arm

Embedded Linux

Linux-based Embedded System Component Stack

Goals

To illustrate the main components of a Linux-based embedded system

To analyze the bootstrap process

Summary

Introduction

Bootloader

Kernel

Device tree

System programs

Application

Root filesystem

Summary

Introduction → component ที่ make up ของ embedded system

Bootloader

Kernel

Device tree

System programs

Application

Root filesystem

Linux-based Embedded System Components

Bootloader

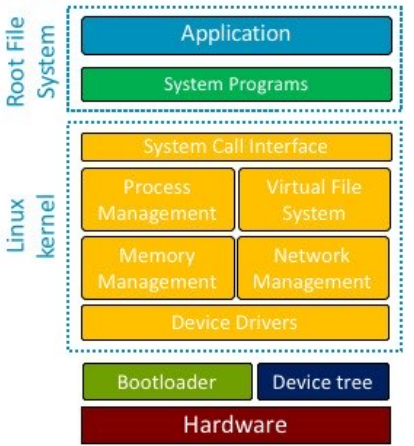
- Software executed at power-up to set-up the hardware to run the operating system

Device tree

- A tree data structure with nodes that describe the physical devices in the hardware needed by the Linux kernel to initialize properly the device drivers

Linux Kernel (main component)

- The operating system code providing all the services to manage the hardware resources



arm

Linux-based Embedded System Components

System programs

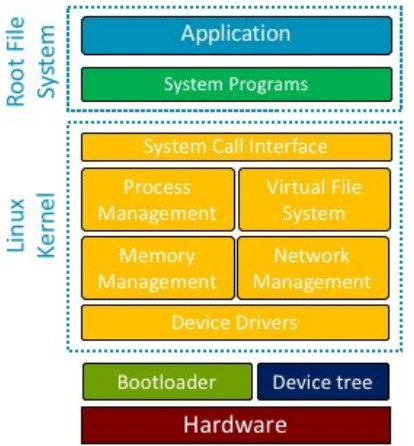
- User-friendly utilities to access operating system services

Application

- Software implementing the functionalities to be delivered to the embedded system user

Root filesystem

- Container for the Linux Kernel configuration files, the system programs, and the application

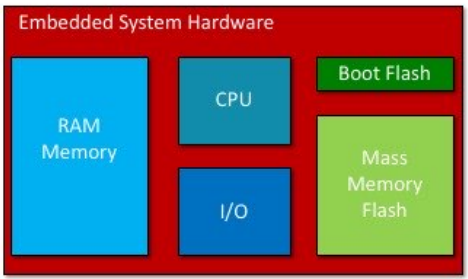


arm

Reference Hardware Model

Embedded System Hardware components:

- RAM memory: volatile memory storing data/code
- CPU: processor running software
- I/O: peripherals to get inputs from the user, and to provide outputs to the user
- Boot Flash: small non-volatile memory needed at power-up (discussed later)
- Mass Memory Flash: large non-volatile memory (discussed later)

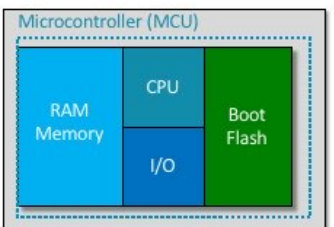


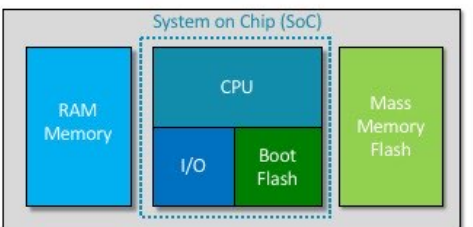
arm

Reference Hardware Model Implementations

Multiple implementations of the reference hardware model are possible. For example:

- Microcontroller-based implementation: a single device hosts most of the reference model components (e.g. CPU, RAM memory, boot flash).
- System-on-Chip implementation: most of the reference model components are discrete components, while the CPU is integrated with some of them (e.g. I/O, boot flash).





arm

CPU Memory Map

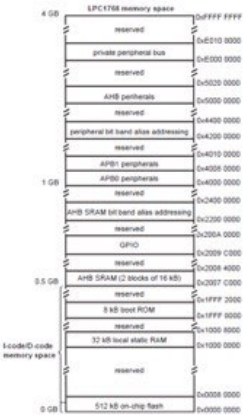
The CPU generates 2^N different addresses ($0 \rightarrow 2^N-1$).

- N=number of bits of the address bus

Each device (memory, I/O, etc.) is associated with a range of addresses.

The memory map describes this association for all the devices.

Memory map of NXP LPC1768 (Cortex-M3)



Summary

Introduction

Bootloader

Kernel

Device tree

System programs

Application

Root filesystem

arm

The Role of the Bootloader

A processor is designed to run software from “somewhere”.

To start running the software, the processor needs to know:

- Where the software is located
- How to get access to the software location
- Where the stack is located

The Role of the Bootloader

ต้องรันเป็นอันแรก

When powered-up, the processor needs a simple way to get the information it needs.

- Where the software is located
- How to get access to the software location
- Where the stack is located

pointer ต่อ address ที่เก็บคำสั่งแรกที่ต้องทำงานของ firmware

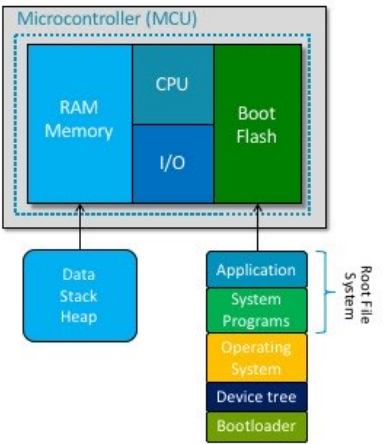
At power-up, the program counter is set to a default known value, the reset vector.

- The software starting at the reset vector, which will load the bootloader, takes care of providing the processor all the needed information.
- The operations performed depend on the system architecture.

Possible Scenarios

Scenario 1, typical of microcontrollers

- All the sw (bootloader + device tree + operating system + root filesystem) is stored in persistent storage (boot flash) embedded in the microcontroller.
- All the sw is executed from the persistent storage.
- The CPU reset vector is located in the boot flash.
- The RAM Memory is embedded in the microcontroller and is used for data, stack and heap only.

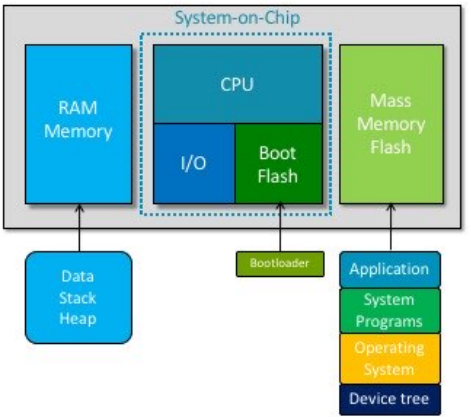


arm

Possible Scenarios

Scenario 2, typical of System-on-Chip

- The bootloader is stored into the boot flash.
- The CPU reset vector is located in the boot flash.
- The root filesystem, operating systems, and device tree are stored in the mass memory flash and loaded in RAM memory by the bootloader.
- The RAM memory is external to the SoC. It will store the operating system + application software, root filesystem (if configured as RAM disk), data, stack, and heap.

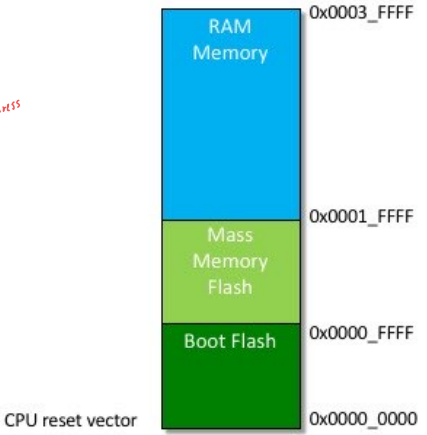


arm

An Example of Bootloader Operations

Let's consider Scenario 2 for a given SoC with the following memory map:

- RAM memory 512 kbytes (arranged as 128 kwords, each 32 bytes long), from 0x0002_0000 to 0x0003_FFFF
- Mass memory flash 256 kbytes (same organization as before), from 0x0001_0000 to 0x0001_FFFF
- Boot flash 256 kbytes (same organization as before), from 0x000_0000 to 0x0000_FFFF



arm

An Example of Bootloader Operations

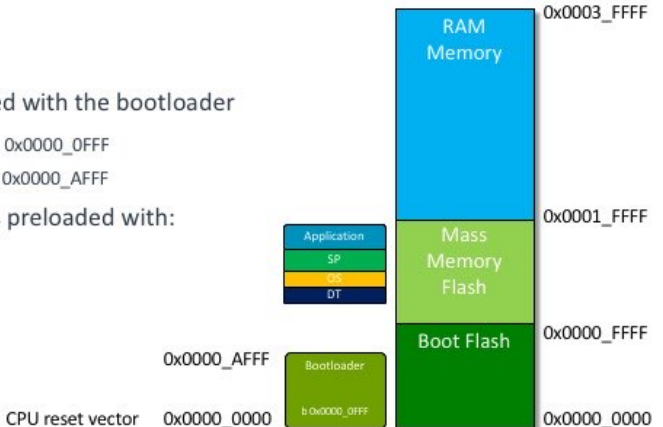
Time = before power up

The boot flash is preloaded with the bootloader

- First bootloader instruction at 0x0000_0FFF
- Last bootloader instruction at 0x0000_AFFF

The mass memory flash is preloaded with:

- Device tree (DT)
- Operating systems (OS)
- System programs (SP)
- Application



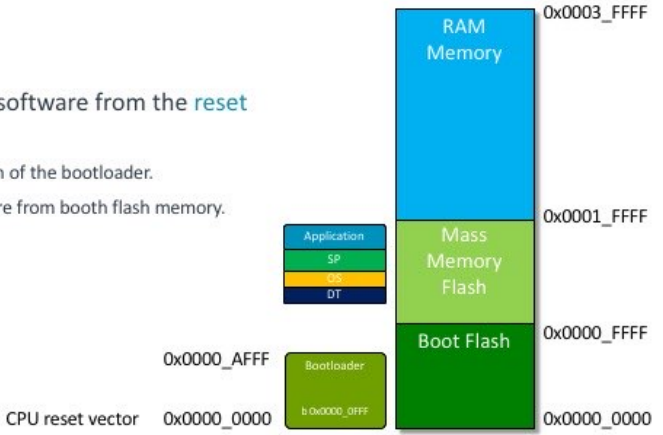
arm

An Example of Bootloader Operations

Time = power up

The CPU starts executing software from the reset vector:

- It jumps to the first instruction of the bootloader.
- It runs the bootloader software from booth flash memory.



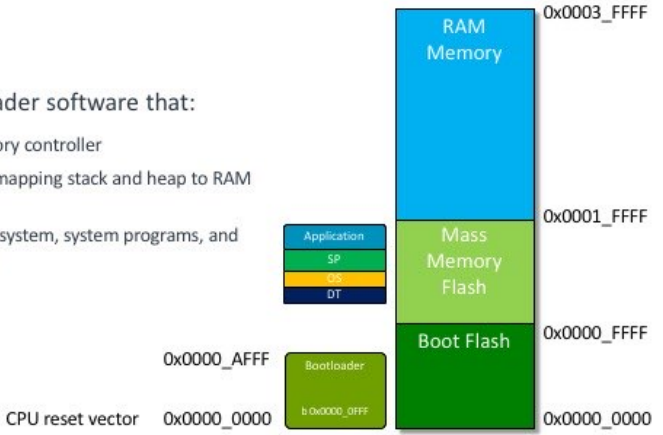
arm

An Example of Bootloader Operations

Time = during bootstrap

The CPU executes bootloader software that:

- Initializes the CPU RAM memory controller
- Sets up the CPU registers for mapping stack and heap to RAM memory
- Copies device tree, operating system, system programs, and applications to RAM memory



arm

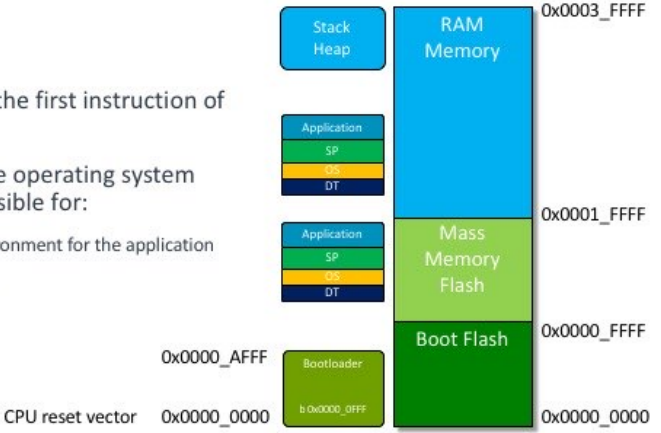
An Example of Bootloader Operations

Time = end of bootstrap

The bootloader jumps to the first instruction of the operating system.

The CPU now executes the operating system software, which is responsible for:

- Setting-up the execution environment for the application
- Starting application execution



arm

Summary

Introduction

Bootloader

Kernel

Device tree

System programs

Application

Root filesystem

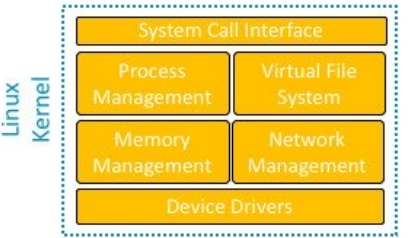
arm

Linux Kernel

The Linux Kernel is the software responsible for optimally managing the embedded system's hardware resources.

It offers services such as:

- Process management → task scheduling
- Process scheduling
- Inter-process communication → share data
- Memory management → share memory in block sizes
- I/O management (device drivers)
- File system
- Networking
- And more

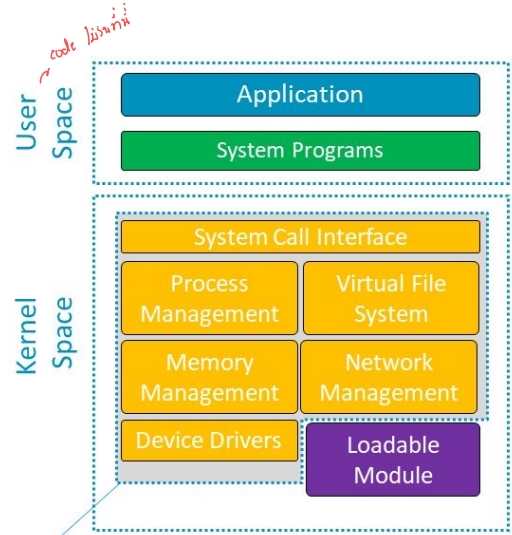


arm

Linux Kernel

The Linux kernel adopts a layered operating system architecture:

- The operating system is divided into two layers, one (user space) built on top of the other (Kernel space).
- User space and Kernel space are different address spaces.
- Basic services are delivered by a single executable, monolithic Kernel.
- Services can be extended at run-time through loadable Kernel modules



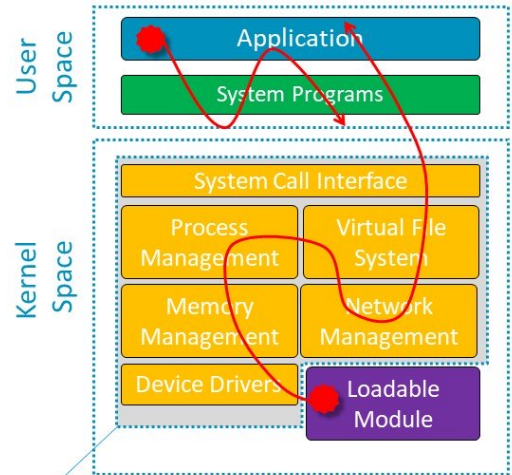
Linux Kernel

Advantage

- Good separation between application/system programs and kernel. Bugs in the user space do not corrupt the kernel.

Disadvantages

- Bugs in one Kernel component (e.g. a new device driver) can crash the whole system.



arm

Summary

Introduction

Bootloader

Kernel

Device tree

System programs

Application

Root filesystem

arm

Device Tree

To manage hardware resources, the Kernel must know which resources are available in the embedded system (i.e. the hardware description: I/O devices, memory, etc).

There are two ways to provide this information to the Kernel:

- Hardcode it into the Kernel binary code. Each modification to the hardware definition requires recompiling the source code.
- Provide it to the Kernel when the bootloader uses a binary file, the device tree blob.

A device tree blob (DTB) file is produced from a device tree source (DTS).

- A hardware definition can be changed more easily as only DTS recompilation is needed.
- Kernel recompilation is not needed upon changes to the hardware definition. This is a big time saver.

Summary

Introduction

Bootloader

Device tree

Kernel

System programs

Application

Root filesystem

System Programs

System programs provide a convenient environment for program development and execution.

They can be divided into:

- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Application programs

Summary

Introduction

Bootloader

Device tree

Kernel

System programs

Application

Root filesystem

Application

The application is the software required to provide the end user service for which the embedded system was conceived.

Examples can be found in many different products:

- Network Attached Storage (NAS)
- Network router
- In-vehicle infotainment
- Specialized lab equipment
- And more

Summary

Introduction

Bootloader

Device tree

Kernel

System programs

Application

Root filesystem

arm

Root Filesystem

The Linux Kernel needs a file system, called a root filesystem, at startup.

- It contains the configuration file needed to prepare the execution environment for the application (e.g. setting up the Ethernet address).
- It contains the first user-level process (init).

The root filesystem can be:

- A portion of the RAM treated as a file system known as Initial RAM Disk (initrd), if the embedded system does not need to store data persistently during its operations
- A persistent storage in the embedded system, if the embedded system has to store data persistently during its operations
- A persistent storage accessed over the network, if developing a Linux-based embedded system

Typical Layout of the Root Filesystem

```
/                                   # Disk root
        /bin                        # Repository for binary files
        /lib                        # Repository for library files
        /dev                        # Repository for device files
                console c 5 1       # Console device file
                null c 1 3          # Null device file
                zero c 1 5          # All-zero device file
                tty c 5 0           # Serial console device file
                tty0 c 4 0          # Serial terminal device file
                tty1 c 4 1          #
                tty2 c 4 2          #
                tty3 c 4 3          #
                tty4 c 4 4          #
                tty5 c 4 5          #
        /etc                        # Repository for config files
                inittab             # The inittab
                /init.d             # Repository for init config files
                        rcS         # The script run at sysinit
        /proc                       # The /proc file system
        /sbin                       # Repository for accessory binary files
        /tmp                        # Repository for temporary files
        /var                        # Repository for optional config files
        /usr                        # Repository for user files
        /sys                        # Repository for system service files
        /media                      # Mount point for removable storage
```