Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

2 U-boot

Youtube: https://youtu.be/Iq6CfYajaSw

Hes·so//FRIBOURG FREIBURG
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

References

[1]:u-boot sources: http://www.denx.de/wiki/U-Boot

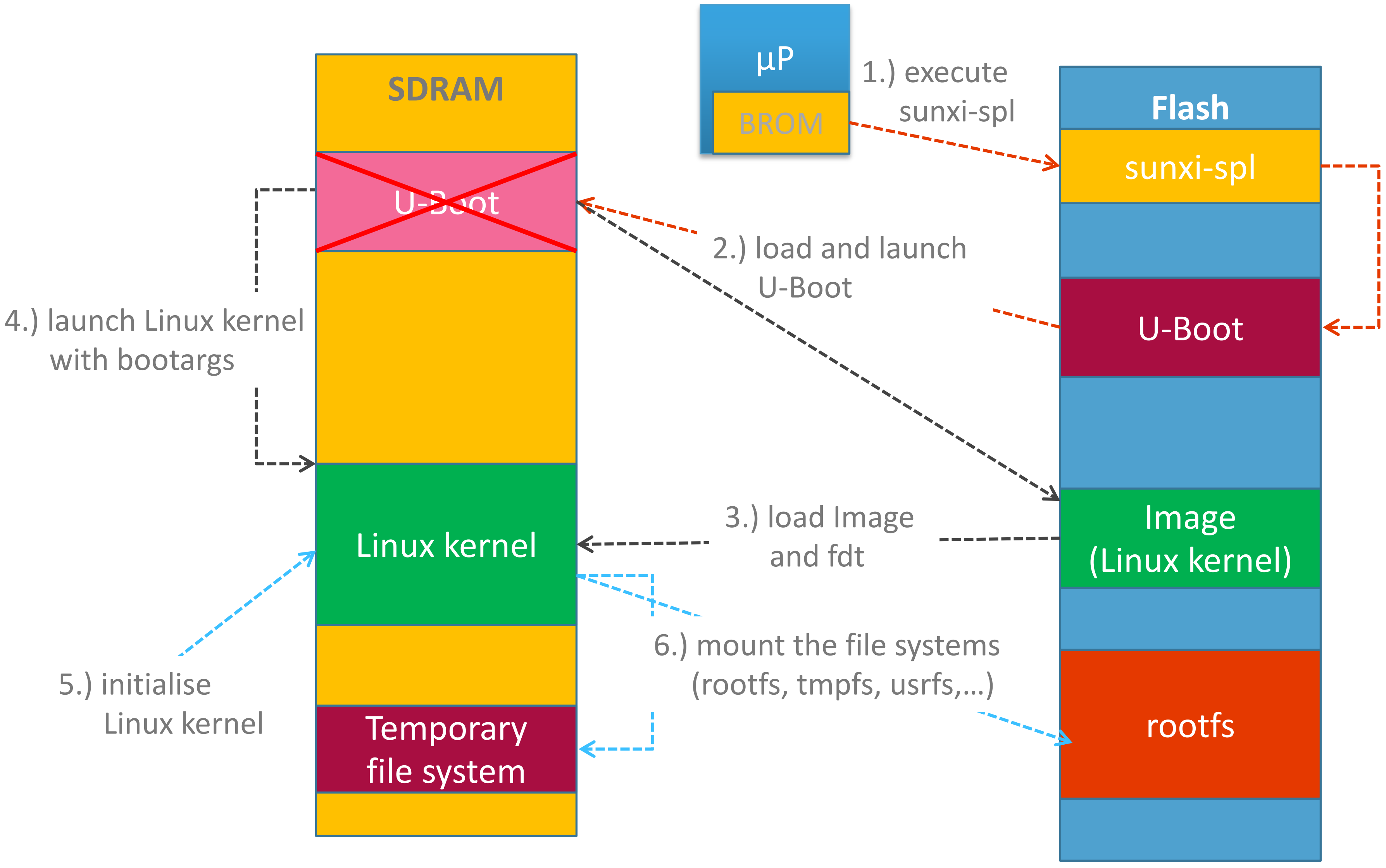
[2]: wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Plus2

[4]: https://github.com/u-boot/u-boot

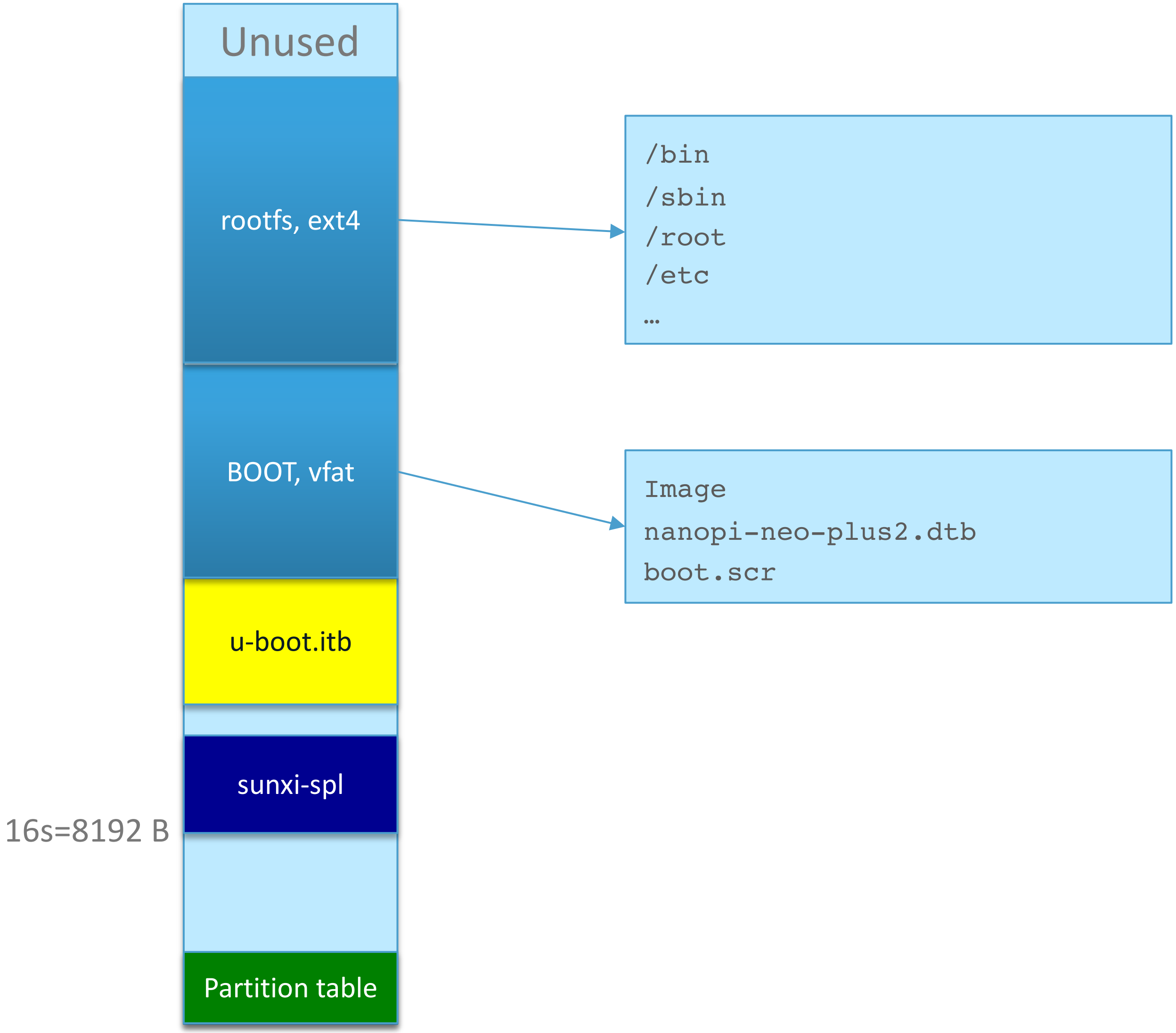
Boot sequence [Cours CSEL, D. Gachet]

- **Le démarrage du NanoPi NEO Plus2 se décompose en 6 phases:**
  - ❑ Lorsque le µP est mis sous tension, le code stocké dans son BROM va charger dans ses 32KiB de SRAM interne le firmware « sunxi-spl » stocké dans le secteur nº 16 de la carte SD / eMMC et l'exécuter.
  - ❑ Le firmware « sunxi-spl » (Secondary Program Loader) initialise les couches basses du µP, puis charge l'U-Boot dans la RAM du µP avant de le lancer.
  - ❑ L'U-Boot va effectuer les initialisations hardware nécessaires (horloges, contrôleurs, …) avant de charger l'image non compressées du noyau Linux dans la RAM, le fichier «Image», ainsi que le fichier de configuration FDT (flattened device tree).
  - ❑ L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (bootargs).
  - ❑ Le noyau Linux procédera à son initialisation sur la base des bootargs et des éléments de configuration contenus dans le fichier FDT (sun50i-h5-nanopi-neo-plus2.dtb).
  - ❑ Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, …) et poursuivra son exécution.

Boot sequence [Cours CSEL, D. Gachet]


µP
BROM
1.) execute sunxi-spl
SDRAM
Flash
sunxi-spl
U-Boot
2.) load and launch U-Boot
U-Boot
4.) launch Linux kernel with bootargs
Linux kernel
3.) load Image and fdt
Image (Linux kernel)
6.) mount the file systems (rootfs, tmpfs, usrfs,…)
5.) initialise Linux kernel
Temporary file system
rootfs

BOOT and rootfs content



```
/bin
/sbin
/root
/etc
…
```

```
Image
nanopi-neo-plus2.dtb
boot.scr
```

U-boot commands

If a key is pressed quickly during NanoPi boot, you can enter to the u-boot mode

```
U-Boot SPL 2019.01 (Aug 29 2019 - 12:49:32 +0200)
DRAM: 1024 MiB
Trying to boot from MMC1


U-Boot 2019.01 (Aug 29 2019 - 12:49:32 +0200) Allwinner Technology

CPU:   Allwinner H5 (SUN50I)
Model: FriendlyARM NanoPi NEO Plus2
DRAM:  1 GiB
MMC:   SUNXI SD/MMC: 0, SUNXI SD/MMC: 1
=>
```

// prompt u-boot

U-boot commands

Type "help" or "?" for a complete listing of available commands.

```
=> ?
 ?        - alias for 'help'
 base     - print or set address offset
 bdinfo   - print Board Info structure
 boot     - boot default, i.e., run 'bootcmd'
 bootd    - boot default, i.e., run 'bootcmd'
 bootefi - Boots an EFI payload from memory
 bootelf - Boot from an ELF image in memory
 booti    - boot arm64 Linux Image image from memory
 bootm    - boot application image from memory
 bootp    - boot image via network using BOOTP/TFTP protocol
 bootvx   - Boot vxWorks from an ELF image

 ext2load- load binary file from a Ext2 filesystem
 ext2ls   - list files in a directory (default /)
 ext4load- load binary file from a Ext4 filesystem
 ext4ls   - list files in a directory (default /)
 ext4size- determine a file's size

 fatinfo - print information about filesystem
 fatload - load binary file from a dos filesystem
 fatls    - list files in a directory (default /)
 fatmkdir- create a directory
 fatrm    - delete a file
 fatsize - determine a file's size
 fatwrite- write file into a dos filesystem
```

U-boot commands

```
…
md      - memory display
mdio    - MDIO utility commands
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
mw      - memory write (fill)
nfs     - boot image via network using NFS protocol

ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables

run     - run commands in an environment variable
save    - save file to a filesystem

Examples:
ext2ls mmc 0:1   // show SDCard 1st partition
ext2ls mmc 0:2   // show SDCard 2nd partition
fatls mmc 1:1    // show eMMC 1st partition
```

Load kernel

The following U-Boot commands load the Linux kernel, Image file, the FDT (Flattened Device Tree) and start Linux

```
=> run bootcmd                  // load and start Image
```

This command searches and executes the boot.scr file in the 1st partition.

Create boot.scr file:

```
cd ~/workspace/nano/buildroot
mkimage -C none -A arm64 -T script -d board/friendlyarm/nanopi-neo-plus2/boot.cmd /home/schuler/workspace/nano/buildroot/output/images/boot.scr
```

Load kernel

Show boot.cmd file: `cat boot.cmd`

```
setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rootwait
fatload mmc 0 $kernel_addr_r Image
fatload mmc 0 $fdt_addr_r nanopi-neo-plus2.dtb
booti $kernel_addr_r - $fdt_addr_r
```

Load Image

Load FDT

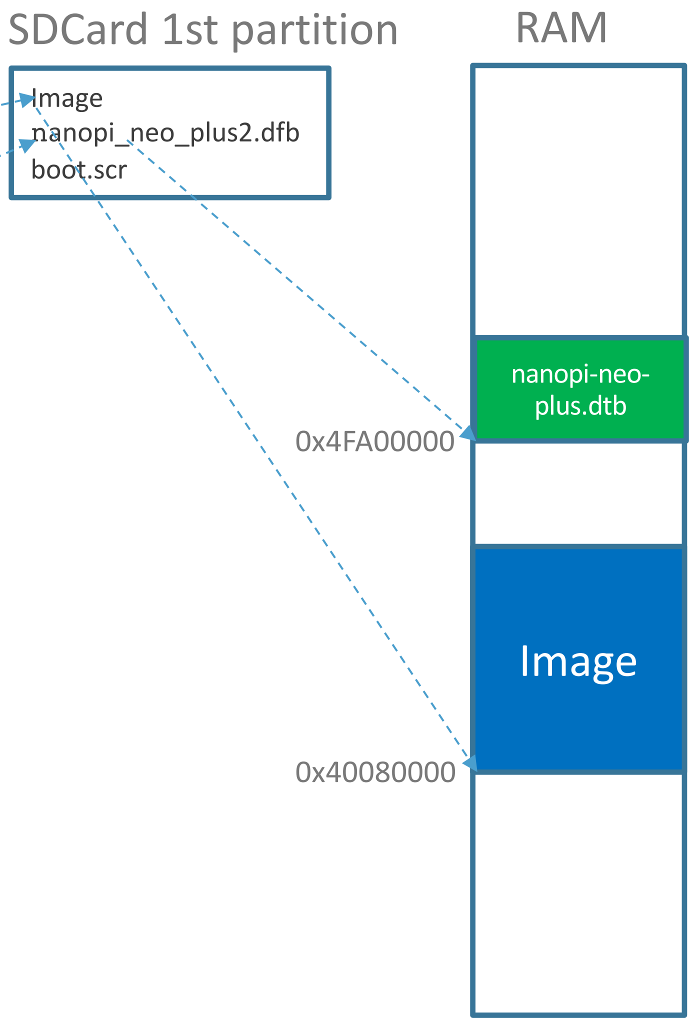
Start Linux

Linux kernel boot parameters

mmc 0: SDCard 1st partition (mmc 0 = mmc 0:1)

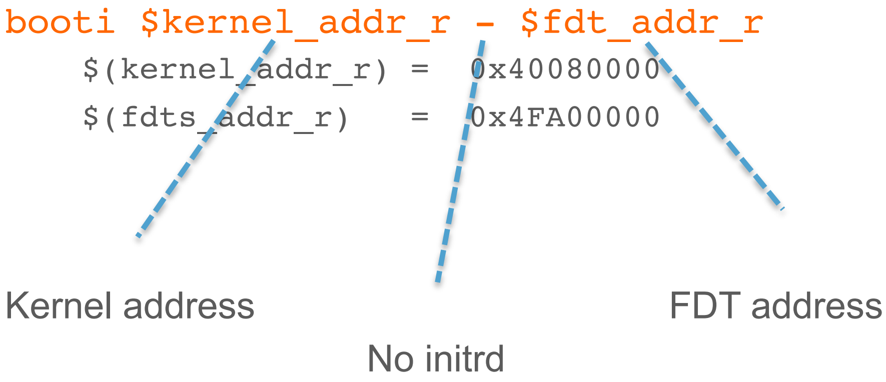
Load kernel

```
fatload mmc 0 $kernel_addr_r Image
fatload mmc 0 $fdt_addr_r nanopi-neo-plus2.dtb

fatls mmc 0
   30210560 Image
   20426    nanopi-neo-plus2.dtb
   271      boot.scr

printenv kernel_addr_r
   kernel_addr_r=0x40080000

printenv fdt_addr_r
   fdt_addr_r=0x4FA00000
```



Load kernel: start Linux

This command starts the Linux kernel



```
booti $kernel_addr_r - $fdt_addr_r
    $(kernel_addr_r) =  0x40080000
    $(fdts_addr_r)   =  0x4FA00000
```

Kernel address

No initrd

FDT address

```
help booti
booti - boot arm64 Linux Image image from memory

Usage:
booti [addr [initrd[:size]] [fdt]]
    - boot arm64 Linux Image stored in memory
        The argument 'initrd' is optional and specifies the address
        of an initrd in memory. The optional parameter ':size' allows
        specifying the size of a RAW initrd.
        Since booting a Linux kernel requires a flat device-tree, a
        third argument providing the address of the device-tree blob
        is required. To boot a kernel with a device-tree blob but
        without an initrd image, use a '-' for the initrd argument.
```

U-boot configuration

U-boot configuration looks like Linux kernel configuration

Configure:

```
cd ~/workspace/nano/buildroot
make uboot-menuconfig
```

Compile: 2 possibilities:

```
1)
cd ~/workspace/nano/buildroot
make uboot-rebuild

2)
cd ~/workspace/nano/buildroot/
rm output/build/uboot-2020.07/.stamp-built
make
```

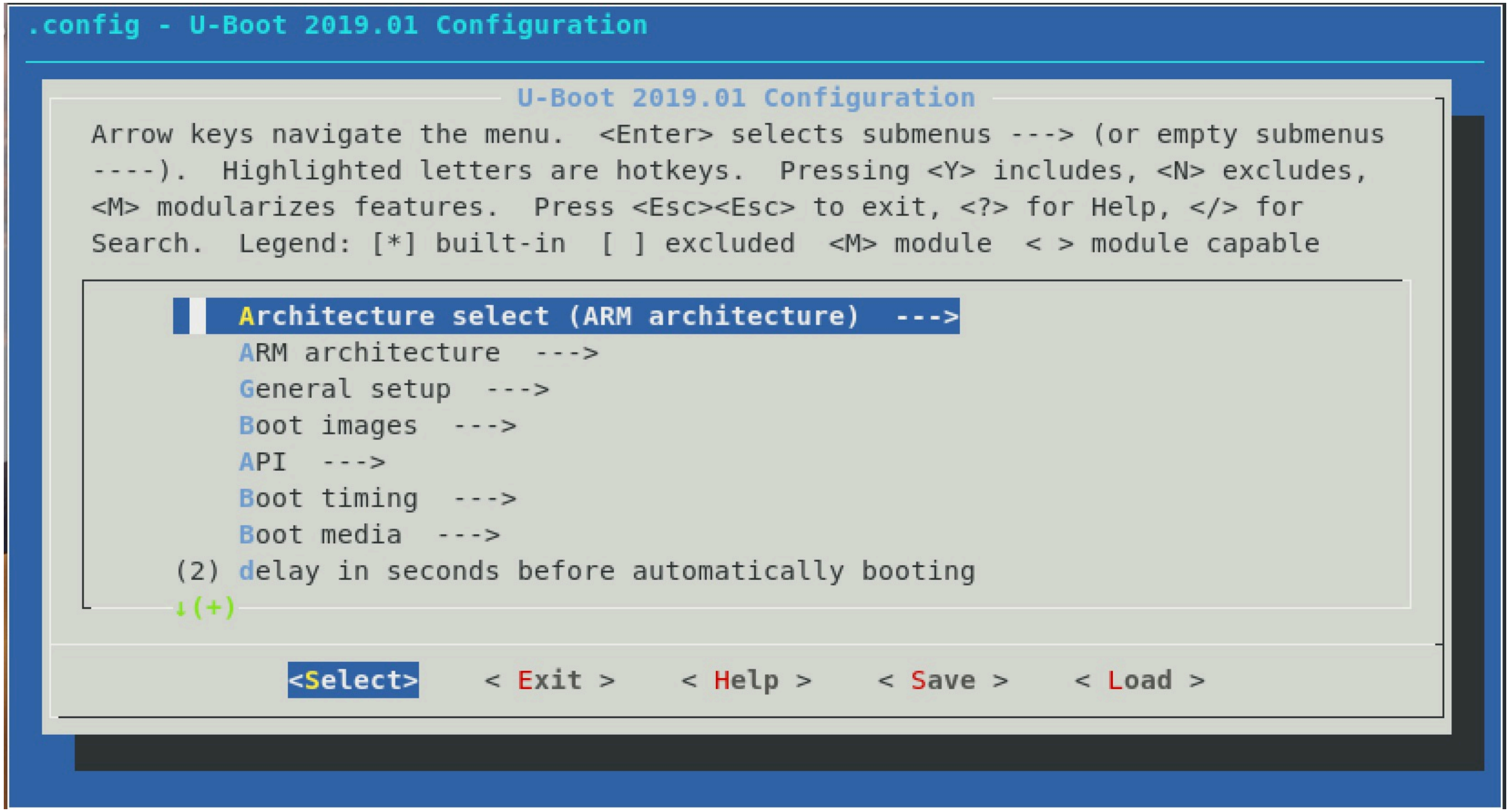
After the make command, two files are created:

```
 ~/workspace/nano/buildroot/output/images/u-boot.itb
 ~/workspace/nano/buildroot/output/images/boot.scr
```

U-boot configuration

```
cd ~/workspace/nano/buildroot
make uboot-menuconfig
```

The configuration file is saved to the ~/workspace/nano/buildroot/output/build/uboot-2020.07/.config file



```
.config - U-Boot 2019.01 Configuration

                    U-Boot 2019.01 Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for
  Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

           Architecture select (ARM architecture)  --->
           ARM architecture  --->
           General setup  --->
           Boot images  --->
           API  --->
           Boot timing  --->
           Boot media  --->
       (2) delay in seconds before automatically booting
       ↓(+)

         <Select>    < Exit >    < Help >    < Save >    < Load >
```

U-boot configuration

It is possible to show the .config file. Don’t modify this file with an editor

```
less .config
      #
      # Automatically generated file; DO NOT EDIT.
      # U-Boot 2019.01 Configuration
      #
      CONFIG_CREATE_ARCH_SYMLINK=y
      # CONFIG_ARC is not set
      CONFIG_ARM=y
      …
      # CONFIG_SH is not set
      # CONFIG_X86 is not set
      # CONFIG_XTENSA is not set
      CONFIG_SYS_ARCH="arm"
      CONFIG_SYS_CPU="armv8"
      …
```

Directories installation [2]

~/workspace	→ working space
/nano	→ working space for NanoPi
/buildroot	→ space for tools, kernel, rootfs generation
/board/friendlyarm/nanopi-neo-plus2	→ genimage.cfg, boot.cmd, sunxi-spl.bin
/dl	→ downloaded « tared » packets: e.g. busybox-1.30.1.tar.bz2
/system/skeleton	→ Rootfs skeleton
/output	
/build	→ source codes and compiled packets, e.g.: linux-5.1.16
/images	→ Image, nanopi-neo-plus2.dtb, rootfs.ext4, u-boot.itb, boot.scr, sunxi-spl.bin
/target	→ rootfs not “tared”
/host/usr/bin	→ cross-compiler: aarch64-linux-gnu-gcc, …

Files u-boot.itb, sunxi-spl.bin, Image, sun50i-h5-nanopi-neo-plus2.dtb, rootfs.ext4, boot.scr will be copied to the uSD card.

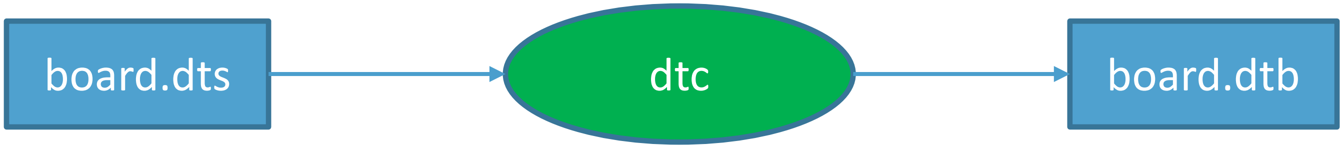
In order to cross-compile, link, … , add the
PATH=$PATH:~/workspace/nano/buildroot/output/host/usr/bin

FDT and FIT

- The Flattened Device-Tree (FDT) was introduced in kernel 2.6. It is a file which contains the hardware description. Linux uses it for its configuration

- FDT has two files:
  - .dts: Device Tree Source, it is an ascii file
  - .dtb: Device Tree Blob, it is a binary file

- `dtc` command transforms `.dts` file to `.dtb` file: `dtc board.dts –o board.dtb`



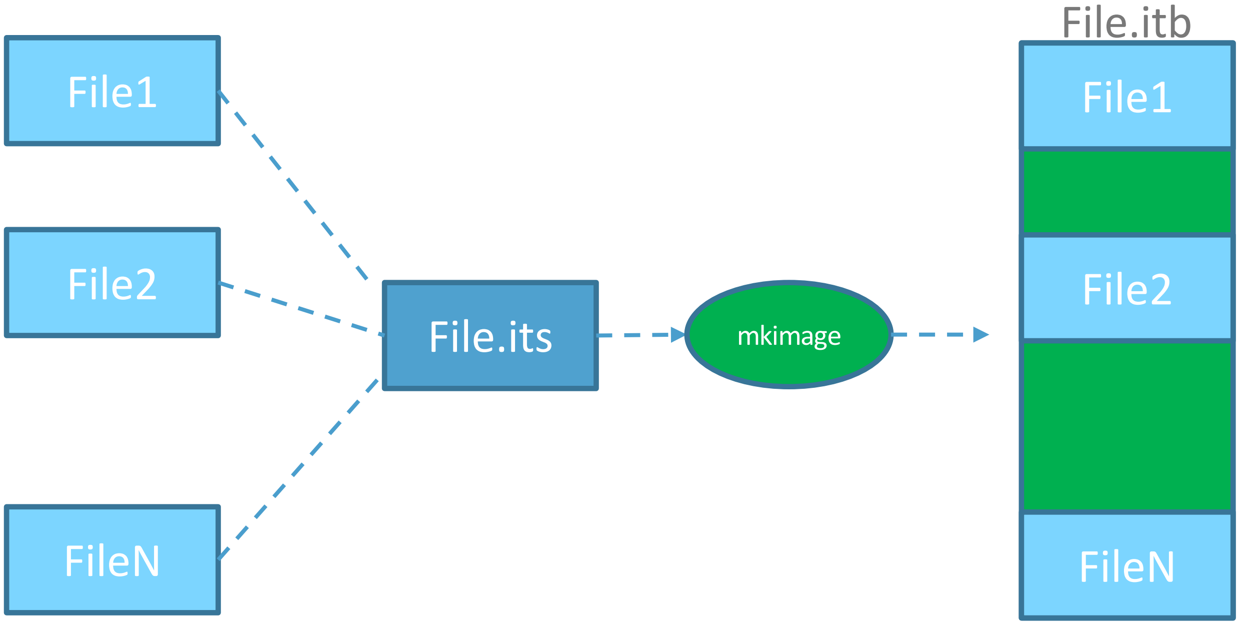
FDT and FIT

- U-boot uses a FTD file in order to describe the hardware.
- The `sun50i-h5-nanopi-neo-plus2.dts` file describes the hardware used by u-boot for the NanoPi (workspace/nano/buildroot/output/build/uboot-2020.07/arch/arm/dts/)

```
cat sun50i-h5-nanopi-neo-plus2.dts
   /dts-v1/;
   #include "sun50i-h5.dtsi"
   #include <dt-bindings/gpio/gpio.h>
   #include <dt-bindings/input/input.h>
   #include <dt-bindings/pinctrl/sun4i-a10.h>
   / {
           model = "FriendlyARM NanoPi NEO Plus2";
           compatible = "friendlyarm,nanopi-neo-plus2", "allwinner,sun50i-h5";

           aliases {
                   ethernet0 = &emac;
                   serial0 = &uart0;
           };

           chosen {
                   stdout-path = "serial0:115200n8";
           };
           …
```

FDT and FIT

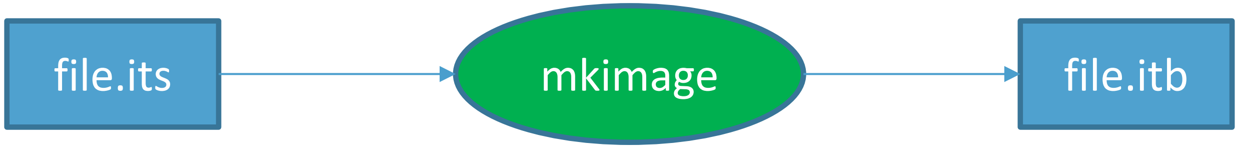
- After the introduction of FTD with the kernel 2.6, a new binary file format was created: FIT (Flattened Image Tree). This format allows to insert different files into a single file
- FIT has two files:
  - its: image source file, it is a text file
  - itb: Image Tree Blob, it is a binary file
- .its file describes which files will be inserted to the .itb file



FDT and FIT

- FIT format allows more flexibility in handling images of various types and also enhances integrity protection of images with hash functions like rsa signature, sha256, sha1, md5, crc32, …
- The mkimage command reads a .its file and creates a .itb file:

```
mkimage  -f file.its -E file.itb
```



U-boot and FIT

- During startup, the Secondary Program Loader (sunxi-spl) loads the `u-boot.itb` file.

- U-boot.itb is built with these commands:

```
cd workspace/nano/buildroot/output/build/uboot-2020.07
mkimage  -f u-boot.its -E u-boot.itb
```

U-boot and FIT

Show u-boot.its file

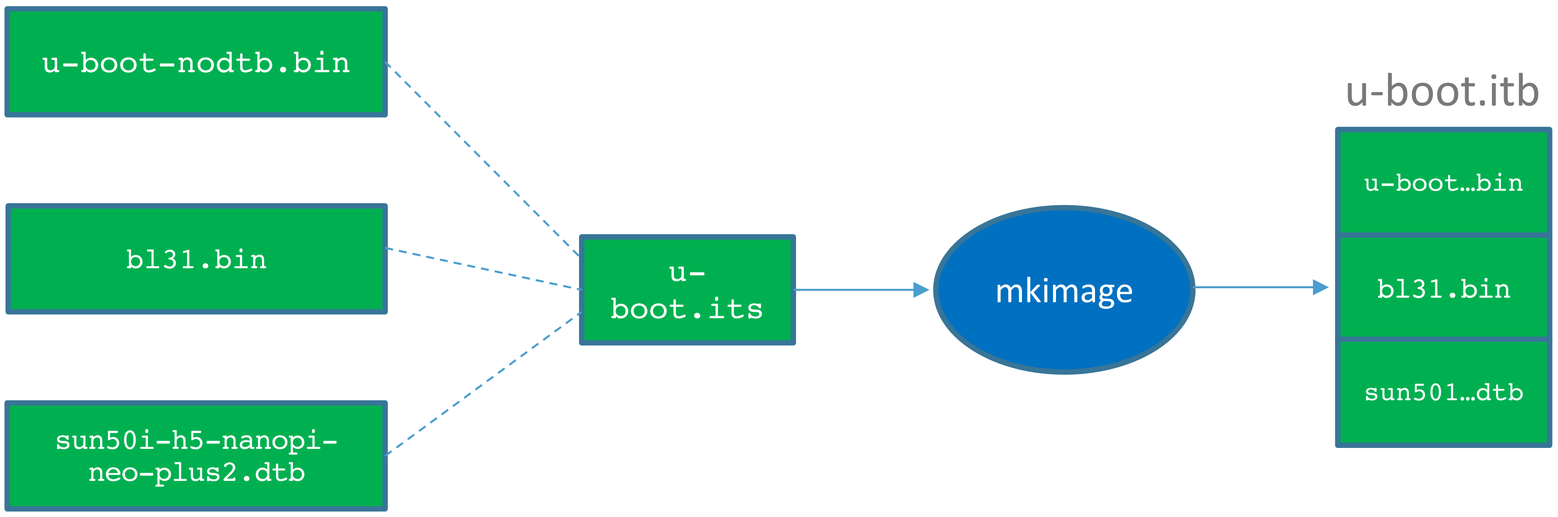
```
cat u-boot.its
  /dts-v1/;
  / {
     description = "Configuration to load ATF before U-Boot";
     #address-cells = <1>;
     images {
           uboot {
              description = "U-Boot (64-bit)";
              data = /incbin/("u-boot-nodtb.bin");
              type = "standalone";
              arch = "arm64";
              compression = "none";
              load = <0x4a000000>;
           };
           atf {
              description = "ARM Trusted Firmware";
              data =
  /incbin/("/home/schuler/workspace/nano/buildroot/output/images/bl31.bin");
              type = "firmware";
              arch = "arm64";
              compression = "none";
              load = <0x44000>;
              entry = <0x44000>;
           };
```

U-boot and FIT

```
            fdt_1 {
               description = "sun50i-h5-nanopi-neo-plus2";
               data = /incbin/("arch/arm/dts/sun50i-h5-nanopi-neo-plus2.dtb");
               type = "flat_dt";
               compression = "none";
            }
      };
      configurations {
            default = "config_1";

            config_1 {
               description = "sun50i-h5-nanopi-neo-plus2";
               firmware = "uboot";
               loadables = "atf";
               fdt = "fdt_1";
            };
}
```

FDT and FIT

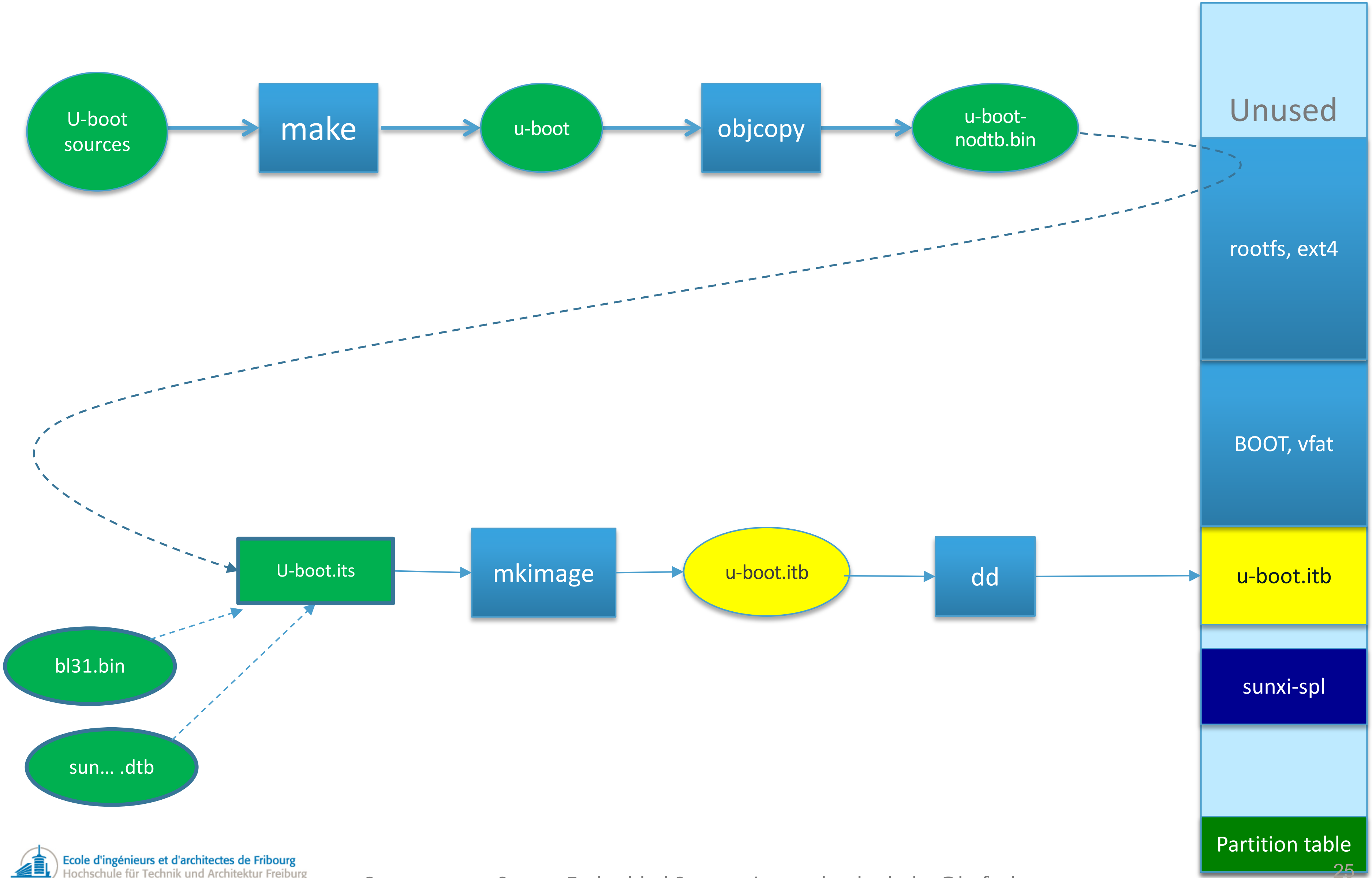
- mkimage reads u-boot.its and builds u-boot.itb.
- u-boot.itb contains
  - u-boot-nodtb.bin: u-boot code
  - bl31.bin: trust zone
  - sun501-h5 … .dtb: Flattened device tree (device Tree Blob)



Where are these files:

- workspace/nano/buildroot/output/build/u-boot.2020.07/u-boot.its
- workspace/nano/buildroot/output/build/u-boot.2020.07/u-boot-nodtb.bin
- workspace/nano/buildroot/output/build/output/image/bl31.bin
- workspace/nano/buildroot/output/build/u-boot.2020.07/arch/arm/dts/sun50i-h5-nanopi-neo-plus2.dtb

Create u-boot.itb image



Create u-boot.itb image

During the make, `u-boot-nodtb.bin` image is created, by the `arm-linux-gnueabihf-objcopy` command

```
aarch64-linux-gnu-objcopy --gap-fill=0xff -O binary u-boot u-boot-nodtb.bin
```

u-boot is created by the linker and has the ELF format

u-boot-nodtb.bin is a raw file with only the loadable sections of the `u-boot` file, other sections (symbols, debug, relocation, …) are not copied to u-boot-nodtb.bin

- `aarch64-linux-gnu-objcopy` generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file.
- All symbols and relocation information will be discarded. The memory dump will start at the load address of the lowest section copied into the output file.

u-boot compilation options

`Important: cd ~/workspace/nano/buildroot/output/build/u-boot-2020.07`

In order to change compilation options, it is necessary to change config.mk or directly Makefile

config.mk:

- PLATFORM_CPPFLAGS, LDFLAGS_FINAL

Makefile:

- KBUILD_CLAGS

These flags modify compilation options

Less Makefile

```
cpp_flags := $(KBUILD_CPPFLAGS) $(PLATFORM_CPPFLAGS) $(UBOOTINCLUDE) \
                              $(NOSTDINC_FLAGS)
c_flags := $(KBUILD_CFLAGS) $(cpp_flags)
```

U-boot-2020.07/README: make CFLAGS=-Wall ---

Analyze the u-boot compilation

By default during the make, the followed gcc options are used

Compilation command: `cd ~/workspace/nano/buildroot`

```
make uboot-rebuild V=1
```

```
aarch64-linux-gnu-gcc -Wp,-MD,arch/arm/cpu/armv8/.fwcall.o.d  -nostdinc -isystem
/home/schuler/workspace/nano/buildroot/output/host/opt/ext-toolchain/bin/../lib/gcc/aarch64-
linux-gnu/8.2.1/include -Iinclude   -I./arch/arm/include -include ./include/linux/kconfig.h -
D__KERNEL__ -D__UBOOT__ -Wall -Wstrict-prototypes -Wno-format-security -fno-builtin -
ffreestanding -std=gnu11 -fshort-wchar -fno-strict-aliasing -fno-PIE -Os -fno-stack-protector
-fno-delete-null-pointer-checks -fmacro-prefix-map=./= -g -fstack-usage -Wno-format-nonliteral
-Werror=date-time -D__ARM__ -fno-pic -mstrict-align -ffunction-sections -fdata-sections -fno-
common -ffixed-r9 -fno-common -ffixed-x18 -pipe -march=armv8-a -D__LINUX_ARM_ARCH__=8 -
I./arch/arm/mach-sunxi/include    -D"KBUILD_STR(s)=#s" -D"KBUILD_BASENAME=KBUILD_STR(fwcall)"
-D"KBUILD_MODNAME=KBUILD_STR(fwcall)" -c -o arch/arm/cpu/armv8/fwcall.o
arch/arm/cpu/armv8/fwcall.c
```

```
-nostdinc: don’t use the standard include directories, use only the –I dirctories
-Os: optimized for size
-fno-stack-protector: no protection against stack smashing
-fno-delete-null-pointer-checks: don’t check null pointer
-g: debug option
-fstack-usage
-fno-common: check multiple-definition of global variables
-Dxxx: same as #define xxx
-I: include directories
```

Improve the u-boot compilation

In order to improve the u-boot code security, it is possible to add these options:

- suppress the –g option: delete the debug information
- Add the -fstack-protector-all option

–fstack-protector-all option adds extra code to check buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions. This variable is called Canary

Example: file.c has a buffer overflow error, the –fstack-protector-all detects that

```
unsigned char localBufferOverflow (void) {
unsigned char val   [16];
int           i;
    for (i=0; i< 18; i++)
        val[i] = 0;
    return val[0];
}
```

```
# gcc -Wall -fstack-protector-all -o file file.C
#./file
*** stack smashing detected ***: ./file terminated
======= Backtrace: =========
/lib/libc.so.6(__fortify_fail+0x45)[0x4de0eb85]
/lib/libc.so.6[0x4de0eb3a]
```

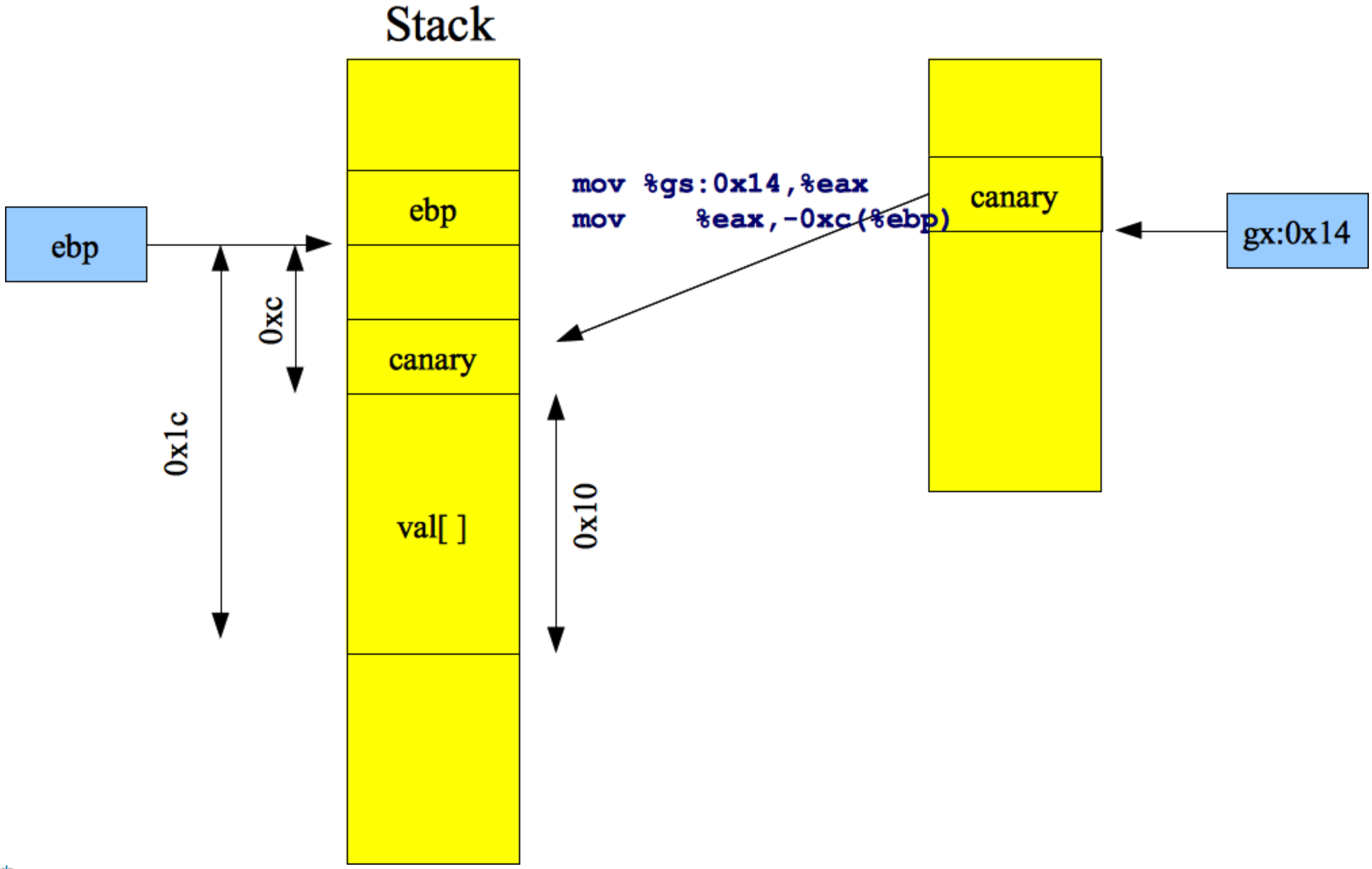
-Fstack-protection-all gcc option, Intel uP

Principe: localBufferOverflow assembler code (Intel code)

```
08048453 <_ZL19localBufferOverflowv>:
 8048453:       55                      push   %ebp
 8048454:       89 e5                   mov    %esp,%ebp
 8048456:       83 ec 28                sub    $0x28,%esp
 8048459:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 804845f:       89 45 f4                mov    %eax,-0xc(%ebp)
 8048462:       31 c0                   xor    %eax,%eax
 ....
 804848b:       8b 55 f4                mov    -0xc(%ebp),%edx
 804848e:       65 33 15 14 00 00 00    xor    %gs:0x14,%edx
 8048495:       74 05                   je     804849c
 8048497:       e8 54 fe ff ff          call   80482f0 <__stack_chk_fail@plt>
 804849c:       c9                      leave
```

If the edx register is not zero, the function __stack_chk_fail@plt is called

-Fstack-protection-all gcc option, Intel uP



```
mov %gs:0x14,%eax
mov     %eax,-0xc(%ebp)
```

-Fstack-protection-all gcc option, ARM uP

```
08048453 <testCanary>:
        1049c:      e58de004        str     lr, [sp, #4]
        104a0:      e28db004        add     fp, sp, #4
        104a4:      e24dd010        sub     sp, sp, #16
        104a8:      e3003f08        movw    r3, #3848       ; 0xf08
        104ac:      e3403002        movt    r3, #2
        104b0:      e5933000        ldr     r3, [r3]
        104b4:      e50b3008        str     r3, [fp, #-8]
        104b8:      e3a03000
        …
        104f8:      e3003f08        movw    r3, #3848       ; 0xf08
        104fc:      e3403002        movt    r3, #2
        10500:      e51b2008        ldr     r2, [fp, #-8]
        10504:      e5933000        ldr     r3, [r3]
        10508:      e1520003        cmp     r2, r3
        1050c:      0a000000        beq     10514 <testCanary+0x7c>
        10510:      ebffff8e        bl      10350 <__stack_chk_fail@plt>
        10514:      e24bd004        sub     sp, fp, #4
        10518:      e59db000        ldr     fp, [sp]
        1051c:      e28dd004        add     sp, sp, #4
        10520:      e49df004        pop     {pc}            ; (ldr pc, [sp],
```

If r2 != r3 --> __stack_chk_fail function is called

U-boot: debug and symbols

In order to improve the security of an ELF executable file, it is necessary to remove debug and symbols information. The strip command deletes this information.

```
aarch64-linux-gnu-strip u-boot
```

Diassemble with symbols

```
arm-linux-gnueabihf-objdump –d u-boot
   43e00058 <reset>:
   43e00058:       eb0004f8        bl      43e01440 <save_boot_params_default>
   43e0005c:       e10f0000        mrs     r0, CPSR
   43e00060:       e3c0001f        bic     r0, r0, #31
   43e00064:       e38000d3        orr     r0, r0, #211    ; 0xd3
   43e00068:       e129f000        msr     CPSR_fc, r0
```

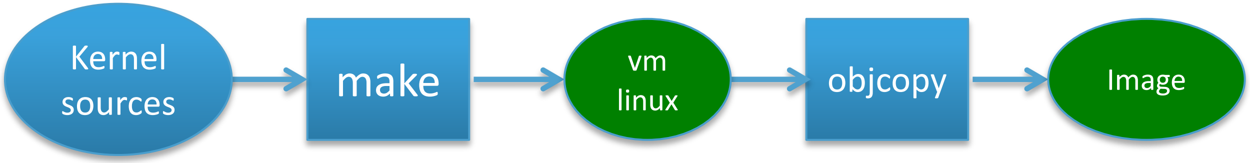
Diassemble without symbols

```
arm-linux-gnueabihf-objdump –d u-boot
   43e00058:       eb0004f8        .word   0xeb0004f8
   43e0005c:       e10f0000        .word   0xe10f0000
   43e00060:       e3c0001f        .word   0xe3c0001f
   43e00064:       e38000d3        .word   0xe38000d3
   43e00068:       e129f000        .word   0xe129f000
```

The command below automatically removes debug and symbols information

```
aarch64-linux-gnu-objcopy --gap-fill=0xff -O binary u-boot u-boot-nodtb.bin
```

Different formats of Linux kernel



- vmlinux: Linux kernel not stripped, elf format
- Image: Linux kernel stripped without some sections (.note, .comment, …)

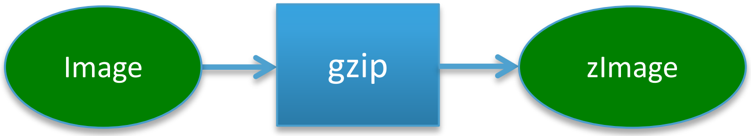
Where are these files:

- `Kernel sources: workspace/nano/buildroot/output/build/linux-xx`
- `vmlinux: workspace/nano/buildroot/output/build/linux-xx/vmlinux`
- `Image: workspace/nano/buildroot/output/build/linux-xx/arch/arm64/boot/Image`

Different formats of Linux kernel

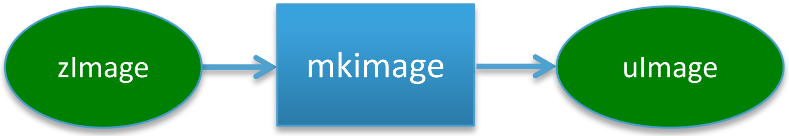
zImage: Compressed Linux

- `cd workspace/nano/buildroot/output/build/linux-xx`
- `cat arch/arm64/boot/Image | gzip -n -f -9 > arch/arm64/boot/zImage`



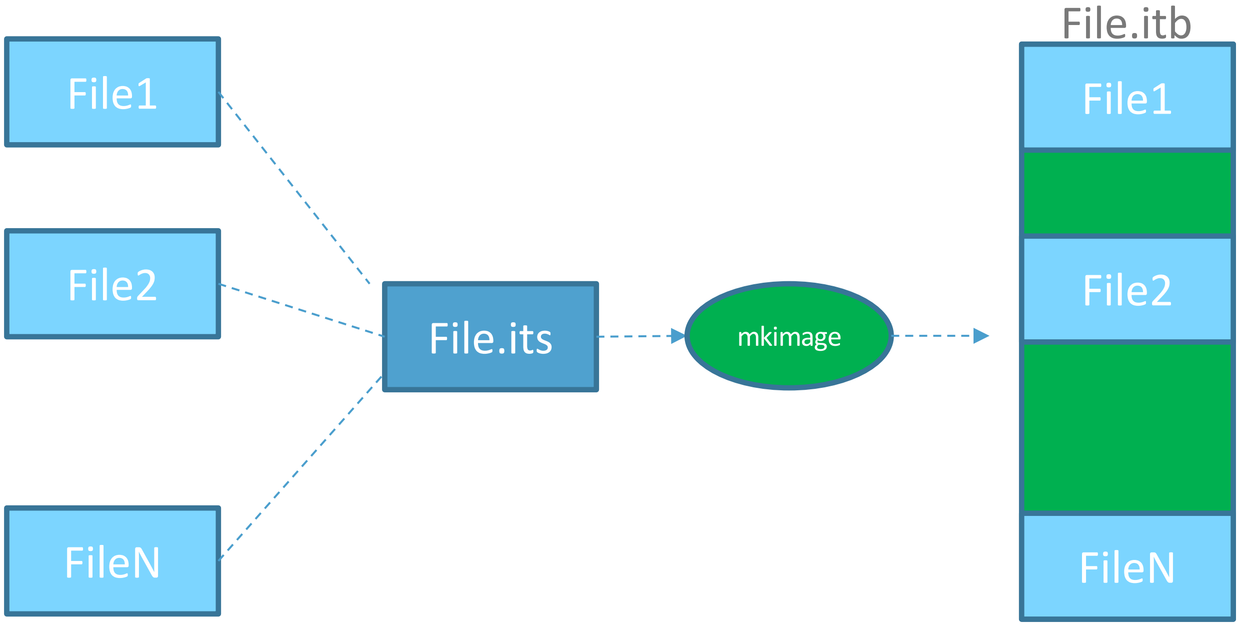
uImage: Compressed Linux with a u-boot header

- `cd workspace/nano/buildroot/output/build/linux-xx`
- `mkimage -A arm64 -O linux -C none  -T kernel -a 0x40080000 -e 0x40080000 -n 'Linux-XX' -d arch/arm64/boot/zImage arch/arm64/boot/uImage`



Different formats of Linux kernel

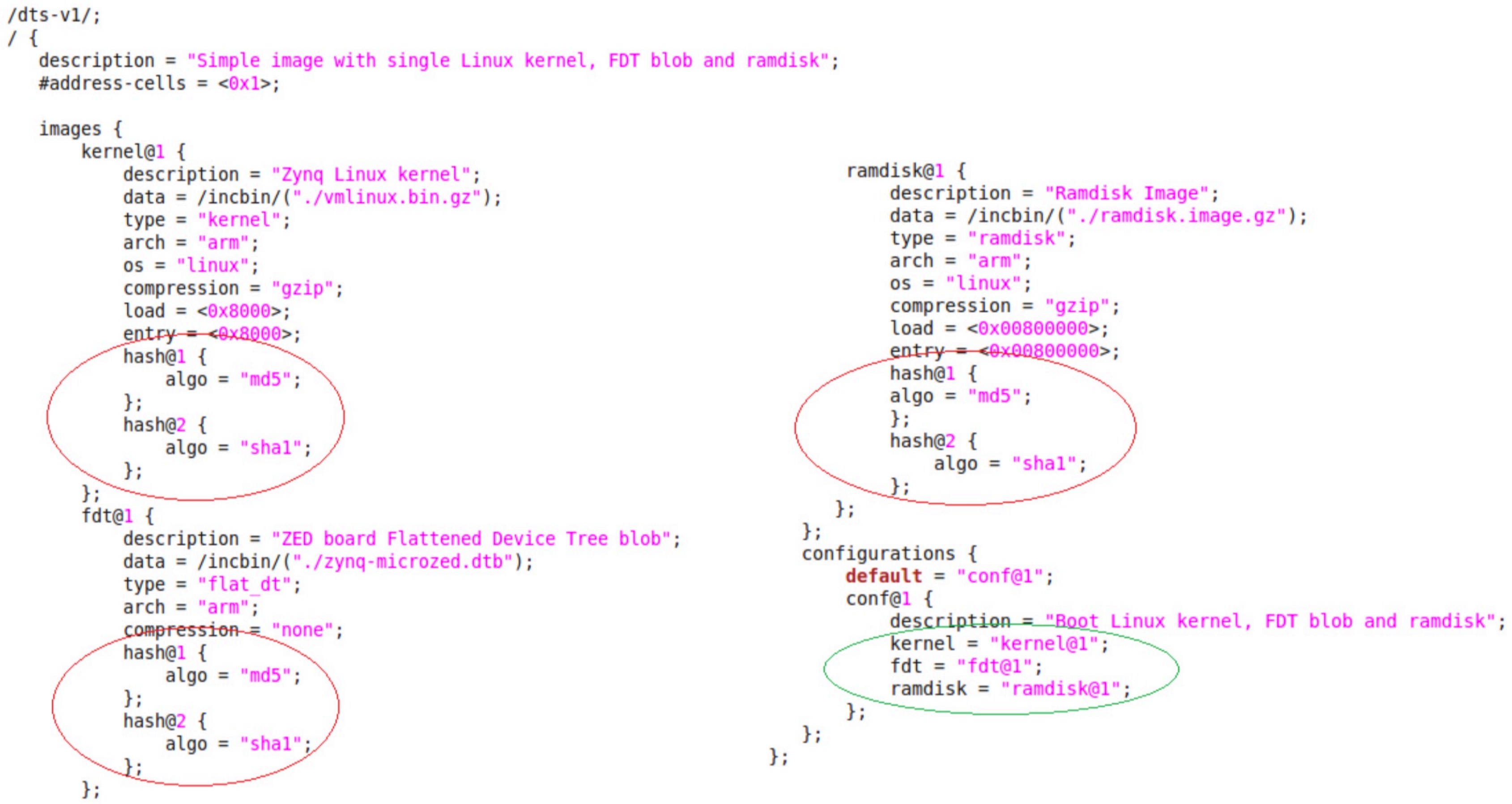
FIT: It is the new format supported by u-boot. A .its file describes information (the structure, the check, ..) about different files inserted into the .itb file



Different formats of Linux kernel

Example: FIT configuration. During linux startup, u-boot can check the integrity of the different files

Source: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images



```
/dts-v1/;
/ {
   description = "Simple image with single Linux kernel, FDT blob and ramdisk";
   #address-cells = <0x1>;

   images {
       kernel@1 {
           description = "Zynq Linux kernel";
           data = /incbin/("./vmlinux.bin.gz");
           type = "kernel";
           arch = "arm";
           os = "linux";
           compression = "gzip";
           load = <0x8000>;
           entry = <0x8000>;
           hash@1 {
               algo = "md5";
           };
           hash@2 {
               algo = "sha1";
           };
       };
       fdt@1 {
           description = "ZED board Flattened Device Tree blob";
           data = /incbin/("./zynq-microzed.dtb");
           type = "flat_dt";
           arch = "arm";
           compression = "none";
           hash@1 {
               algo = "md5";
           };
           hash@2 {
               algo = "sha1";
           };
       };
       ramdisk@1 {
           description = "Ramdisk Image";
           data = /incbin/("./ramdisk.image.gz");
           type = "ramdisk";
           arch = "arm";
           os = "linux";
           compression = "gzip";
           load = <0x00800000>;
           entry = <0x00800000>;
           hash@1 {
           algo = "md5";
           };
           hash@2 {
               algo = "sha1";
           };
       };
   };
   configurations {
       default = "conf@1";
       conf@1 {
           description = "Boot Linux kernel, FDT blob and ramdisk";
           kernel = "kernel@1";
           fdt = "fdt@1";
           ramdisk = "ramdisk@1";
       };
   };
};
```

U-boot loads Linux kernel

U-boot has different commands in order to load the Linux kernel

- booti loads an Image format file
- bootz loads an zImage format file
- bootm loads an uImage format file
- bootm load a fit format file

Example:

```
setenv bootargs console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rootwait
fatload mmc 0 0x40080000 Image
fatload mmc 0 0x4fa00000 sun50i-h5-nanopi-neo-plus2.dtb
booti 0x40080000 – 0x4fa00000
```