Linux BSP 独立仓库
开发指南

版本号: 1.8
发布日期: 2023.12.21

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.06.29	XA0190	初始版本
1.1	2022.08.15	XA0190	根据评审意见修改，重新制定框架
1.2	2022.09.15	XA0190	根据新框架撰写内容
1.3	2022.10.30	XA0190	根据评审意见修改适用范围和平台名称
1.4	2022.11.04	XA0190	根据评审意见修改细节并添加 Q&A 章节
1.5	2022.12.26	XAA0248	根据评审意见修改，兼容 Tina5.0
1.6	2023.3.1	XAA0193	1. 更新了全文表示目录含义的 longan 为 sdk，以兼容适配不同的平台 2. 修改了全文的一些标题格式问题 3. 修改文档名字为开发指南
1.7	2023.6.20	XAA0193	修改了一些标点符号类格式问题
1.8	2023.12.21	XAA0190	修改全文格式问题

目　录

1 概述

1.1 编写目的

本文档作为 Allwinner Linux BSP 独立仓库使用指南，旨在帮助软件开发工程师、技术支持工程师快速上手，熟悉 Linux BSP 独立仓库开发使用流程。本文档以《Longan_linux_ 开发指南》为基础进行撰写，仅对其中差异化进行说明（Tina5.0 下 BSP 独立仓库部署无差异）。

1.2 适用范围

使用 BSP 独立仓库的所有平台。

表 1-1: 当前已适配平台

系统	版本
Tina	Tina-5.0
Longan	无版本区分

1.3 相关人员

本开发指南适用于使用 BSP 独立仓库进行开发的所有软件工程师。

2 方案介绍

2.1 概述

Allwinner 推出 BSP 独立仓库，主要目标是降低 BSP 模块驱动代码与内核原生改代码间的耦合度，从而使 BSP 模块可以便捷地适配各种内核版本，方便产品内核升级。

BSP 独立仓库将 SoC 驱动代码、板级驱动代码、defconfig、SoC 级 dts、ramfs 等文件独立出来，统一放在一个名为bsp的独立 git 仓库下进行管理。

BSP 仓库独立方案从 linux-5.10 开始应用。从整体来看，新方案带来的主要变化有：

- kernel 仓库：kernel 仓库尽量与原生内核版本代码保持一致，不再包含我司开发的驱动代码 (.c/.h/Kconfig/Makefile/dtsi/defconfig)；
- bsp 仓库：在 longan 下新增一个名为 “bsp” 的仓库，用于存放驱动代码及相关文件 (.c/.h/Kconfig/Makefile/dtsi等)；
- device 仓库：device 仓库下新增 defconfig 文件（板级和 SoC 级）。

2.2 目录结构

使用 bsp 独立仓库方案的 longan 开发包总目录结构如下:

```
sdk/
├── brandy
├── bsp    # bsp目录,用于存放allwinner BSP模块代码及相关文件
├── build
├── buildroot
├── build.sh -> build/top_build.sh
├── device  # 新增对板级和SoC级defconfig文件
├── kernel  # 仅存放内核原生代码
├── openwrt
├── out
├── platform
├── prebuilt
├── tee_kit
├── test
└── tools
```

Tina5.0 开发包中 bsp 独立仓库组织方式和 lonan 开发包保持一致，这里不再赘述。

2.2.1 bsp 仓库

bsp 目录下主要存放 allwinner 对应的相关驱动代码及相关配置等，如 drivers、ramfs 及 configs 等目录。

```
|---bsp
    ├──── configs
    ├──── drivers
    ├──── include
    ├──── Kconfig
    ├──── Makefile
    ├──── modules
    ├──── platform
    └──── ramfs
```

表 2-1: bsp 主目录说明

文件	说明
configs	主要存放 SoC 级驱动相关的 dtsi 文件及 defconfig 文件
drivers	存放 AW 对应的驱动代码 (gpu、nand 等模块除外)
include	主要存放模块对外导出的.h 文件
Kconfig	顶层 Kconfig
Makefile	顶层 Makefile
modules	存放 gpu、nand 模块对应驱动
platform	存放 SoC 平台相关文件
ramfs	存放不同架构内存文件系统文件

2.2.1.1 drivers

drivers 目录：主要存放模块对应的驱动代码 (除 nand、gpu 外)，如 usb，dma 等。

```
|---bsp
|   |---drivers
|   |   |---Kconfig
|   |   |---Makefile
|   |   |---moduleX
|   |   |   |---Kconfig
|   |   |   |---Makefile
|   |   |   |---sunxi-moduleX.c
|   |   |---module2
|   |   |   |---Kconfig
|   |   |   |---Makefile
|   |   |   |---sunxi-module2.c
```

2.2.1.2 modules

modules 目录：主要存放特定模块的驱动源码，如 nand 和 gpu。

```
|---bsp
    |---modules
    |   |   |---nand
    |   |   |   |---Kconfig
    |   |   |   |---Makefile
    |   |   |   |---sunxi-moduleX.c
    |   |   |---gpu
    |   |   |   |---Kconfig
    |   |   |   |---Makefile
    |   |   |   |---sunxi-module2.c
```

2.2.1.3 include

include 目录：主要存放模块公共的.h 文件，或需要暴露给其他模块使用的.h 文件，如 clk 相关头文件。

```
|---bsp
    |---include
    |   |   |---sunxi-moduleX.h
    |   |   |---sunxi-module2.h
    |   |   |---module3
    |   |   |   |---sunxi-module3-xxx.h
    |   |   |   |---sunxi-module3-xxx.h
    |   |   |   |---sunxi-module3-xxx.h
    |   |   |---...
```

2.2.1.4 configs

configs 目录：主要存放 SoC 级配置，包含 dtsi 和最小系统 defconfig 文件。

```
|---bsp
    |---configs
    |   |   |---linux-5.10
    |   |   |   |---sun50iw10p1.dtsi
    |   |   |   |---sun50iw10p1_min_defconfig
```

2.2.1.5 ramfs

ramfs 目录：主要存放不同架构内存文件系统相关内容，如 rootfs_arm32 等。

```
|---bsp
    |---ramfs
    |   |   |---rootfs_arm32
    |   |   |---rootfs_arm64
    |   |   |---rootfs_riscv
```

如要适配新的 ramfs，则运行./build.sh menuconfig 依次选择：

```
General setup
Init RAM filesystem and RAM disk (initramfs/initrd) support
```

将新的文件系统文件xxx.cpio.gz放置在: sdk/bsp/ramfs/xxx.cpio.gz

2.2.2 device 仓库

device 仓库里新增对 defconfig 文件的管控。

2.2.2.1 defconfig 适配方法

为满足不同 SoC、不同板子等场景下的 defconfig 个性化适配需求，bsp 独立仓库方案支持在以下三个目录里存放 defconfig 文件:

- 板级配置：sdk/device/config/chips/${SOC}/configs/${BOARD}/${KERN_VER}/xx_defconfig

  举例：sdk/device/config/chips/a133/configs/b6/linux-5.10/bsp_defconfig
- SoC 级配置：sdk/device/config/chips/${SOC}/configs/default/${KERN_VER}/xx_defconfig

  举例：sdk/device/config/chips/a133/default/linux-5.10/bsp_defconfig
- 公共配置：sdk/bsp/configs/${KERN_VER}/xx_defconfig

  举例：sdk/bsp/configs/linux-5.10/sun50iw10p1_min_defconfig

说明

(1) 若三个目录存在相同名称的 defconfig 文件，则默认使用优先级为：板级配置优先级 >SoC 级配置优先级 > 公共配置优先级。
(2) 公共配置对应的 defconfig 文件命名可使用其他命名方式（非bsp_defconfig），一般不建议存放在此（特殊 defconfig 文件除外，如min_defconfig）。

defconfig 适配方法：

- 如要适配新平台，则在板级或 SoC 级相应目录下创建bsp_defconfig。
- 如要适配新内核，则在板级或 SoC 级相应目录下创建${KER_VER}目录，再将 defconfig 文件放入。

2.2.2.2 dts 适配方法

说明：本小节均以 A133 b6 linux-5.10 为示例。

1. 文件存放路径与非独立仓库使用习惯保持一致，如sdk/device/config/chips/a133/configs/b6/linux-5.10/board.dts。
2. 如要适配新 SoC，则在对应目录下 sdk/device/config/chips/${SoC}/configs/${BOARD}/${KERNEL_VER} 创建board.dts。

- 创建${SoC}目录：sdk/device/config/chips/a133。
- 创建板级目录${BOARD}：sdk/device/config/chips/a133/configs/b6/linux-5.10。

```
cd sdk/device/config/chips/${SoC}
mkdir configs && cd configs
mkdir ${BOARD} && cd ${BOARD}
mkdir ${KERNEL_VER} && cd ${KERNEL_VER}
```

- 创建board.dts文件，并进行正确适配:device/config/chips/a133/configs/b6/linux-5.10/board.dts。

```
cd sdk/device/config/chips/a133/configs/b6/linux-5.10
vim board.dts
```

3. 如要适配新内核，则在 sdk/device/config/chips/${SOC}/configs/${BOARD}/下创建${KERNEL_VER}目录，再将放入board.dts 。以 A133 b6 linux-5.10 为例：

- 创建${KERNEL_VER}目录：sdk/device/config/chips/a133/configs/b6/linux-5.10。

```
cd sdk/device/config/chips/${SOC}/configs/${BOARD}
mkdir ${KERNEL_VER}
```

- 移植board.dts文件至上述目录：sdk/device/config/chips/a133/configs/b6/linux-5.10/board.dts。

```
cp xx/board.dts sdk//device/config/chips/${SOC}/configs/${BOARD}/${KERNEL_VER}/
```

2.2.3 kernel

kernel 目录用于存放不同版本的内核原生代码。

```
sdk/kernel
└── linux-5.10
    └── bsp -> ../../bsp/
```

建议：为确保 kernel 仓库的解耦性，建议 kernel 仓库只存放 Linux 或 AOSP 内核原生代码。

注意：编译配置完成后，kernel 仓库下会生成 bsp 仓库对应的软链接，若要修改模块内容，请前往 bsp 仓库修改，不要在软链接修改，避免造成数据丢失等问题。

3 使用方法

3.1 编译打包

在 Allwinner Linux BSP 独立仓库执行编译命令：

1. 进入编译目录

```
cd sdk
```

2. 选择方案：根据需要选择配置，例如依次选择 linux,bsp,a133,b6,default

```
./build.sh config
```

3. SDK 编译

```
./build.sh
```

4. 打包

```
./build.sh pack
```

以上四个步骤即可生成 Linux 固件镜像。

其他编译命令：

1. 单独编译内核：

```
./build.sh kernel
```

2. 内核配置：

```
./build.sh menuconfig
```

3.2 其他打包方式

- 普通固件打包：

```
sdk$ ./build.sh pack        # 生成uart0打印的固件包
sdk$ ./build.sh pack_debug    # 生成card0打印的固件包
sdk$ ./build.sh pack_nor      # 生成spinor&&uart0打印的固件包
sdk$ ./build.sh pack_nor      # 生成spinor&&uart0打印的固件包
```

- 安全固件打包：

```
sdk$ ./build.sh pack_secure         # 生成uart0打印的安全固件包
sdk$ ./build.sh pack_secure_debug     # 生成card0打印的安全固件包
```

如果本地没有固件签名密钥，需要按如下方法手动创建：

```
sdk/build$ ./createkeys      # 注意，只需运行一次，用于创建密钥
```

注：生成的固件包位于longan/out目录下。

3.3 内核编译注意事项

（1）内核所有的编译产物 (vmlinux、config、o 文件等) 均放在longan/out/kernel/目录下，linux-5.10目录下不会再有任何编译产物。

其中：

- sdk/out/kernel/build/ ：.config 文件、vmlinux 文件、o 文件等。
- sdk/out/kernel/staging/：bImage、dtb、ko 等。

（2）必须使用./build.sh menuconfig、./build.sh saveconfig，而不能在内核目录下make ARCH=arm64 menuconfig。

（3）必须使用./build.sh clean、./build.sh distclean，而不能在内核目录下make clean、 make distclean。

（4）如何加载指定的配置文件、更改配置文件、保存配置文件。

- 请使用 ./build.sh loadconfig [xxx_defconfig] 来加载指定的（非默认的）defconfig 文件。
- 然后使用 ./build.sh menuconfig 来更改配置项。
- 最后使用 ./build.sh saveconfig [xxx_defconfig] 来保存更改到默认配置路径如：device/config/chips/${SOC}/configs/default/${KERN_VER}/xx_defconfig 。
- 注意：建议不要直接手动修改 defconfig 文件，建议不要将 .config 文件直接复制为 device/config/chips/${SOC}/configs/default/${KERN_VER}/xx_defconfig 。请使用上述步骤进行 defconfig 配置。

3.4 常用快捷命令

表 3-1: 快捷命令列表

快捷命令	含义
cbsp	进入 Allwinner 驱动代码目录
ckernel	进入内核目录，如 linux-5.10
cconfigs	进入芯片方案配置目录

4 开发说明

4.1 如何添加新驱动

以下均以 moduleX 作为示例，演示在 bsp 仓库里添加一个新驱动的方法。步骤如下：

- 在bsp/drivers目录下，新建 moduleX 目录：

```
bsp/drivers/.
    ├── moduleX
```

- bsp/drivers/Makefile文件内添加模块 moduleX 的编译规则：

```
obj-y += moduleX/
```

- bsp/drivers/Kconfig文件内模块 moduleX Kconfig 规则：

```
source "$(BSP_TOP)drivers/moduleX/Kconfig"
```

4.1.1 添加驱动文件 (.c .h)

在 moduleX 目录下添加相应的.c、.h、Makefile 和 Kconfig 文件：

```
sdk/bsp/modules/moduleX.
.
├── Kconfig
├── Makefile
├── moduleX.c
└── moduleX.h
```

4.1.2 修改设备树

1. 在 bsp/configs 目录添加 moduleX 对应 dtsi 配置。

- 打开 dtsi 文件, 位置为sdk/bsp/configs/${KERNEL_VER}/xx.dtsi，以 linux-5.10 A133 b6 为例：

```
sdk/bsp/configs.
├── linux-5.10
│   ├── sun50iw10p1.dtsi
```

- 在 dtsi 文件内添加 moduleX 模块配置：

```
moduleX: moduleX@xxxx {
    compatible = "allwinner,moduleX";
    reg = <0x0 xxxx 0x0 xxx>;
    interrupts = xxx;
    ...
};
```

2. 在 devices 目录添加 moduleX 对应板级配置：

- 打开对应 dts 文件，位置为sdk/device/config/chips/${SOC}/configs/${BOARD}/${KERNEL_VER}/board.dts; 以 linux-5.10 A133 b6 为例, 位置如下：

```
cp xx/board.dts sdk//device/config/chips/${SOC}/configs/${BOARD}/${KERNEL_VER}/
```

- 在 dts 文件内添加 moduleX 相关配置：

```
&moduleX {
  xxx;
  ...
};
```

4.1.3 修改 defconfig

上述配置完成后，可修改 defconfig 文件对模块进行编译等操作，主要步骤如下：

1. cd sdk/ && ./build.sh menuconfig 选择 moduleX 模块配置。
2. cd sdk/ && ./build.sh saveconfig 将对 moduleX 配置的更改保存至 defconfig 文件。
3. 在sdk/device/config/chips/${SOC}/configs/目录下查看并确认 defconfig 文件已包含 moduleX 对应更改

4.2 如何升级内核

若想给当前产品内核升级 (如 linux-5.4 升级至 linux-5.10), 可采用如下步骤：

4.2.1 适配内核

- 在 kernel 目录下新建 linux-5.10 目录，存放内核原生代码。
- 更改内核 linux-5.10 目录下Kconfig和Makefile，确保 bsp 仓库与内核仓库正确关联起来，更改内容如下：

```
diff --git a/Kconfig b/Kconfig
index 57a142d8d8b4..197dfd3558f3 100644
--- a/Kconfig
+++ b/Kconfig
@@ -7,6 +7,8 @@ mainmenu "Linux/$(ARCH) $(KERNELVERSION) Kernel Configuration"

 source "scripts/Kconfig.include"

+source "bsp/Kconfig"
+
 source "init/Kconfig"

 source "kernel/Kconfig.freezer"
diff --git a/Makefile b/Makefile
index 82effe3d7164..85a7bd23101b 100644
--- a/Makefile
+++ b/Makefile
@@ -574,6 +574,7 @@ LINUXINCLUDE    := \
                -I$(objtree)/arch/$(SRCARCH)/include/generated \
                $(if $(building_out_of_srctree),-I$(srctree)/include) \
                -I$(objtree)/include \
+               -I$(srctree)/bsp/include \
                $(USERINCLUDE)

 KBUILD_AFLAGS   := -D__ASSEMBLY__ -fno-PIE
@@ -763,6 +764,7 @@ ifeq ($(KBUILD_EXTMOD),)
 # Objects we will link into vmlinux / subdirs we need to visit
 core-y          :=
 drivers-y       :=
+drivers-y       += bsp/
 libs-y          := lib/
 endif # KBUILD_EXTMOD
```

4.2.2 适配 bsp 仓库

1. 适配 dtsi：

- bsp/configs目录下新建 linux-5.10 目录。
- 将产品平台的 dtsi 文件移植至上述目录下。

2. 适配驱动：

- 如遇到新内核的 API 发生变化等情况，则需要自行修改驱动代码进行适配。

4.2.3 适配 device 仓库

device 仓库适配主要包含 dts 适配和 defconfig 适配两个步骤，具体操作方法如下。

4.2.3.1 适配 dts

- device 目录下新建 linux-5.10 目录：device/config/chips/${SOC}/configs/${BOARD}/linux-5.10。
- 将产品平台的board.dts文件移至上述目录下。

4.2.3.2 适配 defconfig

- 在板级或 SoC 级目录下首先创建${KER_VER}目录，如mkdir linux-5.10, 再将 defconfig 文件放入创建的${KER_VER}目录，板级目录：sdk/device/config/chips/${SOC}/configs/${BOARD}/${KERN_VER}/xx_defconfig，SoC 级目录：sdk/device/config/chips/${SOC}/configs/default/${KERN_VER}/xx_defconfig。

5 Q&A

5.1 不同内核版本适配不同驱动文件

若不同内核之间，驱动改动太大，需要用不同的 C 文件来区分，可以在 Makefile 中借助内核版本号来条件编译。

例：Linux-5.10 使用 sunxi-module-v200.c，其他内核使用 sunxi-module-v100.c，则模块对应 Makefile 可修改为：

```
$(BSP_TOP)drivers/dma/Makefile

ifeq ($(VERSION).$(PATCHLEVEL),5.10)
obj-$(CONFIG_AW_DMA) += sunxi-dma-v200.o
else
obj-$(CONFIG_AW_DMA) += sunxi-dma-v100.o
endif
```

5.2 不同内核版本适配不同驱动目录

若不同内核之间，驱动改动太大，需要用不同的目录来区分，则除了如上所述修改 Makefile 之外，还需要修改 Kconfig。

例：Linux-5.10.43 内核使用 drivers/module1-v200/ ，其他内核使用 drivers/module1-v100/ ，则需同时修改 Makefile 和 Kconfig：

Makefile 修改方法与前一章节小节一致，Kconfig 里用于判断内核版本号的函数如下，详见 bsp/platform/Kconfig.common ：

```
ker-ver-lt  // 判断当前内核版本 <  指定的版本
ker-ver-le  // 判断当前内核版本 <= 指定的版本
ker-ver-eq  // 判断当前内核版本 == 指定的版本
ker-ver-gt  // 判断当前内核版本 > 指定的版本
ker-ver-ge  // 判断当前内核版本 >= 指定的版本
```

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


、 
、全志科技、  （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。