深圳市华曦达科技股份有限公司		文 档 编 号	版本号	密级
		文档编号	V1.1	密级
文档名称	基础平台源码管理办法		**日期**	2016 年 12 月 12 日

基础平台源码管理办法

文档作者：　　杨宇锋　　日期：

项目经理：　　日期：

审　　核：　　日期：

批　　准：　　日期：

深圳市华曦达科技股份有限公司

文档历史发放及记录

序号	变更（+/-）说明	作者	版本号	日期	批准
1	草稿	杨宇锋	V1.1	2016.12.12	

注意：最新版本放在最前面。

文档简要功能及适用范围

1. 文档的简要功能

文档简要介绍 SDMC 基础平台源码的管理方式:

1) 原厂 SDK 和 SDMC SDK 的管理方式

2) PATCH 命名规范

3) 打包材料的发布说明

2. 文档的适用范围

本文档适用于 SDMC 研发部软件工程师和管理人员

适用平台包括 amlogic、his 等平台

目录

1 编写目的

1） 规范基础软件源码管理和发布
2） 规范基础软件 PATCH 管理和发布
3） 规范研发部软件打包流程，明确相关角色职责，提高执行效率和正确率

2 基础平台 SDK 管理模型

2.1 管理模型

基础平台 SDK 的管理采取如 Figure 1 的模型，以版本控制为中心，多角色参与，共同维护。

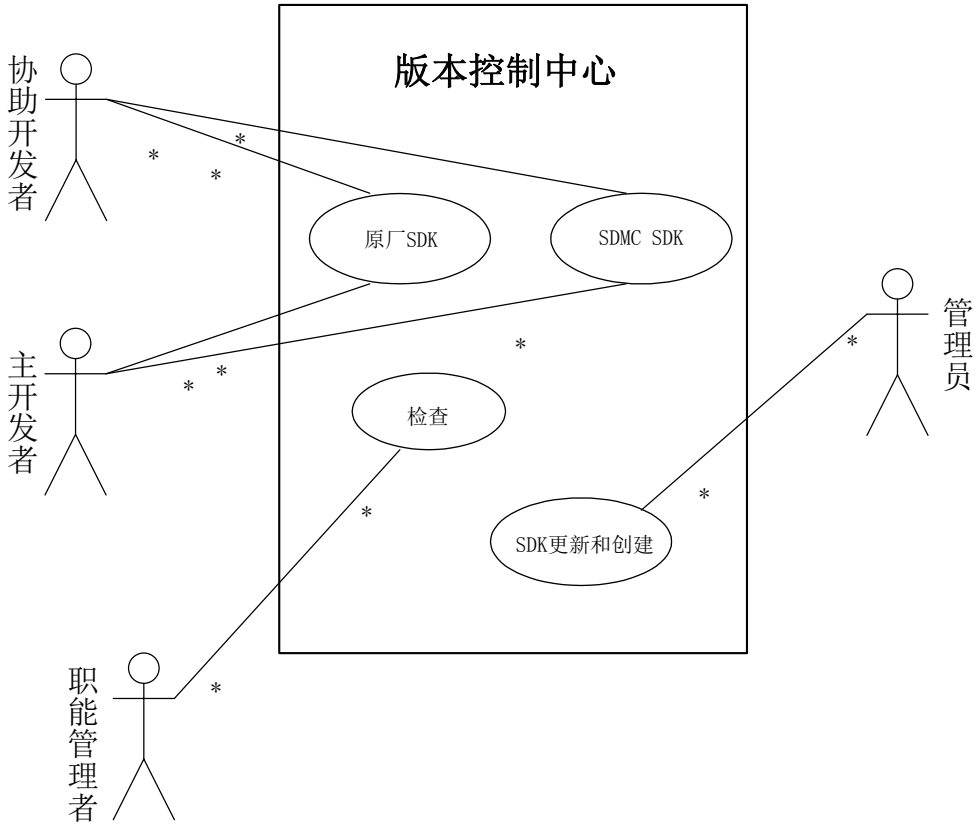

Figure 1

管理员：版本控制中心的管理者，负责创建和更新原厂 SDK 工程以及创建 SDMC SDK

职能管理者：负责监督和检查开发者是否按照规范来维护 SDK

主开发者：SDK 的主要维护者，负责代码或者 PATCH 的提交

协助开发者：协助主开发者来维护 SDK，当协助开发者完善 SDK 的某一个功能后，需要将 PATCH 给予主开发者进行维护。

2.2 架构模型

如 Figure 2，是基础平台 SDK 架构的基本模型，整体上分为两条独立的管理模式，原厂 SDK 的维护和 SDMC SDK 的维护，其中 SDMC SDK 包括基础平台部分源码和 PATCH 集合。原厂 SDK 的更新只依赖原厂，SDMC SDK 的更新只依赖 SDMC,不依赖原厂，其架构如 Figure 3.

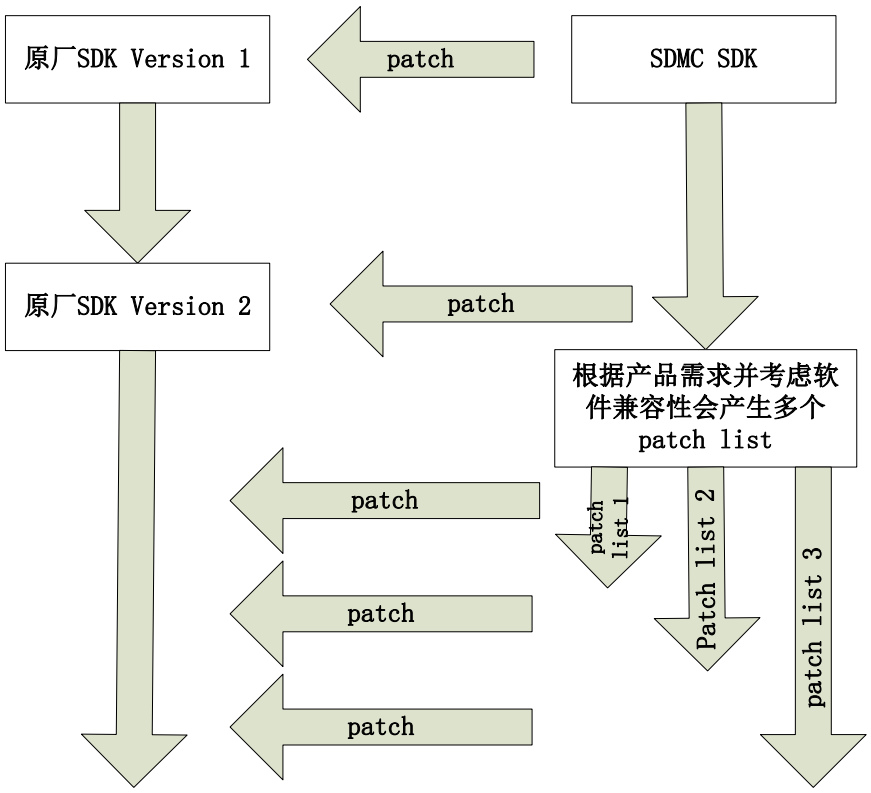

Figure 2

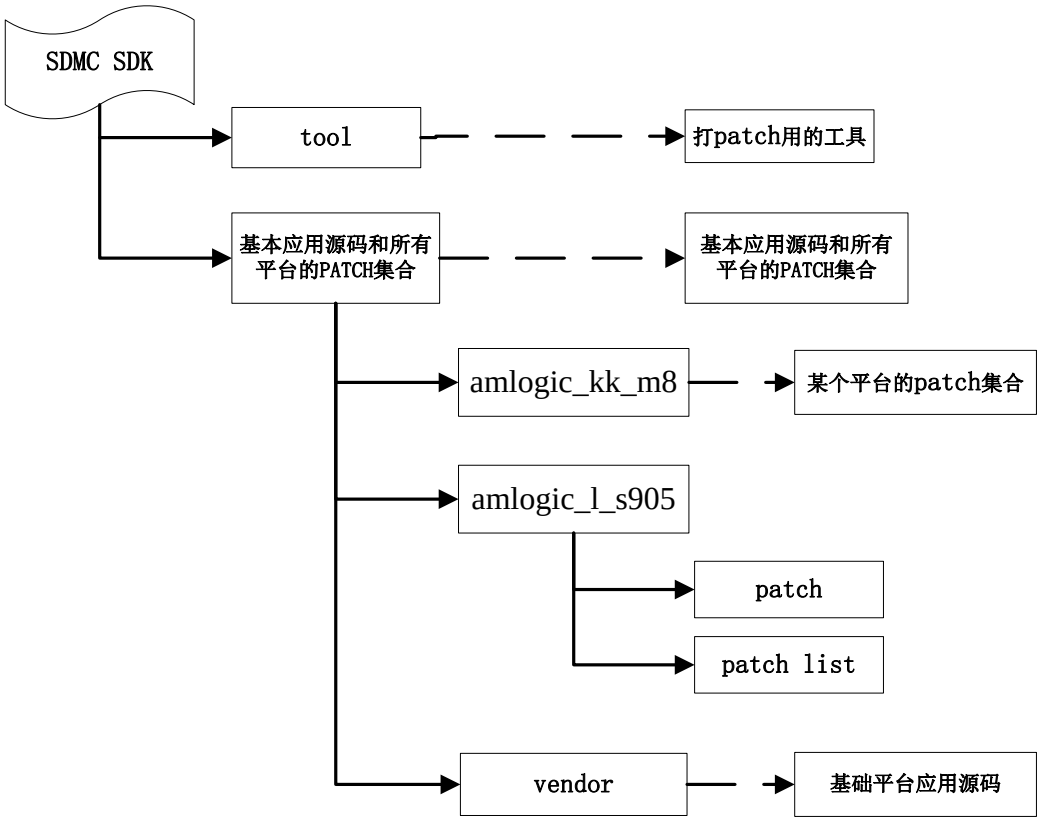

Figure 3

2.3 版本规范

2.3.1 原厂 SDK 版本规范

原厂 SDK 的版本是根据原厂的命名规则来确定，如 20150414。

2.3.2 SDMC SDK 版本规范

SDMC SDK 基础平台版本规范分为 PATCH 版本和 PATCH LIST 版本，其中 PATCH 是某个平台所有 PATCH 的一个集合，根据原厂、系统版本、平台来命名，字母全部小写，如 Table 1

[原厂]_[系统版本]_[平台]	
原厂	芯片厂商名
系统版本	操作系统代号
平台	平台代号

Table 1

如 M8 平台的 PATCH 版本命名为：

amlogic_kk_m8

描述见 Table 2

amlogic_kk_m8	
amlogic	代表芯片原厂
kk	系统版本，Android 4.4 的版本代号为 kitkat，一般简称 kk
m8	芯片型号，m8 代表 s805 和 s812，他们共用一套 sdk

Table 2

2.3.3 PATCH LIST 版本规范

PATCH LIST 是一个 TXT 类型的文件，是指定某一个产品需要打入原厂 SDK 的一个 PATCH 列表，他的产生是由于某一个需求不能和现有的 PATCH LIST 兼容，其命名规范如 Table 3，字母全部小写

patch_list_原厂 SDK 版本_[产品类型]_[客户名]	
patch_list	**必须，固定字段**
原厂 SDK 版本	**必须，指明这个 patch list 打入的原厂 sdk 版本号，根据原厂的命名而来，如 20150414，v64**
产品类型	**非必须，根据平台特性来划分，如 dvb，1G，2G**
客户名	**非必须，如 ais, frontline**

Table 3

如 s905 平台古巴客户的 PATCH LIST 命名为：

patch_list_dvb_cuba.txt

3 SDMC PATCH 命名规范

3.1 PATCH 的组成

每个功能性的 PATCH 以文件夹的形式存放在某个平台 PATCH 集合里面，其结构示意图如 Figure 4。每一个 PATCH 由三部分组成，分别是 PATCH 文件、source_code 和 Readme.txt.

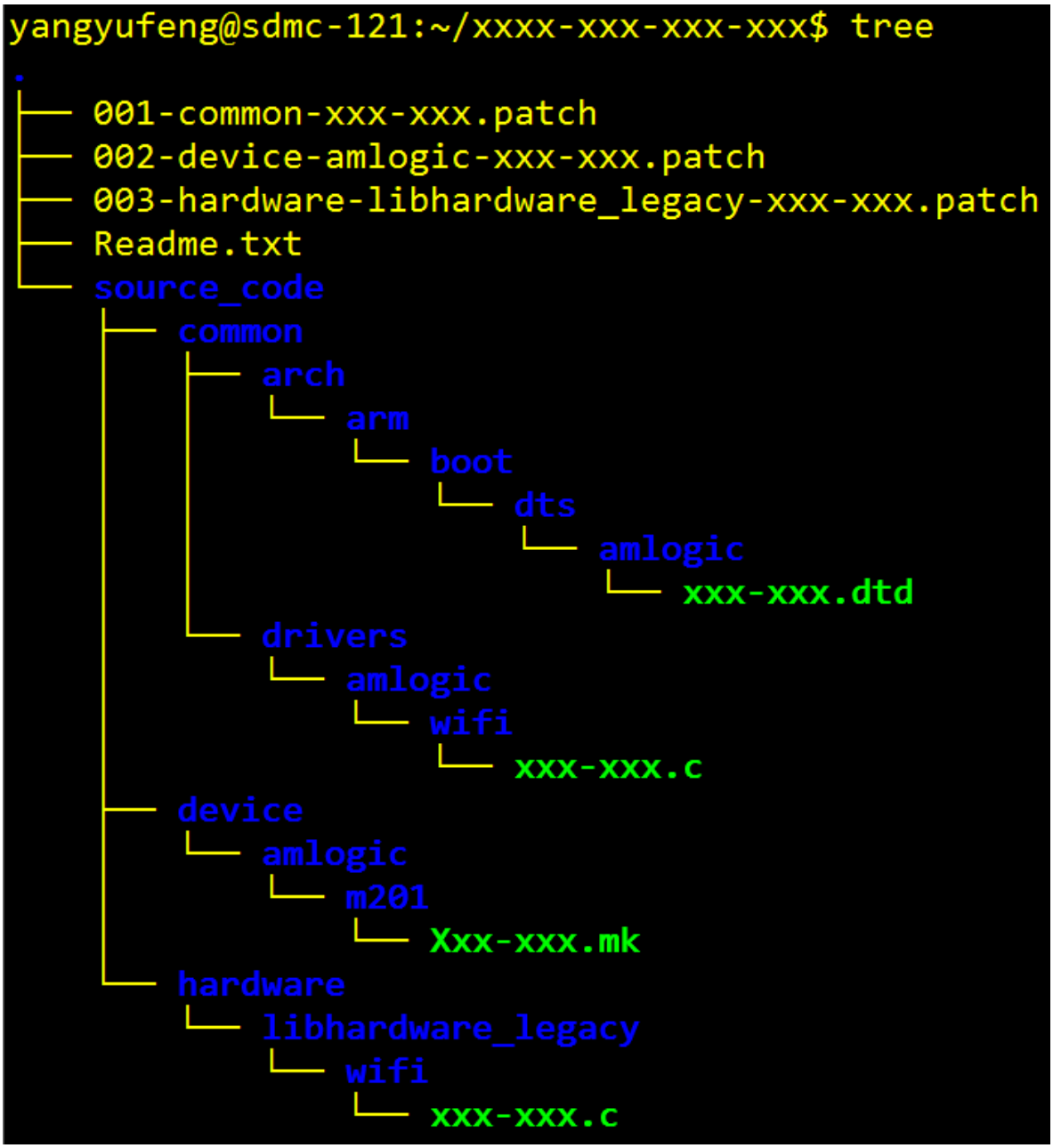

Figure 4

1) PATCH 文件

PATCH 文件夹下的.patch 类型的文件，也是要打入 SDK 的补丁文件，其命名方式见 3.2.2 PATCH 文件的命名规范。

2) source_code

打完 PATCH 后的源文件，保持其在 SDK 中的目录结构存放在 source_code 文件夹下，方便核查

3) Readme

Readme.txt 文件时对这个 PATCH 使用等相关的一个详尽描述，它是一个 TXT 类型的文件，须包含的信息如 Figure 5

```
1、[补丁名字]：xxxx-xxx-xxx-xxx
2、[创建者]：xxx
3、[对修复功能的描述]:xxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxx
4、[patch类型]：必须/非必须，如果是非必须须说明适用的patch list
5、[patch适用的SDK版本]：如20150414
6、[注意事项]
   a、xxxxxxx
   b、xxxxxxx
   c、xxxxxxx
```

Figure 5

如 Figure 6，是对 0062-add-ap6356-wifi-module PATCH 在 Readme.txt 中的描述

```
1、[补丁名字]：0062-add-ap6356-wifi-module
2、[创建者]：杨宇锋
3、[对修复功能的描述]:添加对ap6356 wifi模组的支持
4、[patch类型]：非必须，适用于需要支持ap6356wifi模组的产品
5、[patch适用的SDK版本]：20150414、20150612
6、[注意事项]
   a、由于AP6356 wifi模组适用的是高速SDHC硬件接口，而m8平台
      对SDHC接口兼容性不是很好，只能适用硬件flash是EMMC的产品
```

Figure 6

3.2 命名规范

3.2.1 PATCH 文件夹的命名规范

PATCH 文件夹的命名如 Table 4，字母全部采用小写，单词之间用‘-’隔开

[xxxx]-[describe]	
xxxx	**补丁编号，占四个字节，在 PATCH 集合中根据 PATCH 数量线性递增，为避免多人协作的时候编号冲突，在提交之前先更新到服务器上最新的 PATH**
describe	**PATCH 的功能描述**

Table 4

如：**0062-add-ap6356-wifi-module**
0062：是补丁编号，即第 0062 号 PATCH
add-ap6356-wifi-module：PATCH 的功能描述，即“add ap6356 wifi module”

3.2.2 PATCH 文件的命名规范

PATCH 文件命名格式如 Table 5，字母全部采用小写，单词之间用'-'隔开

[xxx]-[path]-[describe]	
xxx	**补丁编号，占三个字节，根据 PATH 的不同来进行编号**
path	**合并目录，补丁打入的 SDK 路径(须是 git 的根目录)**
describe	**PATCH 的功能描述**

Table 5

如：003-hardware-libhardware_legacy-add-ap6356-wifi-module.patch
003：补丁编号，占三个字节，根据 PATH 的不同来进行编号
hardware-libhardware_legacy：合并目录，即这个补丁要打入的 SDK 路径是 hardware / libhardware_legacy，这个路径是 git 的第一级路径，叫做 git 的根目录
add-ap6356-wifi-module：PATH 的功能描述，即"add ap6356 wifi module"

3.2.3 PATCH LIST 的命名规范

PATCH LIST 是需要打的补丁集合列表，是一个 TXT 类型的文件，格式如 Table 6，字母全部采用小写，单词之间用'-'隔开，补丁之间用"&"隔开，合并目录和补丁之间用"|"隔开。合并目录和补丁表示一个单独的补丁。

[patch 文件夹]&[path]| [patch]& [patch]| [patch]...	
patch 文件夹	**用来存放补丁相关文件，其命名方式见 3.2.1 PATCH 文件夹的命名规范**
path	**见 Table 5 中关于 path 的说明**
patch	**要打入 SDK 的补丁文件，其命名方式见 3.2.2 PATCH 文件的命名规范**

Table 6

如：**0042-add-demo-reset&uboot|001-uboot-add-demo-reset.patch**
0042：补丁的编号，占四个字节，是在 PATCH 集合中根据 PATCH 数量线性递增的编号
add-demo-reset：补丁提交的 commit 注释，即用 git commit 最后提交的注释是 add demo reset
uboot：合并补丁的 git 根目录
001-uboot-add-demo-reset.patch：这个才是要打入 SDK 中的 PATCH 文件

3.3 PATCH 管理

3.3.1 权限管理

整个 PATCH 的参与者及其拥有的权限如 Table 7。

SDK	原厂 SDK	SDMC SDK
管理员	RW	RW
职能管理者	RW	RW
主开发者	R	RW
协助开发者	R	R

Table 7

3.3.2 PATCH 协作

当多个开发者参与到 SDK 的维护中时，为了便于管理，协助开发者只需要按照本规范来生成 PATCH，以邮件的形式发送给主开发者，并抄送给相关人员，由主开发者提交到版本控制中心。如果 PATCH 不够规范，主开发者可以要求协助开发者重新整理 PATCH。

3.3.3 PATCH 的更新和删除

1) PATCH 更新

当某个 PATCH 有更新时，需要重新生成一个完整的 PATCH，不接受在已有的 PATCH 上打 PATCH。重新生成后，同样以邮件的形式发送给主开发者，并抄送给相关人员，由主开发者提交到版本控制中心。

2) PATCH 删除

当需要删除某一个已有的 PATCH 或者有协助开发者告知主开发者需要删除某个 PATCH 时，主开发者只需将这个 PATCH 从 PATCH LIST 中删除即可。

3.3.4 PATCH LIST 的产生

产品研发初期，PATCH LIST 需要从零开始产生。这个时候需要按照 3.2.3 PATCH LIST 的命名规范一条一条的创建。

当某一个平台的某一个产品需求不能兼容现有的 PATCH LIST 时，需要在现有的 PATCH LIST 中，根据需求拷贝一份，并按照 2.3.3 PATCH LIST 的命名规范重新命名，然后将新的 PATCH 按照 3.2.3 PATCH LIST 的命名规范添加进新的 PATCH LIST。

注意：由于 shell 读行时是以回车符来确认的，在最后一个功能补丁后面记得回车换行，不然最后一个补丁合并不进去。

4 SDK 下载说明

4.1 权限管理

基础平台 SDK 的权限管理是以最低权限为原则来管理的，即每个工程师默认是没有下载 SDK 的权限的，只有向相关人员申请到下载 SDK 的权限才有权限下载。

4.2 下载方法

申请到权限后，原厂 SDK 和 SDMC SDK 下载方法请参考文档《基础平台操作方法》相关章节，此文档存放在“svn://10.10.121.5/project/technology_docs/05.平台资料/04.平台操作指南文档/”路径下，会持续更新。

5 打包材料发布说明

打包材料的发布随着打包工程的变更，需要发布的打包材料也有微小变动，但大致需要发布 Figure 7 中几个部分，即 bootloader、kernel、frameworks、pkg（主要用来验证基础平台问题），发布规范需遵循《产品软件版本管理规范》中关于模块发布的规范。

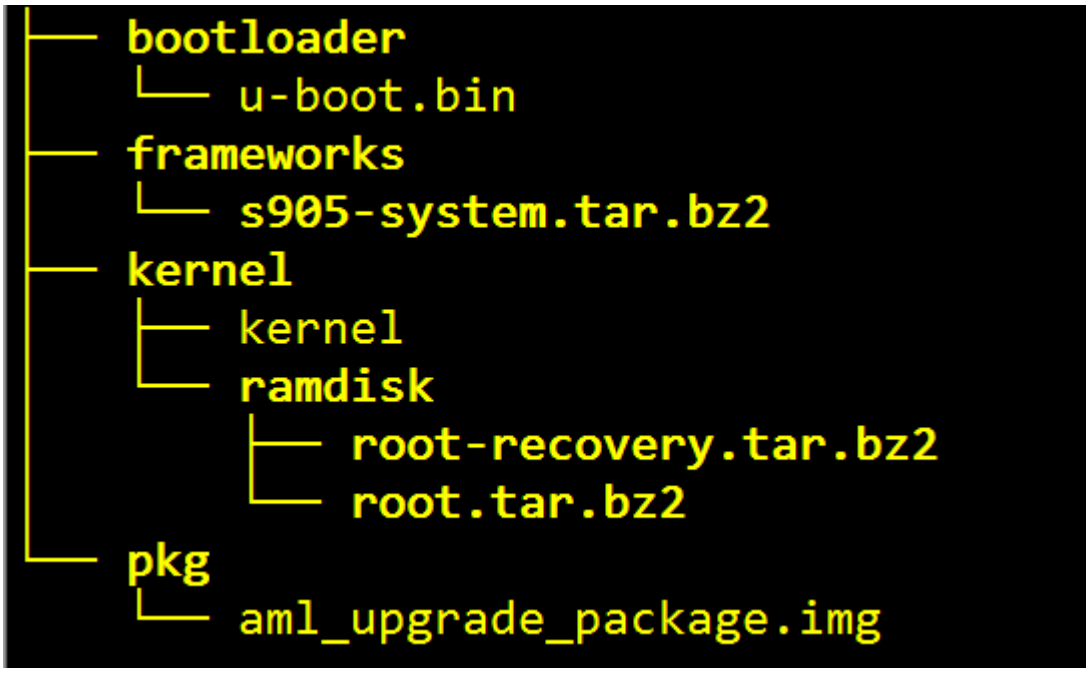

Figure 7