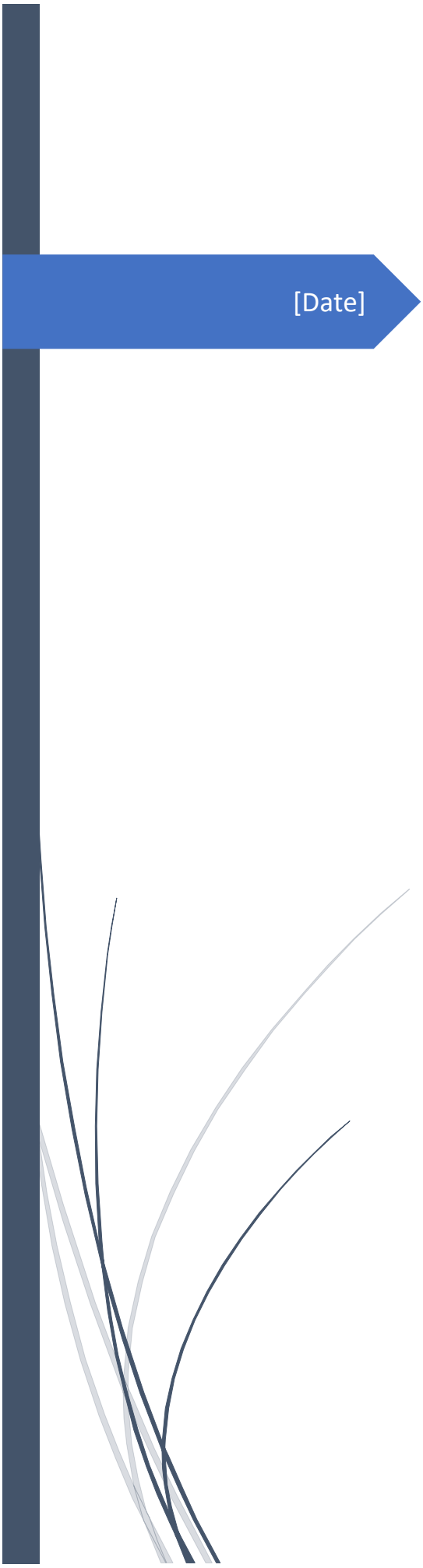
[Date]

[Bootloader, Startup code, Flash sections, Hex File Format]



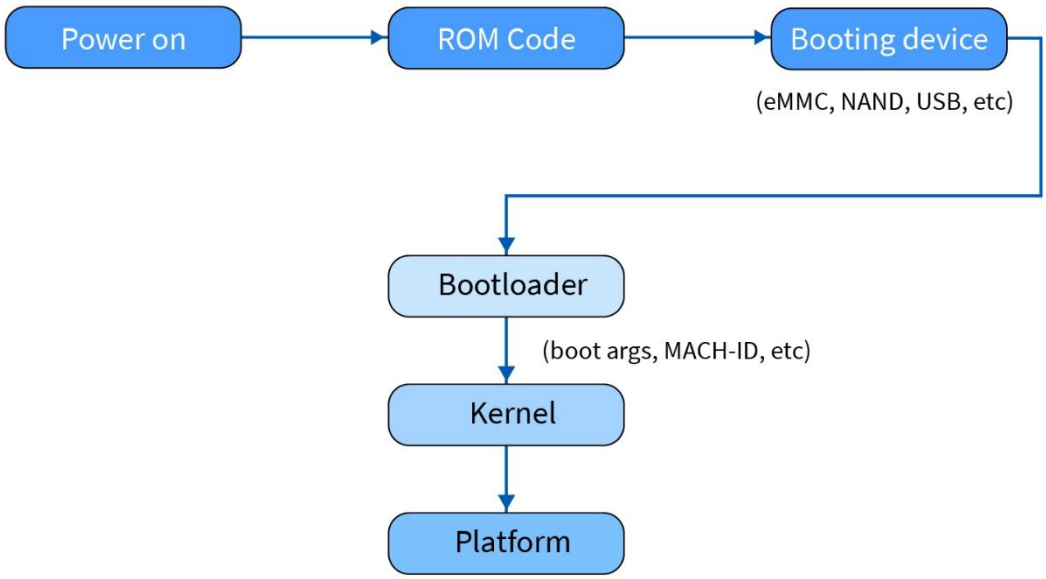
Eslam Ahmed Ali
ITI . EMBEDDED ADV. AL-AZHAR G4 23

1-Bootloader

Bootloader stands for Bootstrap Loader which is the compact software which is responsible to load the operating system into the memory of the computer. The bootloader will always run whenever the computer system is started or restarted.

When you start your computer, the operating system gets loaded and after that, you are able to login into the system and use the system, but who loads the operating system into the memory of your computer? **It's called Bootstrap Loader which loads the operating system into the memory.**

So, when we start out the system, all the hardware components receive the power signal and get initialized. Now BIOS which is a basic input-output system will read the instructions and based on the instructions BIOS will look for a bootable device. Once the bootable device is found, the BIOS loads the bootloader. **Bootloader will load the operating system into memory. This entire process is called booting**. Any kind of hardware device which contains the bootloader is called Bootable Devive.



- When we press the power button of our system, the hardware components get initialized and we see the different pieces of information about the hardware of the system.

- As the main memory of the system is volatile so it won't have any content present in it. In order to start the execution, the initial instructions are fetched from non-volatile memory called ROM(Read Only Memory), and this initial set of instructions is called BIOS.
- BIOS is a compact program that will find the bootable device present in the system, eg. Floppy disk, Hard disk, CD etc.
- Once the bootable device is found, BIOS search for Master Boot Record (MBR) which contains the bootloader.
- If the MBR is found, BIOS will load the bootloader present in it.
- From here on, the bootloader takes the control of CPU and loads the operating system into the main memory of our system.
- Once the operating system is loaded into the main memory it takes the control of the entire system.

Note: *Non-Volatile Memory* is the memory that can hold the information even if we restart the device. eg. ROM (Read Only Memory) Whereas, *Volatile Memory* is the memory that gets erased if we restart the device. eg. RAM (Random Access Memory)

If BIOS fails to find MBR in any of the bootable devices it will return an error message.

Where Exactly are Bootloaders Stored?

Bootloaders are generally stored in the *first sector* of a bootable device called Master Boot Record. So, whenever the BIOS finds the bootable device, BIOS simply reads the data present in the **first sector**.

We can keep some data that is not related to the loading of the Operating System into the first sector of the bootable device detected by BIOS. Many gaming vendors keep the code responsible to start the game in the first sector of the bootable device as a result when we start the system, BIOS looks for the bootable device, and as the bootable device now contains the code which can start the game so rather than loading the operating system bootloader will execute the instructions responsible to start the game.

There is a second way in which the bootloader is stored. With the advancement of technology in many systems, the bootloader is present in *specific partition* of the bootable medium and BIOS are smart enough to identify in which sector the bootloader is present.

It is possible that we can have more than one bootloader. There are systems that have a primary bootloader that is very small in size and whose only work is to load the secondary bootloader. Once the secondary bootloader is loaded it will be responsible to load the operating system.

A Summary of a Bootloader's Functions

- The main function of the bootloader is to load the operating system.
- If there is more than one bootloader, the primary bootloader has to load the secondary bootloader.
- Bootloaders are also used to launch the application programs. (e.g., games)
- Bootloaders are also used to expand or add the missing function's firmware.
- Bootloaders are used to load the alternative firmware.

2-Start up code

Startup code typically refers to the initial set of instructions executed when a computer system or software application starts up. Its purpose is to prepare the environment and set the stage for the main execution of the program. Here are some important points to discuss regarding startup code:

1. Bootstrapping: Startup code is often responsible for bootstrapping the system or application. This involves initializing the necessary hardware components, such as the CPU, memory, and peripherals, to bring the system to a functional state.

2. Configuration and Initialization: Startup code handles the configuration and initialization of various software components. This may include setting up system-wide variables, establishing communication channels, configuring hardware devices, and initializing data structures.

3. Setting Up Interrupts and Exception Handling: Startup code typically configures interrupt handlers and exception handling mechanisms. Interrupts allow the system to respond to external events, while exception handling deals with unexpected errors or exceptional conditions that may occur during program execution.

4. Memory Management: Startup code often plays a role in memory management. It may allocate memory for the program's data, stack, and heap segments. Additionally, it sets up the memory protection mechanisms to ensure proper access control and prevent unauthorized operations.

5. Loading and Linking: In the case of executable programs, startup code handles the loading and linking of the program's binary code into memory. This involves reading the executable file from storage, resolving symbols and dependencies, and setting up the program's execution context.

6. Environment Setup: Startup code can establish the runtime environment for the program, including setting up the necessary libraries, frameworks, and runtime configurations. It may also handle command-line argument parsing and initialization of global variables.

7. Error Handling and Recovery: Startup code typically includes error handling mechanisms to detect and handle critical errors that occur during startup. This may involve logging errors, displaying error messages, and taking appropriate recovery actions.

8. Main Function Invocation: Once the startup tasks are completed, the startup code typically invokes the main function of the program. The main function acts as the entry point for the program's execution, where the primary logic and functionality reside.

3-Flash sections

Flash sections, also known as memory sections or memory segments, are specific regions in the memory of a microcontroller or embedded system that are allocated for storing different types of data or code. These sections are defined in the linker script, which is used during the compilation and linking process of the software.

Here are some commonly used flash sections:

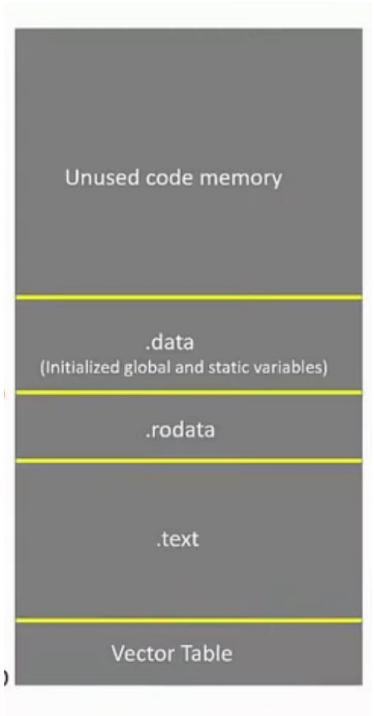
1. Text Section (Code Section): This section contains the executable code of the program. It typically includes the main function and other functions or routines that are called during program execution. The text section is usually read-only and is stored in non-volatile memory (such as flash memory) to retain the code even when the system is powered off.

2. Data Section: The data section stores initialized global and static variables. These variables have predefined values assigned to them at the time of declaration. The data section is typically stored in flash memory during the initial programming of the microcontroller or embedded system and may be copied to RAM during program execution for faster access.

3. BSS Section: The BSS (Block Started by Symbol) section is used to store uninitialized global and static variables. These variables are initialized to zero or null values by default. As with the data section, the BSS section is typically stored in flash memory and may be copied to RAM during program startup.

4. Constant Section: The constant section, also known as the read-only data section or rodata section, is used to store read-only data that is accessed during program execution. This may include constants, string literals, and other immutable data. The constant section is usually stored in flash memory.

5. Interrupt Vector Table (IVT): The IVT is a special section that contains the addresses of interrupt service routines (ISRs). When an interrupt occurs, the microcontroller jumps to the corresponding ISR address stored in the IVT. The IVT is typically located at a fixed memory address and is often stored in flash memory.



4-Hex File Format

The hex file format is a common file format used to represent binary data, typically the machine code instructions or data that will be programmed into a microcontroller or other embedded systems. Hex files are often generated by compilers, assemblers, or other development tools as an output format for the resulting binary code.

Here's a brief overview of the hex file format:

1. Structure: A hex file consists of a series of text records, each representing a block of data. Each record starts with a colon (':') character as the record marker.

2. Record Format: Each record is composed of several fields:

- Start Code: The first field is a one-byte hexadecimal value (two ASCII characters) representing the count of the remaining fields in the record.

- Address Fields: The next two fields represent the address of the data. The first field is a two-byte hexadecimal value representing the address offset, and the second field is a one-byte hexadecimal value representing the record type.

- Data Field: The following fields contain the actual data. The number of data fields depends on the count specified in the start code field.

- Checksum: The last field is a one-byte hexadecimal value representing the checksum. It is calculated by summing all the bytes in the record and taking the two's complement of the least significant byte of the sum. The checksum ensures data integrity during transmission or storage.

3. Record Types: The record type field specifies the purpose of the data in the record. Common record types include:

- Data Record (Type 00): Represents a block of data to be loaded into memory at the specified address.

- End of File Record (Type 01): Marks the end of the hex file.

- Extended Segment Address Record (Type 02): Specifies the upper 16 bits of the data address for subsequent data records.

- Start Segment Address Record (Type 03): Specifies the execution start address for the program.

- Extended Linear Address Record (Type 04): Specifies the upper 16 bits of the data address for subsequent data records in a 32-bit linear address space.

- Start Linear Address Record (Type 05): Specifies the execution start address for the program in a 32-bit linear address space.

Here's an example of a hex file record:

```
:10010000214601360121470136007EFE09D2190140
^ ^^^^ ^^ ^^^^ ^^^^^^^^^^^^^^^^^^^^ ^^
| |  | |  |            |  |
| |  | |  |            |  Checksum
| |  | |  |            Data
| |  | |  Record Type
| |  |  Address
| |  Start Code
Record Marker
```

In this example, the record has a start code of 0x10 (16 in decimal), an address of 0x0100, a record type of 0x00 (data record), and the data field contains 16 bytes of data. The checksum is calculated based on these values.

Hex files are commonly used in the programming and debugging of microcontrollers and embedded systems as they provide a human-readable representation of binary data and can be easily parsed by programming tools to program the target device's memory.