高通 msp 合并程序分析

版本 0.3
日期：2016 年 4 月 19 日
最后修改日期：2016 年 4 月 23 日
源程序：msp.py
分析后带注释的程序：msp_comment.py
本文档使用的编辑器：LibreOffice Writer
编辑器网址：https://www.libreoffice.org/zh-cn

主要功能：program 编程功能，将 rawprogram.xml 配置文件指定的一系列的文件合并为一个单一的 singleimage.bin 文件
次要功能：patch 补丁功能，将 patch.xml 指定的一系列补丁文件，写入到 singleimage.bin
次要功能：Read 读功能，按照 xml 配置文件指定从 singleimage.bin 读出数据写到指定的文件中，xml 可以一次指定多个读操作。

特点：
1，以上三个功能中的 singleimage.bin 可以用 SD 卡的取代。
2，程序使用 python 开发，支持 Linux 和 windows
3，扇区大小支持 512 和 4K
4，如果存在 testsparse.py 文件，可以完成附加的安全检查，sparse 稀疏文件。
5，singleimage 现在支持 GPT 分区

选项：
-n noprompt 无提示静默模式
-t 指定输出文件夹
-r –rawprogram 这个选项指定一个 xml 配置，包含全部的输入文件和相应的起始地址等。
-b –sectorsize 设置磁盘映像文件的扇区尺寸
-p –patch 指定补丁文件的 xml 配置文件
-d –dest #目标文件，分两种情况：1)singleimage.bin 的尺寸，2) /dev/sdb 等驱动器被指定
-i –interactive #交互模式
-s –search_path 索搜路径，xml 中的文件的搜索路径
-v –verbose 冗余模式，更多的输出信息，更多的日志信息
-f –file_list #装载 xml 配置文件中的一个子集，列表中文件之间用逗号分隔的。

程序中的关键变量常量
BLOCK_SIZE = 0x200　　#块尺寸，0x200=512,就是磁盘扇区大小

#emmc 最大块尺寸，64TBytes
EMMCBLD_MAX_DISK_SIZE_IN_BYTES　= (64*1024*1024*1024*1024) # 64TB - Terabytes

#分割前的最大文件尺寸 10M，拷贝文件的时候，一次最多读 10M，避免消耗太多内存
MAX_FILE_SIZE_BEFORE_SPLIT = (10*1024*1024)
#MAX_FILE_SIZE_BEFORE_SPLIT = (2048) # testing purposes

#扇区尺寸
SECTOR_SIZE = 512

函数分析：
def HandleNUM_DISK_SECTORS(field)
这个函数理出 域中的表达式，表达式如 "NUM_DISK_SECTORS-(\d+)" 或 NUM_DISK_SECTORS，或者直接指定扇区数值。NUM_DISK_SECTORS 其实指 singleimage.bin 中的扇区数，这个扇区数是用户在命令行中指定的。使用表达式的原因应该是为了在映像文件末尾保存数据，比如 gpt_backup 分区就保存在映像文件的末尾。

函数分析：
def ReturnParsedValues(element)
功能，解析 XML 后，用这个函数进一步处理， 首先将 element 解析值存到 MyDict 中，然后进行兼容性处理，对老的变量参数进行转换。下一步，同扇区数有关的表达式进行处理，解析表达式，计算出具体的扇区值。最后对补丁文件的参数进行处理。

函数分析
def ParseXML(xml_filename):　## this function updates all the global arrays
　　global WriteArray,PatchArray,ReadArray,MinDiskSizeInSectors

多个 XML 配置文件中， XML 元素仅有有三种标签，read,program,patch
放入对应的三个全局数组中 WriteArray,PatchArray,ReadArray
由于 python 的强大，解析 xml 语言只使用了短短的一段程序

函数分析
def find_file(filename, search_paths):
#查找文件，在索搜路径和工作路径下，返回全路径文件名

函数分析
def PerformRead():
　　global ReadArray, search_paths, interactive, Filename
#　　完成读操作
读命令指 将 singleimage.bin 的指定扇区（起始扇区，长度）
读到 xml 配置文件 指定的文件中，（起始位置总为 0，长度）

函数分析
def PerformWrite():
　　global WriteSorted, LoadSubsetOfFiles, search_paths, interactive, Filename

完成写操作
将 xml 配置文件中指定的数据（起始地址，长度）写到 singleimage.bin 中去（起始位置，长度）

函数分析

```
def PerformPatching():
      global PatchArray,Patching
#    完成补丁操作。
# 将-p 参数指定的 xml 配置文件中的文件  写到 singleimage.bin 中。
```

MTK 新合并程序需求分析

1，只支持合并程序，也就是 program 部分，不支持读和补丁操作。

2，只写到磁盘映像文件 singleimage.bin 中，不支持写 SD 卡

3，扇区大小支持 512 和 4K

4，支持 GPT 分区（暂时不支持）

5，不支持 sparse 稀疏文件检擦

6，支持 window7,64 位和 32 位；不支持 linux。

7，使用压缩的 singleimage.bin 格式，不保存空白扇区。

8，只支持 pid = 0 的分区

XML 可能遇到的属性值解析

输入文件
filename : 输入的原文件，二进制
file_sector_offset： 输入文件的扇区偏移,实际编程的时候，从这里开始，之前的数据不处理
label： 文件类别，程序无需处理
num_partition_sectors ：分区扇区数，在映像文件中分配给这个文件的扇区数，
必须要大于文件的实际尺寸，否则报错。

physical_partition_number：物理分区号，必须是零，否则 msp.py 不能处理
size_in_KB： 用 KB 计算的大小 #未使用，源程序中搜索不到
sparse ： 稀疏文件，不处理 #未用
start_byte_hex: 起始字节，16 进制 #程序中未使用，搜索不到
start_sector 起始扇区，输入文件保存到目标文件的起始扇区
sector_size_in_bytes 扇区尺寸，用字节计算，一般 512，也有 4k=4096 的

以下用于补丁文件，忽略
function： 补丁文件的功能，CRC32 等
arg 参数 0，
arg1 参数 1
value 值 #补丁文件用
byte_offset 偏移字节，#字节偏移量，只用于补丁文件，按字节打补丁。
size_in_bytes 用字节计算的尺寸，补丁文件用

合并文件算法：
1，解析 XML 文件，文件名，起始地址等保存到 Write
2，将 Write 按起始地址排序存到 WriteSorted 中
3，计算最小磁盘尺寸,起始扇区+扇区占用数，最大值就是磁盘尺寸
计算 singleimage.bin 大小，磁盘剩余空间要比这个大。
更正：计算最小磁盘空间，要加上起始扇区
4，删除老的 singleimage.bin 文件
5，打开输入文件 WriteSorted[i],
6，Seek(file_sector_offset*SECTOR_SIZE)
7，文件分割成最大 MAX_FILE_SIZE_BEFORE_SPLIT 的数据块
8，循环处理上面分割的数据块，写到目标 singleimage.bin 中
9，如果有剩余数据，最后剩余的数据不到一个扇区，剩余空间填补 0x0。
10，循环处理 全部的 WriteSorted 数据
10，关闭文件

需要用到的技术
1 地址的排序使用 Tlist 控件，需要继承 Tlist，扩充类，并且自己分配存储空间。
这个控件使用失败，最后使用自己编写的排序控件成功！

2016 年 4 月 20 日
高通 XML 解析：

1，解析 xml 配置文件

2，决定特殊区块的真实的起始地址

从磁盘底部倒着计算的起始扇区的块，如 GPT_backup 磁盘分区表保存在磁盘的最后，起始扇区位置用 **NUM_DISK_SECTORS – 33** 类似的表达式来表示。 **NUM_DISK_SECTORS** 表示目标磁盘的总扇区数，也就是芯片用扇区数表示的大小当前 **8G**，**16G**，**32G**，**64G** 芯片比较常见。

对于特殊区块定位分两种情况，

1）如果用户指定了磁盘大小，就直接定位这些特殊区块了，这些区块也就是普通 区块了。下面就可以进入排序过程了。

2）用户没有指定磁盘大小，我们需要计算最小磁盘尺寸，然后再对起始地址定位。为了便于后面的计算，对于这样的特殊块 暂时先使用 **64T + 33** 来替代，保证在后面排序的时候，把这样的块排在最后。**64T** 是我们程序能处理的最大磁盘大小，**33** 为从磁盘空间倒数的扇区起始地址。

3，起始扇区地址排序

高通经过上面的处理，起始扇区全部都是数字了，因此可以排序了。按照起始扇区地址从小到大排列。，以用冒泡法。

4，计算最小磁盘尺寸 **MinDiskSizeInSec**

算法：全部普通块所占用的磁盘空间 **= max(startsec+num_block)**，对于排序后的情形，就是最后一个普通块 **startsec+num_block** 的值。然后再把特殊块的大小全部加上，这个值是 高通的计算是 **sum(startsec)**， 上面的例子中 **startsec=33**。起始可以更精确一点，**max(startsec),**因为从空间底部算起的起始扇区地址，必然是绝对地址大的包含绝对地址小的。比如倒数起始地址 **33**，必然包含倒数起始地址 **30** 的空间，第一块最多 从 **-33** 到**-31**，第二个块从**-30** 到 **-0**

当然为了有更多冗余，计算和理解简单。也可以用 **sum(startsec)** 的公式。

5， 要求最小磁盘尺寸不能大于用户给定的 磁盘尺寸，就是芯片的尺寸（**8G**，**16G, 32G**，**64G**）。否则报错，停止处理。

6，重新定位特殊块

用户给定的磁盘大小为 **DiskSizeInSec** 必须大于 **MinDiskSizeInSec**

对起始扇区地址 **startsec** 大于 **64T** 的，用 **DiskSizeInSec - (startsec-64T)**重新定位扇区的起始位置，这是这些扇区在最终目标位置的起始地址。

7，至此 **xml** 分析完毕，要写的文件的信息，特别是起始地址都已经排序，定好绝对位置。将这些记录送到 已经调试好的 **TMergeFile** 对象，就可以生成我们需要的单个合并后的文件**(singleimage)**

对高通 **xml** 参数的分析：

1 先取出一条解析

<program start_sector="131072" start_byte_hex="0x4000000" sparse="false" size_in_KB="86016.0" physical_partition_number="0" num_partition_sectors="172032" label="modem" filename="NON-HLOS.bin" file_sector_offset="0" SECTOR_SIZE_IN_BYTES="512"/>

1)， start_sector="131072" start_byte_hex="0x4000000"

表示的是一个意思，数据块在目标芯片的定位，起始位置。使用计算器验证： **0x4000000 bytes = 67108864 bytes = 67108864 div 512 sector = 131072 sector**

2)，size_in_KB="86016.0" num_partition_sectors="172032"

172032 sectors = 172032 * 512 bytes = 88080384 bytes = 88080384 /1024 Kbytes=86016 KB

刚好相等，我们再来查看下文件的实际尺寸：**81.0 MB (85,023,232** 字节**)**

占用空间 **81.0MB**（**85,024,768 bytes**）**= 83032KB** 比 分配的空间 **86016 KB** 略小

2，再取出来一条

<program start_sector="393216" start_byte_hex="0xc000000" sparse="false" size_in_KB="1.0" physical_partition_number="0" num_partition_sectors="2" label="fsc" filename="" file_sector_offset="0" SECTOR_SIZE_IN_BYTES="512"/>

1）start_sector="393216" start_byte_hex="0xc000000"

0xc000000 bytes = 201326592 bytes / (512 bytes/sec) = 201326592/512 sec= 393216 sec

刚好相等

2) size_in_KB="1.0" num_partition_sectors="2"

这个一眼就能看出来，**1KB = 2sec**

2 sec * 512bytes/sec = 1024B=1KB

filename=””,无法验证了，这个记录忽略

3，取最后一条记录 **<program start_sector="NUM_DISK_SECTORS-33." start_byte_hex="(512*NUM_DISK_SECTORS)-16896." sparse="false" size_in_KB="16.5" physical_partition_number="0" num_partition_sectors="33" label="BackupGPT" filename="gpt_backup0.bin" file_sector_offset="0" SECTOR_SIZE_IN_BYTES="512"/>**

1)start_sector="NUM_DISK_SECTORS-33."

start_byte_hex="(512*NUM_DISK_SECTORS)-16896."

起始扇区数就是从空间底部数起，**33** 扇区，和 **16896** 字节，判断这两个数是否相等即可

33*512=16896，也刚好相等

2) size_in_KB="16.5" num_partition_sectors="33"

33 * 512 = 16896 bytes

16.5KB = 16.5* 1024 = 16896 bytes 两个数刚好相等。 **33Sec=16.5K**

看一下 **gpt_backup0.bin** 文件的大小 **16.5 KB (16,896** 字节**)** ，刚好等于 我们前面的计算值 **16,896**

总结：拷贝一个数据块需要 **3** 个要素**:** 起始地址，数据长度，数据类型长度

起始地址： **start_sector， start_byte_hex** 分别是用扇区和字节计数的起始地址，总相等

数据长度： **num_partition_sectors，size_in_KB** 分别是用扇区和字节计数的长度，总相等

数据类型长度： **SECTOR_SIZE_IN_BYTES** 字节计算的数据类型长度

注释：文件的实际长度 总是小于等于 **num_partition_sectors** 和 **size_in_KB**

其他几个参数：

file_sector_offset

#输入文件的扇区偏移，拷贝数据的起始位置并不是总是从 **0** 扇区开始。不过目前这个配置文件的配置记录全都是 **0**。

physical_partition_number="0"

物理分区号，配置文件记录里面全都是 **0**，而且高通 **msp.py** 里面要求 **physical_partition_number** 要求 必须是零，否则报错。配置文件从 **filename** 和 **label**

label="PrimaryGPT" filename="gpt_main0.bin"
label="aboot" filename="emmc_appsboot.mbn"
label="abootbak" filename="emmc_appsboot.mbn"
label="boot" filename="boot.img"
label="OEMCONFIG" filename="oemconfig.img"

看有很多特殊块。看来这个同我们的 **partition ID** 不同。

术语表：

GPT 分区

GUID 磁碟分割表（GUID **Partition Table**，缩写：**GPT**）是一个實體硬盘的分区表的结构布局的标准。它是可扩展固件接口（EFI）标准（被 Intel 用于替代个人计算机的 BIOS）的一部分，被用于替代 BIOS 系统中的一 **32bits** 来存储逻辑块地址和大小信息的主開機紀錄（**MBR**）分区表。对于那些扇区为 **512** 字节的磁盘，MBR 分区表不支持容量大于 **2.2**TB（**2.2×10**^{12} 字节）[1]的分区，然而，一些硬盘制造商（诸如希捷和西部数据）注意到这个局限性，并且将他们的容量较大的磁盘升级到 **4KB** 的扇区，这意味着 **MBR** 的有效容量上限提升到 **16 TiB**。 这个看似“正确的”解决方案，在临时地降低人们对改进磁盘分配表的需求的同时，也给市场带来关于在有较大的块（block）的设备上从 **BIOS** 启动时，如何最佳的划分磁盘分区的困惑。**GPT** 分配 **64bits** 给逻辑块地址，因而使得最大分区大小在 **2**^{64}**-1** 个扇区成为可能。对于每个扇区大小为 **512** 字节的磁盘，那意味着可以有 **9.4**ZB（**9.4×10**^{21} 字节）或 **8** ZiB 个 **512** 字节（**9,444,732,965,739,290,426,880** 字节或 **18,446,744,073,709,551,615**（**2**^{64}**-1**）个扇区 **×512**（**2**^{9}）字节每扇区）[1][2]。

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。在使用 64 位 Windows Server 2003 的机器上，最多可以创建 128 个分区，即分区表中保留了 128 个项，其中每个都是 128 字节。（EFI 标准要求分区表最小要有 16,384 字节，即 128 个分区项的大小）

分区表头还记录了这块硬盘的 GUID，记录了分区表头本身的位置和大小（位置总是在 LBA 1）以及备份分区表头和分区表的位置和大小（在硬盘的最后）。它还储存着它本身和分区表的 CRC32 校验。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。所以 GPT 硬盘的分区表不可以直接使用 16 进制编辑器修改。

百度百科：**http://baike.baidu.com/view/10461841.htm?fromtitle=gpt%E4%BF%9D%E6%8A%A4%E5%88%86%E5%8C%BA&fromid=9003078&type=syn**

维基百科：**https://zh.wikipedia.org/wiki/GUID%E7%A3%81%E7%A2%9F%E5%88%86%E5%89%B2%E8%A1%A8**