/sys_config/a10/script.bin /mnt/boot
umount /mnt/
```

GPT (experimental)

There is 8kb space for partition data. MBR uses only the first sector and allows for 4 partitions. If you are concerned about the 4 partition limitation you can try different partitioning scheme. While GPT standard mandates that GPT should have at least 128 entries gdisk can resize a GPT partition to 56 entries which fit into the 7kb that follow the protective MBR header and GPT header. Linux understands such GPT but some tools refuse it since it does not adhere to the standard. YMMV[1]

The GPT partition table can also be moved out of the way of the SPL and U-Boot. This has the advantage that the full 128 or more partition table entries mandated by the GPT standard can be used. The start of the partition table is stored in the GPT header (LBA 1), and is usually set to 2. Version 1.0.3 and later of the gdisk program has the ability to change this value (command j in the "extra functionality" menu). The following table shows the card layout with the partition table start relocated to LBA 2048.

start	size	usage
0	0.5KB	Protective MBR
1	0.5KB	GPT header
2	7KB	Unused
8	32KB	Initial SPL loader
40	504KB	U-Boot
544	128KB	environment
672	128KB	Falcon mode boot params
800	-	Falcon mode kernel start
1024	16KB	Partition table
1056	-	Free for partitions

Boot Script

Preparation of a boot script is described on the U-Boot configuration page.

Rootfs

This depends on what distribution you want to install. Which partition layout you use does not matter much, since the root device is passed to the kernel as argument. You might need tweaks to `/etc/fstab` or other files if your layout does not match what the rootfs expects. As of this writing most available images use two partitions with separate `/boot`.

Using rootfs tarball

```
mount ${card}${p}2 /mnt/
tar -C /mnt/ -xjpf my-chosen-rootfs.tar.bz2
umount /mnt
```

Linaro rootfs

Linaro offers a set of different root filesystems (https://wiki.linaro.org/Platform/DevPlatform/Rootfs). A retention policy of 30 days applies to Linaro rootfs on snapshot servers. New snapshots can be generated on request. Latest snapshots can be made from sources such as Ubuntu Build Service (https://git.linaro.org/gitweb?p=ci/ubuntu-build-service.git)

In any case, you can get the actual rootfs tarballs here (http://snapshots.linaro.org/ubuntu/images/). ALIP is a minimal LXDE based desktop environment which might me useful to most allwinner users.

Note that recent (2015, and maybe even earlier) versions of ALIP/Linaro/Ubuntu and any other rootfs that makes use of *systemd* (and possibly also *upstart*) can only be used with a kernel compiled with `CONFIG_FHANDLE=y`[2]. In the default configuration of the Sunxi-3.4 kernel this option is not set (It says "*# CONFIG_FHANDLE is not set*" in `.config`). So you must take care of this yourself during kernel configuration (*"General Setup", "Open by fhandle syscalls"*).

Otherwise your kernel will boot, rootfs will mount and after that nothing will happen: no login prompt will appear on any console. If you must use a kernel without `CONFIG_FHANDLE`, try using a Debian rootfs with *sysvinit*.

Rootfs from LinuxContainers

LinuxContainers projects has various downloadable rootfs images (https://images.linuxcontainers.org/images/).

Using debootstrap - Debian/Ubuntu based distributions

Using Debian's Debootstrap, you can create your own rootfs from scratch. The process is described in the debootstrap howto.

Kernel modules

When you have copied rootfs to your card you might want to copy the kernel modules as well.

Troubleshooting

- **Check partitioning** - if you did the partitioning yourself read back the layout with `sfdisk` in sectors. `sfdisk` and `gparted` sometimes apply weird rounding when using megabytes.

```
sfdisk -uS -d /dev/sdd
# partition table of /dev/sdd
unit: sectors

/dev/sdd1 : start=     2048, size= 16150528, Id=83
/dev/sdd2 : start=        0, size=        0, Id= 0
/dev/sdd3 : start=        0, size=        0, Id= 0
/dev/sdd4 : start=        0, size=        0, Id= 0
```

- **Re-check that you have written the image correctly** Check image checksum if provided. Re-read writing instructions carefully. Try another writing method if available - dd / phoenixsuit / win32-diskimage. Especially writing on Windows tends to cause trouble. If your board is new you can try an image for similar board with the same CPU. Use console cable if you have one to check the boot messages.
- **Power off the board completely before booting** If you are using a console cable the board may not power off completely. There is a possiblility that self-powered USB peripherials or USB hubs may cause sililar issue. The red power light would get dimmer when the board is off but does not turn off completely. In this case the mmc controller may not get reset properly and the board boots from nand. Power off the board, disconnect all peripherials, and disconnect the serial console cable. Try booting again. You can re-connect your peripherials before booting. This issue does not seem to happen when the kernel powers down the mmc controller properly but is common when the kernel crashes.
- **Check for bad micro-SD card contact** This is common issue on boards that use micro-SD socket. Try removing and re-inserting the card, cleaning the contacs on the card and dusting off the SD card socket. Some people report that inserting the card together with a piece of paper improves contact and allows booting cards which are too loose in the socket.

See also

- FEL (Special formatted SD card for FEL boot)
- U-Boot

External

- Additional info on sunxi's flavor of U-Boot (https://github.com/linux-sunxi/u-boot-sunxi/wiki)

References

1. https://en.wiktionary.org/wiki/YMMV
2. http://unix.stackexchange.com/questions/169935/no-login-prompt-on-serial-console