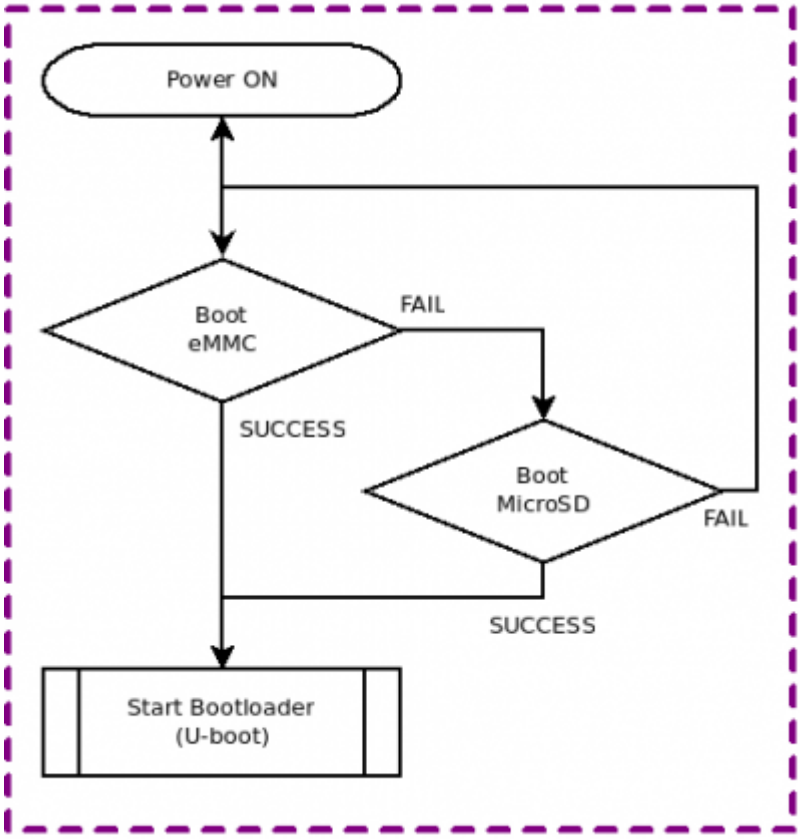
Boot sequence

Stage #1: Power cycle

This stage is aimed to fetch the bootloader software from the inserted to the board. ODROID-C4 offers two different boot storages, please visit this page to figure out the location.

- eMMC : Removable (custom design by Hardkernel, eMMC 5.0)
- Micro SD : Removable (UHS-I)

The bootloader in the eMMC or the Micro SD will be fetched by CPU. Since the eMMC has a high priority, CPU will fetch from Micro SD if eMMC does not have the bootloader.



Stage #2: Bootloader

The second stage to fetch the relatively huge software binaries from attached storage that can be accessible by the bootloader loaded at stage #1. In this stage, the system becomes more flexible to access various storage types and vendors. Even the OS software can be loaded through network protocol or USB storages. Also, the bootloader also offers a command line shell so a user can run commands manually or set/store the bootloader variables.

Boot from eMMC/Micro SD

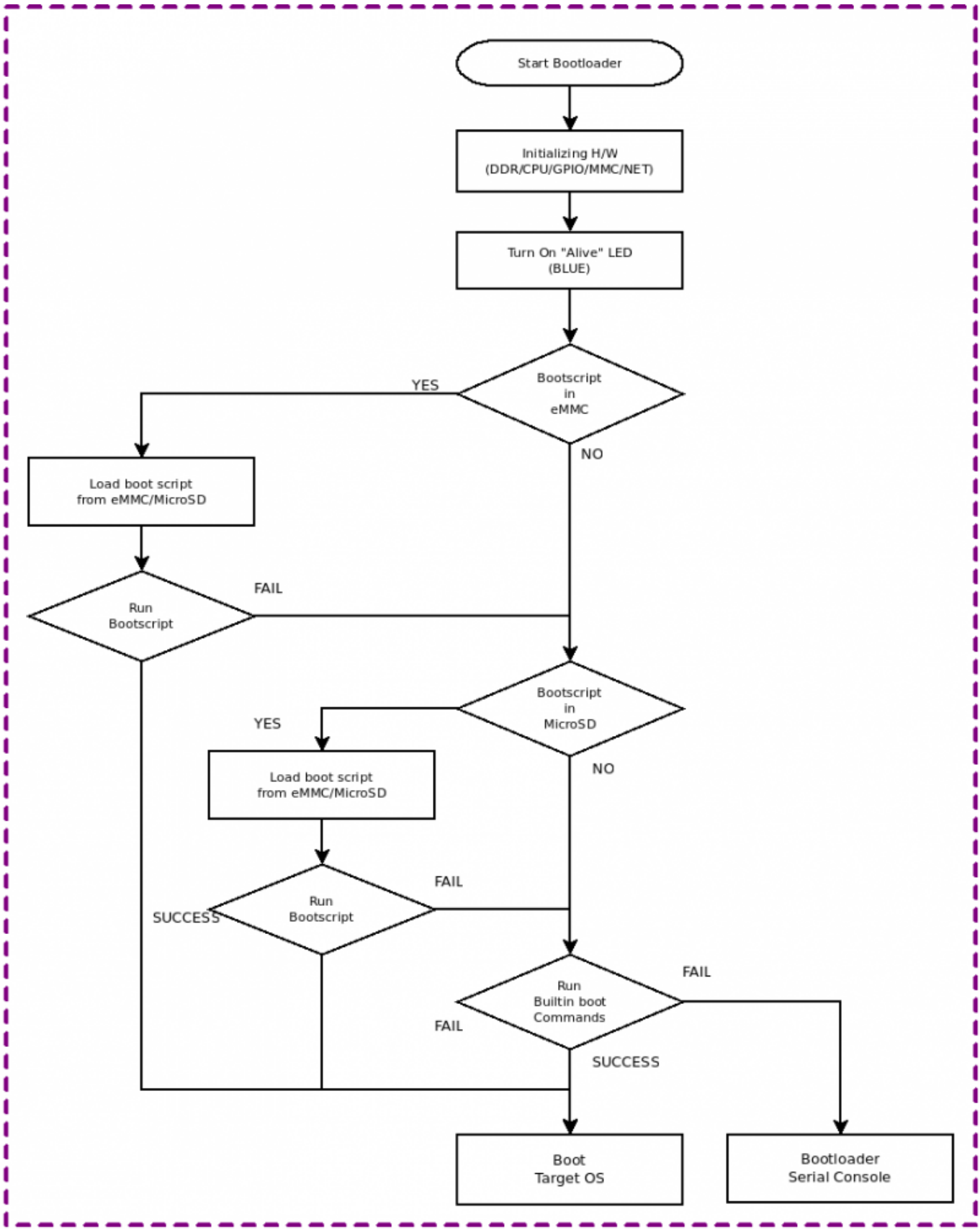
If ODROID-C4 is set to boot from eMMC or Micro SD, the bootloader looks up the first partition of each storage device from eMMC and Micro SD in order to find out the boot script. Actually, the contents in the boot script are the set of commands supported by the bootloader, `U-boot`, such as `setenv`, `load` and `bootm`. The current bootloader supports two types of boot scripts.

boot.ini

This is a purely text-based file, human-readable and editable by any text editor so very handy to change its contents at any platform. This script type is traditionally used by Hardkernel's ODROID SBCs. If any text file with `boot.ini` contains the board specific magic string, `ODROIDC4-BOOT-CONFIG` for ODROID-C4, at the first line, the bootloader starts to execute the script.

boot.scr

This is a legacy script format of U-boot which contains the 64 bytes binary header and test-based commands. Since this file is also editable in a text editor, one can change the commands in it. But, unlike `boot.ini`, `boot.scr` has the checksum at the header in the first 64 bytes, the header must be regenerated by the tool `mkimage` which is one of U-boot tools whenever the contents of the script. This script is not handy but can prevent to be modified and executed unexpectedly.



Debugging at U-boot

As shown in the diagram, the bootloader traverses the boot storage one after another until boot successfully. For some reason, the bootloader cannot start the next software, expensive OS like Linux, it stops and gives a shell prompt through the serial port. With USB-UART Module Kit, one can read

logging message from U-boot or gives commands directly.

Builtin boot commands

The one may be curious what is the builtin boot command and what commands are. Basically, this is the command set to read various blobs from the boot memory card among eMMC or Micro SD that the bootloader is loaded and practically to boot Android platform that requires multiple blobs in different sizes and types more than 4 partitions which is the amount of the primary partitions in MBR (Master Boot Record). So in order to store numerous blobs for Android in the memory card, the partition table is designed and noted in the U-boot code. This partitions can be updated through the `fastboot` that is a popular tool to update Android system through a USB cable. So technically, if you have software blobs that will store in the memory card as a raw format other than a file in a file system, you could design the partition table and update them in U-boot. This page introduces the current partition table for Android.