Firmware format

The firmware file is made up of the following components, each located in the firmware file after one another:

- Firmware header
- Module header table
- Module data

This can be extracted using the following Kaitai Struct definition:

Kaitai Struct definition

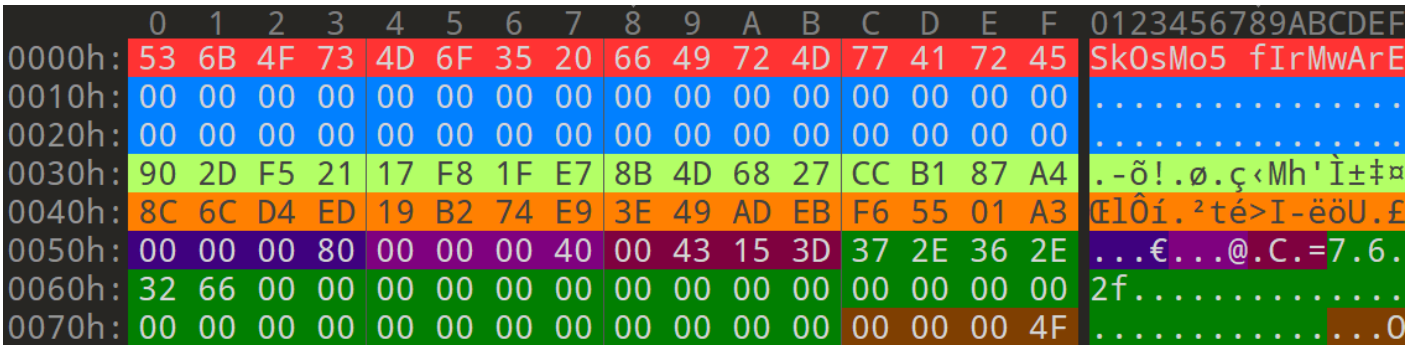
```
meta:
  id: spa504g
  endian: be
  license: Butlersaurus
  title: SPA504G firmware
  bit-endian: be
seq:
  - id: header
    type: header
  - id: modules
    type: module
    repeat: expr
    repeat-expr: header.module_count
types:
  header:
    seq:
      - id: magic
        contents: SkOsMo5 fIrMwArE
      - id: signature
        size: 32
      - id: digest
        size: 16
      - id: random_sequence
```

```
      size: 16
    - id: header_length
      type: u4
    - id: module_header_length
      type: u4
    - id: file_length
      type: u4
    - id: version
      type: str
      encoding: utf8
      size: 32
    - id: module_count
      type: u4
    - id: padding
      size: header_length - 128
module:
  seq:
    - id: module_sequence_number
      type: u2
    - id: module_compressed_flag
      type: u2
    - id: module_length
      type: u4
    - id: module_offset
      type: u4
    - id: module_digest
      size: 16
    - id: padding
      size: _parent.header.module_header_length - 28
  instances:
    body:
      pos: module_offset
      size: module_length
      process: zlib
```

Firmware header

The firmware header is located at the very beginning of the file. Its length is defined by the 32 bit unsigned integer (big endian) located at offset `0x0050`. For example: `0x00000080` (128 bytes).

The format for the **firmware header** is described below:



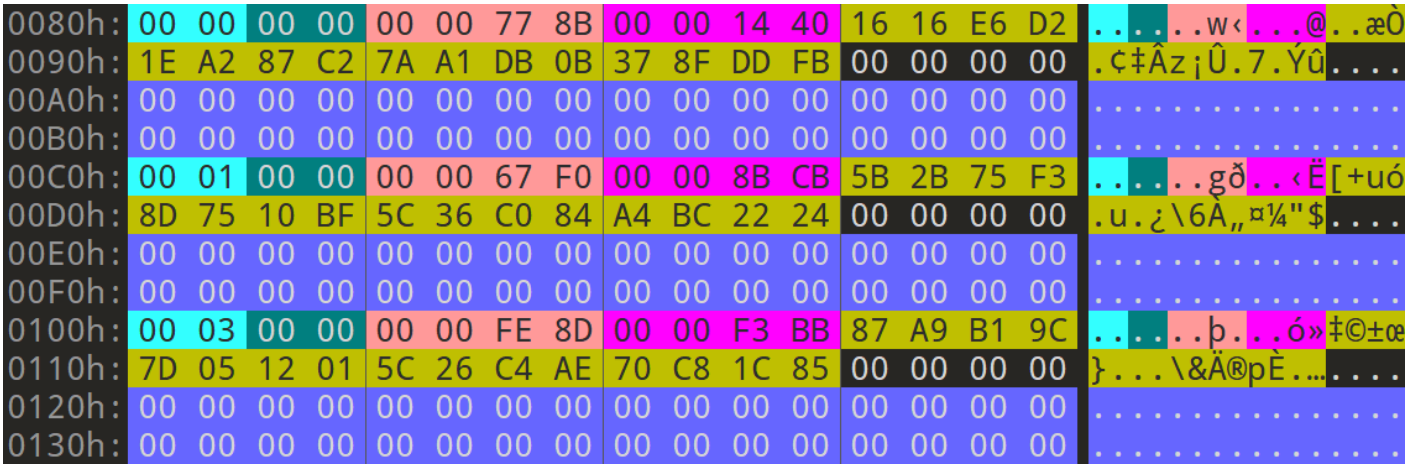
Size (byte)	Type	Description	Example
16	Byte array	Identifier	`SkOsMo5 fIrMwArE`
32	Byte array	Signature	Not used
16	Byte array	MD5 of **firmware header** + **module headers**	`90 2D F5 21 17 F8 1F E7 8B 4D 68 27 CC B1 87 A4`
16	Byte array	A random sequence	`8C 6C D4 ED 19 B2 74 E9 3E 49 AD EB F6 55 01 A3`
4	Unsigned BE Int	Length of the entire **firmware header**	`0x00000080` (128 bytes)
4	Unsigned BE Int	Length of each **module header**	`0x00000040` (64 bytes)
4	Unsigned BE Int	Length of entire file	`0x0043153D` (4396349 bytes)
32	Byte array	Firmware version number	`7.6.2f`
4	Unsigned BE Int	Number of modules present in the firmware	`0x0000004F` (79 modules)

Module header table

There are multiple modules in the firmware file. The number of modules is defined at offset `0x007C` in the **firmware header**. For example: `0x0000004F` (79 modules).

Each module has a **module header**, located in a **module header table**. These are of a length defined at offset `0x0054` in the **firmware header**. For example: `0x00000040` (64 bytes). Each **module header** is concatenated in order, directly after the **firmware header** (in this case, from offset `0x0080`).

The format for each **module header** is described below. The screenshot contains three of the 79 **module headers** present in this particular firmware (7.6.2f).



Size (byte)	Type	Description	Examples
2	Unsigned BE Short	Sector ID	0x00 (0) 0x01 (1) 0x03 (3) This was observed to increase from 0 to 127, skipping the following values: • 2 • 5 to 52 (inclusive)
2	Unsigned BE Short	Compressed flag?	0x00
4	Unsigned BE Int	Length of module data	0x0000778B (30,603 bytes) 0x000067F0 (26,608 bytes) 0x0000FE8D (65,165 bytes) …
4	Unsigned BE Int	Offset to module data from start of file	0x00001440 (5,184 bytes) 0x00008BCB (35,787 bytes) 0x0000F3BB (62,395 bytes) …
16	Byte array	MD5 digest of the uncompressed module	16 16 E6 D2 1E A2 87 C2 7A A1 DB 0B 37 8F DD FB 5B 2B 75 F3 8D 75 10 BF 5C 36 C0 84 A4 BC 22 24 87 A9 B1 9C 7D 05 12 01 5C 26 C4 AE 70 C8 1C 85 …
4	Byte array	Unknown	

Size (byte)	Type	Description	Examples
32	Padding	Zero padding	

Module data

The offset and size for each module's data is defined in the corresponding **module header** in the **module header table**.

For example, the first module in this particular firmware (7.6.2f) is located at offset `0x00001440` (5,184 bytes) from the beginning of the file, and is `0x0000778B` (30,603 bytes) long.

It looks like all of this data is zlib compressed with a 'windowBits' parameter of `15`, as determined by reversing the 'uncompress' method in the psbl.elf binary.

Decompile: uncompress - (psbl.elf)

```
1
2  int uncompress(uchar *_destination,ulong _destinationLength,uchar *_source,ulong _sourceLength)
3
4  {
5    int ret;
6    uchar *source;
7    ulong sourceLength;
8    uchar *destination;
9    ulong destinationLength;
10   undefined4 local_3c;
11   code *libupg_zcalloc;
12   code *libupg_zfree;
13
14   memset(&source,0,0x38);
15   destinationLength = *(ulong *)_destinationLength;
16   libupg_zcalloc = ::libupg_zcalloc;
17   libupg_zfree = ::libupg_zfree;
18   source = _source;
19   sourceLength = _sourceLength;
20   destination = _destination;
21   ret = inflateInit2(&source,15);
22   if (ret == 0) {
23     ret = inflate(&source,4);
24     if (ret == 1) {
25       *(undefined4 *)_destinationLength = local_3c;
26       ret = inflateEnd(&source);
27     }
28     else {
29       inflateEnd(&source);
30       if (ret == 0) {
31         ret = -5;
32       }
33     }
34   }
35   return ret;
36 }
37
```

Using Python, these can be deflated trivially. A PoC deflation script is shown below:

Python deflate script

```
import zlib


compressed_data = open('spa50x-30x-...-module.bin', 'rb').read()
decompressed_data = zlib.decompress(compressed_data, 15) # windowBits of 15.


with open('spa50x-30x-...-module_deflated.bin', 'wb') as f:
    f.write(decompressed_data)
```

Revision #20
Created 2 April 2022 22:05:18 by Jack
Updated 3 April 2022 18:07:39 by Aidan