Image preparation for SD card

Author:

Armando Emmanuel Correa Amorelli

Zapopan, Jalisco, May 18, 2023

Table of Contents

Introduction....3
Objective....3
Boot stages of a Linux-based embedded system....4
Bootloader Depuration....5
Analysis....5
Adding a console log at booting....6
Source code modification....6
Source code modification Result....7
Patch....7
Conclusions....8
Bibliography....9

Introduction

Objective

Identifying the booting process of the platform and the different peripherals involved in this process, as well as understanding and comparing the steps and actions by which the booting media can be changed.

Boot stages of a Linux-based embedded system

The boot process of a Linux-based embedded system consists of several stages that initialize and prepare the system for full operation. These stages include powering up the system to executing the Linux kernel and loading the runtime environment.

The different stages of the boot process of a Linux-based embedded system are the follow:

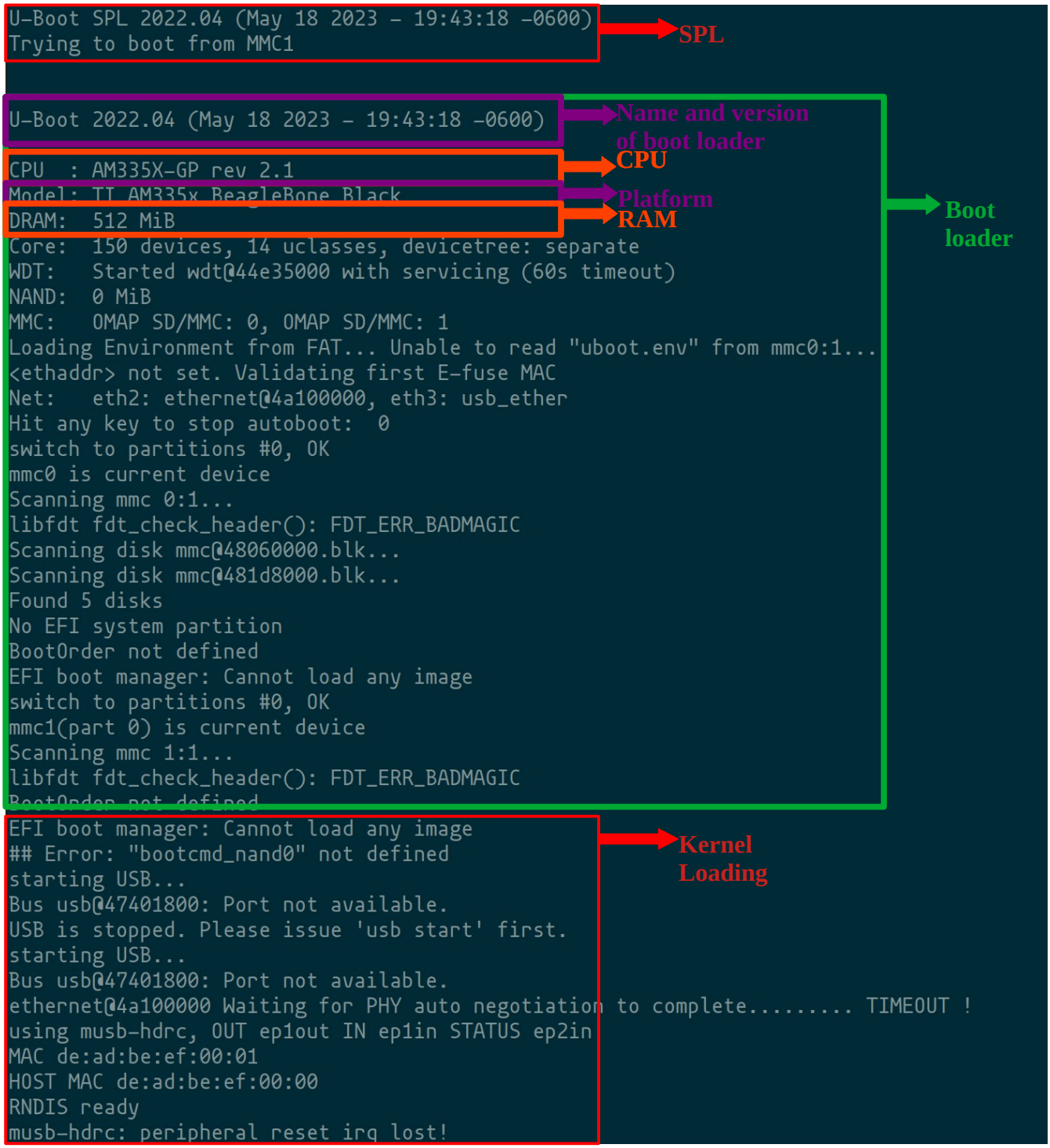
1. **Power-up and hardware initialization(IPL):** In this stage, the embedded system is powered up, and basic hardware initialization takes place. This includes activating the power source, configuring system clocks, initializing buses, and activating essential peripherals.
2. **Bootloader(SPL):** The bootloader is software stored in non-volatile memory, such as ROM or flash. It runs after hardware initialization. Its main function is to load the Linux kernel into memory and transfer control to it. Popular bootloader examples include GRUB, U-Boot, and Das U-Boot.
3. **Linux kernel loading:** In this stage, the bootloader loads the Linux kernel into the system's memory and performs necessary configurations for its execution. This involves setting initialization parameters like command-line arguments, hardware configuration, and required device drivers.
4. **Linux kernel initialization:** Once the Linux kernel is loaded into memory, its execution begins. During this process, the kernel performs various tasks such as memory configuration, device driver initialization, hardware detection, interrupt configuration, and creating the runtime environment.
5. **Root file system loading and execution:** After the Linux kernel is up and running, the root file system is loaded. It contains all the necessary files for the system's operation. The root file system can reside in flash memory, an SD card, or other storage media. After loading the root file system, it is mounted as the system's root directory.
6. **User space startup:** Once the root file system is loaded and mounted, the Linux kernel starts the user space. This involves executing processes and services necessary for the system to be fully functional. Initial processes may include network configuration, loading kernel modules, setting up services, and running applications required for the embedded system.

To identify these stages during the boot process of a Linux-based embedded system, you can follow these steps:

- **Observe the boot sequence on the screen:** When powering on the embedded system, a boot sequence is typically displayed on the screen, indicating the different stages of the boot process. This may include information about hardware initialization, the bootloader, kernel loading, and other components.
- **Review bootloader and kernel logs:** Both the bootloader and the Linux kernel generate logs that can provide information about the boot stages. By examining the logs, you can identify the sequence of events and the progress of each stage.

Bootloader Depuration

Analysis



Adding a console log at booting

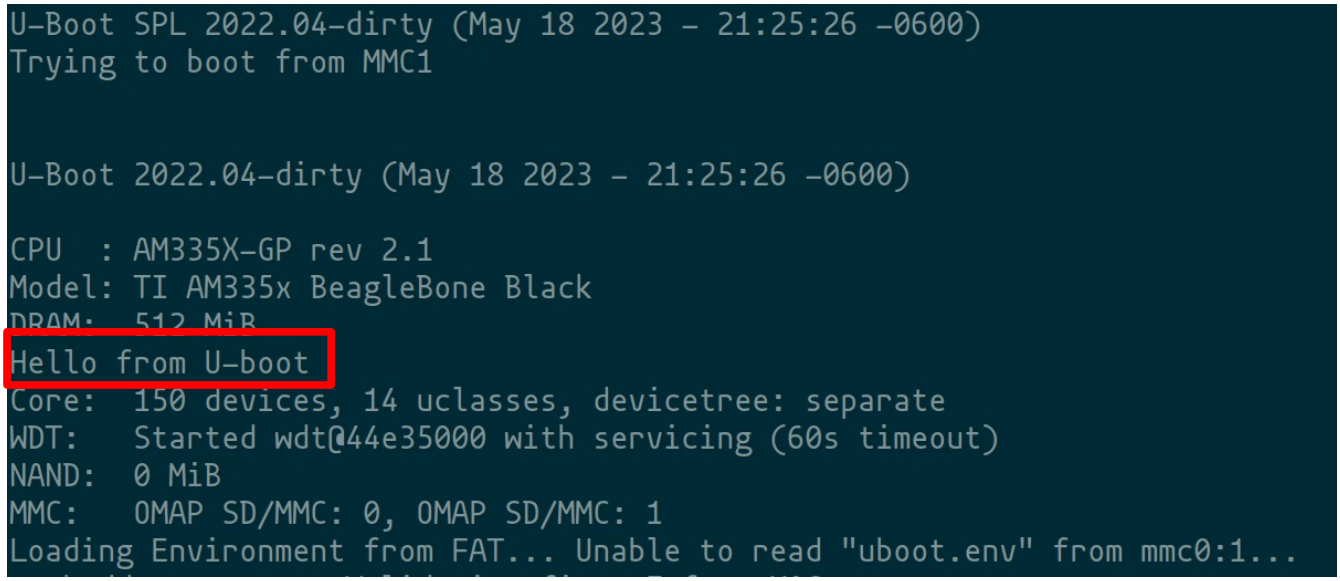
Source code modification

1. **Locate the file:** Look for the file responsible for displaying pre-boot messages. Typically, this file is named *`board.c`* and is located in the *`board`* directory of the U-Boot source tree.
2. **Modify the file:** Open the file in a text editor and locate the function that initializes the board. This function is typically named *`board_init()`* or similar.
3. Add your console message: Within the *`board_init()`* function, add a call to the console output function, such as *`printf()`* or *`puts()`*, to display your message. For example:

```
        rv = setup_clock_synthesizer(&cdce913_data);
        if (rv) {
            printf("Clock synthesizer setup failed %d\n", rv);
            return rv;
        }

        /* reset PHYs */
        gpio_set_value(GPIO_PHY_RESET, 0);
        udelay(2);  /* PHY datasheet states 1uS min. */
        gpio_set_value(GPIO_PHY_RESET, 1);
    }
#endif

    printf("Hello from U-boot\n")

    return 0;
}
```

4. Save changes
5. Compile U-Boot
6. Flash SD card with updated U-Boot

Source code modification Result



Patch

See attach file hello_from_uboot.patch

Conclusions

Being able to modify and personalize the bootload is, from my point of view, a strong and powerful tool. Because, it allow us to debug a new board or give some better experience to the user while the system is still booting.

Bibliography

[1]“The U-Boot Documentation — Das U-Boot unknown version documentation,” *u-boot.readthedocs.io*. https://u-boot.readthedocs.io/en/latest/ (accessed May 18, 2023).

[2]K. Klein-Wengel, “Understanding the (Embedded) Linux boot process,” *Klein Embedded*, Mar. 31, 2023. https://kleinembedded.com/understanding-the-embedded-linux-boot-process/ (accessed May 18, 2023).

[3]M. Jones, “Inside the Linux boot process,” *IBM Developer*, May 31, 2006. https://developer.ibm.com/articles/l-linuxboot/

[4]P. Divate, “Embedded Linux booting process,” *Medium*, Feb. 28, 2023. https://prashant-divate.medium.com/embedded-linux-booting-process-f3c09e3c1a8f (accessed May 18, 2023).