>>> Operating Systems And Applications For Embedded Systems
>>> Bootloaders

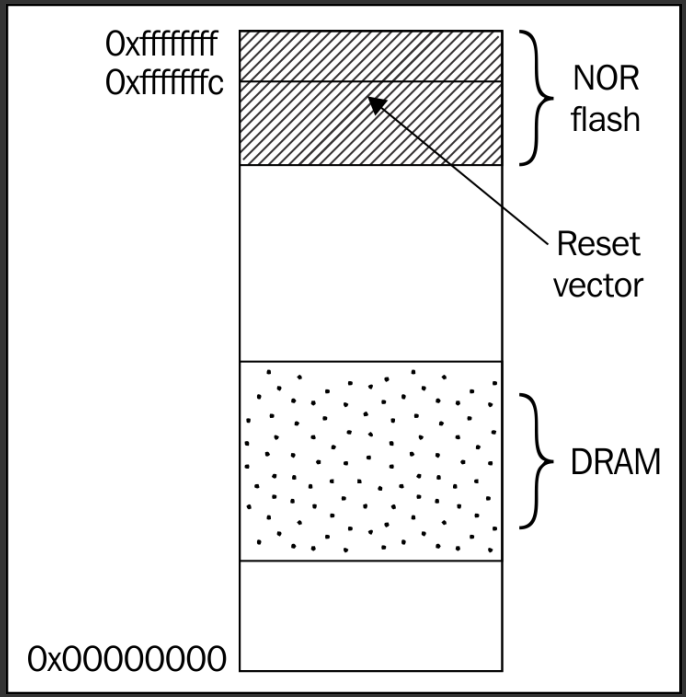
Name: Mariusz Naumowicz
Date: 27 sierpnia 2018

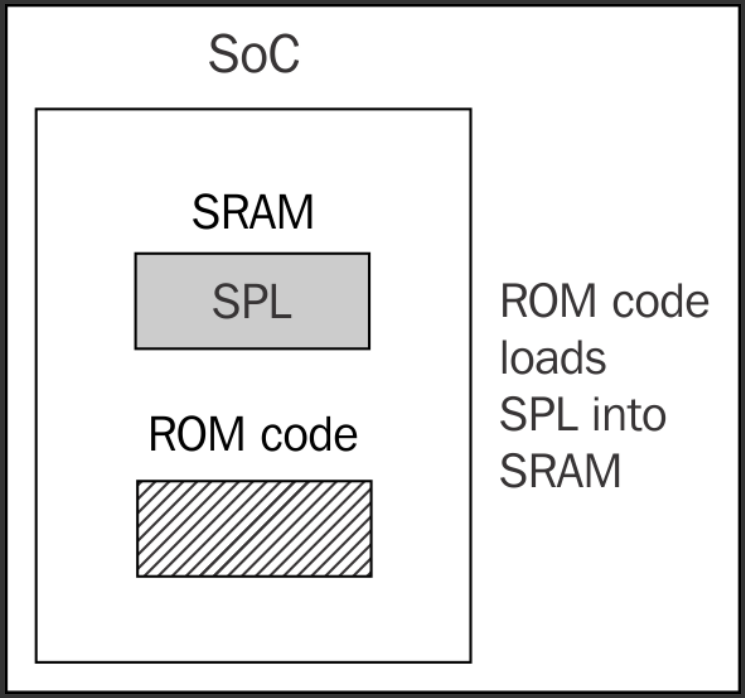
>>> Plan

1. The boot sequence
   - Phase 1: ROM code
   - Phase 2: SPL
   - Phase 3: TPL
   - UEFI firmware
   - Choosing a bootloader

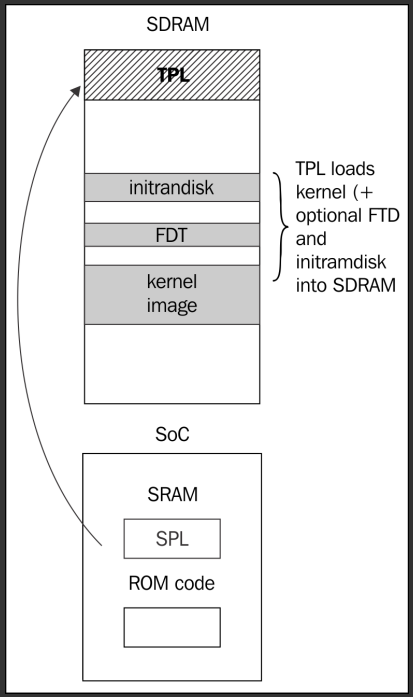
2. Wyniki
   - U-Boot
   - Building U-Boot
   - Installing U-Boot
   - Using U-Boot
   - Boot image format
   - Loading images
   - Booting Linux

>>> Phase 1: ROM code




SoC
SRAM
SPL
ROM code
ROM code loads SPL into SRAM

>>> Phase 3: TPL


SDRAM
TPL
initrandisk
FDT
kernel
image
TPL loads
kernel (+
optional FTD
and
initramdisk
into SDRAM
SoC
SRAM
SPL
ROM code

>>> UEFI firmware

Most embedded PC designs and some ARM designs have firmware based on the Universal Extensible Firmware Interface (UEFI) standard, see the offcial website at http://www.uefi.org for more information. The boot sequence is fundamentally the same as described in the preceding section:

* Phase 1: The processor loads the UEFI boot manager firmware from flash memory. In some designs, it is loaded directly from NOR flash memory, in others there is ROM code on-chip which loads the boot manager from SPI flash memory
* Phase 2: The boot manager loads the boot firmware from the EFI System Partition (ESP) or a hard disk or SSD, or from a network server via PXE boot.
* Phase 3: The TPL in this case has to be a bootloader that is capable of loading a Linux kernel and an optional RAM disk into memory. Common choices are:
  * GRUB 2: This is the GNU Grand Unified Bootloader, version 2, and it is the most commonly used Linux loader on PC platforms. However, there is one controversy in that it is licensed under GPL v3, which may make it incompatible with secure booting since the license requires the boot keys to to be supplied with the code. The website is https://www.gnu.org/ software/grub/.
  * gummiboot: This is a simple UEFI-compatible bootloader which has since been integrated into systemd, and is licensed under LGPL v2.1 The website is https://wiki.archlinux.org/index.php/Systemd-boot.

>>> Choosing a bootloader

Name	Architectures
Das U-Boot	ARM, Blackfin, MIPS, PowerPC, SH
Barebox	ARM, Blackfin, MIPS, PowerPC
GRUB 2	X86, X86_64
RedBoot	ARM, MIPS, PowerPC, SH
CFE	Broadcom MIPS
YAMON	MIPS

>>> U-Boot

U-Boot, or to give its full name, Das U-Boot, began life as an open source bootloader for embedded PowerPC boards. Then, it was ported to ARM-based boards and later to other architectures, including MIPS, SH, and x86. It is hosted and maintained by Denx Software Engineering. There is plenty of information available, and a good place to start is www.denx.de/wiki/U-Boot. There is also a mailing list at u-boot@lists.denx.de.

>>> Building U-Boot

```
git clone git://git.denx.de/u-boot.git
cd u-boot
make CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf- am335x_boneblack_defconfig
make CROSS_COMPILE=arm-cortex_a8-linux-gnueabihf-
```

The results of the compilation are:

* u-boot: This is U-Boot in ELF object format, suitable for use with a debugger
* u-boot.map: This is the symbol table
* u-boot.bin: This is U-Boot in raw binary format, suitable for running on your device
* u-boot.img: This is u-boot.bin with a U-Boot header added, suitable for uploading to a running copy of U-Boot
* u-boot.srec: This is U-Boot in Motorola srec format, suitable for transferring over a serial connection

>>> Installing U-Boot

```
sudo sfdisk -D -H 255 -S 63 /dev/mmcblk0 « EOF
,9,0x0C,*
",-
EOF
sudo mkfs.vfat -F 16 -n boot /dev/mmcblk0p1
cp MLO u-boot.img /media/chris/boot
gtkterm -p /dev/ttyUSB0 -s 115200
U-Boot#
```

>>> Using U-Boot

Usually, U-Boot offers a command-line interface over a serial port. It gives a command prompt which is customized for each board. In the examples, I will use U-Boot#. Typing help prints out all the commands confgured in this version of U-Boot; typing help <command> prints out more information about a particular command.

>>> Boot image format

```
mkimage
Usage: mkimage -l image
-l ==> list image header information
mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name -d
data_file[:data_file...] image
-A ==> set architecture to 'arch'
-O ==> set operating system to 'os'
-T ==> set image type to 'type'
-C ==> set compression type 'comp'
-a ==> set load address to 'addr' (hex)
-e ==> set entry point to 'ep' (hex)
-n ==> set image name to 'name'
-d ==> use image data from 'datafile'
-x ==> set XIP (execute in place)
mkimage [-D dtc_options] -f fit-image.its fit-image
mkimage -V ==> print version information and exit
For example, to prepare a kernel image for an ARM processor, the command is:
mkimage -A arm -O linux -T kernel -C gzip -a 0x80008000 -e 0x80008000 -n 'Linux' -d
zImage uImage
```

>>> Loading images I

```
U-Boot# mmc rescan
U-Boot# fatload mmc 0:1 82000000 uimage
reading uimage
4605000 bytes read in 254 ms (17.3 MiB/s)
U-Boot# iminfo 82000000
## Checking Image at 82000000 ...
Legacy image found
Image Name: Linux-3.18.0
Created: 2014-12-23 21:08:07 UTC
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 4604936 Bytes = 4.4 MiB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
U-Boot# setenv ipaddr 192.168.159.42
U-Boot# setenv serverip 192.168.159.99
U-Boot# tftp 82000000 uImage
link up on port 0, speed 100, full duplex
Using cpsw device
```

>>> Loading images II

```
TFTP from server 192.168.159.99; our IP address is 192.168.159.42
Filename 'uImage'.
Load address: 0x82000000
Loading:
3 MiB/s
done
Bytes transferred = 4605000 (464448 hex)
U-Boot# fatload mmc 0:1 82000000 uimage
reading uimage
4605000 bytes read in 254 ms (17.3 MiB/s)
U-Boot# nandecc hw
U-Boot# nand erase 280000 400000
NAND erase: device 0 offset 0x280000, size 0x400000
Erasing at 0x660000 - 100% complete.
OK
U-Boot# nand write 82000000 280000 400000
NAND write: device 0 offset 0x280000, size 0x400000
4194304 bytes written: OK
Now you can load the kernel from flash memory using nand read:
```

>>> Loading images III

```
U-Boot# nand read 82000000 280000 400000
```

The bootm command starts a kernel image running. The syntax is: bootm [address of kernel] [address of ramdisk] [address of dtb]. The address of the kernel image is necessary, but the address of ramdisk and dtb can be omitted if the kernel configuration does not need them. If there is a dtb but no ramdisk, the second address can be replaced with a dash (-). That would look like this:
U-Boot# bootm 82000000 - 83000000

>>> References

C. Simmonds.
Mastering Embedded Linux Programming.
Packt Publishing, 2015.