Tina Linux 打包流程
说明指南

版本号: 1.1
发布日期: 2022.02.22

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.03.27	AWA1615	first version
1.1	2022.02.22	AWA1315	添加 pack out of TIna & 数据流程分析 & FAQ

目　录

1 概述

1.1 编写目的

介绍 Allwinner 平台上打包流程。

1.2 适用范围

Allwinner 软件平台 Tina v3.0 版本以上。

1.3 相关人员

适用 Tina 平台的广大客户，想了解 Tina 打包流程的开发人员。

2 固件打包简介

固件打包是指将我们编译出来的 bootloader、内核和根文件系统一起写到一个镜像文件中，这个镜像文件也叫固件。然后可以将这个镜像写到 nand、nor flash 或是 sd 卡上，从而启动系统。打包成固件时需要使用到一些打包工具，打包脚本以及打包配置文件。本文主要就是介绍打包时需要哪些工具，需要哪些配置文件，以及固件的生成流程。

3 打包工具介绍

本文只介绍 Tina 打包时特有的工具，其他通用工具如 unix2dos 等请自行百度。在 tina SDK 中特有的打包工具保存在如下路径：

```
tina/tools/pack-bintools/src
```

3.1 update_mbr

工具名称	update_mbr
功能说明	根据分区配置文件，更新主引导目录文件 sunxi_mbr.fex ，sunxi_gpt.fex 及分区的下载文件列表 dlinfo.fex 。
使用方法	update_mbr <partition_file> (mbr_count) 如果不指定 mbr_count，mbr_count = 4; update_mbr <partition_file> <mbr_countnt> <output_name> 使用此用法必须指定 mbr_count，本来输出的 sunxi_mbr.fex 会改名为 output_name。
参数说明	partition_file：分区配置文件，如 sys_partition.bin mbr_count：mbr 的备份数量，如果不指定，缺省 mbr_count = 4; output_name：修改 sunxi_mbr.fex 的输出名，没有特殊需求不建议使用此用法。
应用举例	update_mbr sys_partition.bin 4 update_mbr sys_partition.bin 1 sunxi_mbr_tmp.fex （没有特殊需求不建议使用此用法）

3.2 merge_full_img

工具名称	merge_full_img
功能说明	指定起始逻辑地址，把 boot0，boot1，mbr 及分区文件下载列表里的文件合并成 img 固件包，应用于小容量的 nor flash。此时没有分区的概念。
使用方法	merge_full_img –out <outfile> \ –boot0 <boot0.fex> \ –boot1 <boot1.fex> \ –mbr <mbr.fex> \ –partition <partition.fex> \

工具名称	merge_full_img
	–logic_start <512|256> –help
参数说明	–out <outfile>：指定输出目标文件 –boot0 <boot0.fex>：指定输入的 boot0 文件 –boot1 <boot1.fex>：指定输入的 boot1 文件 –mbr <mbr.fex>：指定输入的 mbr 文件 –partition <partition.fex>：指定输入的分区配置文件 –logic_start <512|256>：指定起始逻辑地址 –help：显示使用方法
应用举例	merge_full_img –out full_img.fex \ –boot0 boot0_spinor.fex \ –boot1 ${BOOT1_FILE} \ –mbr sunxi_mbr.fex \ –logic_start ${LOGIC_START} \ –partition_file

3.3 script

(1) 注意: 此处讲述的不是 Linux 通用的 script 工具（Linux 下 script 工具用于终端会话录制）

(2) 它是全志实现的一个同名工具，工具功能说明如下：

工具名称	script
功能说明	解析输入文本文件的所有数据项，生成新的二进制 bin 文件，以便程序解析。 生成的目标文件与源文件名字 (除后缀) 一样，但后缀为.bin。
使用方法	script <source_file>
参数说明	source_file：输入的文本文件，可多个
应用举例	scriptsys_config.fex scriptsys_partition.fex

3.4 dragonsecboot

工具名称	dragonsecboot
功能说明	1) 根据指定的 keys 生成 toc0 文件。 2) 根据指定的 keys 和 cnfbase 生成 toc1 文件。 3) 根据配置文件生成 keys。 4) 按配置文件的配置进行打包生成目的文件。

工具名称	dragonsecboot
使用方法	dragonsecboot -toc0 <cfg_file> <keypath> <version_file> dragonsecboot -toc1 <cfg_file> <keypath> <cnfbase> <version_file> dragonsecboot -key <cfg_file> <keypath> dragonsecboot -pack <cfg_file>
参数说明	-toc0：表示要生成 toc0 文件 -toc1：表示要生成 toc1 文件 -key：表示要生成 key -pack：表示进行打包 cfg_file：配置文件 keypath：key 的路径 cnfbase：输入的 cnf_base.cnf 文件 version_file：固件防回滚配置文件
应用举例	dragonsecboot -pack boot_package.cfg dragonsecboot -key dragon_toc.cfg keys dragonsecboot -toc0 dragon_toc.cfg keys version_base.mk dragonsecboot -toc1 dragon_toc.cfg keys cnf_base.cnf version_base.mk

3.5 update_boot0

工具名称	update_boot0
功能说明	根据配置脚本内容，修正 boot0 头部的参数。修正参数：debug_mode、dram_para 参数、uart 参数、bootcpu、jtag 参数、NAND 参数等。
使用方法	update_boot0 <boot0> <sys_config_file> <storage_type>
参数说明	boot0：boot0 文件 sys_config_file：系统配置文件 storage_type：存储介质类型
应用举例	update_boot0 boot0_nand.fex sys_config.bin NAND update_boot0 boot0_sdcard.fex sys_config.bin SDMMC_CARD update_boot0 boot0_spinor.fex sys_config.bin SDMMC_CARD

3.6 update_dtb

工具名称	update_dtb
功能说明	把 Linux 设备树二进制 dtb 文件进行 512 字节对齐后再预留空间。
使用方法	update_dtb <dtb_file> <reserve_size>
参数说明	dtb_file：输入的 Linux 设备树二进制 dtb 文件

工具名称	update_dtb
	reserve_size：输出目标文件预留多少字节
应用举例	update_dtb sunxi.fex 4096

3.7 update_fes1

工具名称	update_fes1
功能说明	从系统配置文件中取出数据对 fes1 头部相关参数进行修正。 修正参数包括：DRAM 参数、UART 参数、JTAG 参数等。
使用方法	update_fes1 <fes1_file> <config_file>
参数说明	fes1_file：更修正的 FES1 文件 config_file：输入的系统配置文件
应用举例	update_fes1 fes1.fex sys_config.bin

3.8 signature

工具名称	signature
功能说明	对 MBR 指定要进行签名的分区文件进行签名。
使用方法	signature <sunxi_mbr_file> <dlinfo_file>
参数说明	sunxi_mbr_file：输入的 MBR 文件 dlinfo_file：输入的分区下载列表文件
应用举例	signature sunxi_mbr.fex dlinfo.fex

3.9 update_toc0

工具名称	update_toc0
功能说明	从系统配置文件中取出相关参数对 toc0 配置参数进行修正，修正参数包括： DRAM、UART、JTAG、NAND、卡 0 、卡 2 、secure 参数等。
使用方法	update_toc0 <toc0_file> <config_file>
参数说明	toc0_file：输入的 toc0 文件 config_file：输入的系统配置文件
应用举例	update_toc0 toc0.fex sys_config.bin

3.10 update_uboot

工具名称	update_uboot
功能说明	从系统配置文件中取出相关参数对 uboot 头部参数进行修正。 修正参数包括：UART 参数、TWI 参数、target 参数、SDCARD 参数等。
使用方法	update_uboot <uboot_file> <config_file> update_uboot -merge <uboot_file> <config_file> update_uboot -no_merge <uboot_file> <config_file>
参数说明	uboot_file：要更新的 uboot 文件 config_file：系统配置文件 -merge：系统配置文件会拼接在 uboot 文件尾部 -no_merge：系统配置文件不会拼接在 uboot 文件尾部 注意：没有显式指明-no_merge 参数默认会把系统配置文件拼接在 uboot 文件尾部
应用举例	update_uboot u-boot.fex sys_config.bin update_uboot -merge u-boot.fex sys_config.bin update_uboot -no_merge u-boot.fex sys_config.bin

3.11 update_scp

工具名称	update_scp
功能说明	从系统配置文件中取出相关参数对 scp （小 cpu 运行代码只有带有小 cpu 方案的芯片会用到）头部参数进行修正。修正参数包括：UART 参数、dram_para 参数等。
使用方法	update_scp <scp_file> <config_file>
参数说明	uboot_file：要更新的 scp 文件 config_file：系统配置文件
应用举例	update_scp scp.fex sunxi.fex

3.12 u_boot_env_gen

工具名称	u_boot_env_gen
功能说明	解析 env 文件生成 uboot 能识别的 env 二进制数据文件，功能与标准的 mkenvimage 工具类似。
使用方法	u_boot_env_gen <env_file> <env_bin_file>
参数说明	env_file：输入的 evn 文件 env_bin_file：输出的 env 二进制文件

工具名称	u_boot_env_gen
应用举例	u_boot_env_gen env.cfg env.fex

3.13 fsbuild

工具名称	fsbuild
功能说明	根据 boot-resource.ini 生成 fat 格式文件。
使用方法	fsbuild <rootfs_config_file> <magic_file>
参数说明	rootfs_config_file：fat 系统配置文件 magic：用于 fat 文件系统校验
应用举例	fsbuild boot-resource.ini split_xxxx.fex

3.14 update_toc1（旧版本工具，后面会弃用，可以不理会）

工具名称	update_toc1
功能说明	从系统配置文件中取出相关参数对 toc1 配置参数进行修正，修正参数包括： board_id_simple_gpio 参数（需配置启用，目前已没有方案使用）。
使用方法	update_toc1 <toc1_file> <config_file>
参数说明	toc1_file：输入的 toc1 文件 config_file：输入的系统配置文件
应用举例	update_toc1 toc1.fex sys_config.bin

3.15 programmer_img

工具名称	programmer_img
功能说明	生成 mmc 介质的烧录固件。
使用方法	programmer_img <boot0_file> <uboot_file> <out_img> programmer_img <partition file> <mbr_file> <out_img> <in_img>
参数说明	in_img：输入的文件或镜像 boot0_file：boot0 文件 uboot_file：uboot 文件 partition_file：分区配置文件 mbr_file：sunxi_mbr 文件

工具名称	programmer_img
应用举例	output_img：输出的文件或镜像 programmer_img boot0_sdcard.fex boot_package.fex ${out_img} programmer_img sys_partition.bin sunxi_mbr.fex ${out_img} ${in_img}

3.16 dragon

工具名称	dragon
功能说明	根据 img 配置文件和分区配置文件生成固件。
使用方法	dragon <img_config> <partition_file>
参数说明	img_config：配置文件，描述 img 文件格式和包含其他文件列表 partition_file：分区配置文件
应用举例	dragon image.cfg sys_partition.fex

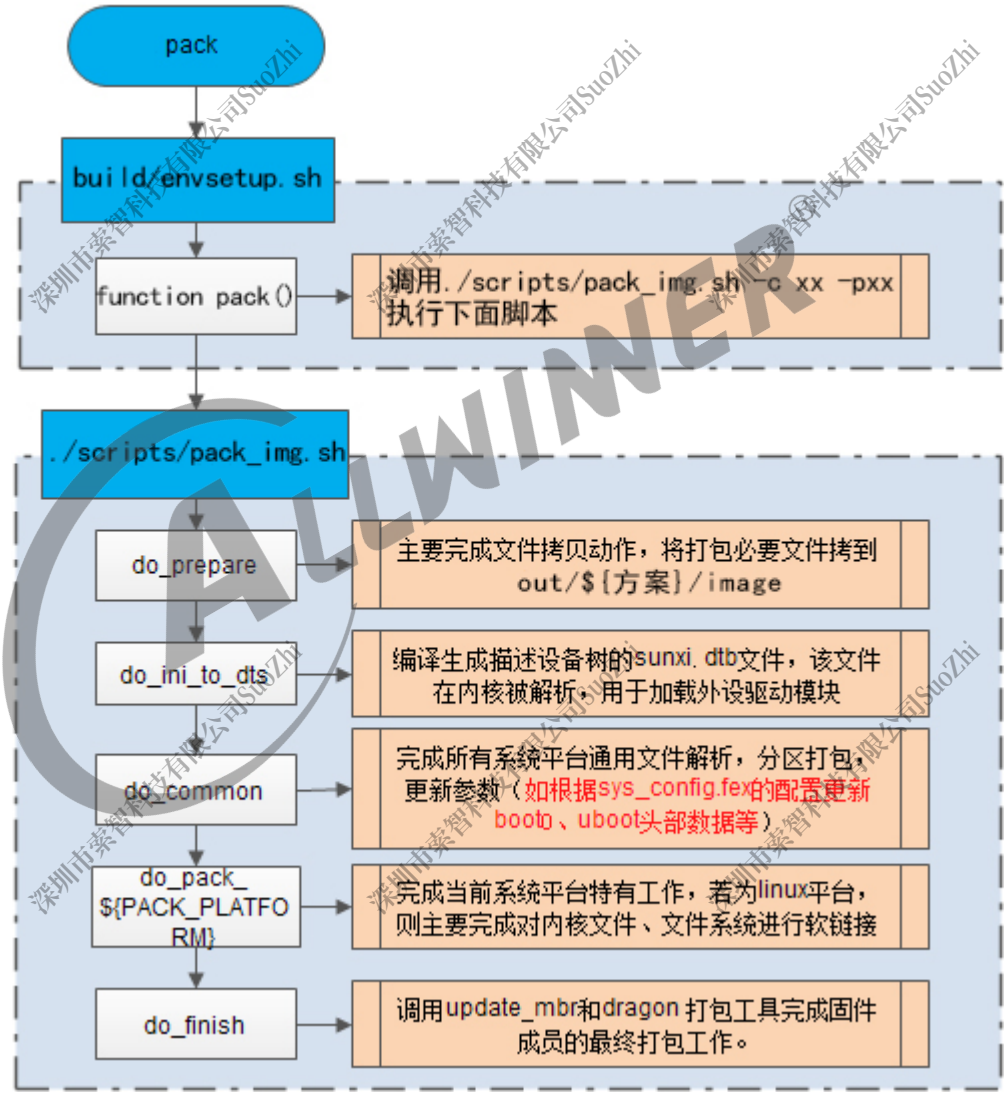
3.17 sigbootimg

工具名称	sigbootimg
功能说明	在输入文件的后面添加一个证书，并输出一个带证书的文件。
使用方法	sigbootimg –image <input_img> –cert <cert_file> –output <output_img>
参数说明	input_img：输入的文件或镜像 cert_file：文件或镜像对应的证书 output_img：带证书的文件或镜像
应用举例	sigbootimg –image boot.fex –cert boot.fex –output boot_sig.fex

4 打包脚本分析

4.1 脚本的调用流程

在 Tina 主目录下执行 make -j16 编译完成后便可以执行 pack 进行打包工作，整个打包过程大概如下图所示：



结合上面的打包流程图分析，打包方法在 Tina SDK 根目录下运行：

```
pack
```

最终打包出来的固件放在目录 tina/out/$platform-${board}/下。

pack 命令实质上是 tina SDK 内置的一个环境变量命令。

在使用 tina SDK 时需要执行 source build/envsetup.sh 这个命令。

这个命令是把 tina SDK 实现的一些 shell 命令 export 到当前 shell 中。

打开 build/envsetup.sh 脚本，可以发现里面实现了一个 shell 函数：

```
function  pack()
```

在 tina 根目录下执行 pack 命令后调用到的就是 build/envsetup.sh 脚本中的 function pack() 函数，function pack() 函数进行一些参数设置后最终调用到以下语句：

```
$T/scripts/pack_img.sh -c $chip -p $platform -b $board -d $debug -s $sigmode -m $mode -w
    $programmer -v $securemode -i $tar_image -t $T


-c: 输入的芯片类型例如: sun8iw18p1

-p: 输入的平台例如: tina

-b: 输入的板级方案例如: r328s2-perf1

-d: 输入调试时log输入方式例如: uart0/card0

-s: 输入是否打包成安全固件例如: none/secure

-m: 输入是正常固件还是调试用的dump固件: normal/dump

-w:

-v: 输入打包时是否使用安全boot，对于tinaSDK来说该参数功能已由-s替代，参数只是历史遗留或做兼容用，客户可以不
    用理会该参数

-i: 输入是否制作压缩包none/tar_image，调试时用，客户可以不用理会该参数

-t: Tina根目录的路径
```

从上面可以看出 function pack() 函数最终调用到 tina/scripts/pack_img.sh 这个脚本文件，这个脚本文件实现了打包的最终流程。

目前打包脚本主要分为 5 个阶段（其他阶段都是一些特殊化处理），分别是：

```
do_prepare

do_ini_to_dts

do_common

do_pack_tina

do_finish
```

4.2 打包的各阶段分析

4.2.1 do_prepare 阶段

此阶段完成文件拷贝动作。打包时需要拷贝若干文件到 tina/out/xxxplatform/image 目录下，目前脚本对其进行了分类，分别是 tools_file_list，configs_file_list，boot_resource_list 和 boot_file_list，boot_file_secure，a64_boot_file_secure 如有新增文件，可以归入其中一类或者创建新类，后续打包会使用到这些文件。

```
function do_prepare()

{

    ......

    #拷贝 tools_file_list 类文件到 tina/out/xxxplatform/image 目录下

    printf "copying tools file\n"

    for file in ${tools_file_list[@]} ; do

        cp -f  $file ${ROOT_DIR}/image/ 2> /dev/null

    done

    ......

    #拷贝 configs_file_list 类文件到 tina/out/xxxplatform/image 目录下

    printf "copying configs file\n"

    for file in ${configs_file_list[@]} ; do

        cp -f  $file ${ROOT_DIR}/image/ 2> /dev/null

    done

    ......

    #拷贝 boot_resource_list 类文件到 tina/out/xxxplatform/image 目录下

    printf "copying boot resource\n"

    #根据不同的arm架构拷贝 不同的boot_file_secure 类文件到tina/out/xxxplatform/image 目录下

    #32位系统

    printf "copying secure boot file\n"

    for file in ${boot_file_secure[@]} ; do

        cp -f `echo $file | awk -F: '{print $1}'` \

            ${ROOT_DIR}/`echo $file | awk -F: '{print $2}'`
```

```
    done
    #64位系统
    printf "copying arm64 secure boot file\n"
    for file in ${a64_boot_file_secure[@]} ; do
        cp -f `echo $file | awk -F: '{print $1}'` \
            ${ROOT_DIR}/`echo $file | awk -F: '{print $2}'`
    done
}
```

4.2.2 do_ini_to_dts 阶段

在 linux-3.10，引入了 linux 设备树的概念。此阶段主要是编译生成描述设备树的 sunxi.dtb 文件。该文件在 linux 内核启动过程中会被解析，根据该文件中设备列表进行加载各个外设的设备驱动模块。具体实现分析如下：

```
function do_ini_to_dts()
{
    if [ "x${PACK_KERN}" == "xlinux-3.4" ] ;
        then return
    fi
    ......
    #根据不同的内核设置不同的参数，最后调用下的命令编译生成.dbt 文件
    $DTC_COMPILER  ${DTC_FLAGS} -O dtb -o ${ROOT_DIR}/image/sunxi{SUFFIX}.dtb \
                                -b 0\
                                -i $DTC_SRC_PATH\
                                -F $DTC_INI_FILE\
                                -d $DTC_DEP_FILE $DTC_SRC_FILE    &int>  /dev/null
    if [ $? -ne 0 ]; then
        pack_error "Conver script to dts failed" exit 1
    fi
    printf "Conver script to dts ok.\n"
    ......
```

```
}
```

4.2.3 do_common 阶段

此阶段完成所有系统平台通用的文件解析，分区打包。具体实现分析如下。(代码顺序与脚本的不一致，主要是为了方便说明)，该阶段与存储介质、内核版本等有耦合。因此比较复杂，但主要包括下面的 5 个阶段：

(1) 使用 unix2dos 工具确保文本文件为 dos 格式。

(2) 使用 script 工具解析文本文件，生成对应的二进制文件，便于后续工具解析。

(3) 更新 boot0,uboot,scp 的头部参数。

(4) 生成 boot_package。

(5) 生成 env 分区数据 env.fex。

具体实现分析如下：

```
function do_common()

{

    busybox unix2dos sys_config.fex

    busybox unix2dos sys_partition.fex

    busybox unix2dos sys_partition_nor.fex

    #使用 script 程序解析文本文件 sys_config.fex 和 sys_partition.fex/sys_partition_nor.fex

    #生成相应的二进制文件 sys_config.bin 和 sys_partition.bin 便于后续工具程序解析

    script    sys_config.fex > /dev/null

    script    sys_partition.fex > /dev/null

    script    sys_partition_nor.fex > /dev/null

    #根据 sys_config.bin 参数,取出 DRAM,UART 等参数更新 boot0 头部参数

    update_boot0    boot0_nand.fex    sys_config.bin    NAND > /dev/null

    update_boot0    boot0_sdcard.fex    sys_config.bin     SDMMC_CARD > /dev/null

    #根据 sys_config.bin 参数设置,更新 uboot 头部参数

    update_uboot    u-boot.fex    sys_config.bin > /dev/null

    #根据 sys_config.bin 参数设置,更新 fes1.fex 参数

    update_fes1    fes1.fex    sys_config.bin > /dev/null
```

```
    #制作启动过程相关资源的分区镜像

    fsbuildboot-resource.inisplit_xxxx.fex > /dev/null

    #根据配置生成 uboot 基本配置二进制文件env.fex

    mkenvimage -r -p 0x00 -s ${env_size} -o env.fex env_burn.cfg

    u_boot_env_gen env.cfg env.fex > /dev/null

    #根据boot_package.cfg配置生成 boot_package

    echo "pack boot package"

    busybox unix2dos boot_package.cfg

    dragonsecboot -pack boot_package.cfg
}
```

4.2.4 do_pack_tina 阶段

此阶段完成当前系统平台特有的工作以及安全相关的工作，主要对内核文件，文件系统等进行软链接，以及安全件的签名和 toc0 的生成。

具体实现分析如下：

```
function do_pack_tina()

{

    #软链接boot.fex，rootfs.fex

    ln -s ${ROOT_DIR}/boot.img     boot.fex

    ln -s ${ROOT_DIR}/rootfs.img     rootfs.fex

    ......

    #如果需要打包成安全固件就会调用do_signature函数

    do_signature

    {

        #生成toc0文件

        dragonsecboot -toc0 dragon_toc.cfg ${ROOT_DIR}/keys ${ROOT_DIR}/image/version_base.
    mk

        #根据 sys_config.bin 参数，取出 DRAM，UART 等参数更新 toc0 头部参数

        update_toc0    toc0.fex    sys_config.bin

        ......
```

```
        #生成toc1文件
        dragonsecboot -toc1 dragon_toc.cfg ${ROOT_DIR}/keys \
                           ${CNF_BASE_FILE} \
                           ${ROOT_DIR}/image/version_base.mk
        #对内核进行签名
        sigbootimg --image boot.fex --cert toc1/cert/boot.der --output boot_sig.fex
        #根据 sys_config.bin 参数，取出 DRAM，UART 等参数更新 toc1 头部参数
         update_toc1  toc1.fex           sys_config.bin
    }
}
```

4.2.5 do_finish 阶段

此阶段根据指定的固件成员完成打包。

具体实现分析如下：

```
function do_finish()
{
    ......
    #生成分区结构文件 sunxi_mbr.fex 及分区下载文件列表文件 dlinfo.fex
    update_mbr    sys_partition.bin    4 > /dev/null
    #根据所列的成员文件及分区信息,组合完成打包
    dragon    image.cfg    sys_partition.fex
    ......
}
```

4.3 打包过程数据流分析

上一章节讲述了打包脚本的几个阶段，本节我们换个视角看打包过程。打包过程是将编译好后的二进制镜像和各种配置文件，通过一些加工、转换和合成最终生成固件包。

如下图所示，讲述了需要打包的各种文件在哪个位置，会经过如何处理，合成什么文件，最终生成了固件包。

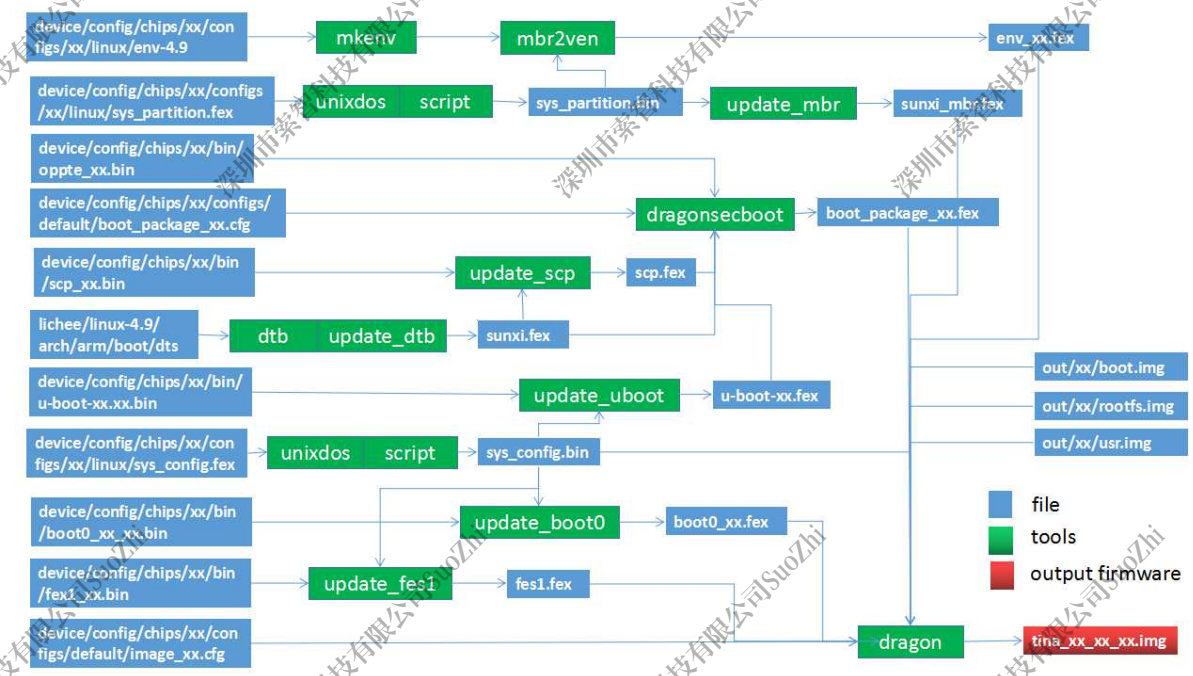


图 4-1：打包数据流

其中蓝色的是源文件或者生成的中间文件；

绿色的是打包用到的工具；

红色的是最终生成的固件。

固件里面包含哪些文件，可以参考下一章节。

这里展示另一张图，可以从 Tina SDK 全局看打包的作用位置，以及数据的流动。

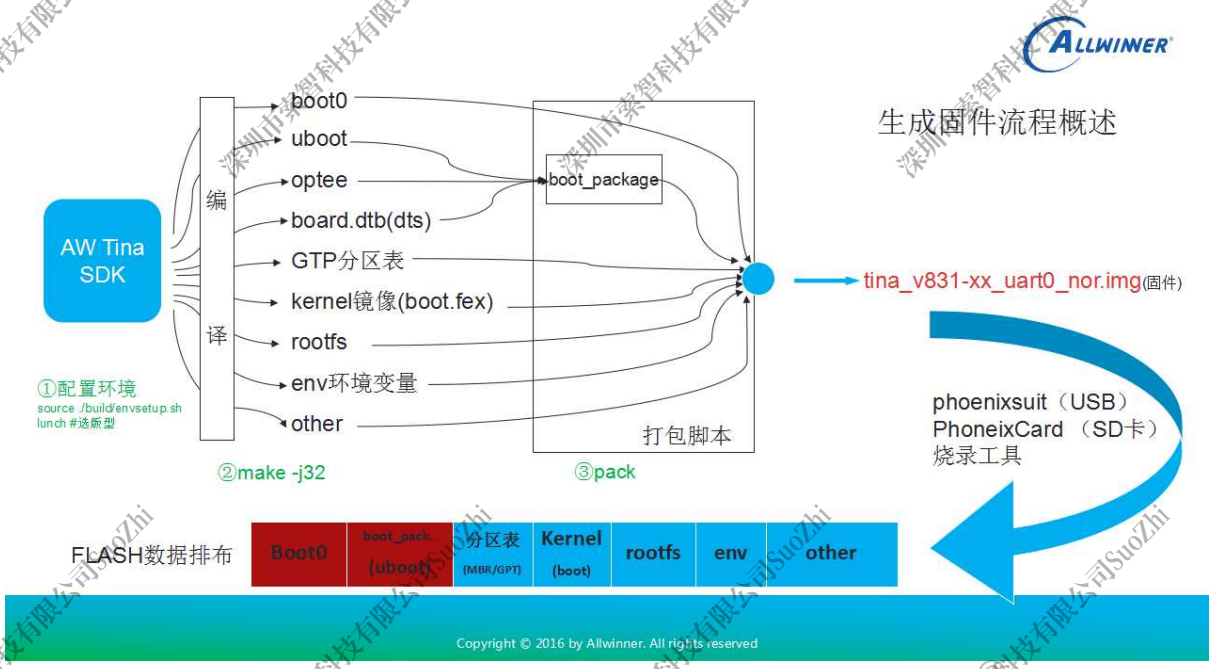


图 4-2: SDK 整体数据流程

4.4 固件组成成员分析

固件包本质是由一系列的文件组成，类似于一个压缩包，把多个文件压缩成了一个固件包。这里通过一个描述性的配置文件 (image.cfg)，把需要添加到固件包的文件枚举出来。然后打包过程就读取这个配置文件，生成了最终的固件包。由 do_finish 函数可以知道，生成固件的工具是 dragon，dragon 工具需要 2 个配置文件 image.cfg 和 sys_partition.fex，下面将会分析这 2 个配置文件。

4.4.1 image.cfg 配置文件分析

用文本方式，打开 tina/out/xxxplatform/image/image.cfg 文件，可以看到大致如下的内容：

```
[FILELIST]

    {filename = "sys_config.fex",       maintype = ITEM_COMMON,       subtype = "
    SYS_CONFIG100000",},

    {filename = "config.fex",           maintype = ITEM_COMMON,       subtype = "
    SYS_CONFIG_BIN00",},

    {filename = "board.fex",            maintype = ITEM_COMMON,       subtype = "
    BOARD_CONFIG_BIN",},

    {filename = "split_xxxx.fex",       maintype = ITEM_COMMON,       subtype = "
    SPLIT_0000000000",},
```

```
    {filename = "sys_partition.fex",    maintype = ITEM_COMMON,       subtype = "
    SYS_CONFIG000000",},

    {filename = "sunxi.fex",            maintype = ITEM_COMMON,       subtype = "
    DTB_CONFIG000000",},


    {filename = "boot0_nand.fex",       maintype = ITEM_BOOT,         subtype = "
    BOOT0_0000000000",},

    {filename = "boot0_sdcard.fex",     maintype = "12345678",        subtype = "1234567890
    BOOT_0",},

    {filename = "u-boot.fex",           maintype = "12345678",        subtype = "
    UBOOT_0000000000",},

    {filename = "toc1.fex",             maintype = "12345678",        subtype = "
    TOC1_00000000000",},

    {filename = "toc0.fex",             maintype = "12345678",        subtype = "
    TOC0_00000000000",},

    {filename = "fes1.fex",             maintype = ITEM_FES,          subtype = "FES_1
    -0000000000",},

    {filename = "boot_package.fex",     maintype = "12345678",        subtype = "BOOTPKG
    -00000000",},


    ;-------------------------------usb量产部分-------------------------------------;

    ;-->tools文件

    {filename = "usbtool.fex",          maintype = "PXTOOLSB",        subtype = "
    xxxxxxxxxxxxxxxx",},

    {filename = "aultools.fex",         maintype = "UPFLYTLS",        subtype = "
    xxxxxxxxxxxxxxxx",},

    {filename = "aultls32.fex",         maintype = "UPFLTL32",        subtype = "
    xxxxxxxxxxxxxxxx",},


    ;-------------------------------卡量产部分-------------------------------------;

    ;-->固定不变的PC使用

    {filename = "cardtool.fex",         maintype = "12345678",        subtype = "1234567890
    cardtl",},

    {filename = "cardscript.fex",       maintype = "12345678",        subtype = "1234567890
    script",},


    ;-->需要烧写到卡上的文件

    {filename = "sunxi_mbr.fex",        maintype = "12345678",        subtype = "1234567890
    __MBR",},
```

```
    {filename = "dlinfo.fex",           maintype = "12345678",        subtype = "1234567890
    DLINFO",},

    {filename = "arisc.fex",            maintype = "12345678",        subtype = "1234567890
    ARISC" ,},

    ;镜像配置信息

[IMAGE_CFG]

    version = 0x100234                ;-->Image的版本

    pid = 0x00001234                  ;-->产品ID

    vid = 0x00008743                  ;-->供应商ID

    hardwareid = 0x100                ;-->硬件ID bootrom

    firmwareid = 0x100                ;-->固件ID bootrom

    bootromconfig = "bootrom_071203_00001234.cfg"

    rootfsconfig = "rootfs.cfg"

    filelist = FILELIST

    imagename = tina_XXXXXX.img
```

该文件项的格式：

```
filename= name,maintype=ITEM_ROOTFSFAT16,subtype = user_define
```

当用户需要添加文件的时候，按照同样的格式，把自己需要的文件写到脚本文件中即可。

- **filename：打包文件**

是指文件的全路径。可以使用相对路径，如上述文件中，就使用了相对路径。

- **maintype：打包格式**

表明文件的格式类型，该文件有此类型定义的列表。

- **subtype：自定义名称**

用户自己定义的名称，使用数字和英文字符 (区分大小写)，最大长度必须为 16 字节。

只要按照上述规则书写，并放到文件的 [FILELIST] 之后，等到打包的时候就会自动把文件添加到固件包中。

下表描述 image.cfg 文件中的各固件成员的作用。

固件成员	成员作用
sys_config.fex	从具体的 board 配置目录拷贝而来，此文件描述了一些硬件相关的配置, 用户可以修改该配置。如果不关心某项配置，则可以直接删除该项，此时该项使用默认值。
config.fex	硬件相关的配置信息的二进制文件，用于程序解析。
split_xxxx.fex	作为 fsbuild 的其中一个输入参数。
sys_partition.fex	规划分区文件，指明存储设备上的分区个数，并由用户定义分区属性。当烧写固件包后，存储设备上就会存在这样由用户定义的分区。
sunxi.fex	描述设备树的配置信息，内核会将这些资源展开相应的设备。
boot0_nand.fex	boot0 编译生成的 nand 启动目标代码，在 SRAM 中运行，主要作用是初始化 DRAM，并从外部存储器 nand 中加载 UBOOT。对 UBOOT 做效验并跳转到 UBOOT 执行。
boot0_sdcard.fex	boot0 编译生成的 sdcard 启动目标代码，在 SRAM 中运行，主要作用是初始化 DRAM，并从外部存储器 sdcard 中加载 UBOOT。对 UBOOT 做效验并跳转到 UBOOT 执行。
u-boot.fex	u-boot 编译生成的目标代码，主要作用是初始化时钟设置，电源管理，卡量产，USB 烧写等，最后加载内核。
toc1.fex	用于 secboot 。
toc0.fex	用于 secboot 。
fes1.fex	用于初始化 DRAM，并返回初始化的结果。在小机进行 USB 量产或升级时，需要先运行这段代码。
boot_package.fex	对指定文件进行打包，目前只包含 uboot，便于扩展
full_img.fex	用于小容量的外部存储器如 nor flash，此时没有分区概念。如果不是 nor flash 启动，这个文件会置空，该文件也可以用做烧录器使用。
usbtool.fex	usb 烧写工具插件，处理 USB 烧写的整个过程，适用于 windows 系统。
aultools.fex	usb 烧写工具插件，处理 USB 烧写的整个过程，适用于 linux 64 位系统。
aultls32.fex	usb 烧写工具插件，处理 USB 烧写的整个过程，适用于 linux 32 位系统。
cardtool.fex	Card 烧写工具，处理 card 烧写的整个过程。
cardscript.fex	指定 Card 烧写的各分区文件。
sunxi_mbr.fex	分区主引导记录。
dlinfo.fex	指定分区 download 的文件。
arisc.fex	小 CPU 的一段可执行代码，用于管理 standby，电源管理等。

4.4.2 sys_partition.fex 配置文件分析

除 image.cfg 文件所列的文件，固件还包含了 sys_partition.fex 所列的分区的文件。用文本文件打开 sys_partition.fex，可以看到大致如下的内容（主要分区有 3 个，不同方案分区表可能不一样，用户也可以添加自己的分区）：

```
[partition_start]

[partition]
    name         = env
    size         = 32768
    downloadfile = "env.fex"
    user_type    = 0x8000

[partition]
    name         = boot
    size         = 131072
    downloadfile = "boot.fex"
    user_type    = 0x8000

[partition]
    name         = rootfs
    size         = 1048576
    downloadfile = "rootfs.fex"
    user_type    = 0x8000
```

这是一个规划磁盘分区的文件，一个分区的属性，有如下几项：

- 分区名称
- 分区的大小
- 下载的文件
- 分区的用户属性

以下是文件中所描述的一个分区的属性：

- **name：分区名称**

分区名称由用户自定义。当用户在定义一个分区的时候，可以把这里改成自己希望的字符串，但是长度不能超过 16 个字节。

- **size: 分区的大小**

定义该分区的大小，以扇区的单位。

- **downloadfile: 下载的文件**

下载文件的路径和名称，可以使用相对路径，相对是指相对于 image.cfg 文件所在分区，也可以使用绝对路径。

- **user_type：分区的用户属性**

目前该标志位只有 spi nand 的 ubi 文件系统还在使用，是历史遗留问题，客户可以不理会，仿照文档中的分区填写即可（例如 0x8000）。

下表描述了 sys_partition.fex 文件指定的分区里的文件。

固件成员	成员作用
env.fex	u-boot 的基本配置文件
boot.fex	tina SDK 生成的 boot.img 的软链接，主要包含 kernel
rootfs.fex	tina SDK 生成的 rootfs 镜像的软链接，根文件系统

5 获得打包后的分区镜像文件

由于方案的差异化不同，一些方案需要获取到每个分区的镜像文件而非全志的整个固件烧录包。这些分区镜像可能主要用来做 OTA 升级，或者创建自己的烧录固件。

出于这方面考虑，Tina SDK 提供一种可以将分区镜像文件整理输出，同时输出后的可以脱离 Tina SDK 再次进行打包的方法。方便可移植到贵方 SDK 中进行二次修改和再次打包。

5.1 开启 [pack out of tina] 功能

上述功能需要开启 [pack out of tina] 功能

```
make meunconfig
    Target Image -->
        [*] support pack out of tina
```

开启 [pack out of tina] 功能后，直接 pack 即可看到打印提示多生成了一个文件夹：

```
out/xx方案/aw_pack_src
```

5.2 aw_pack_src 目录介绍

```
./aw_pack_src
|--aw_pack.sh  #执行此脚本即可在aw_pack_src/out/目录生成固件
|--config      #打包配置文件
|--image       #各种镜像文件，可替换，但不能改文件名
|    |--boot0_nand.fex         #nand介质boot0镜像
|    |--boot0_sdcard.fex       #SD卡boot0镜像
|    |--boot0_spinor.fex       #nor介质boot0镜像
|    |--boot0_spinor.fex       #nor介质boot0镜像
|    |--boot_package.fex       #nand和SD卡uboot镜像
|    |--boot_package_nor.fex   #nor介质uboot镜像
|    |--env.fex                #env环境变量镜像
|    |--boot.fex               #内核镜像
|    |--rootfs.fex             #rootfs镜像
|--other       #打包所需的其他文件
|--out         #固件生成目录
|--tmp         #打包使用的临时目录
|--rootfs      #存放rootfs的tar.gz打包,给二次修改使用
|--tools       #工具
|--lib_aw      #拷贝全志方案的库文件，如多媒体组件eyesempp等,给应用app编译链接使用(没有选择这些库，则可
    能是空文件).
```

```
|--README        #关于板级方案的一些说明，例如分区布局等等(无说明则没有这个文件夹)。
```

1. aw_pack_src 目录以及里面的文件，您可以移植到贵方 SDK 上。当您重新执行 aw_pack.sh 的时候，即会根据这个目录里面的分区文件和分区信息重新打包成生成全志的固件包。基于此，您可以手动二次修改分区镜像文件后再重新打包。
2. 您也可以将分区文件单独拿出来，去做自己的 OTA 升级包，或者做自己的 flash 固件。

说明

***Tina SDK* 原本设定好的分区大小和分区名字在 *aw_pack_src* 是无法改变的。**

6 打包常见问题 FAQ

- **Q1：镜像文件的大小超过规划的分区大小**

```
...
ERROR: dl file rootfs.fex size too large
ERROR: filename = rootfs.fex
ERROR: dl_file_size = 5888 sector
ERROR: part_size = 4864 sector
ERROR: update mbr file fail
ERROR: update_mbr failed
```

A：这里表明 rootfs 分区所关联的 rootfs.fex 镜像文件 (dl_file_size: 5888 x 512 byte) 大于分区规划的大小 (part_size = 4864 x 512 byte)

需要将 sys_partition.fex (NOR 案：sys_partition_nor.fex) 里面的分区大小改大，以便可以装下分区镜像的大小。

说明

***sys_partition.fex* 里面的分区都需要按照*flash*的擦除块大小来对齐。 例如*SPINOR Flash*，应该按照*64K* 来对齐。**

- **Q2：找不到指定的分区镜像**

```
...
ERROR:  unable to open file usr.fex
update_for_part_info -1
ERROR: update mbr file fail
ERROR: update_mbr failed
```

A: 这个错误是说明你设定的分区指定了分区镜像文件，但在打包的时候没有找到。

像这个 usr.fex 是需要开启 [CONFIG_SUNXI_SMALL_STORAGE_OTA] 才会主动生成的。

其他自建的分区镜像文件，改后缀名 xx.fex 并放以下目录即可自动找到：

```
device/config/chips/${TARGET_PLATFORM}/configs/${TARGET_PLAN}/linux
```

或者在 sys_partition.fex(NOR 方案：sys_partition_nor.fex) 里面写入绝对路径

```
...
    downloadfile = "/home/aw1315/img/DIY.fex"
```

- **Q3: NOR 方案分区设置太大**

```
...
load file: env_nor.fex ok
load file: bootlogo.fex ok
error:offset(67252224) is too large MAX_IMAGE_SIZE:67108864!
merge_package fail
```

```
...
scripts/pack_img.sh: line 1441: 73617 Segmentation fault      (core dumped) merge_full_img
    --out full_img.fex --boot0 boot0_spinor.fex --boot1 ${BOOT1_FILE} --mbr ${mbr_file} --
    logic_start ${LOGIC_START} --uboot_start ${UBOOT_START} --partition sys_partition_nor.
    bin
ERROR: merge_full_img failed
```

A：以上两种情况都是属于在 sys_partition_nor.fex 里面设置太大的分区了，需要到合适大小。考虑到 SPINOR Flash 不会超过 64M, 所以做了这个限制。

这个也是 merge_full_img 这个应用的缺陷，如果确实需要打包这么大的 Nor 固件，可能需要把 merge_full_img 这个工具注释掉。

```
#create img for nor programmer
merge_full_img --out full_img.fex \
      --boot0 boot0_spinor.fex \
      --boot1 ${BOOT1_FILE} \
      --mbr ${mbr_file} \
      --logic_start ${LOGIC_START} \
      --uboot_start ${UBOOT_START} \
      --partition sys_partition_nor.bin
if [ $? -ne 0 ]; then
        pack_error "merge_full_img failed"
        exit 1
fi
```

将 scripts/pack_img.sh 里面两处有上述命令的地方，使用“#”号注释掉即可。

说明

***merge_full_img* 只是用来做烧录器固件，不影响全志固件的生成。**

7 打包流程总结

(1) 最终打包生成固件的工具是 dragon。

(2)dragon 工具需要 2 个配置文件 image.cfg，sys_partition.fex。

(3)dragon 工具就是根据 image.cfg 和 sys_partition.fex 描述进行固件文件的打包。

(4) 整个打包流程实质上就是在处理 image.cfg 和 sys_partition.fex 里描述的文件。

(5) 整个打包流程可以简单理解为下面 3 个步骤：

- 生成或拷贝 image.cfg 和 sys_partition.fex 描述的文件。
- 对描述的文件进行一些中间处理，例如更新一些配置到文件里面等。
- 用 dragon 工具生成最终固件。

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明


、 

、 
（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。