2. KernelBoot

NXKR_YoungSikYang

Exported on 02/27/2024

Table of Contents

1 Difference between kernel images

1.1 Image

- **Image** refers to the uncompressed, raw binary image of the Linux kernel. This is the most basic form of the kernel image, directly produced by the kernel compilation process. However, this raw Image format is not directly used in most embedded systems due to its size and lack of metadata for bootloaders.

1.2 zImage

- **zImage** is a compressed version of the Linux kernel(ARM32), built directly from the kernel source. The kernel is compressed to reduce its size, which is crucial for embedded systems with limited storage space. The zImage includes a small decompression stub at the beginning of the image, which is executed by the bootloader and unpacks the kernel into memory before it starts.

1.3 uImage

- **uImage** is a format specific to the U-Boot bootloader, widely used in embedded systems. It wraps around the kernel image (which can be an Image, zImage, or even an initramfs image) with a header that includes metadata like the image's size, load address, entry point, and compression type. This metadata is used for the bootloader to correctly load and execute the kernel image. The uImage format is designed to be versatile, supporting different types of payloads and compression, making it suitable for a wide range of embedded devices.

2 How to compile the linux kernel to make Image and zImage

2.1 Install dependencies

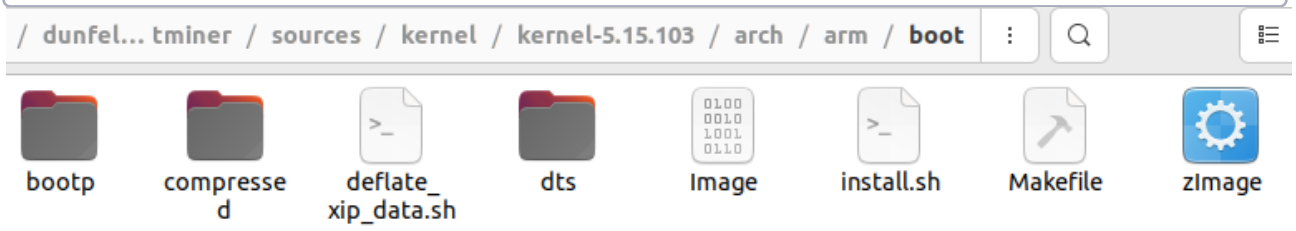
```
sudo apt-get install make build-essential libncurses-dev bison flex libssl-dev
libelf-dev
```

2.2 Build (for ARM32)

```
sudo apt-get install gcc-arm-linux-gnueabihf # Cross compiler
make menuconfig # Save menuconfig and create .config
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
make
# make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu-
```

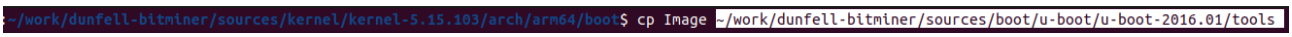
2.3 Build (for ARM64)

```
sudo apt-get install gcc-aarch64-linux-gnu # Cross compiler
make menuconfig # Save menuconfig and create .config
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
make
# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
```

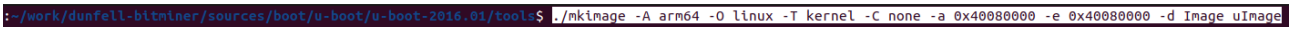


3 How to make uImage

3.1 Copy the compiled kernel image to wrap around the uImage into uboot-2016.01/tools



3.2 Create uImage




```
./mkimage -A arm64 -O linux -T kernel -C none -a 0x40080000 -e 0x40080000 -d Image
uImage
```

4 How to boot the created image (uImage in this example)

4.1 Check the boot command with $ `printenv`

```
boot_cmd_mmcboot=ext4load mmc 0:1 0x40080000 Image;ext4load mmc 0:1 0x49000000 s5p6818-bitminer-rev01.dtb;run dtb_reserve;booti 0x40080000 - 0x49000000
```

This shows the kernel image is loaded into 0x40080000 and the DTB into 0x49000000

4.2 Load the image into the RAM of the board

```
udown 0x40008000 # Enter this in minicom to prepare to load the image into the RAM of
the board
./usb-downloader -t slsiap -f ~/work/dunfell-bitminer/sources/boot/u-boot/u-
boot-2016.01/tools/uImage
udown 0x49000000 # for the DTB
./usb-downloader -t slsiap -f ~/work/dunfell-bitminer/build_bitminer_bitminer/tmp/
deploy/images/s5p6818/s5p6818-bitminer-rev01.dtb
```

```
bitminer# udown 0x40008000
Download Address 40008000 Size  20429376(hex : 137ba40)
Download complete
```

```
USB Download Tool

=> Header Mode
============================================================
 Nexell USB Downloader Version 1.4.2-artik, Mar  3 2017, 15:36:32
 Send Image width NSHI Header
 processor type  : slsiap
 nsih file       : NULL
 bin file        : /home/youngsik/work/dunfell-bitminer/sources/boot/u-boot/u-bo
ot-2016.01/tools/uImage
 secondboot file : NULL
 download addr   : default
 start addr      : default
=> Downloading 20429888 bytes
=> Download Success!!!
```

4.3 Run the uImage in the target board

```
bootm 0x40008000 - 0x49000000
```

```
bitminer# bootm 0x40008000 - 0x49000000
## Booting kernel from Legacy Image at 40008000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    20429312 Bytes = 19.5 MiB
   Load Address: 40080000
   Entry Point:  40080000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 49000000
   Booting using the fdt blob at 0x49000000
   Loading Kernel Image ... Image too large: increase CONFIG_SYS_BOOTM_LEN
Must RESET board to recover
resetting ...
b◆
U-BootDRAM:  219 MiB
POFFHIS: NONE
PONHIS : PWRONPON
CHGS   : CHG OFF
DCDC1  : 1250mV, En, dclim:1303023664, dclimsden:0
DCDC2  : 1200mV, En, dclim:1303023664, dclimsden:0
DCDC3  : 3300mV, En, dclim:1303023664, dclimsden:0
DCDC4  : 1500mV, En, dclim:1303023664, dclimsden:0
DCDC5  : 1500mV, En, dclim:1303023664, dclimsden:0
MMC:   NEXELL DWMMC: 0
In:    serial
Out:   serial
Err:   serial
Net:
Warning: ethernet@c0060000 (eth0) using random MAC address - e6:44:71:53:be:f9
eth0: ethernet@c0060000
Hit any key to stop autoboot:  0
11926784 bytes read in 561 ms (20.3 MiB/s)
41888 bytes read in 8 ms (5 MiB/s)
## Error: "dtb_reserve" not defined
## Flattened Device Tree blob at 49000000
   Booting using the fdt blob at 0x49000000
   Using Device Tree in place at 0000000049000000, end 000000004900d39f

Starting kernel ...
```

5 booti, bootz

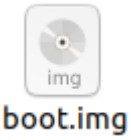
- booti: The default boot command of this target board is booti to boot an uncompressed Image. (e.g. booti 0x40008000 - 0x49000000)
- bootz: bootz is disabled by default and should be enabled if it needs to be used. (e.g. bootz 0x40008000 - 0x49000000)

Add this in common/kconfig to enable bootz

```
config CMD_BOOTZ
        bool "bootz"
        default y
        help
          Boot a zImage from the memory.
```

6 How to upload data into the eMMC using fastboot

6.1 File system



- **Kernel Image (`Image`):** This is the compiled Linux kernel. It's the core of the operating system that manages the hardware and allows other programs to run.
- **Device Tree Blob (`*.dtb`):** This file describes the hardware in the system so that the kernel knows what devices are present and how they are configured.
- **Ramdisk (`initrd`):** This is a temporary root file system used during the boot process before the real root filesystem is mounted.

These components are combined into a flashable boot.img to get flashed onto the file system of the eMMC

6.1.1 How to check if boot.img containts the kernel Image

```
def find_pattern(source, pattern):
    position = source.find(pattern)
    if position != -1:
        print(f"Found the pattern at position: {position}")
    else:
        print("The pattern not found in the file.")

with open('boot.img', 'rb') as f:
    boot_img = bytearray(f.read())
with open('Image', 'rb') as f2:
    kernel = bytearray(f2.read())
    find_pattern(boot_img, kernel)
    kernel[1] = 1
    find_pattern(boot_img, kernel)
```

```
Found the pattern at position: 5300372
The pattern not found in the file.
```

6.2 partmap_emmc.txt

```
flash=mmc,0:2ndboot:2nd:0x200,0x10000:bl1-emmcboot.img;
flash=mmc,0:fip-loader:boot:0x10200,0x50000:fip-loader-emmc.img;
flash=mmc,0:fip-secure:boot:0x60200,0x180000:fip-secure.img;
flash=mmc,0:fip-nonsecure:boot:0x1E0200,0x100000:fip-nonsecure.img;
flash=mmc,0:env:env:0x2E0200,0x4000:params.bin;
flash=mmc,0:boot:ext4:0x400000,0x4000000:boot.img;
flash=mmc,0:rootfs:ext4: 0x4400000,0x3b400000:rootfs.img;
flash=mmc,0:user:ext4: 0x40000000,0:userdata.img;
```

6.3 Upload

6.3.1 Host PC

```
sudo fastboot flash partmap boot-binary/partmap_emmc.txt # Map partitions in the eMMC
# Upload
sudo fastboot flash 2ndboot boot-binary/bl1-emmcboot.img
sudo fastboot flash fip-loader boot-binary/fip-loader-emmc.img
sudo fastboot flash fip-secure boot-binary/fip-secure.img
sudo fastboot flash fip-nonsecure boot-binary/fip-nonsecure.img
sudo fastboot flash env boot-binary/params.bin
sudo fastboot flash boot boot.img
sudo fastboot flash rootfs rootfs.img
sudo fastboot flash user userdata.img
```

6.3.2 Target board (in u-boot)

```
fast 0
```