



现代 PC 的启动过程

-主要基于 WINDOWS 系统

DEZEMING FAMILY

DEZEMING

Copyright © 2022-01-06 Dezeming Family

Copying prohibited

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, without the prior written permission of the publisher.

Art. No 0

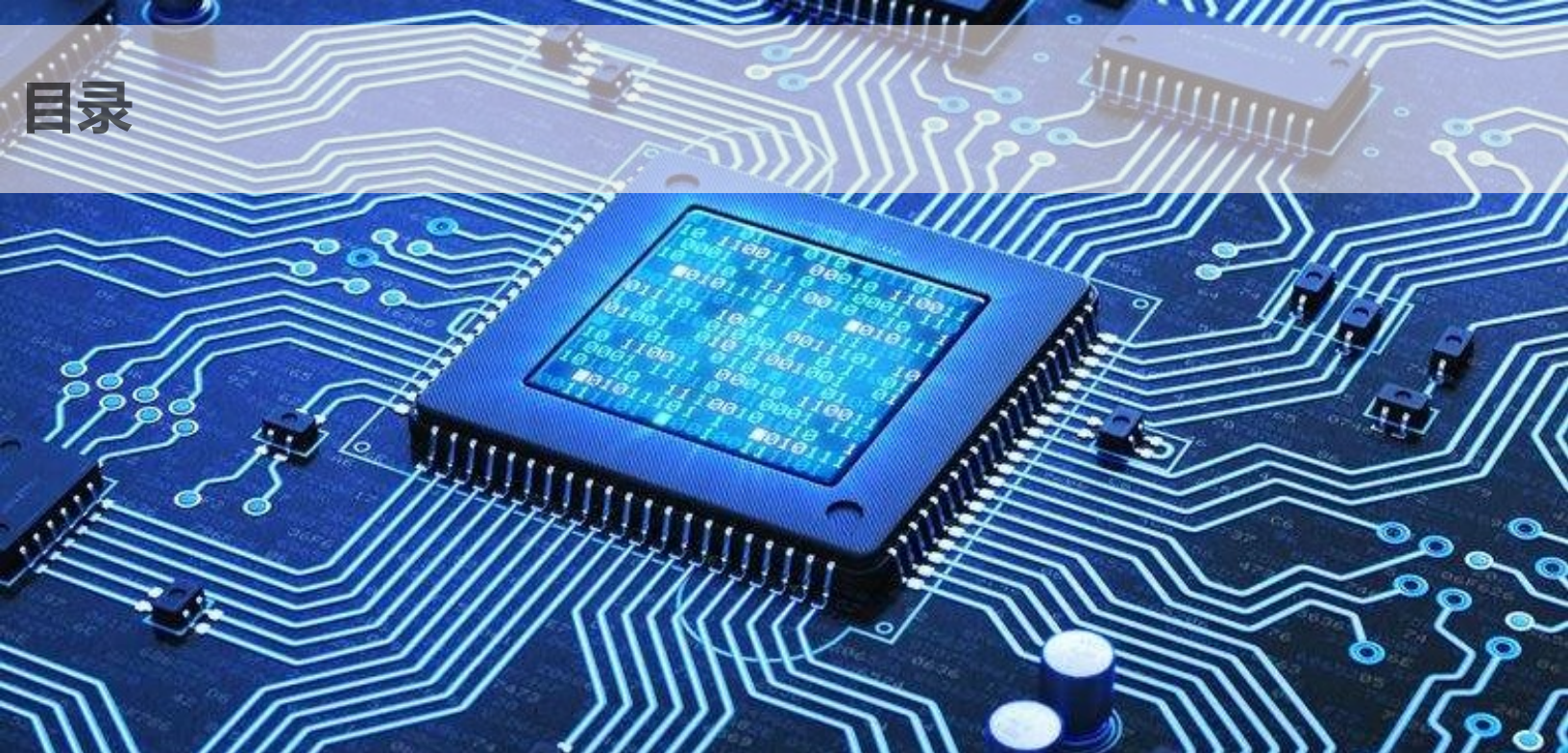
ISBN 000-00-0000-00-0

Edition 0.0

Cover design by Dezeming Family

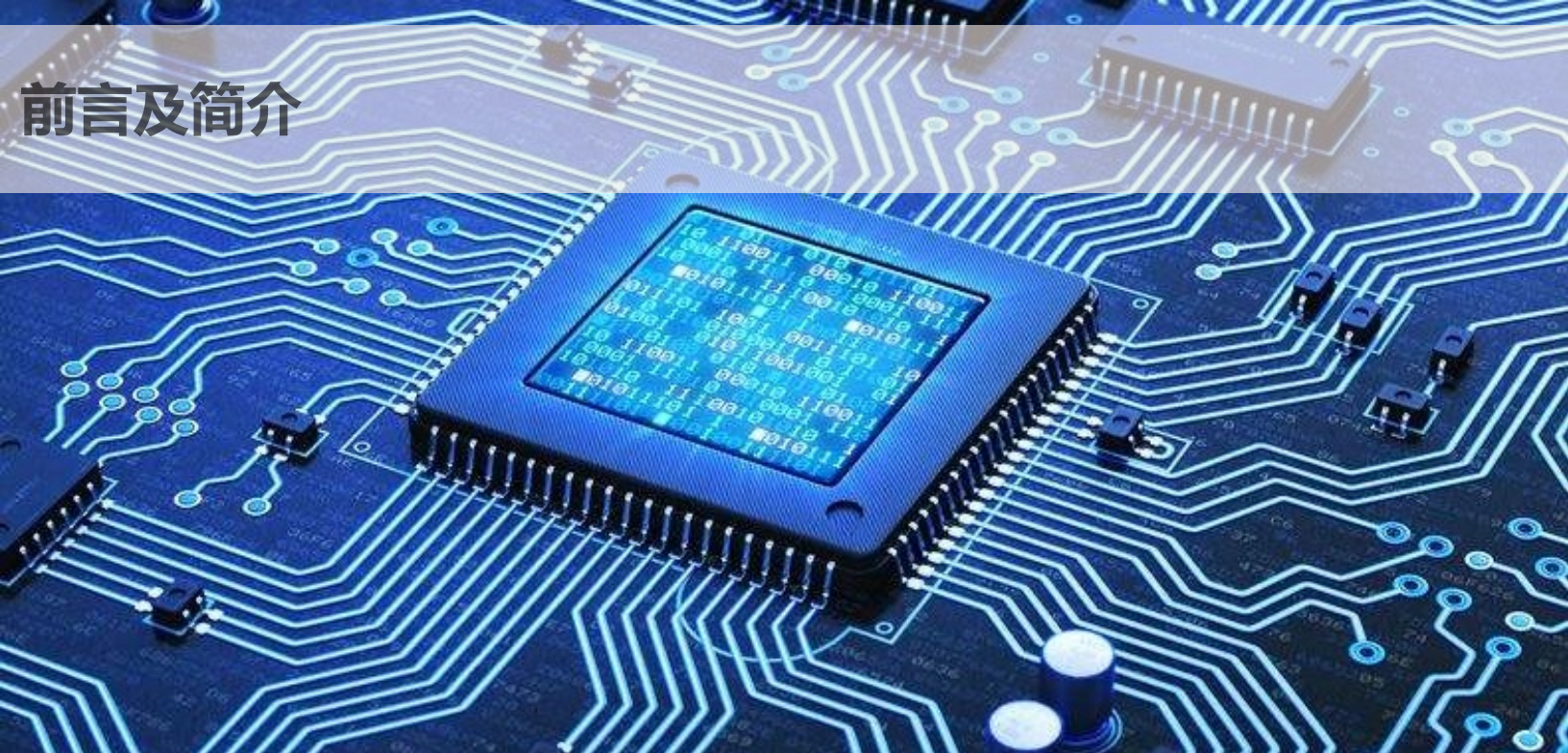
Published by Dezeming

Printed in China



目录

0.1	本书前言	5
1	硬件与开机过程	6
1.1	开机	6
1.2	BIOS 启动做了些什么	7
1.3	主引导记录 MBR	9
2	UEFI 与 BIOS	10
2.1	传统模式与 UEFI 模式简述	10
2.2	MBR 的问题	11
2.3	GPT 分区格式	11
2.4	UEFI 启动过程	13
2.5	UEFI 与 CSM 模式和安全启动	15
3	磁盘结构查看与验证	16
3.1	BIOS 启动与 MBR 磁盘	16
3.2	UEFI 启动与 GPT 磁盘	17
3.3	总结	20
	Literature	20



前言及简介

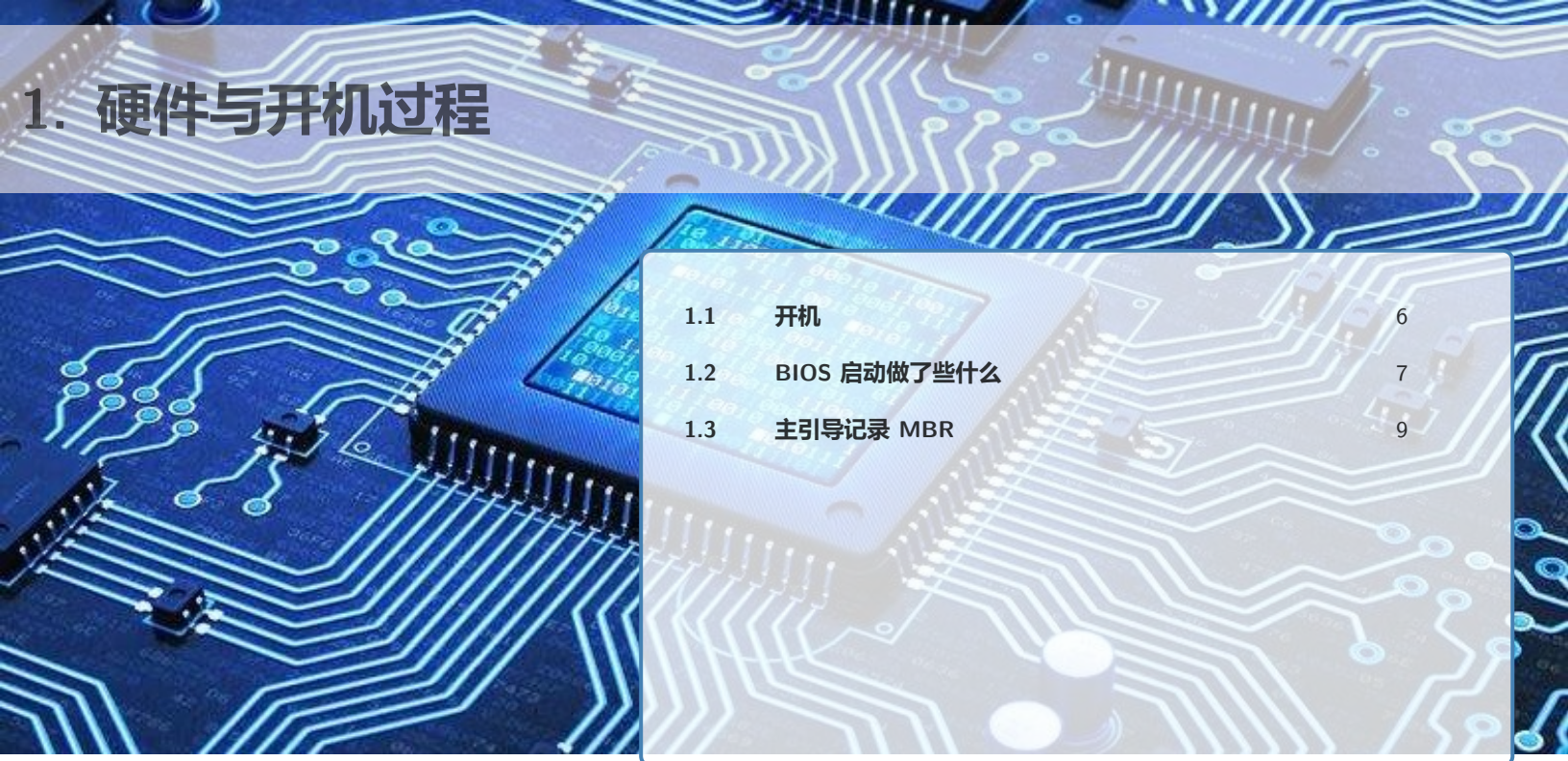
DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列书，可以从我们的网站 [<https://dezeming.top/>] 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

0.1 本书前言

在我以前研究嵌入式系统时，对系统的启动流程有过一些了解和认识，但对于现代 PC 机仍然有很多不了解的地方，于是凭借本文的内容来进行一些详细和细致的介绍。

一台现代的 Windows 计算机，当你按下主机电源以后，它到底经历了哪些过程，运行了哪里程序，何时加载操作系统，这些内容将会在本文中进行详细的介绍。

对于计算机中各种“鸡生蛋”和“蛋生鸡”的问题，这里不会做太多解释，只简单提一句：最开始的时候，代码都是手工打在纸条带上的，后来出现了存储器，人们也需要提前设置一些代码，然后用这些代码去启动计算机。之后人们在启动的计算机上用交互工具（例如键盘）来读写操作存储器，写出更多的代码，一步一步，直到慢慢构建出现在庞大和复杂的计算机软件系统和环境。之后，功能强大的计算机系统又可以给其他的计算机写一些启动程序和开发环境。



1. 硬件与开机过程

- 1.1 开机
- 1.2 BIOS 启动做了些什么
- 1.3 主引导记录 MBR

6
7
9

本章描述计算机系统的硬件和开机阶段的过程。

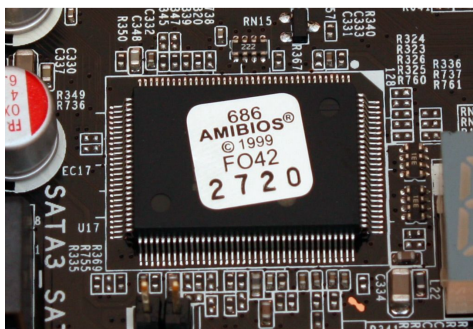
1.1 开机

“启动”就是 boot，这里的 boot 来自于 bootstrap(鞋带)[1]，这是由于一句谚语：“pull oneself up by one’s bootstraps”，即拽着鞋带把自己拉起来——正如计算机所做的事情：需要运行程序来启动计算机，但是计算机不启动就无法运行程序——总之是一件很矛盾的事情。

而可不可以事先搞一段程序，让计算机打开电源以后自动去执行这里的代码呢？学习过数字电路的话，我们就知道计算机可以在时钟信号中不断自动执行程序，进行读入和读出。因此，这个方法理论上是可行的。

早期的计算机启动程序可以在各种存储器上保存，比如磁芯存储。我们需要设计合理的接口，让 CPU 通电以后，能把它们读出来并执行相应的功能即可。后来，当只读内存 (read-only memory，又叫做 ROM) 出现以后，大家就开始广泛使用它来存储计算机通电以后首先读取和执行的程序了。

在现代 PC 机上，这个通电以后首先读取和执行的 ROM 叫做 BIOS (Basic Input/Output System，基本输入/输出系统)，通常是一个 BIOS 芯片：



值得一提的是，我们可以让电脑去升级 ROM 固件，也就是重写 ROM (刷 BIOS)，以更好地

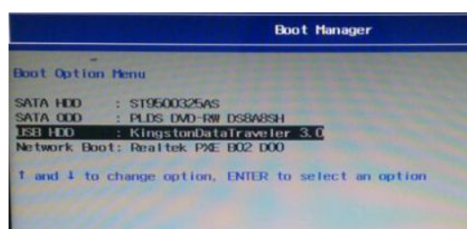
做一些启动相关的事项（例如硬件自检）。如果不小心刷错了，就可能会把电脑变成一个板砖——无法再启动，这时，我们就需要借助其他的计算机硬件设施来重刷我们当前的这台计算机了。

1.2 BIOS 启动做了些什么

BIOS 首先会进行硬件自检。我们在装机时，会选择不同厂家的主板，不同的主板上硬件也不同，所以 BIOS 里面的程序也应该是不同的。

BIOS 会检查计算机硬件是否满足可以运行的条件，比如内存是否能用。如果硬件出现问题，则蜂鸣器就会发出代表不同含义的蜂鸣声，然后终止启动过程。否则，说明硬件没有问题，可以继续运行。

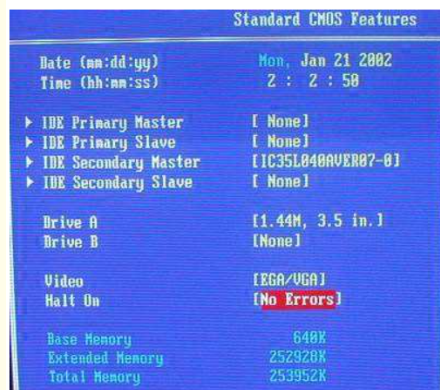
如果这时我们按下了一些特殊的按键（有的电脑需要按 F1，有的需要 Del 等），就会进入到 BIOS 中。我们可以从中看到启动顺序表，BIOS 会首先从表中的第一个设备开始读取：



这个启动顺序如果想更改的话，我们可以在 BIOS 中设置，不过此时我们要设置的并不是 BIOS，而是 CMOS（本意是指互补金属氧化物半导体存储器，是一种大规模应用于集成电路芯片制造的原料 [2]）。我们通常会说是进入 BIOS 进行设置 [2]，这是因为 CMOS 一般会被集成到 BIOS 芯片内部。

Explanation 1.1 (CMOS 的解释) CMOS 通常全名叫 CMOS RAM，是一块可读写的 RAM 芯片，由后备电池供电，因此信息不会消失，它里面包含了很多可以我们自由设置的信息。当我们设置了以后，BIOS 就可以从中读取一些信息，比如下一步应该首先从哪个硬件设备来读取和运行程序。由于可以进行读写，所以可以用户自定义的内容就可以设置和保存在 CMOS 里。

首先注意下图的 IDE 设备，电脑里有两根 IDE 数据线，而每根 IDE 数据线又可以同时接两个硬盘或者一个硬盘一个光驱，所以总共可以接 4 个设备，如果在一根线上接了两个设备，就有主从关系，谁是主设备 (Master)，谁是从设备 (Slave)。然后再注意 Drive A，这是一个软盘，标准的英特尔 1.44M，3.5 寸软盘（又叫软驱），现在很少有人安装这个东西，但是下图中所述的电脑安装了它。Drive B 是预留的 5.2 寸软驱，下图中所述的电脑并没有安装它。



Standard CMOS Features	
Date (mm:dd:yy)	Mon, Jan 21 2002
Time (hh:mm:ss)	2 : 2 : 58
IDE Primary Master	[None]
IDE Primary Slave	[None]
IDE Secondary Master	[IC35L040AVER07-01]
IDE Secondary Slave	[None]
Drive A	[1.44M, 3.5 in.]
Drive B	[None]
Video	[EGA/VGA]
Halt On	[No Errors]
Base Memory	640K
Extended Memory	252928K
Total Memory	253952K

BIOS 中的系统设置程序是进行 CMOS 参数设置的手段 [2], CMOS RAM 既是 BIOS 设定系统参数的存放场所, 又是 BIOS 设定系统参数的结果。因此, 完整的说法应该是“通过 BIOS 设置程序对 CMOS 参数进行设置”。由于 BIOS 和 CMOS 都跟系统设置密切相关, 所以在实际使用过程中造成了 BIOS 设置和 CMOS 设置的说法, 其实指的都是同一回事, 但 BIOS 与 CMOS 却是两个完全不同的概念, 不可混淆。

我们已经说过, CMOS 中存储了外部存储设备的排序, 叫做启动顺序。BIOS 根据设置的启动顺序来选择首先读取的硬件存储设备。计算机会首先读取该存储设备的第一个扇区, 也就是该设备的最前面的 512 个字节。如果这最后两个字节是 0x55 和 0xAA, 则说明这个设备可以用来启动, 否则就从启动顺序中找下一个设备进行启动。0x55 和 0xAA 的二进制表示为 (01010101) 和 (10101010), 也就是 0 和 1 交替出现, 这样在通信中可以提高错误的检出率, 所以经常被沿用为一段二进制码的结尾标志。

BIOS 当前已经知道了下一个阶段的启动程序 (这个程序并不一定是操作系统 (OS), 还可能是其他程序, 比如操作系统加载程序 BootLoader) 在哪个设备里, 并且读取了出来 (最后两个字节校验正确), 现在可以去运行这里的程序了。

关于操作系统加载程序 BootLoader

做过嵌入式系统的朋友应该听说过系统启动的 BootLoader, 由于嵌入式系统很多硬件设备都是我们自己设计和添加的, 所以 BootLoader 也经常需要自己去写或修改, BootLoader 会进行一些基本的硬件初始化工作, 并引导和运行操作系统。

现代 PC 机中一般没有单独的 BootLoader 程序, 而是放入了 BIOS 中, 操作系统由 BIOS 来启动。这是由于现代 PC 机一般都是固定厂家生产的, 主板结构也很复杂, 自己改造主板并不现实, 所以我们不会自己去写 BIOS。

在一些嵌入式系统中, 开机启动的代码并不包含 BootLoader, 而是去判断 BootLoader 和 OS 在哪个外设中 (比如是否在 SD 卡或者 NandFlash 中), 然后再去将这里的 BootLoader 读出, 去做一些事情。比如有的 BootLoader 会直接去把动态内存 (DRAM) 进行初始化, 这样就能把庞大的 OS 读入到 DRAM (叫做运行内存, 容量比较大, 比较流行的大小是 4GB、8GB) 中去执行。

1.3 主引导记录 MBR

“主引导记录 (Master Boot Record)”就是 BIOS 读取的可以用于启动的第 0 个扇区的内容，只有 512 个字节：

- 第 0-445 字节表示调用操作系统的机器码。
- 第 446-509 字节表示分区表 (Partition table)。
- 第 510-511 字节就是主引导记录签名，也就是校验位 (0x55 和 0xAA)。

一个硬盘就是一个设备，分区表就是把这个设备进行分区，每个区都可以装不同的操作系统。分区表一共有 64 个字节，分为四项，每项有 16 个字节，也就是说一个硬盘只能分为四个区（这四个区都是一级分区，又叫主分区）。每个主分区的 16 个字节由下面的 6 个部分组成：

- 第 0 个字节如果是 0x80，则该分区是激活分区，控制权要交给这个分区。四个主分区里只能有一个激活分区。
- 第 1-3 个字节是主分区第一个扇区的物理位置（包括柱面、磁头和扇区号，这个第二章会介绍）。
- 第 4 个字节是主分区类型（简单来说，就是这个主分区的类别 ID，不同的 ID 对应了不同的公司和标准）。
- 第 5-7 个字节是这个主分区最后一个扇区的物理位置。
- 第 8 到 11 字节表示这个主分区第一个扇区的逻辑地址。扇区的逻辑地址是 2^{32} 位。
- 第 12-15 字节表示这个主分区的扇区总数。由于是四个字节，所以扇区数不能超过 2^{32} 个，考虑到每个扇区的字节数为 512，也就是说一个分区最大不能超过 2TB，这与前面的逻辑地址也是相同的。

Explanation 1.2 (U 盘启动) 经常会有人制作 U 盘启动盘来装系统，也就是说我们可以让 BIOS 从 U 盘来启动系统。

在网上搜索系统盘制作工具，系统盘制作工具一般都是先把 U 盘进行分区，然后激活分区并写入启动文件。制作完成以后，启动分区就会被隐藏起来（避免被破坏）。

我们下一章会详细介绍另一种启动方式，UEFI，它是为了解决 BIOS 启动的一些问题而设计的。如果你的电脑并不是 UEFI 启动，则下一章你可以跳过，或者简单了解一下即可。

2. UEFI 与 BIOS

2.1	传统模式与 UEFI 模式简述	10
2.2	MBR 的问题	11
2.3	GPT 分区格式	11
2.4	UEFI 启动过程	13
2.5	UEFI 与 CSM 模式和安全启动	15

本章详细介绍 UEFI 模式和 BIOS 的区别与联系。

2.1 传统模式与 UEFI 模式简述

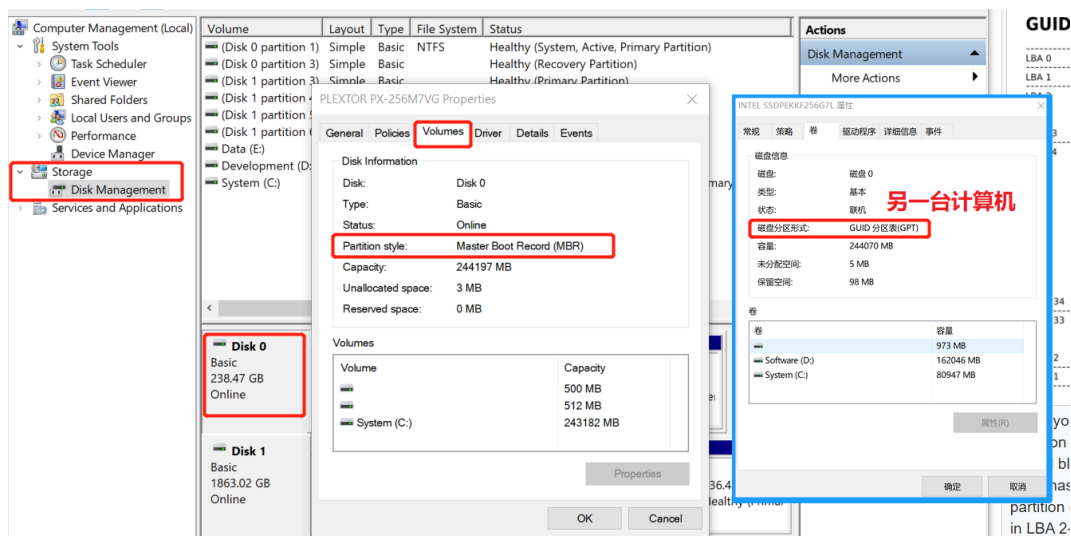
上面所述的 BIOS 启动是属于传统模式的方法，本节我们介绍最主要的两种硬盘系统启动方式：传统模式 (Legacy) 和 UEFI(Unified Extensible Firmware Interface, 统一的可扩展固件接口) 模式。

Legacy 模式对应于 MBR (Master Boot Record, 主引导记录) 硬盘，是磁盘驱动器最开始的一个扇区 (也就是前面所说的第 0 个扇区，一共 512 个字节的数据)，这个扇区里包含了已安装的操作系统信息，并且用 0-445 的字节机器码来启动系统。(Linux 系统的 MBR 通常是 GRUB 加载器，类似于 U-boot 这种 BootLoader)。BIOS 固件其实只是识别系统包含的磁盘，在执行启动装载程序之后，BIOS 固件本身就不再参与启动。

UEFI 模式对应于 GPT (GUID(Globally Unique Identifier) Partition Table) 硬盘，即全局唯一标识磁盘分区表，这是更先进的磁盘组织形式。MBR 分区只支持最大 2TB 的磁盘，而 GPT 分区理论上是不受限的。MBR 最多支持四个主分区，而 GPT 则理论上支持无限多个分区。

我们把传统的方式称为 BIOS 固件 (firmware)，它主要是用于 IBM PC 兼容计算机；而 UEFI 具有更强的通用性，可用于各种非 IBM PC 兼容 (比如 ARM 核心的计算机) 的计算机上。UEFI 的前身是 EFI，这是由 Intel 开发的，而之后 UEFI Forum 来管理它 [5]。

我们可以找到磁盘管理器，然后右键某个磁盘查看属性，来确认我们自己的电脑上的磁盘是 MBR 分区还是 GPT 分区：



2.2 MBR 的问题

我们需要特殊工具来写入 MBR，如果要查看 MBR 中包含的内容，唯一的方法几乎就是把 MBR 拷贝出来然后进行检查（注意这玩意并不在分区里）。

MBR 本身只有 512 个字节，系统引导部分的 446 个字节肯定不足以容纳许多现代启动装载程序（操作系统引导代码）。这些启动装载程序会将自身的一小部分安装在 MBR 中，而将其他部分安装到磁盘上的可用空间中——这段可用空间位于常规 MBR 末尾和第一个分区的起始位置之间（需要注意，一般 MBR 后面并不会直接跟着第一个分区，这是可以自己设定的）。这就会造成很大的问题（其实这整个设计就是个大问题）：对于第一个分区的起始位置，并没有可靠的规定，因此难以确保空间足够。只有一件事情是肯定的：这段空间不足以容纳某些启动装载程序的配置。

同时，MBR 没有提供从除了磁盘以外的设备（例如服务器远程启动）的标准和规范，只能说大家爱怎么设计就怎么设计，非常混乱（UEFI 提供了标准方法）。

此外，如果用户想选择从 U 盘启动 OS 而不是磁盘，MBR 模式唯一的方法就是由启动装载程序（在 BIOS 里选择）进行处理，但是并没有很好的规定和标准。注意，编写启动程序难度很大，对于想有多个可选的启动对象的实现而言，MBR 没有很好的标准和方法。

2.3 GPT 分区格式

在继续了解 UEFI 之前，我们还是先了解 GPT 格式。GPT 的分区格式包含传统的 MBR、分区表头、分区表、备份分区表头、备份分区表以及数据区。

传统的寻址方式是 CHS (cylinders-heads-sectors, 磁柱-磁头-扇区)，也就是说区块的地址必须由这三个分量构成。后来使用 4 个字节的 LBA(Logical Block Address, 逻辑区块地址)，LBA 非常简单：从 0 开始编号来定位区块，第一区块 LBA=0，第二区块 LBA=1。MBR 和 GPT 都采样了 LBA 寻址方式。

传统的 MBR 会放入 LBA 0 处，以防不支持 GPT 的硬盘管理软件错误识别并破坏硬盘数据 [7]。在这个 MBR 中，只有一个标志为 0xEE 的分区（在第 450 个字节，是第一个分区的第 4 个

表 2.1: 分区表头

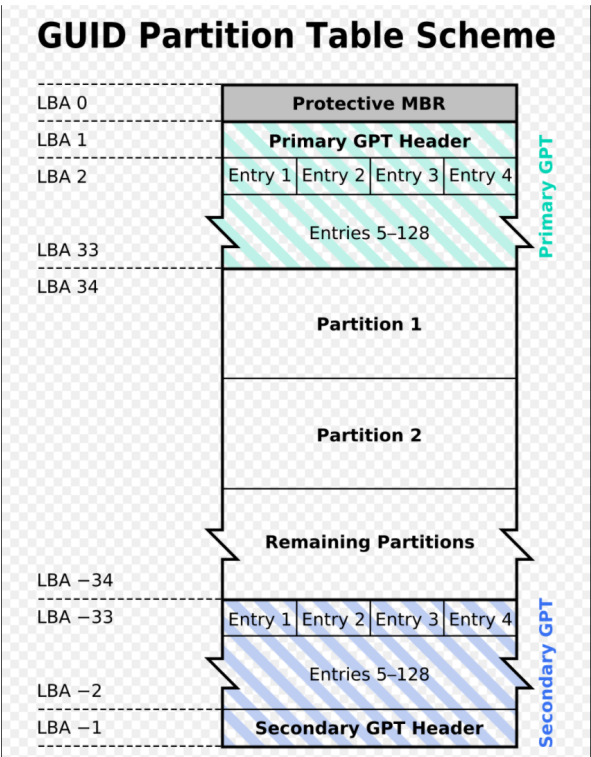
起始字节	长度	内容
0	8 字节	签名 (“EFI PART”, 45 46 49 20 50 41 52 54)
8	4 字节	修订 (在 1.0 版中, 值是 00 00 01 00)
12	4 字节	分区表头的大小 (单位是字节, 通常是 92 字节, 即 5C 00 00 00)
16	4 字节	分区表头 (第 0 – 91 字节) 的 CRC32 校验
20	4 字节	保留, 必须是 0
24	8 字节	当前 LBA (这个分区表头的位置)
32	8 字节	备份 LBA (备份分区表头的位置)
40	8 字节	第一个可用于分区的 LBA (主分区表的最后一个 LBA + 1)
48	8 字节	最后一个可用于分区的 LBA (备份分区表的第一个 LBA – 1)
56	16 字节	硬盘 GUID (在类 UNIX 系统中也叫 UUID)
72	8 字节	分区表项的起始 LBA (在主分区表中是 LBA 2)
80	4 字节	分区表项的数量
84	4 字节	一个分区表项的大小 (通常是 128 字节)
88	4 字节	分区序列的 CRC32 校验
92	*	保留, 剩余的字节必须是 0

字节, 也就是这个主分区的类型, 大家可以去第一章查看), 以此表示这块硬盘使用 GPT 分区格式。不支持 GPT 分区格式的软件, 会识别出未知类型的分区; 支持 GPT 分区格式的软件, 可正确识别 GPT 分区磁盘。

GPT 分区表头被存储在 LBA 1 处, 分区表头定义了硬盘的可用空间以及组成分区表的项的大小和数量, 比如最多可以创建多少个分区、每个分区表大小是多少个字节 (一般是 128 个字节)。分区表头占据了一个扇区, 但是只用到了 92 个字节, 其他的字节保留, 并必须为 0 (对于一个扇区 LBA 是 512 字节的分区表头来说, 最后 420 个字节都是 0)。

上表为分区表头各个字节的意义。注意分区表头储存着表头本身和分区表序列的 CRC32 校验 (在计算时, 把 CRC32 校验字段作为 0 处理, 需要计算出分区序列的 CRC32 校验后再计算本字段)。固件、引导程序和操作系统在启动时可以根据这个校验值来判断分区表是否出错, 如果出错了, 可以使用软件从硬盘最后的备份 GPT 中恢复整个分区表, 如果备份 GPT 也校验错误, 硬盘将不可使用。备份分区表头位于硬盘最后一个扇区, 备份分区表头中的信息是关于备份分区表的, 可以找到备份分区表的位置。

我们再来对照一下 [9] 给出的 GPT 表:



注意上表的每一个 Entry 都是一个分区表，每个分区表是 128 个字节。

2.4 UEFI 启动过程

同 BIOS 一样，UEFI 也是一种主板引导项（严格来说是一个标准），只是它相对来说更复杂。

在这里我们不去按照它的各个启动步骤这样一步一步地来介绍，因为这并不能让我们很轻松的就了解 UEFI 启动机制。通常情况下我们只需要简单了解大体过程即可。

- 通电以后，首先进行安全验证，该阶段会在 Cache 上开辟一段空间作为内存（由于此时内存并没有初始化，所以还不能运行 C 语言风格生成的代码，开辟内存以后才可以）。
- 之后就是对主板硬件（CPU 和内存等）的初始化，并加载固件中所有的驱动，完成驱动设备初始化。
- 之后由用户指定操作系统的启动项，加载操作系统，然后操作系统就开始执行了。操作系统完全获得控制权以后，会回收一些 UEFI 占用资源。

UEFI 可以运行 C 风格的代码，所以可以做更多事情，比如 UEFI 固件可以读取 FAT 格式（及其变种）的 filesystem 类型。

我们知道，BIOS 会从 MBR 的前 445 个字节开始引导系统，那么 UEFI 启动也是类似的，它会在标准位置查找启动装载程序代码。固件会按照磁盘上的分区顺序遍历磁盘上的每个 EFI 系统分区（EFI 分区，即 EFI system partition，一般简写成 EFI 分区或者 ESP），UEFI 固件会查找文件 \EFI\BOOT\BOOT{计算机类型简称（对于 x86-64 的 PC 为 x64，对于 32 位 ARM 为

ARM) } .EFI (或者 .efi), UEFI 固件将执行找到的第一个有效文件 (文件需要符合 UEFI 规范中定义的可执行格式)。

Explanation 2.1 (EFI 系统分区)

EFI 系统分区 (ESP) 是一个使用 FAT32 格式化的
小分区, 通常为 100MB, 存储已安装系统的 EFI 引导加载程序以及启动时固件使用的应用程序。
如果硬盘驱动器为 GPT 格式, 它将在安装 Windows 或 Mac 操作系统 (OS) 后生成
EFI 系统分区:

卷	布局	类型	文件系统	状态	容量	可用空间	% 可用
— (磁盘 0 磁盘分区 1)	简单	基本		状态良好 (EFI 系统分区)	98 MB	98 MB	100 %

我们早已知道, BIOS 是没有办法引导装在 GPT 磁盘上的操作系统的, 必须要是
EFI 才能引导。不同操作系统使用 GPT 进行 EFI 分区的情况都是不同的: Mac 在创建
GPT 磁盘时都会在 LBA40 扇区开始分 209MB 出来作为 EFI 系统分区; 微软则会有一个
LBA2048 开始的 100MB EFI 系统分区和 128MB 的 MSR 保留分区。

UEFI 固件可以识别 FAT 文件系统格式, 因此可以找到根目录并找到要启动的 .efi 文
件。主板 UEFI 初始化后, 就可以找到默认启动项 Windows Boot Manager[11] 然后运行这
里的 .efi 文件。

与 BIOS 的区别

同 BIOS 一样, 用于 UEFI 启动的程序肯定是一通电就会自动执行的程序, 因此也会放在类
似 BIOS 芯片的硬件中。现代的主板很多都是 BIOS-UEFI 主板。有人看到这里可能会有些不解,
这个启动过程不是跟 BIOS 差不多吗? 先初始化硬件, 再启动 OS。实际上就是没有本质区别, 都
是为了引导操作系统嘛。但我们还可以比较一下他们的其他区别。

UEFI 规范定义了一种可执行文件格式, 并要求所有 UEFI 固件能够执行此格式的代码。当
开发人员为纯粹的 UEFI (naive UEFI, 表示非 BIOS-UEFI 混合的、单纯使用 UEFI 的启动方
式) 编写启动装载程序时, 就必须按照这种格式编写。

- UEFI 是用模块化, C 语言风格的参数堆栈传递方式, 动态链接的形式构建的系统, 较 BIOS
而言更易于实现, 容错和纠错特性更强, 缩短了系统研发的时间 (前面说过, BIOS 的开发
很困难)。
- UEFI 可以达到处理器的最大寻址。
- UEFI 利用加载 UEFI 驱动程序的形式, 识别及操作硬件, 不同于 BIOS 利用挂载真实模式
中断的方式增加硬件功能。
- 在 BIOS 中添加几个简单的 USB 设备支持都非常困难。
- UEFI 可以使用图形化界面。

与操作系统的关系

UEFI 在概念上类似于一个低阶的操作系统 [9], 并且具有操控所有硬件资源的能力。

不少人感觉它的不断发展将有可能代替现代的操作系统，而事实上，EFI 的缔造者们在第一版规范出台时就将 EFI 的能力限制于不足以威胁操作系统的统治地位：

- UEFI 只是硬件和预启动软件间的接口规范。
- UEFI 环境下不提供中断的机制，也就是说每个 UEFI 驱动程序必须用轮询（polling）的方式来检查硬件状态，并且需要以解释的方式运行，较操作系统下的机器码驱动效率更低。
- UEFI 系统不提供复杂的缓存器保护功能，它只具备简单的缓存器管理机制。

2.5 UEFI 与 CSM 模式和安全启动

BIOS 的 Boot 选项中有一个关于 UEFI 的子项目：CSM（Compatibility Support Module，兼容支持模块）。

许多 UEFI 固件实现了某种 BIOS 兼容模式，即 CSM。许多 UEFI 固件可以像 BIOS 固件一样启动系统，它们可以查找磁盘上的 MBR，然后从 MBR 中执行启动装载程序，接着将后续工作完全交给启动装载程序。有时候，其他人误将此功能称为“禁用 UEFI”，其实是不对的，因为固件不能被禁用。如果我们选择禁用 CSM，则系统只能用 UEFI 方式来启动，否则就可以使用传统的 MBR 方式来启动。所以可以说，使用传统方式来启动系统是 UEFI 的一个可支持的功能选项。

在启动时，有时候我们会看到“安全启动 (Secure Boot)”，它和 CSM 属于并列项，它与 UEFI 并不是同一个概念，它其实是 UEFI 规范的一个可选功能。安全启动其实很复杂，里面有一堆需要注意的条款和条例，所以这里不再讲解，只需要知道，它是为一系列安全问题（比如文件正确性）设计的。

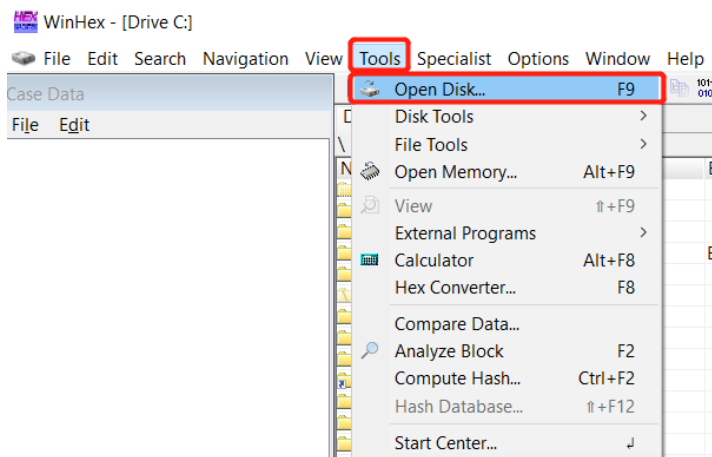
3. 磁盘结构查看与验证

3.1	BIOS 启动与 MBR 磁盘	16
3.2	UEFI 启动与 GPT 磁盘	17
3.3	总结	20

本章描述如何通过查看磁盘内容来熟悉上面的介绍。

3.1 BIOS 启动与 MBR 磁盘

我们下载一下 winhex 软件 [12]，该软件可以免费使用 45 天。我们用管理员运行，就可以打开磁盘：



然后打开物理硬盘：



我的磁盘是 MBR 格式的，打开以后看到下面的表（这里只截取了前 512 个字节，我把不同的部分标注为了不同的颜色）：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	ANSI ASCII		
0000000000	33	C0	8E	D0	BC	00	7C	8E	C0	8E	D8	BE	00	7C	BF	00	06	B9	00	02	FC	F3	A4	50	68	1C	06	CB	FB	B9	04	00	3A2D4	12A204	l; i u0=Ph E01
0000000020	BD	BE	07	80	7E	00	00	7C	0B	0F	85	0E	01	83	C5	10	E2	F1	CD	18	88	56	00	55	C6	46	11	05	C6	46	10	00	44	e~ ... fA Afí 'v UZF EF	
0000000040	B4	41	BB	AA	55	CD	13	5D	72	0F	81	FB	55	AA	75	09	F7	C1	01	00	74	03	FE	46	10	66	60	80	7E	10	00	74	'Ae*Uí jr GU*u +Á t bF f'e~ t		
0000000060	26	66	68	00	00	00	00	66	FF	76	08	68	00	00	68	00	7C	68	01	00	68	10	00	B4	42	8A	56	00	8B	F4	CD	13	afh	fyv h h h h 'BSV <ôf	
0000000080	9F	83	C4	10	9E	EB	14	B8	01	02	BB	00	7C	8A	56	00	8A	76	01	8A	4E	02	8A	6E	03	CD	13	66	61	73	1C	FE	YfA Ze , » ŠV Šv ŠN Šn i fas p		
00000000A0	4E	11	75	0C	80	7E	00	80	0F	84	8A	00	B2	80	EB	84	55	32	E4	8A	56	00	CD	13	5D	EB	9E	81	3E	FE	7D	55	N u e~ e „Š eee„U2aŠv i jež >p)U		
00000000C0	AA	75	6E	FF	76	00	E8	8D	00	75	17	FA	B0	D1	E6	64	E8	83	00	B0	DF	E6	60	E8	7C	00	B0	FF	E6	64	E8	75	*unyv è u ū*Nadèf "Ba"èl "yedèu		
00000000E0	00	FB	B8	00	BB	CD	1A	66	23	C0	75	3B	66	81	FB	54	43	50	41	75	32	81	F9	02	01	72	2C	66	68	07	BB	00	ù „xif #Au;f ūTCPAu2 ū r„fh »		
0000000100	00	66	68	00	02	00	00	66	68	08	00	00	00	66	53	66	53	66	55	66	68	00	00	00	00	66	68	00	7C	00	00	66	fh fh fšfšTŪfh fh l f		
0000000120	61	68	00	00	07	CD	1A	5A	32	F6	EA	00	7C	00	00	CD	18	A0	B7	07	EB	08	A0	B6	07	EB	03	A0	B5	07	32	E4	ah i 22èè i . e q e p 2a		
0000000140	05	00	07	8B	F0	AC	3C	00	74	09	BB	07	00	B4	0E	CD	10	EB	F2	F4	EB	FD	2B	C9	E4	64	EB	00	24	02	E0	F8	<ô-< t » ' i èòèy+Èadè Š àe		
0000000160	24	02	C3	49	6E	76	61	6C	69	64	20	70	61	72	74	69	74	69	6F	6E	20	74	61	62	6C	65	00	45	72	72	6F	72	\$ ÅInvalid partition table Error		
0000000180	20	6C	6F	61	64	69	6E	67	20	6F	70	65	72	61	74	69	6E	67	20	73	79	73	74	65	6D	00	4D	69	73	73	69	6E	loading operating system Missin		
00000001A0	67	20	6F	70	65	72	61	74	69	6E	67	20	73	79	73	74	65	6D	00	00	00	63	7B	9A	B5	7C	FB	76	00	00	80	20	g operating system c(è)ŭv e		
00000001C0	21	00	07	DD	1E	3F	00	08	00	00	00	A0	0F	00	00	DD	1F	3F	07	FE	FF	FF	00	A8	0F	00	0E	72	AF	1D	00	FE	i Y ? ŷ ? byy " r" b		
00000001E0	FF	FF	27	FE	FF	FF	00	20	BF	1D	00	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55 AA	yy'byy ; B*	

可以看到，最后两个字节就是 0x55 和 0xAA。前面不同颜色的部分表示四个分区表的信息。可以看到第一个分区表的第 0 个字节是 0x80，表示激活的分区；其他分区表的第 0 个字节是 0x00，表示未激活。

我们看一下分区，最开始的地方就是 Start sectors，它一共有 1.0M 的字节：

Drive C: Hard disk 0							
Partitioning style: MBR							
Name	Ext.	Size	Created	Modified	Record changed	Attr.	1st sector ^
Start sectors		1.0 MB					0
Partition 1	NTFS	500 MB					2,048
Partition 2 (C:)	NTFS	237 GB					1,026,048
Partition gap		761 KB					499,063,3...
Partition 3	NTFS	512 MB					499,064,8...
Unpartitionable space		2.3 MB					500,113,4...

而第一个分区的起始扇区是 2048，考虑到一个扇区是 512 字节，所以其实正好就对应了这个 1.0M 字节。

再看分区表中第一个分区的信息，最后四位 00 A0 0F 00 表示扇区总数，这里应该这么去理解：000FA000，大小正好是 1000x1024 个扇区，由于每个扇区是 512 个字节，所以正好是 500MB 的大小。然后再往前数 4 位，即 00 08 00 00 表示该主分区第一个扇区的逻辑地址，即 00000800，8x256x512 字节，等于 1024x1024 字节，正好是 1MB。

3.2 UEFI 启动与 GPT 磁盘

对于 UEFI 启动的系统，我们使用 winhex 打开系统所在的硬盘，字节如下：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		ANSI	ASCII
0000000000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000001C0	02	00	EE	FF	FF	FF	01	00	00	00	FF	FF	FF	FF	00	00		û Ås	
00000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		ïyyy	yyyy
00000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
00000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			U*

在第一个分区的第 4 个字节为 0xEE，符合前面所述的 MBR 保留区的特征。
磁盘整体的分区结构如下：

文件名称	扩展名	文件大小	创建时间	修改时间	记录更新时间	文件属性	第1扇区
起始扇区		1.0 MB					0
分区 1	FAT32	300 MB				H	2,048
分区 2	?	128 MB				H	616,448
分区 3 (C:)	NTFS	182 GB					878,592
分区间隙		930 KB					382,107,...
分区 4	NTFS	634 MB				SH	382,109,...
分区间隙		1.0 MB					383,408,...
分区 5 (F:)	NTFS	97.7 GB					383,410,...
分区间隙		1.0 MB					588,208,...
分区 6	NTFS	900 MB				SH	588,210,...
分区 7 (D:)	NTFS	176 GB					590,053,...
分区 8	NTFS	19.1 GB				S	960,196,...
分区间隙		344 KB					1,000,21,...

我们说过，从 LBA 1 开始就是分区表头：

0000000200	45 46 49 20 50 41 52 54	00 00 01 00 5c 00 00 00	EFI PART \
0000000210	58 68 ED FB 00 00 00 00	01 00 00 00 00 00 00 00	Xhiú
0000000220	AF 12 9E 3B 00 00 00 00	22 00 00 00 00 00 00 00	— ž; "
0000000230	8E 12 9E 3B 00 00 00 00	C3 F8 B1 08 22 D3 F7 40	ž ž; Å± "Ó÷@
0000000240	BC 89 3B 52 F2 93 E3 AB	02 00 00 00 00 00 00 00	4; Rò"ā«
0000000250	80 00 00 00 80 00 00 00	7B 65 72 5E 00 00 00 00	€ € {er^
0000000260	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000270	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000280	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000290	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002B0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000002E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

分区表头的结构在第一章我们已经进行了详细的描述，这里我们再重复一下某些内容。对照一下就可以看到，分区表数量为 0x80，也就是 128 个（但我们可能并不会分这么多区）。同时，每个分区表大小也是 0x80，即 128 字节（我们下面会描述这 128 个字节分别代表什么含义）。分区表起始的 LBA 为 2，这也符合我们前面的描述（前面有个 GPT 表图，可以看到第一个 Entry（也就是第一个分区表）就是在 LBA 2 的位置）。

现在开始看第一个分区表，在看之前，我们先给出分区表的结构，参照表 3.1。

我只列出两个分区表的项，可以看到，这两个分区表一个名字为系统分区，另一个是微软保

表 3.1: 分区表

起始字节	长度	内容
0	16 字节	分区类型 GUID
16	16 字节	分区 GUID
32	8 字节	起始 LBA（小端序）
40	8 字节	末尾 LBA
48	8 字节	属性标签（如：60 表示“只读”）
56	72	分区名（可以包括 36 个 UTF-16（小端序）字符）

留分区。

0000000400	28 73 2A C1 1F F8 D2 11	BA 4B 00 A0 C9 3E C9 3B	(s*Ã øÛ °K É>É;
0000000410	BE 4D AC 52 2F 56 78 4B	A5 13 5A 4B 16 3B 60 19	¼M~R/VxK¥ ZK ;`
0000000420	00 08 00 00 00 00 00 00	FF 67 09 00 00 00 00 00	ÿg
0000000430	00 00 00 00 00 00 00 80	45 00 46 00 49 00 20 00	€E F I
0000000440	73 00 79 00 73 00 74 00	65 00 6D 00 20 00 70 00	s y s t e m p
0000000450	61 00 72 00 74 00 69 00	74 00 69 00 6F 00 6E 00	a r t i t i o n
0000000460	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000470	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000480	16 E3 C9 E3 5C 0B B8 4D	81 7D F9 2D F0 02 15 AE	ãĖã\ ,M }ù-ð @
0000000490	68 D4 1D 01 DF 74 E9 46	A5 AF EF 7F 9D 02 58 5C	hÔ ßtéF¥~i X\
00000004A0	00 68 09 00 00 00 00 00	FF 67 0D 00 00 00 00 00	h ÿg
00000004B0	00 00 00 00 00 00 00 80	4D 00 69 00 63 00 72 00	€M i c r
00000004C0	6F 00 73 00 6F 00 66 00	74 00 20 00 72 00 65 00	o s o f t r e
00000004D0	73 00 65 00 72 00 76 00	65 00 64 00 20 00 70 00	s e r v e d p
00000004E0	61 00 72 00 74 00 69 00	74 00 69 00 6F 00 6E 00	a r t i t i o n
00000004F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

我们先来看第一个分区表，末尾 LBA 是 0x967FF，起始 LBA 是 0x800，用末尾 LBA 减去起始 LBA 再加 1，就能得到 0x96000，换算成十进制是 614400，再乘以 512 就是 314572800 个字节。我们看前面的硬盘分区图的分区 1，大小是 300MB，300x1024x1024=314572800。所以验证成功。

我们再来看看第二个分区表，末尾是 0x0D67FF，起始是 0x096800，总共 262144 个 LBA，也就是 134217728 个字节，等同于 128MB，也就是微软保留区。我们可以看到，和上面的分区二完全一致。

我们看到第一个分区是从 LBA 2 开始的（从第 2048 个字节开始），而前面我们说过，微软会留出 100MB 的 EFI 系统分区和 128M 的微软保留分区，这样看来，有些时候也是会有例外的，比如这里的 EFI 系统分区就是 300MB。我们可以通过 Windows 的磁盘管理器来确认：

卷	布局	类型	文件系统	状态	容量	可用空间
(D:)	简单	基本	NTFS	状态良好 (基本数据分区)	176.50 GB	116.66 GB
(磁盘 0 磁盘分区 1)	简单	基本		状态良好 (EFI 系统分区)	300 MB	300 MB
(磁盘 0 磁盘分区 4)	简单	基本		状态良好 (恢复分区)	634 MB	634 MB
(磁盘 0 磁盘分区 6)	简单	基本		状态良好 (恢复分区)	900 MB	900 MB
(磁盘 0 磁盘分区 8)	简单	基本		状态良好 (恢复分区)	19.08 GB	19.08 GB
Windows (C:)	简单	基本	NTFS	状态良好 (启动, 页面文件, 故障转储, 基本数据分区)	181.78 GB	102.03 GB

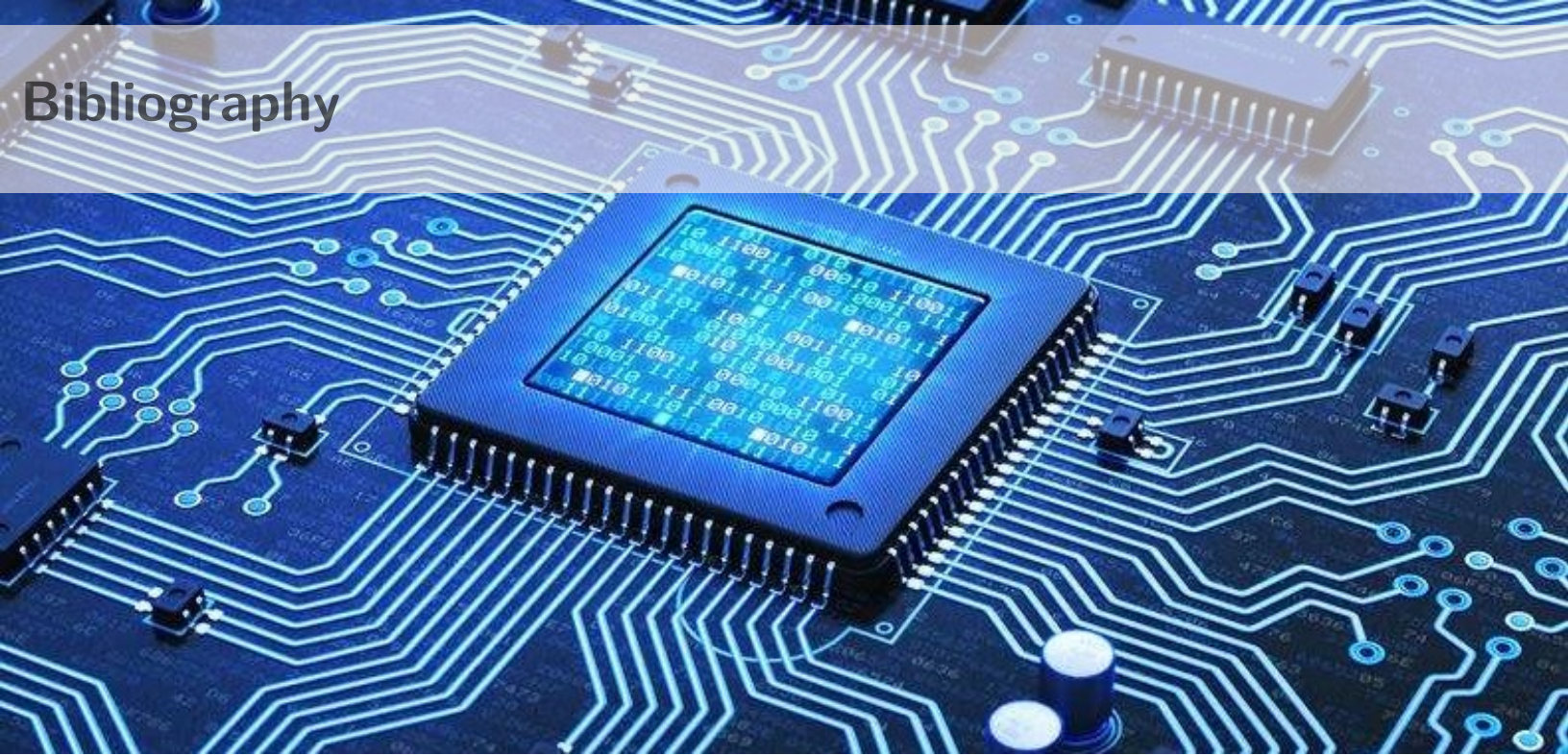
我们还需要注意的是，分区一的扩展名是 FAT32，这也符合 UEFI 启动对于该区域文件系统的要求。

3.3 总结

到这里，你可能会出现不少疑问，这是很正常的——对于一个复杂的东西，我们每了解其中的一部分内容，就又会有一堆新的东西等待着我们去研究。

为了能更好的掌握本文的内容，一定要多去尝试，用 winhex 以及磁盘管理器、磁盘精灵等软件去分析和理解。同时，不要在意自己哪些地方没弄懂，而是要在意自己学会了什么——这样才能不断增强自己的信心。

本文的写作计划来源于我正在进行的项目——制作一个可运行在现代 PC 上的操作系统。我认为，了解现代 PC 的启动过程是非常重要的，这对我们了解自己的电脑、学会管理和优化自己的电脑都很有意义。



Bibliography

- [1] <http://www.ruanyifeng.com/blog/2013/02/booting.html>
- [2] <https://product.pconline.com.cn/itbk/software/dnwt/1310/3602136.html>
- [3] http://www.360doc.com/content/18/0901/23/11935121_783145790.shtml
- [4] <https://www.happyassassin.net/posts/2014/01/25/uefi-boot-how-does-that-actually-work-then/>
- [5] <https://uefi.org/>
- [6] <https://www.cnblogs.com/mahuan2/p/4801218.html>
- [7] <https://www.cnblogs.com/cwcheng/p/11270774.html>
- [8] <https://zh.wikipedia.org/wiki/GUID>
- [9] https://en.wikipedia.org/wiki/GUID_Partition_Table
- [10] http://blog.sina.com.cn/s/blog_be992a830102vua4.html
- [11] <https://www.cnblogs.com/pipci/p/13280030.html>
- [12] <http://www.x-ways.net/winhex/>