Booting Linux

Alexander Ulmer

January 24, 2024

Overview

How to boot Linux?

- Solution: Well, you use a bootloader

Thank you for your attention!

Overview

Chicken and egg problem: How to load the bootloader in the first place?

- Hardware: reset circuit and boot ROM
- Firmware: BIOS, Xilinx BOOT.BIN, etc.
- Bootloaders and boot protocols
- Linux platform initialisation

Power-up procedure.

System power-up

Asserted reset line on power-up will cause CPU to reset its state and start executing at a *predefined address*. There must be some kind of *boot ROM* mapped to it:

- **x86-PC**: BIOS ROM at `0xffff_fff0`.
- **ARM Cortex A9**: Starts executing at `0x0000_0000` or `0xffff_0000` depending on `VINITHI` input pin sampled on reset [1]

Boot ROM

Read-only memory that contains firmware code. It must be able to load the code for the first-stage bootloader.

- Can be external or embedded into the (SoC-)package.
- On x86-PCs, it is usually attached to the *Low Pin Count (LPC)* bus and contains the BIOS.

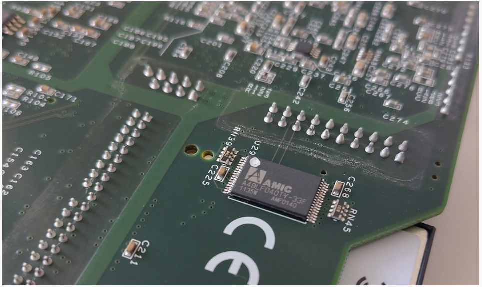


Figure 1: LPC flash chip on an *AMD Geode* based network router circuit board (PC Engines ALIX series).

Example: PC BIOS

The *basic input output system* (BIOS) is stored on the boot ROM (*BIOS ROM*).

- Starts executing code using the processor's cache (no RAM yet!)
- Performs *power-on-self-test* (POST)
- Initialises chipset and memory controller
- Performs enumeration of the PCI bus
- **Loads the bootloader** from the boot device's *master boot record* (512 bytes) to address `0x7C00`.

Details: BIOS BOOT Specification (1996) [2]

Example: PC BIOS

Detailed BIOS boot example [2]:

1. BIOS loads 1st stage bootloader (512 bytes) from master boot record into RAM.
2. 1st stage bootloader uses BIOS interrupt calls to load 2nd stage bootloader from disk sectors before filesystem starts.
3. 2nd stage bootloader knows filesystems and loads actual bootloader (e.g. GRUB) from a file.
4. Now the bootloader can continue loading the operating system (Linux).

Example: Unified Extensible Firmware Interface (UEFI)

UEFI is an improved interface between firmware and operating systems published by Intel in 1998. [3]

- Lets you write your own firmware plug-ins easily (*EFI Applications*).
- Booting works by loading an EFI application from the boot device, which *contains the bootloader* (e.g. GRUB).
- Usually, systems have UEFI and legacy-BIOS coexist on the same device. However, vendors recently started to drop legacy-BIOS support [4].

Example: Zybo Z7

- Cold reset: latch boot mode jumper state into register.
- Warm reset: mode register left unchanged.

Then execute code on boot ROM that loads the next-stage bootloader.

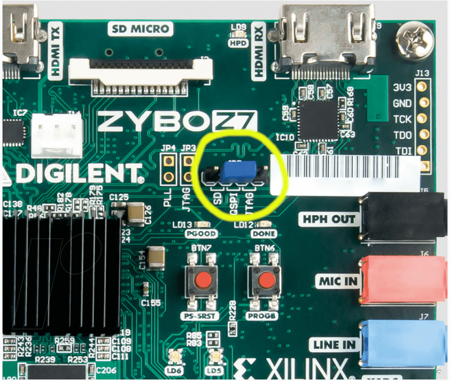


Figure 2: Boot mode jumpers on a Zybo Z7 development board [5].

Example: Zybo Z7: First stage bootloader (FSBL)

Code on boot ROM will load *first stage bootloader* (`BOOT.BIN`) from selected boot-source into on-chip static RAM (SRAM). Next steps:

- set up the DDRAM controller and map RAM to `0x4000_0000`.
- if bitstream is available, configure FPGA.
- load bare metal program or next-stage bootloader (e.g. U-Boot) into DDRAM.

Details: Zybo Zynq Z7 Reference Manual [5]

Bootloaders and boot protocols

What is a bootloader?

Bootloaders do all the setup crap operating system developers don't want to deal with:

```
/*
 * Kernel startup entry point.
 * ---------------------------
 * [...]
 * We're trying to keep crap to a minimum; DO NOT add any machine
 * specific crap here - that's what the boot loader (or in
 * extreme, well justified circumstances, zImage) is for.
 */

```

(From linux source code, `/arch/arm/kernel/head.S`)

What is a boot protocol?

Bootloaders **implement** one or more *boot protocols*.

Boot protocols specify:

- **Machine state** after jump to kernel
- **What** information to pass along
- **How** to pass this information

Examples: Multiboot, Multiboot2, *the linux boot protocol*

Device Tree

Configuration information needs to be provided to the kernel. One way to do this is a *device tree*. It includes:

- Machine type
- Memory layout
- Kernel command line
- Initramdisk address in memory
- hardware devices and their MMIO addresses
- ...

Configuration can also be hardcoded, but usage of a **device tree** is "highly recommended" [6].

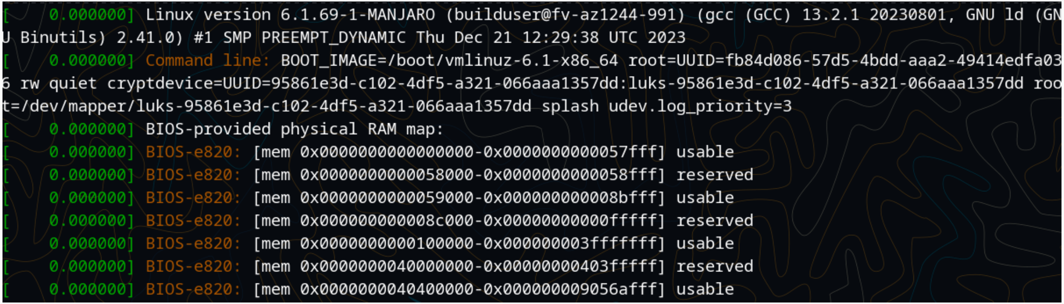
Tagged lists

Historically, configuration was provided via a **tagged list**. For example, on ARM:

- `r1`: Machine type (`linux/arch/arm/tools/mach-types`)
- `r2`: Physical address of tagged list

Again, this got completely replaced by the **device tree**.

Example: E820 memory map



```
[    0.000000] Linux version 6.1.69-1-MANJARO (builduser@fv-az1244-991) (gcc (GCC) 13.2.1 20230801, GNU ld (GN
U Binutils) 2.41.0) #1 SMP PREEMPT_DYNAMIC Thu Dec 21 12:29:38 UTC 2023
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-6.1-x86_64 root=UUID=fb84d086-57d5-4bdd-aaa2-49414edfa03
6 rw quiet cryptdevice=UUID=95861e3d-c102-4df5-a321-066aaa1357dd:luks-95861e3d-c102-4df5-a321-066aaa1357dd roo
t=/dev/mapper/luks-95861e3d-c102-4df5-a321-066aaa1357dd splash udev.log_priority=3
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000057fff] usable
[    0.000000] BIOS-e820: [mem 0x0000000000058000-0x0000000000058fff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000059000-0x000000000008bfff] usable
[    0.000000] BIOS-e820: [mem 0x000000000008c000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000003fffffff] usable
[    0.000000] BIOS-e820: [mem 0x0000000040000000-0x00000000403fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000040400000-0x000000009056afff] usable
```

Figure 3: Screenshot showing the early kernel log of a Thinkpad T480. The E820 memory map is being displayed in the top. It was retrieved by calling the `GetMemoryMap()` function of the UEFI boot services [7]

Linux boot protocol

The *Linux Boot Protocol* [8] is different for every architecture. Things all architectures have in common:

- Location of the kernel image (*zImage*), stack and heap
- Layout of kernel image header structure layout:
    - Location of the *initial RAM disk* (initrd)
    - Kernel command line
    - ...
- **Lots** of other details...

zImage and initial RAM disk

- To save space, the linux kernel image can be compressed. In this case it contains code to decompress itself. This is then called a *zImage* or *bzImage*.
- Since the root filesystem cannot always be mounted directly, the kernel image can be accompanied by an *initial RAM disk* or *initramfs* containing...
  - ... `/etc/fstab`: Mount points
  - ... `/init`: The init program
  - ... `/lib`: Shared libraries and kernel modules

Linux platform initialisation

Linux entry point (ARM) [9]

Entry point in `arch/arm/kernel/head.S` (only 600 lines):

- `__lookup_processor_type()`, `__lookup_machine_type()`
- `__create_page_tables()`, `__enable_mmu()`
- `__enable_mmu()`:
  - set TTB register (ARM's cr3), enable MMU
- `__mmap_switched()`:
  - clear BSS
  - jump to `start_kernel()` (architecture-independent)

Generic initialisation: `start_kernel()`

- Initialize console and debug output
- Start handling interrupts
- Bring up the other CPU cores
- Setup virtual file system
- Mount initial RAM disk
- ...
- **Spawn the init process**: `rest_init()`

Spawning the init process

- `rest_init()`:
  - Spawning the init process:

```
    pid = user_mode_thread(kernel_init, NULL, CLONE_FS)}

```

  - Set the system state to scheduling and switch to the idle task:

```
    system_state = SYSTEM_SCHEDULING;
    [...]
    cpu_startup_entry(CPUHP_ONLINE);
}

```

Now the init program is responsible for bringing up userspace and shell.

Bonus: Symmetric multiprocessing (SMP) initialisation

Symmetric multiprocessing initialisation brings up all the other processor cores, if they are present.

- On ARM, when linux takes over, all cores execute the same code: Bad
- Solution: Boot one core, then wake up the others
- Non-boot cores are looping on WFI instruction (like HLT) until they get an IPI [10]:

```
while (pen_release != read_core_id()) {
    __asm__ ("wfi");
}
boot();
```

References

[1] ARM, Cortex-A9 Technical Reference Manual. 2008-2011, p. 174.

[2] Intel-Compaq-Phoenix, **Bios boot specification version 1.01,** (1996), [Online]. Available: https://www.scs.stanford.edu/nyu/04fa/lab/specsbbs101.pdf (visited on 01/23/2024).

[3] T. Leemhuis, **Was firmware, bios, uefi alles meinen können. in: Heise online,** (2019), [Online]. Available: https://heise.de/-4405251 (visited on 01/23/2024).

[4] C. Windeck, **Intel: Uefi-bios verliert 2020 die bios-kompatibilität. in: Heise online,** (2017), [Online]. Available: https://heise.de/-3890747 (visited on 01/23/2024).

[5] W. Deacon, **Digilent zynq z7 reference manual,** (2018), [Online]. Available: https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf (visited on 01/21/2024).

[6] R. King, **Linux kernel documentation: Booting arm,** (2002), [Online]. Available: https://docs.kernel.org/arch/arm/booting.html (visited on 01/21/2024).

[7] **Uefi specification: Boot services,** (Version 2.9A), [Online]. Available: https://uefi.org/specs/UEFI/2.9_A/07_Services_Boot_Services.html (visited on 01/24/2024).

[8] **Linux kernel documentation: The linux boot protocol (x86),** (Version 2.15), [Online]. Available: https://www.kernel.org/doc/html/v5.6/x86/boot.html (visited on 01/23/2024).

[9] J. Sevy, **Arm linux boot sequence,** (2007), [Online]. Available: `https://jsevy.com/linux/ARM_Linux_boot_sequence.html` (visited on 01/21/2024).

[10] ARM, **Arm developer documentation: Smp boot in linux,** (Version 4.0), [Online]. Available: `https://developer.arm.com/documentation/den0013/d/Multi-core-processors/Booting-SMP-systems/SMP-boot-in-Linux` (visited on 01/21/2024).

[11] W. Deacon, **Linux kernel documentation: Booting aarch64,** (2012), [Online]. Available: `https://www.kernel.org/doc/Documentation/arm64/booting.txt` (visited on 01/21/2024).

[12] TheRasteri, **Add an isa slot to modern motherboards!** (2023), [Online]. Available: `https://www.youtube.com/watch?v=putHMSzu5og` (visited on 01/21/2024).