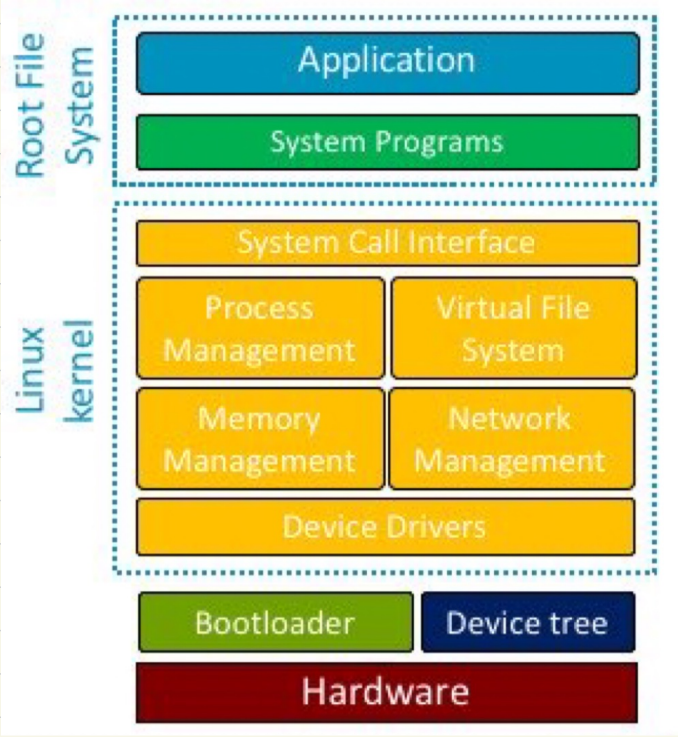
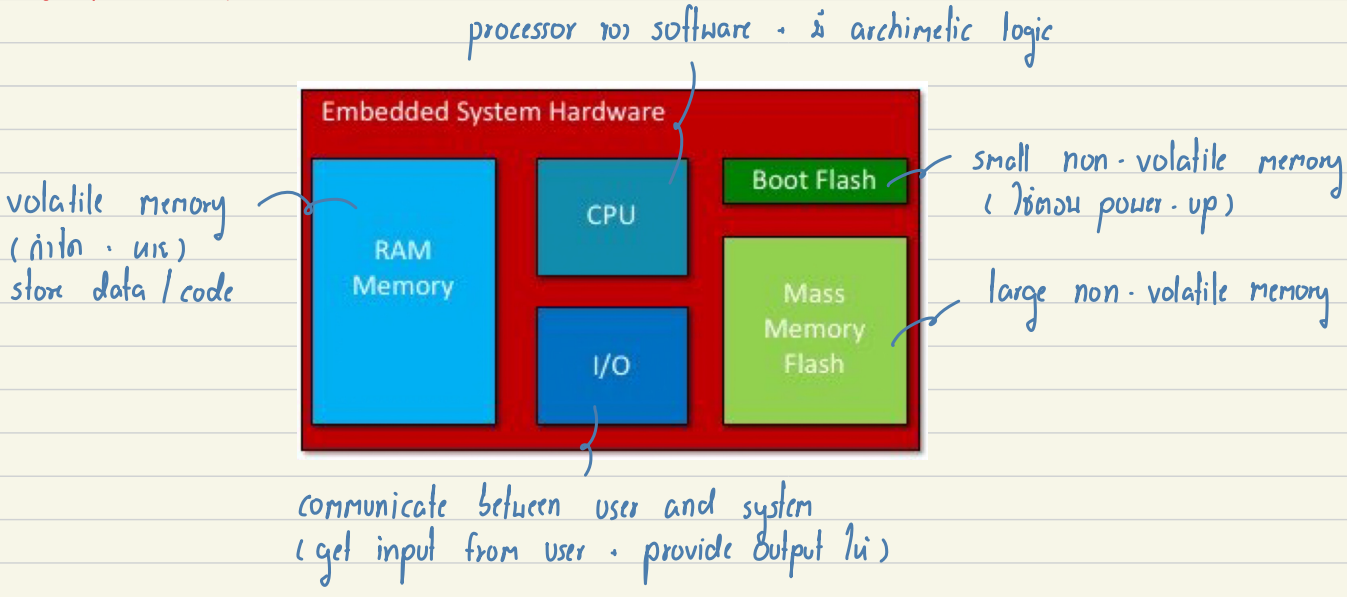
Linux based embedded system component

1. Bootloader → software ที่รัน execute ตอนที่ power-up เพื่อ set-up hardware ในการ run OS
2. Device tree → describe physical device ใน hardware ที่ linux kernel ต้องใช้ในการ init device drivers
3. Linux Kernel → OS code ที่ providing service ในการ manage hardware resource
4. System program
5. Application → Software ที่ implement functionalities ให้ embedded system user



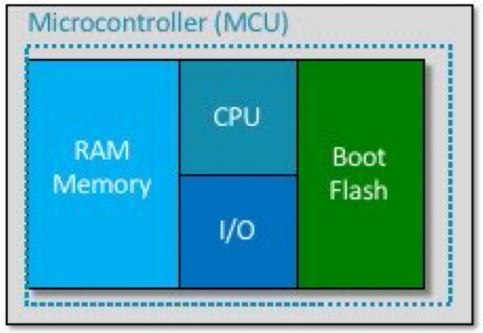
6 Root filesystem → เป็น container ที่เก็บ Linux kernel configuration file, system program, application

Reference Hardware model

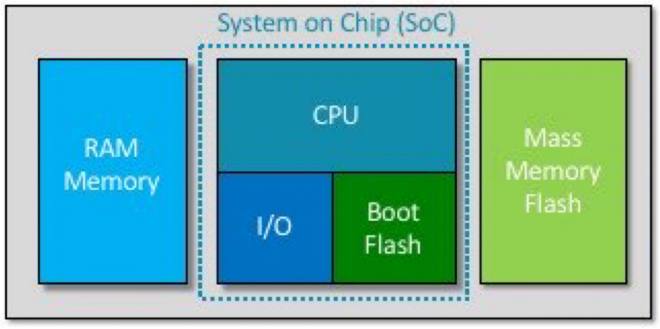


Reference Hardware Model Implementation

1. single device hosts most of reference model component



2. reference model component ส่วนมากเป็น discrete CPU รวมกับ some of them (ex. I/O , file)



CPU memory map

- มีมาก generate ได้ 2^N difference address ; N = จำนวน bits ของ address bus
- แต่ละ device มี address point อยู่ (memory and I/O จะ require memory ใน space)
- memory map จะ describe ตามลำดับหน่วยความจำของ device

The role of the Bootloader

thing need to know

- ตำแหน่ง software ที่อยู่
- วิธี access software location
- stack เก็บที่ไหน

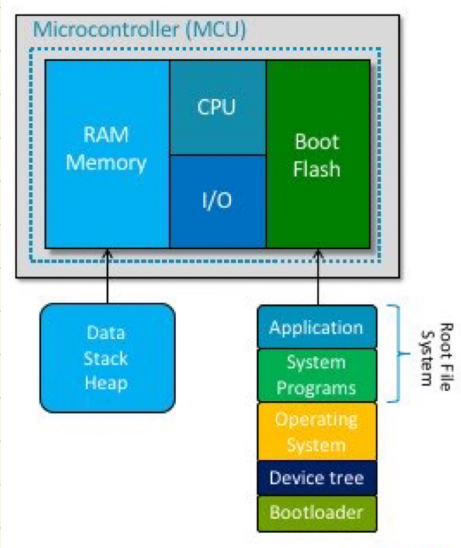
At power-up → program counter จะถูก set default known value และ execute reset vector

program counter = register ที่เก็บค่า address ของ instruction ต่อไปที่จะ run

reset vector = pointer ที่ชี้ไปที่ address ที่เก็บ instruction แรกที่จะ run firmware

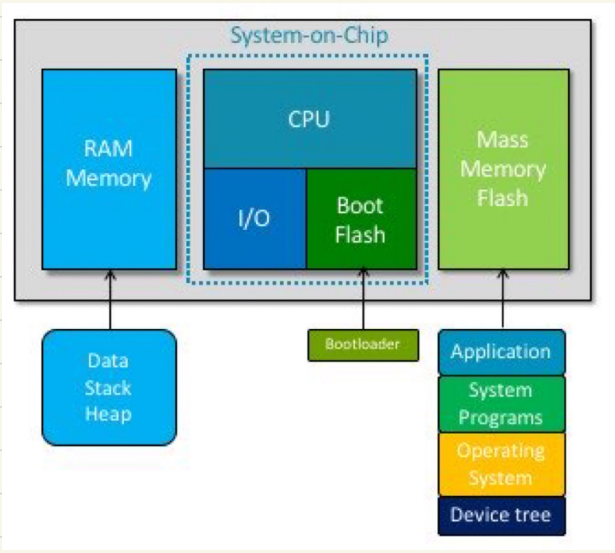
Possible Scenarios

1.) Microcontroller



- 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.

2.) System on chip



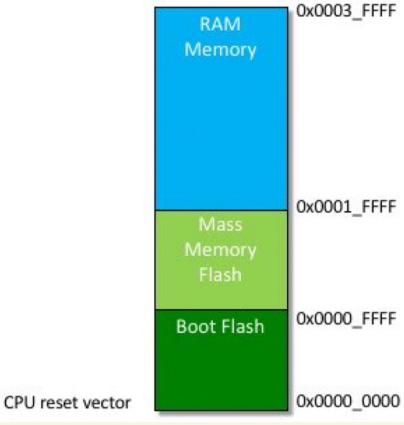
- The bootloader is stored into the boot flash.
- The CPU reset vector is located in the boot flash.

} aziz MCU

- 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.

Example of bootloader Operations (SoC)

- 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



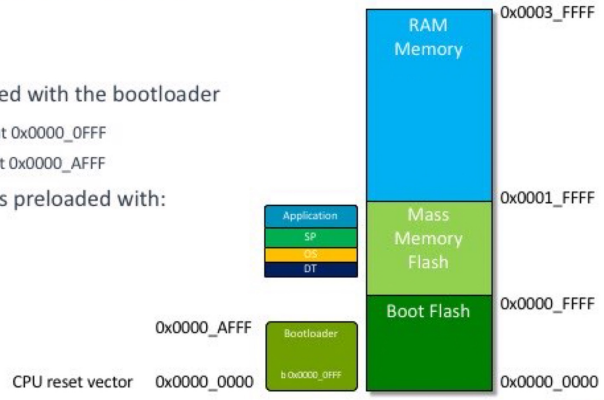
Time = before power up

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:

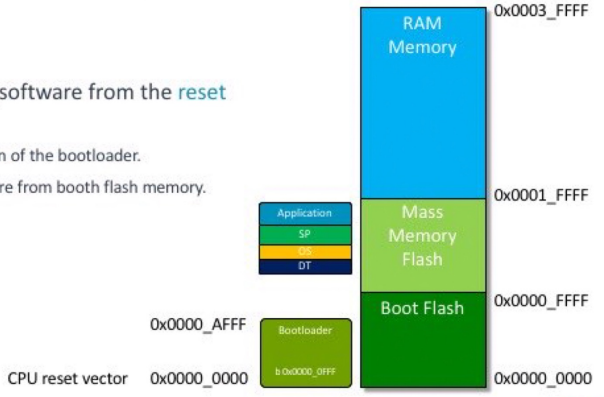
- Device tree (DT)
- Operating systems (OS)
- System programs (SP)
- Application



Time = power up

The CPU starts executing software from the reset vector:

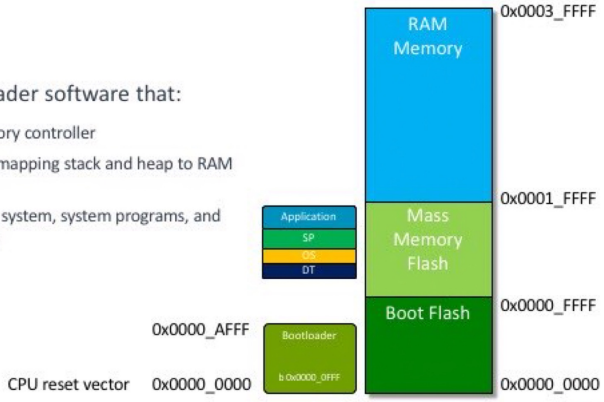
- It jumps to the first instruction of the bootloader.
- It runs the bootloader software from booth flash memory.



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

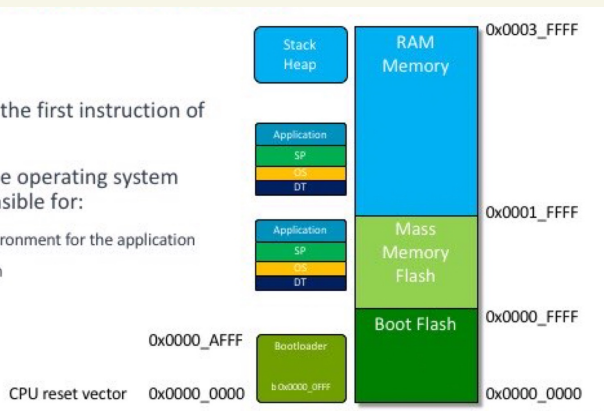


Time = end of bootstrap

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

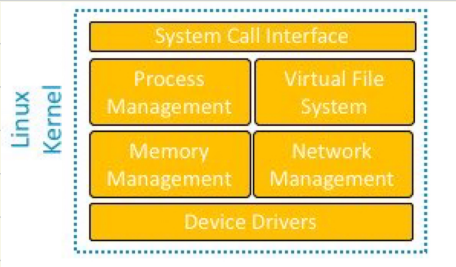
The CPU now executes the operating system software, which is responsible for:

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



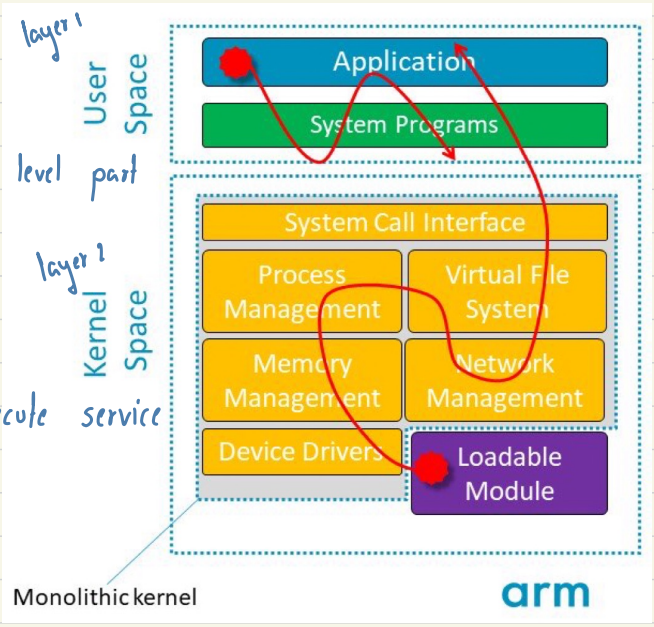
Linux Kernel

เป็น software ที่ดูแลจัดการ hardware resource ของ embedded system



It offers services such as:

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



ทั้ง 2 layer จะมี address space ที่ต่างกัน

- basic service → delivered by single executable
- service ที่เหลือ extended ที่ run-time เช่น loadable kernel modules

advantage

- มีการแบ่งแยกระหว่าง application, system program, kernel
- Bugs ใน user space ไม่ส่งผลต่อ kernel

disadvantage

- bugs แต่ละ component ทั้งใน kernel ก็ส่งผลกับทั้งหมดได้

Device tree

ในการ manage hardware resource , kernel จะต้องรู้ว่า resource ในระบบ embedded system มีอะไร

1.) hardcode ลงไปใน kernel binary code (ทุกครั้งที่มีการแก้ไข hardware definition ต้อง compile source code ใหม่)

2.) provide it to kernel ตอน bootloader ให้ binary file (device tree blob) DTB

ซึ่งมาจาก 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.

System program

คือ executable จาก command line (ex. ls, cd)

โดยจะเป็นตัวจัดการ environment ให้

program development และ execution

They can be divided into:

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

Application

↳ software ที่ให้ตอบสนอง user service

Examples can be found in many different products:

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

Root filesystem

ตอน startup linux, kernel จะใช้ filesystem (root filesystem) ซึ่งจะ contain configuration file ที่ใช้ในการเตรียม execution environment ร่วมกับ application และ first user-level process (init)

```
/                               # 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
```

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

LAB - Yocto Project

- ไม่แนะนำให้ใช้กับ ubuntu 20 ขึ้นไป -- 16, 18 ใช้ได้
- ไม่แนะนำ clone จาก branch krogoth ขึ้นไป -- ต่ำกว่านั้น branch sumo
- ใน rpi-build/conf/bblayers.conf ต่อท้ายด้วย ${BSPDIR}/sources/meta-openembedded/meta-python \
- ติดตั้ง Etcher -- รองรับกับ ubuntu 18 ขึ้นไป

* ปัญหาที่เกิดขึ้นจาก bitbake rpi-basic-image