Initrd

After kernel booted, it tries to mount a file system. Using Linux on DaVinci, there are several options where this file system can come from. Options are

- Harddisk or CompactFlash card
- MMC/SD card
- NFS [1] (see DVEVM Getting Started Guide, sprue66c.pdf, section 4.3.4 [2])
- (Flash) file system in NOR or NAND
- Ramdisk

The last option, an initial file system in a ram disk is called *initrd* [3] (**init**ial **r**am**d**isk). Note that using an initrd with recent kernels is still possible and has some advantages, but isn't recommended any more. Using initramfs [4] is the preferred way today.

initrd/initramfs (and NFS) is a file system option not permanently stored at the target device and thus normally used while development. initrd/initramfs is typically new installed/downloaded each time the target board is power cycled (using e.g. boot loader U-Boot). The two other options above (hard disk and flash file system) give the target system permanently access to its file system without any external debug connection (e.g. network download) and thus are used *after* development in production ready devices.

initrd

Creation

To create an (initially empty) initrd use the following steps:

Note: Change *count* to your required filesystem size. E.g. with count=8192 you will get a 8MB ramdisk.

```
host > dd if=/dev/zero of=/dev/ram0 bs=1k count=<count>
host > mke2fs -vm0 /dev/ram0 <count>
host > tune2fs -c 0 /dev/ram0
host > dd if=/dev/ram0 bs=1k count=<count> | gzip -v9 > ramdisk.gz
```

Now, we have a (empty) gzipped ramdisk image with (extracted) size of <*count*>.

Filling

To fill empty ramdisk created above with all files needed for ramdisk, mount the image and fill it. Content would be e.g. BusyBox [5] and/or other applications and/or libraries.

```
host > mkdir mnt
host > gunzip ramdisk.gz
host > mount -o loop ramdisk mnt/
host > ... copy stuff you want to have in ramdisk to mnt...
host > umount mnt
host > gzip -v9 ramdisk
```

The resulting ramdisk.gz is now ready for usage. Note its size is smaller than <*count*> cause of compression.

Note: Don't forget to create/copy some basic /dev/xxx nodes to ramdisk.

Note: If BusyBox or applications in ramdisk are linked dynamically, don't forget to copy dynamic libraries (*.so) to ramdisk (to correct directory) as well.

Kernel options

To make initrd work, you have to configure kernel properly:

```
#
# General setup
#
...
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_SOURCE=""
...

#
# UBI - Unsorted block images
#
...
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=1
CONFIG_BLK_DEV_RAM_SIZE=8192
CONFIG_BLK_DEV_RAM_BLOCKSIZE=1024
...
```

Note: The ramdisk size e.g. 8192 above has to be configured for your individual setup.

Installation

Now, you can install the ramdisk via u-boot e.g. in NOR flash. For this copy filled ramdisk created above to your tftpboot directory on host (e.g. /tftpboot/ramdisk.gz). Then start target and copy the data into RAM and flash:

```
UBOOT # tftp 0x87000000 ramdisk.gz
UBOOT # erase 0x2200000 +0x<filesize>
UBOOT # cp.b 0x87000000 0x2200000 0x<filesize>
```

Note: Replace *filesize* above by the value the tftp download command gives you as *Bytes transferred*.

Now, last step is to update kernel boot parameters and save them

```
UBOOT # setenv bootargs ... root=/dev/ram0 rw initrd=0x87000000,8M
UBOOT # setenv bootcmd cp.b 0x2200000 0x87000000 0x<filesize>; bootm
UBOOT # saveenv
```

Note: In example above with "8M" we assume that your ramdisk is 8MBytes. Adapt this to your needs.

Note: Your ramdisk filled above should have a /dev/ram0 node

```
brw-rw---- 1 root disk 1, 0 Sep 11 1999 /dev/ram0
```

to make this work properly.

Now you should be able to start your kernel and it should find and mount the initrd:

```
Linux version 2.6.23-davinci1 ...
...
checking if image is initramfs...it isn't (no cpio magic); looks like an initrd
Freeing initrd memory: 8192K
```

```
...
RAMDISK driver initialized: 1 RAM disks of 8192K size 1024 blocksize
...
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem).
Freeing init memory: ...
...
```

initramfs

To use initramfs a cpio [6] archive is embedded directly into the kernel. I.e. you don't create an additional (ramdisk) image. Instead, the initial file system is directly incorporated into the kernel. With this, the kernel size increases by the file system size. It's like you embed above ramdisk directly into the kernel.

Creation

Cause initramfs is directly embedded in the the kernel, its creation is simpler. No dd & mount & gzip stuff like with ramdisk above. You simply have to fill a directory on your host with the target filesystem you like and then pass the path to this directory to the kernel build process.

Create target file system

```
host > mkdir target_fs
host > ... copy stuff you want to have in initramfs to target_fs...
```

Note: cpio system used for initramfs can't handle hard links [7]. If you e.g. created your BusyBox using hard links, you will get a quite large initramfs cause each command is taken with its size and not as hard link. In cpio initramfs use symbolic/soft links [8] instead.

Note: To be able to detect initramfs by kernel properly, the top level directory has to contain a program called *init*. For example, this can be done by using a soft link from top level init to /bin/busybox (assuming you are using BusyBox in your initramfs).

```
/init -> /bin/busybox
```

To create the cpio archive execute the following commands.

```
host > cd target_fs
host > find . | cpio -H newc -o > ../target_fs.cpio
```

Kernel options

The only difference from creating an initrd is to give the kernel the path to the target file system you like to embed:

```
#
# General setup
#
...
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_SOURCE="<path_to>/target_fs>"
...

#
```

```
# UBI - Unsorted block images
#
...
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=1
CONFIG_BLK_DEV_RAM_SIZE=8192
CONFIG_BLK_DEV_RAM_BLOCKSIZE=1024
...
```

Then, if you compile the kernel, e.g. by make uImage, the cpio archive is generated and embedded into the kernel:

```
 ...
 CHK     include/linux/compile.h
 GEN     usr/initramfs_data.cpio.gz
 AS      usr/initramfs_data.o
 LD      usr/built-in.o
 ...
```

Installation

No special installation like above with initrd is necessary. The initramfs is already in the kernel. If you start the kernel, the initramfs is already there. Therefore, there is **no** *root=/dev/ram0 rw initrd=0x87000000,8M* bootargs option necessary. Remove this if you still have it!

initrd vs. initramfs

- Using initrd, kernel and initial file system are splitted. Making changes to kernel or filesystem doesn't touch the other one. The download size (e.g. while development) of one component is smaller.
- Creating and modifying an initramfs is easier than with initrd (unzip & mount & unmount & zip)
- Having one big image (kernel & initramfs) is easier to handle (e.g. download or flashing) than having two splitted images.

References

[1] http://en.wikipedia.org/wiki/Network_file_system
[2] http://focus.ti.com/general/docs/techdocsabstract.tsp?abstractName=sprue66c
[3] http://www.ibm.com/developerworks/linux/library/l-initrd.html
[4] http://linuxdevices.com/articles/AT4017834659.html
[5] http://www.busybox.net/
[6] http://en.wikipedia.org/wiki/Cpio
[7] http://en.wikipedia.org/wiki/Hard_link
[8] http://en.wikipedia.org/wiki/Soft_link

Article Sources and Contributors

Initrd *Source*: http://processors.wiki.ti.com/index.php?oldid=47945 *Contributors*: Db, Dirk, Isaac, Jonhunter