**Intel HEX文件解析**

2016年04月05日 17:22:50 [CouchDB](https://me.csdn.net/u011000290) 阅读数：7341 标签： [Intel](http://so.csdn.net/so/search/s.do?q=Intel&t=blog)[单片机](http://so.csdn.net/so/search/s.do?q=%E5%8D%95%E7%89%87%E6%9C%BA&t=blog)[bootloader](http://so.csdn.net/so/search/s.do?q=bootloader&t=blog)[数据解析](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE%E8%A7%A3%E6%9E%90&t=blog) 更多

个人分类： [C++](https://blog.csdn.net/u011000290/article/category/3167669)[Linux C](https://blog.csdn.net/u011000290/article/category/5858247)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u011000290/article/details/51066338

最近有一个需求就是为Arduino开发板做一个基于蓝牙的无线烧录程序。目前的Arduino程序都是通过USB线连接到电脑的主机上，实际的传输过程是基于USB协议的，这个过程还是比较麻烦的。因为每次的编译完以后都需要通过一个USB线来完成传输烧录的工作，这个还是很麻烦的。

**原理解读**

在Arduino中，利用USB来完成传输烧录大概是这么一个过程。每个Arduino源程序，即sketch文件，经过一系列的编译处理以后，最终会形成一个Intel HEX格式的文件，这个HEX文件其实就一个被封装好的数据包，包含头部、长度、偏移量、数据类型、数据和校验和等6个字段。再利用USB协议传输这个数据包，再通过板子上已有的bootloader程序把这个HEX文件中的有效字节码写到板子上的Flash存储器中。就是因为这个bootloader程序的存在，才使得多数的单片机具有**片内引导程序自编程功能**。MCU通过运行这个常驻Flash的bootloader程序，就可以利用任何可用的数据接口读取代码(注：在Arduino中，这里的代码是指HEX文件中的有效数据字段，是最终字节码的形式)后写入到自身的flash存储器中，从而实现了自编程的功能。

理解了上面的原理以后，你比如说我要做一个支持无线传输的类似bootloader的东西。无线传输采用蓝牙协议。蓝牙的接收端将接收到的数据全部存在串口中，所以核心的技术除了蓝牙收发部分以及对flash存储器的读写和存储空间的管理以外，还需要实现对这个HEX文件的解析，更准确、形象地说应该是对这些数据包的解析。怎么把这个数据包拆开，取出有效的字节码数据，然后把这些字节码按顺序分块组装，写到flash存储器，再把PC指针指到开始的地方就可以了。不过这东西确实很底层，就是在直接跟内存打交道，怎么管理内存，怎么读写内存，怎么拆分组装数据，出错了以后怎么擦除已经写的内容，如何优化等等一些列问题都需要解决。

**事非经过不知难，经过方知难与易**。慢慢做吧！先来第一步，对HEX格式的文件进行解析。解析之前自然要弄懂HEX文件的具体格式。

**HEX文件的格式**

Intel HEX格式的文件是由多条记录组成，而每条记录又是由6个字段组成。这些记录是由一些代表机器语言代码和常量的16进制数据组成的。Intel HEX 文件常用来传输要存储在 ROM 或者 EPROM 中的程序和数据。大部分的 EPROM 编程器能使用 Intel HEX文件记录的基本格式如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RecordMark | RecordLength | LoadOffset | RecordType | Data | Checksum |
| 记录标志 | 记录长度 | 装载偏移 | 记录类型 | 数据 | 校验和 |

其中，RecordMark字段其实就是每条记录的首部，其值为0x3A，在ASCII码中就是冒号“:”。该字段在HEX文件中，这个头部只占有一个字节。RecordLength表示每条记录包含的数据的长度，以**字节**为单位，最大描述255个字节，表现为2个16进制的字符，该字段在HEX文件中占2个字节。LoadOffset表示该记录中的数据在整个存取器空间中的偏移，用4个十六进制字符描述一个16位数据，在HEX文件中该字段占有4个字节。RecordType表示记录类型，表现为2个十六进制字符。取值有以下几种：

00表示数据记录；

01表示文件结束记录；

02描述拓展段地址记录；

03描述开始段地址记录；

04描述扩展线性地址记录；

05描述开始线性地址记录。

Data字段表示数据的具体内容，描述方法仍是两个16进制的字符表示1字节的数据。此字段的长度由该记录的RecordLength决定，数据的解释取决于记录类型(RecordType)。Checksum字段为校验和。这个校验和是这么来的，将RecordMark(“:”)后的所有的数据按字节相加，即成对相加起来，然后模除256得到余数，再对这个余数求补码，最终得出的结果就是校验和。所以检测方法也很简单：在每一条记录内，将RecordMark(“:”)后的所有数据(包括Checksum)按字节相加后得到的8位数据为0，则说明数据无误，否则说明出错了。

至于什么是拓展段地址记录、开始段地址记录、扩展线性地址记录、开始线性地址记录这里不做详细的介绍，在芯艺的《AVR单片机GCC程序设计》的附录部分有详细的说明。而在Arduino的HEX文件中，记录类型只有两种，数据记录和文件结束记录。所以RecordType这个字段的值不是0x00就是0x01。**数据记录**适用于8位、16位和32位格式，其详细格式如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 记录名 | RecordMark | RecordLength | LoadOffset | RecordType | Data | Checksum |
| 记录标志 | 记录长度 | 装载偏移 | 记录类型 | 数据 | 校验和 |
| 内容 | “:” | X | - | “00” | - | - |
| 字节数 | 1 | 1 | 2 | 1 | X | 1 |

**文件结束记录**适用于8位、16位和32位格式，其详细格式如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 记录名 | RecordMark | RecordLength | LoadOffset | RecordType | Checksum |
| 记录标志 | 记录长度 | 装载偏移 | 记录类型 | 校验和 |
| 内容 | “:” | “00” | “0000” | “01” | “FF” |
| 字节数 | 1 | 1 | 2 | 1 | 1 |

比如说，有如下一条数据记录：“:1001A000808184608083808182608083808181609F”，则，其RecordMark为“:”，RecordLength为”10”，这里是16进制的，对应10进制为16，也就是说Data字段有16个字节；LoadOffset的值为01A0，也就是说在该条记录中，数据字段在内存中的起始地址为01A0；RecordType为00，表示是记录类型；Data的值为80818460808380818260808380818160，一共有16个字节；Checksum的值为9F。用计算器按照上面的方式验证一下是可以得到一个低8位为0的数据，也就是说这条记录是合法的。下面开始编码实现对HEX文件的解析。

**HEX文件解析的实现**

解析过程还是比较简单的，这里直接附上源码。

HexLexer.h头文件中定义了相关的类的属性和操作。

1. #ifndef \_HEXLEXER\_H\_
2. #define \_HEXLEXER\_H\_
3. #include <cstdio>
4. #include <cstring>
5. #include <cstdlib>
6. */\**
7. *Intel Hex文件解析器V1.0*
8. *Hex文件的格式如下：*
9. *RecordMark RecordLength LoadOffset RecordType Data Checksum*
10. *在Intel Hex文件中，RecordMark规定为“:”*
11. *\*/*
12. #pragma warning(disable:4996)
13. #define MAX\_BUFFER\_SIZE 43
14. class Hex
15. {
16. public:
17. Hex(char mark);
18. ~Hex();
19. void ParseHex(char \*data);*//解析Hex文件*
20. void ParseRecord(char ch);*//解析每一条记录*
21. size\_t GetRecordLength();*//获取记录长度*
22. char GetRecordMark();*//获取记录标识*
23. char \*GetLoadOffset();*//获取内存装载偏移*
24. char \*GetRecordType();*//获取记录类型*
25. char \*GetData();*//获取数据*
26. char \*GetChecksum();*//获取校验和*
28. private:
29. char m\_cBuffer[MAX\_BUFFER\_SIZE];*//存储待解析的记录*
30. char m\_cRecordMark;*//记录标识*
31. size\_t m\_nRecordLength;*//记录长度*
32. char \*m\_pLoadOffset;*//装载偏移*
33. char \*m\_pRecordType;*//记录类型*
34. char \*m\_pData;*//数据字段*
35. char \*m\_pChecksum;*//校验和*
36. bool m\_bRecvStatus;*//接收状态标识*
37. *//size\_t m\_nIndex;//缓存的字符索引值*
38. };
40. Hex::Hex(char mark)
41. {
42. this->m\_cRecordMark = mark;
43. m\_cBuffer[0] = '\0';
44. *//m\_pBuffer = NULL;*
45. m\_nRecordLength = 0;
46. m\_pLoadOffset = NULL;
47. m\_pRecordType = NULL;
48. m\_pData = NULL;
49. m\_pChecksum = NULL;
50. m\_bRecvStatus = false;
51. *//m\_nIndex = 0;*
52. }
54. Hex::~Hex()
55. {
56. delete m\_pLoadOffset, m\_pRecordType, m\_pData, m\_pChecksum;
57. }
59. #endif

HexLexer.cpp文件实现了头文件中函数，并给出了测试用例。

1. #include "HexLexer.h"
2. #include <iostream>
3. using namespace std;
4. *//获取记录标识*
5. char Hex::GetRecordMark()
6. {
7. return this->m\_cRecordMark;
8. }
9. *//获取每条记录的长度*
10. size\_t Hex::GetRecordLength()
11. {
12. *//char \*len = (char\*)malloc(sizeof(char)\* 3);*
13. if (strlen(m\_cBuffer)>=2)
14. {
15. char len[3];
16. len[0] = m\_cBuffer[0];
17. len[1] = m\_cBuffer[1];
18. len[2] = '\0';
19. char \*p = NULL;
20. return strtol(len, &p, 16);
21. }
22. else
23. {
24. return 0;
25. }
26. }
27. *//获取装载偏移*
28. char\* Hex::GetLoadOffset()
29. {
30. if (strlen(m\_cBuffer) == (GetRecordLength() + 5) \* 2)
31. {
32. char \*offset = (char\*)malloc(sizeof(char)\* 5);
33. for (int i = 0; i < 4; ++i)
34. {
35. offset[i] = m\_cBuffer[i + 2];
36. }
37. offset[4] = '\0';
38. m\_pLoadOffset = offset;
39. offset = NULL;
40. }
41. return m\_pLoadOffset;
42. }
43. *//获取记录类型*
44. char\* Hex::GetRecordType()
45. {
46. if (strlen(m\_cBuffer) == (GetRecordLength() + 5) \* 2)
47. {
48. char \*type=(char\*)malloc(sizeof(char)\*3);
49. type[0] = m\_cBuffer[6];
50. type[1] = m\_cBuffer[7];
51. type[2] = '\0';
52. m\_pRecordType = type;
53. type = NULL;
54. }
55. return m\_pRecordType;
56. }
57. *//获取数据*
58. char\* Hex::GetData()
59. {
60. if (strlen(m\_cBuffer) == (GetRecordLength() + 5) \* 2)
61. {
62. int len = GetRecordLength();
63. char \*data = (char\*)malloc(sizeof(char)\*(len \* 2 + 1));
64. for (int i = 0; i < len \* 2;++i)
65. {
66. data[i] = m\_cBuffer[i + 8];
67. }
68. data[len \* 2] = '\0';
69. m\_pData = data;
70. data = NULL;
71. }
72. return m\_pData;
73. }
74. *//获取校验和*
75. char\* Hex::GetChecksum()
76. {
77. int len = GetRecordLength();
78. if (strlen(m\_cBuffer) == (len + 5) \* 2)
79. {
80. char \*checksum=(char\*)malloc(sizeof(char)\*3);
81. checksum[0] = m\_cBuffer[(len + 5) \* 2 - 2];
82. checksum[1] = m\_cBuffer[(len + 5) \* 2-1];
83. checksum[2] = '\0';
84. m\_pChecksum = checksum;
85. checksum=NULL;
86. }
87. return m\_pChecksum;
88. }
89. *//解析Hex文件中的每一条记录*
90. void Hex::ParseRecord(char ch)
91. {
92. size\_t buf\_len = strlen(m\_cBuffer);
93. if (GetRecordMark()==ch)
94. {
95. m\_bRecvStatus = true;
96. m\_cBuffer[0] = '\0';
97. *//m\_nIndex = 0;*
98. return;
99. }
100. if ((buf\_len==(GetRecordLength()+5)\*2-1))
101. {
102. *//接收最后一个字符*
103. m\_cBuffer[buf\_len] = ch;
104. m\_cBuffer[buf\_len + 1] = '\0';
105. *//检验接收的数据*
106. char temp[3];
107. char \*p = NULL;
108. long int checksum = 0;
109. for (int i = 0; i < strlen(m\_cBuffer);i+=2)
110. {
111. temp[0] = m\_cBuffer[i];
112. temp[1] = m\_cBuffer[i + 1];
113. temp[2] = '\0';
114. checksum += strtol(temp, &p, 16);
115. temp[0] = '\0';
116. }
117. checksum &= 0x00ff;*//取计算结果的低8位*
118. if (checksum==0)*//checksum为0说明接收的数据无误*
119. {
120. cout << "RecordMark " << GetRecordMark() << endl;
121. cout << "RecordLength " << GetRecordLength() << endl;
122. cout << "LoadOffset " << GetLoadOffset() << endl;
123. cout << "RecordType " << GetRecordType() << endl;
124. cout << "Data " << GetData() << endl;
125. cout << "Checksum " << GetChecksum() << endl;
126. }
127. else*//否则接收数据有误*
128. {
129. cout << "Error!" << endl;
130. }
131. m\_cBuffer[0] = '\0';
132. m\_bRecvStatus = false;
133. m\_nRecordLength = 0;
134. m\_pLoadOffset = NULL;
135. m\_pRecordType = NULL;
136. m\_pChecksum = NULL;
137. m\_bRecvStatus = false;
138. }
139. else if (m\_bRecvStatus)
140. {
141. m\_cBuffer[buf\_len] = ch;
142. m\_cBuffer[buf\_len + 1] = '\0';
143. *//m\_nIndex++;*
144. }
145. }
146. *//解析Hex文件*
147. void Hex::ParseHex(char \*data)
148. {
149. for (int i = 0; i < strlen(data);++i)
150. {
151. ParseRecord(data[i]);
152. }
153. }
154. int main(int argc, char \*argv[])
155. {
156. freopen("in.txt", "r", stdin);
157. freopen("out.txt", "w", stdout);
159. Hex hex(':');
160. char ch;
161. while (cin>>ch)
162. {
163. hex.ParseRecord(ch);
164. }
165. fclose(stdout);
166. fclose(stdin);
167. return 0;
168. }

in.txt是在Arduino IDE编译blink程序生成的blik.cpp.hex的原始内容，直接拷贝到记事本中，其内容如下：

:100000000C945C000C946E000C946E000C946E00CA

:100010000C946E000C946E000C946E000C946E00A8

:100020000C946E000C946E000C946E000C946E0098

:100030000C946E000C946E000C946E000C946E0088

:100040000C9488000C946E000C946E000C946E005E

:100050000C946E000C946E000C946E000C946E0068

:100060000C946E000C946E00000000080002010069

:100070000003040700000000000000000102040863

:100080001020408001020408102001020408102002

:10009000040404040404040402020202020203032E

:1000A0000303030300000000250028002B000000CC

:1000B0000000240027002A0011241FBECFEFD8E043

:1000C000DEBFCDBF21E0A0E0B1E001C01D92A930AC

:1000D000B207E1F70E94F1010C9401020C940000B8

:1000E00061E08DE00C94810161E08DE00E94BA0135

:1000F00068EE73E080E090E00E94F50060E08DE043

:100100000E94BA0168EE73E080E090E00C94F50084

:100110001F920F920FB60F9211242F933F938F933C

:100120009F93AF93BF938091010190910201A091A1

:100130000301B09104013091000123E0230F2D371A

:1001400020F40196A11DB11D05C026E8230F0296DB

:10015000A11DB11D20930001809301019093020124

:10016000A0930301B09304018091050190910601D1

:10017000A0910701B09108010196A11DB11D8093C6

:10018000050190930601A0930701B0930801BF9168

:10019000AF919F918F913F912F910F900FBE0F9034

:1001A0001F9018953FB7F894809105019091060132

:1001B000A0910701B091080126B5A89B05C02F3F6B

:1001C00019F00196A11DB11D3FBF6627782F892F19

:1001D0009A2F620F711D811D911D42E0660F771FDE

:1001E000881F991F4A95D1F70895CF92DF92EF9219

:1001F000FF92CF93DF936B017C010E94D200EB0151

:10020000C114D104E104F10489F00E9400020E94AB

:10021000D2006C1B7D0B683E734090F381E0C81ADE

:10022000D108E108F108C851DC4FEACFDF91CF9146

:10023000FF90EF90DF90CF900895789484B582601E

:1002400084BD84B5816084BD85B5826085BD85B57A

:10025000816085BDEEE6F0E0808181608083E1E829

:10026000F0E0108280818260808380818160808361

:10027000E0E8F0E0808181608083E1EBF0E0808164

:1002800084608083E0EBF0E0808181608083EAE736

:10029000F0E08081846080838081826080838081BF

:1002A000816080838081806880831092C10008957E

:1002B000833081F028F4813099F08230A1F00895E4

:1002C0008730A9F08830B9F08430D1F48091800073

:1002D0008F7D03C0809180008F7780938000089588

:1002E00084B58F7702C084B58F7D84BD08958091D9

:1002F000B0008F7703C08091B0008F7D8093B000F5

:100300000895CF93DF9390E0FC01E458FF4F2491D0

:10031000FC01E057FF4F8491882349F190E0880F5A

:10032000991FFC01E255FF4FA591B4918C559F4F49

:10033000FC01C591D4919FB7611108C0F8948C91CC

:10034000209582238C93888182230AC0623051F4E5

:10035000F8948C91322F309583238C938881822B53

:10036000888304C0F8948C91822B8C939FBFDF917B

:10037000CF9108950F931F93CF93DF931F92CDB723

:10038000DEB7282F30E0F901E859FF4F8491F901D9

:10039000E458FF4F1491F901E057FF4F04910023F7

:1003A000C9F0882321F069830E9458016981E02FF8

:1003B000F0E0EE0FFF1FEC55FF4FA591B4919FB7F2

:1003C000F8948C91611103C01095812301C0812B99

:1003D0008C939FBF0F90DF91CF911F910F91089544

:1003E00008950E941D010E94F0010E947000C0E06B

:1003F000D0E00E9474002097E1F30E940000F9CF42

:060400000895F894FFCFFF

:00000001FF

解析的结果直接写入到记事本中，结果如下：

RecordMark :

RecordLength 16

LoadOffset 0000

RecordType 00

Data 0C945C000C946E000C946E000C946E00

Checksum CA

RecordMark :

RecordLength 16

LoadOffset 0010

RecordType 00

Data 0C946E000C946E000C946E000C946E00

Checksum A8

RecordMark :

RecordLength 16

LoadOffset 0020

RecordType 00

Data 0C946E000C946E000C946E000C946E00

Checksum 98

RecordMark :

RecordLength 16

LoadOffset 0030

RecordType 00

Data 0C946E000C946E000C946E000C946E00

Checksum 88

RecordMark :

RecordLength 16

LoadOffset 0040

RecordType 00

Data 0C9488000C946E000C946E000C946E00

Checksum 5E

RecordMark :

RecordLength 16

LoadOffset 0050

RecordType 00

Data 0C946E000C946E000C946E000C946E00

Checksum 68

RecordMark :

RecordLength 16

LoadOffset 0060

RecordType 00

Data 0C946E000C946E000000000800020100

Checksum 69

RecordMark :

RecordLength 16

LoadOffset 0070

RecordType 00

Data 00030407000000000000000001020408

Checksum 63

RecordMark :

RecordLength 16

LoadOffset 0080

RecordType 00

Data 10204080010204081020010204081020

Checksum 02

RecordMark :

RecordLength 16

LoadOffset 0090

RecordType 00

Data 04040404040404040202020202020303

Checksum 2E

RecordMark :

RecordLength 16

LoadOffset 00A0

RecordType 00

Data 0303030300000000250028002B000000

Checksum CC

RecordMark :

RecordLength 16

LoadOffset 00B0

RecordType 00

Data 0000240027002A0011241FBECFEFD8E0

Checksum 43

RecordMark :

RecordLength 16

LoadOffset 00C0

RecordType 00

Data DEBFCDBF21E0A0E0B1E001C01D92A930

Checksum AC

RecordMark :

RecordLength 16

LoadOffset 00D0

RecordType 00

Data B207E1F70E94F1010C9401020C940000

Checksum B8

RecordMark :

RecordLength 16

LoadOffset 00E0

RecordType 00

Data 61E08DE00C94810161E08DE00E94BA01

Checksum 35

RecordMark :

RecordLength 16

LoadOffset 00F0

RecordType 00

Data 68EE73E080E090E00E94F50060E08DE0

Checksum 43

RecordMark :

RecordLength 16

LoadOffset 0100

RecordType 00

Data 0E94BA0168EE73E080E090E00C94F500

Checksum 84

RecordMark :

RecordLength 16

LoadOffset 0110

RecordType 00

Data 1F920F920FB60F9211242F933F938F93

Checksum 3C

RecordMark :

RecordLength 16

LoadOffset 0120

RecordType 00

Data 9F93AF93BF938091010190910201A091

Checksum A1

RecordMark :

RecordLength 16

LoadOffset 0130

RecordType 00

Data 0301B09104013091000123E0230F2D37

Checksum 1A

RecordMark :

RecordLength 16

LoadOffset 0140

RecordType 00

Data 20F40196A11DB11D05C026E8230F0296

Checksum DB

RecordMark :

RecordLength 16

LoadOffset 0150

RecordType 00

Data A11DB11D209300018093010190930201

Checksum 24

RecordMark :

RecordLength 16

LoadOffset 0160

RecordType 00

Data A0930301B09304018091050190910601

Checksum D1

RecordMark :

RecordLength 16

LoadOffset 0170

RecordType 00

Data A0910701B09108010196A11DB11D8093

Checksum C6

RecordMark :

RecordLength 16

LoadOffset 0180

RecordType 00

Data 050190930601A0930701B0930801BF91

Checksum 68

RecordMark :

RecordLength 16

LoadOffset 0190

RecordType 00

Data AF919F918F913F912F910F900FBE0F90

Checksum 34

RecordMark :

RecordLength 16

LoadOffset 01A0

RecordType 00

Data 1F9018953FB7F8948091050190910601

Checksum 32

RecordMark :

RecordLength 16

LoadOffset 01B0

RecordType 00

Data A0910701B091080126B5A89B05C02F3F

Checksum 6B

RecordMark :

RecordLength 16

LoadOffset 01C0

RecordType 00

Data 19F00196A11DB11D3FBF6627782F892F

Checksum 19

RecordMark :

RecordLength 16

LoadOffset 01D0

RecordType 00

Data 9A2F620F711D811D911D42E0660F771F

Checksum DE

RecordMark :

RecordLength 16

LoadOffset 01E0

RecordType 00

Data 881F991F4A95D1F70895CF92DF92EF92

Checksum 19

RecordMark :

RecordLength 16

LoadOffset 01F0

RecordType 00

Data FF92CF93DF936B017C010E94D200EB01

Checksum 51

RecordMark :

RecordLength 16

LoadOffset 0200

RecordType 00

Data C114D104E104F10489F00E9400020E94

Checksum AB

RecordMark :

RecordLength 16

LoadOffset 0210

RecordType 00

Data D2006C1B7D0B683E734090F381E0C81A

Checksum DE

RecordMark :

RecordLength 16

LoadOffset 0220

RecordType 00

Data D108E108F108C851DC4FEACFDF91CF91

Checksum 46

RecordMark :

RecordLength 16

LoadOffset 0230

RecordType 00

Data FF90EF90DF90CF900895789484B58260

Checksum 1E

RecordMark :

RecordLength 16

LoadOffset 0240

RecordType 00

Data 84BD84B5816084BD85B5826085BD85B5

Checksum 7A

RecordMark :

RecordLength 16

LoadOffset 0250

RecordType 00

Data 816085BDEEE6F0E0808181608083E1E8

Checksum 29

RecordMark :

RecordLength 16

LoadOffset 0260

RecordType 00

Data F0E01082808182608083808181608083

Checksum 61

RecordMark :

RecordLength 16

LoadOffset 0270

RecordType 00

Data E0E8F0E0808181608083E1EBF0E08081

Checksum 64

RecordMark :

RecordLength 16

LoadOffset 0280

RecordType 00

Data 84608083E0EBF0E0808181608083EAE7

Checksum 36

RecordMark :

RecordLength 16

LoadOffset 0290

RecordType 00

Data F0E08081846080838081826080838081

Checksum BF

RecordMark :

RecordLength 16

LoadOffset 02A0

RecordType 00

Data 816080838081806880831092C1000895

Checksum 7E

RecordMark :

RecordLength 16

LoadOffset 02B0

RecordType 00

Data 833081F028F4813099F08230A1F00895

Checksum E4

RecordMark :

RecordLength 16

LoadOffset 02C0

RecordType 00

Data 8730A9F08830B9F08430D1F480918000

Checksum 73

RecordMark :

RecordLength 16

LoadOffset 02D0

RecordType 00

Data 8F7D03C0809180008F77809380000895

Checksum 88

RecordMark :

RecordLength 16

LoadOffset 02E0

RecordType 00

Data 84B58F7702C084B58F7D84BD08958091

Checksum D9

RecordMark :

RecordLength 16

LoadOffset 02F0

RecordType 00

Data B0008F7703C08091B0008F7D8093B000

Checksum F5

RecordMark :

RecordLength 16

LoadOffset 0300

RecordType 00

Data 0895CF93DF9390E0FC01E458FF4F2491

Checksum D0

RecordMark :

RecordLength 16

LoadOffset 0310

RecordType 00

Data FC01E057FF4F8491882349F190E0880F

Checksum 5A

RecordMark :

RecordLength 16

LoadOffset 0320

RecordType 00

Data 991FFC01E255FF4FA591B4918C559F4F

Checksum 49

RecordMark :

RecordLength 16

LoadOffset 0330

RecordType 00

Data FC01C591D4919FB7611108C0F8948C91

Checksum CC

RecordMark :

RecordLength 16

LoadOffset 0340

RecordType 00

Data 209582238C93888182230AC0623051F4

Checksum E5

RecordMark :

RecordLength 16

LoadOffset 0350

RecordType 00

Data F8948C91322F309583238C938881822B

Checksum 53

RecordMark :

RecordLength 16

LoadOffset 0360

RecordType 00

Data 888304C0F8948C91822B8C939FBFDF91

Checksum 7B

RecordMark :

RecordLength 16

LoadOffset 0370

RecordType 00

Data CF9108950F931F93CF93DF931F92CDB7

Checksum 23

RecordMark :

RecordLength 16

LoadOffset 0380

RecordType 00

Data DEB7282F30E0F901E859FF4F8491F901

Checksum D9

RecordMark :

RecordLength 16

LoadOffset 0390

RecordType 00

Data E458FF4F1491F901E057FF4F04910023

Checksum F7

RecordMark :

RecordLength 16

LoadOffset 03A0

RecordType 00

Data C9F0882321F069830E9458016981E02F

Checksum F8

RecordMark :

RecordLength 16

LoadOffset 03B0

RecordType 00

Data F0E0EE0FFF1FEC55FF4FA591B4919FB7

Checksum F2

RecordMark :

RecordLength 16

LoadOffset 03C0

RecordType 00

Data F8948C91611103C01095812301C0812B

Checksum 99

RecordMark :

RecordLength 16

LoadOffset 03D0

RecordType 00

Data 8C939FBF0F90DF91CF911F910F910895

Checksum 44

RecordMark :

RecordLength 16

LoadOffset 03E0

RecordType 00

Data 08950E941D010E94F0010E947000C0E0

Checksum 6B

RecordMark :

RecordLength 16

LoadOffset 03F0

RecordType 00

Data D0E00E9474002097E1F30E940000F9CF

Checksum 42

RecordMark :

RecordLength 6

LoadOffset 0400

RecordType 00

Data 0895F894FFCF

Checksum FF

RecordMark :

RecordLength 0

LoadOffset 0000

RecordType 01

Data

Checksum FF

**参考文献**

[1]Wikipedia.Intel HEX[DB/OL].<http://en.wikipedia.org/wiki/Intel_HEX>,2016,4