UEFI Development Exploration 34 – Option ROM Prequel 1

原创 luobing4365 Posted on 2019-10-15 23:24:26 Read 1.9k Collection 17 Likes 2

Category columns: UEFI Development Article Tags: UEFI Programming Option ROM Low-level programming Embedded in BIOS Assembly language

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

Now let's move on to the main theme of this blog series - the development of Option ROM.

I plan to explain this issue in 3 articles, including Oprom under Legacy BIOS, and how to build Oprom under UEFI, including several tools I made for developing Oprom.

I once dealt with the Option ROM of Legacy BIOS for a long time. In the absence of sample programs and no one to guide me, I struggled to explore it. It was painful but fulfilling.

It all starts with this document:

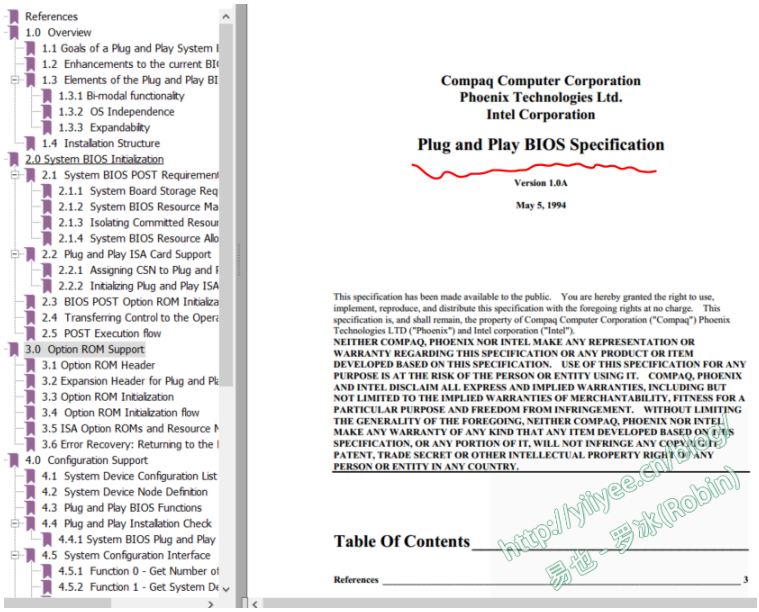

Figure 1 PNP BIOS spec

1 Origin

About two years after graduation, the company planned to develop a physically isolated motherboard. Our chief engineer only had experience in board development and had never developed such a complex product. Moreover, one of the product goals was to modify the ACPI process. None of us had relevant experience and we were all confused.

I will not discuss the hardware work for now, the software is my responsibility. It probably includes:

1) MCU firmware code as the protocol center provides a communication channel between Windows App and the board, and provides a communication channel between PCI Option ROM and the board;
2) Windows App provides a user interface at the operating system layer;
3) PCI Option ROM provides a bottom-level user interface.

After further discussion, it was found that PCI Option ROM could not be used because it would require adding a WCH365 costing nearly 20 yuan, and the driver code was found to be unstable.

Finally, we can only consider embedding Option ROM into BIOS and using a Cypress chip to open two communication channels at the same time.

Just like that, without any relevant knowledge (I hadn't even learned assembly language at the time), I plunged into this huge problem vortex, overcoming various problems encountered along the way bit by bit, until I miraculously made the first Oprom that printed "Hello, world!"

2. Explore and think

I searched all the libraries in Nanjing, and all the reference books I could find. Finally, I saw in a computer magazine in the library of my alma mater that the BIOS file can be extracted through Cbrom.exe.

I read the article again and again as if I had found a treasure. Then I found a tool called bfdisk on the Internet. In the article introducing this tool, there is this sentence (marked in red in the picture):

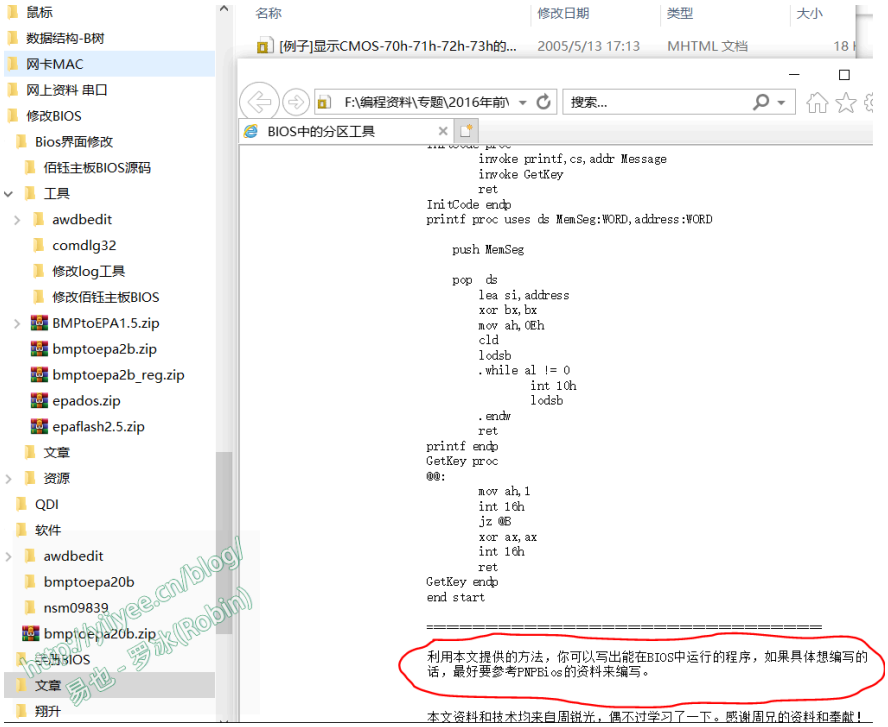

Figure 2 Partitioning tool embedded in BIOS

It was this sentence that helped me find the basis for how to write the Oprom format.

Of course, even if you know the format, there are still many problems to be solved, such as writing code in assembler, converting compiled files to ROM files, image programming knowledge, various peripheral access methods, etc.

This UEFI exploration blog is actually similar to what I did back then, except that I now have much more knowledge than I did back then, and can quickly get into various UEFI developments.

I absorbed new knowledge like crazy, and I still remember the feeling of my soul burning. I really hope I can keep the thirst for new knowledge throughout my life.

3 Option ROM format

The so-called Option ROM is a read-only memory located on a PCI or ISA device. Because this memory is not required to be implemented according to the bus standard, it is called Option ROM (optionally implemented ROM).

Option ROM usually contains data and code for initializing the device. Graphics cards and network cards usually have Option ROM. In simple terms, at the beginning of it, there is always a fixed header structure, called PnP Option ROM Header.

The offsets 18h and 1Ah of the header structure can point to two other structures, which are called PCI data structure and PnP Expansion Header structure (PEH for short). There is a Next field (offset 06h, length WORD) in PEH that acts as a linked list and is used to describe the offset of the next expansion structure. As shown in the figure:

偏移	长度（字节）	值	描述
0h	2h	AA55h	Rom 标志
2h	1h	自定义	Option ROM 总长
3h	4h	自定义	入口向量
7h	13h	自定义	保留
1Ah	2h	自定义	指向扩展头结构

Figure 3 Option ROM header structure

The ISA ROM structure does not need to be written completely according to the above structure. There are two points to note:

1) Fill in the length of Option ROM at offset 2h, with 2k as a unit. That is, if the entire ROM code is 20k, fill in 0ah.
2) The entire ROM code must be checked and 0. That is, from the first byte of the ROM to the last byte, the sum of these bytes must be 0.

If the BIOS detects that the ROM code complies with the pnp BIOS specification, it will use FAR CALL to call the entry vector at offset 3h, and the control will be transferred to the Oprion ROM code. Of course, in order to return the control to the BIOS normally, RETF must be used at the end of the Option ROM code to hand over the control.

4 Code Structure

After understanding the structure of Option ROM, you can start programming. It should be noted that when Option ROM is running, it can only call BIOS interrupts.

Considering the control of the structure, the library linking problem and the file size, it is best to write it in assembly language. Use an assembly compiler to compile it into an execution file, and use a tool to convert it into a BIN file that meets the requirements.

The typical program structure is as follows:

```
.MODEL TINY
.486
.CODE
ORG 0H
START:
DW 0AA55h ; Extended BIOS flag
DB 40h
Call Main ; Main is the main program entry
Retf ; Control is returned to BIOS
```

Just implement the required functions in the Main function .

The compiler uses MASM6.11, adopts the code of the above ISA ROM module, and uses Link /T to link the obj file to generate a COM file. Then, according to the requirements of PNP BIOS, a tool program is written to convert the COM file into a BIN file with a byte checksum of 0, and this tool program is used to convert the COM file into a BIN file.

After that, embed the bin file into BIOS, and the Option ROM development process is completed.

(to be continued…)