

珠海六点智能科技有限公司lindeni



珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

Longan Linux 开发指南

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

版本号: 1.0
发布日期: 2020.06.24

珠海六点智能科技有限公司lindeni

珠海六点智能科技有限公司lindeni

珠海六点智能

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.06.24	AW1637	version 1.0



目 录

1 前言	1
1.1 文档简介	1
1.2 文档简介	1
1.3 使用范围	1
2 环境介绍	2
2.1 系统介绍	2
2.1.1 longan 系统介绍	2
2.2 软件环境	2
2.3 目录结构	2
2.4 longan 下载方式	3
2.4.1 公司内部下载方式	3
2.4.2 公司外部下载方式	4
2.5 longan 编译及烧录	4
2.5.1 longan 的编译	4
2.5.2 SDK 固件的烧录	6
2.5.3 SDK 的调试	7
3 longan 配置说明	8
3.1 编译配置规则	8
3.2 修改 longan 平台配置	8
3.2.1 修改编译规则 BoardConfig.mk	8
3.2.2 相关文件覆盖规则	9
3.2.2.1 BoardConfig.mk/env.cfg/sys_partition.fex 覆盖规则	9
3.2.3 kernel 更换交叉编译工具链	12
3.2.4 修改 linux 的根文件系统	13
3.2.5 修改 kernel 的 defconfig	13
3.2.6 修改 linux 的设备树文件	14
3.2.7 支持编译 buildroot	15
3.2.8 向文件系统增加文件	16
3.2.8.1 方式一	16
3.2.8.2 方式二	16
3.2.8.3 方式三	16
4 linux 使用指南	17
4.1 U-boot 调试指南	17
4.2 Kernel 调试指南	17
4.2.1 打印/设置寄存器	17
4.2.2 挂载 debugfs	17
4.2.3 电脑与主机端传输文件	18

1 前言

1.1 文档简介

本文档介绍全志科技的 longan 系统 (Linux SDK) 的了解开发环境、目录结构、编译和打包，主要目的用于指导用户如何定制和使用 Linux SDK。

1.2 文档简介

基于 longan 系统的 linux SDK 开发者

1.3 使用范围

表 1-1: 适用产品列表

内核版本
Linux-4.4
Linux-4.9
Linux-5.4

2 环境介绍

2.1 系统介绍

2.1.1 longan 系统介绍

longan 是 lichee 和 kunos 合并后的名称，是全志平台统一使用的 linux 开发平台。它集成了 BSP，构建系统，独立 IP 和测试，既可作为 BSP 开发和 IP 验证平台，也可以作为量产的嵌入式 linux 系统。

longan 的功能包括以下四部分：

1. BSP 开发，包括 bootloader, uboot 和 kernel
2. Linux SDK 开发，包括量产的嵌入式 linux 系统
3. IP 的验证和发布平台，包括 gpu, cedarx, gstreamer, drm/weston, security 以及其他的私有软件包。IP 随 longan 的发布而发布，减少使用邮件发布；并且给出 IP 的使用方法和系统集成的 demo 程序，方便第三方快速使用
4. 测试，包括板级测试和系统测试，如 SATA 和 drangonboard。

2.2 软件环境

Ubuntu 12.04 及以上版本

2.3 目录结构

```
longan/
├── brandy          //uboot和boot0代码
│   ├── brandy-1.0
│   └── brandy-2.0
├── build          //编译打包脚本
│   ├── bin       //制作文件系统工具
│   ├── createkeys //创建安全方案密钥工具
│   ├── envsetup.sh //配置环境变量
│   ├── Makefile
│   ├── mkcmd.sh  //主要编译脚本
│   └── mkcommon.sh //编译脚本
```

```

├── mksetup.sh
├── pack          //打包脚本
├── toolchain     //编译工具链
├── top_build.sh
├── build.sh -> build/top_build.sh //顶级编译脚本, 链接到build/top_build.sh
├── device //方案配置文档
│   ├── config
│   │   ├── chips                //板级配置
│   │   │   ├── t509            //t509配置
│   │   │   │   ├── bin        //uboot和烧写程序
│   │   │   │   ├── boot-resource //启动资源文件, 如bootlogo
│   │   │   │   ├── config     //方案板配置
│   │   │   │   ├── default    //默认配置和通用配置
│   │   │   │   ├── demo      //t509-demo方案板配置
│   │   │   │   ├── longan     //linux sdk配置
│   │   │   │   └── bin        //linux sdk 专用boot启动程序
│   ├── product -> ./config/chips/t509
│   └── target //方案配置目录
├── kernel //各个版本的内核文件
│   ├── linux-3.10
│   ├── linux-3.4
│   ├── linux-4.4
│   ├── linux-4.9
│   └── linux-5.4
├── out
│   ├── gcc-linaro-* //kernel交叉编译工具链
│   ├── pack_out     //打包输出目录
│   ├── t509         //t509输出目录
│   │   ├── demo    //t509-demo输出目录
│   │   └── bsp     //bsp输出目录
│   └── t509_linux_demo_uart0.img //t509生成固件
├── test
│   ├── dragonboard //dragonboard测试系统
│   └── SATA        //sata测试系统
├── tools //pc工具
│   ├── build
│   ├── codecheck //代码检查工具
│   ├── pack
│   └── tools_win //windows软件工具

```

2.4 longan 下载方式

2.4.1 公司内部下载方式

```

mkdir longan
cd longan
repo init -u http://gerrit.allwinnertech.com:8081/manifest.git -b lichee -m sunxi_dev.xml
repo sync
repo start sunxi-dev --all //如果用的是老平台, linux-5.4
repo start sunxi-product-q --all //A100, H616, T507

```

2.4.2 公司外部下载方式

对于公司外部人员，会统一发布某一平台的 SDK 代码包。

2.5 longan 编译及烧录

2.5.1 longan 的编译

longan 编译的步骤包括配置，编译，打包三个步骤。这三个步骤都在 longan 的根目录下进行，具体的步骤如下所示：

- 步骤 1, 进行 sdk 环境配置，详细看下一节

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

下面就以编译 T509demo 板子 longan 系统为例展示一个 longan 配置过程，跳转到 longan 根目录：

```
longan$ ./build.sh config

Welcome to mkscript setup progress
All available platform:
  0. android
  1. linux
Choice [linux]:1
All available linux_dev:
  0. bsp
  1. dragonboard
  2. sata
  3. longan
  4. tinyos
Choice [bsp]:0
All available kern_ver:
  0. linux-3.10
  1. linux-3.4
  2. linux-4.4
  3. linux-4.9
  4. linux-5.4
Choice [linux-5.4]:
All available ic:
  0. a100
  1. a133
  2. a50
  3. a63
  4. a64
  5. f113
  6. f133
  7. h3
  8. h6
  9. h616
```

```
10. r328-s2
11. r328-s3
12. r328
13. r329
14. r528
15. t3
16. t507
17. t7
18. t8
19. tv303
20. v316
21. v5
22. v533
23. v536
24. v831
25. v833
```

```
Choice [a100]:
```

```
All available board:
```

```
0. b1
1. b3
2. b4
3. evb
4. fpga
5. ft
6. perf1
7. perf2
8. qa
9. ver
```

```
Choice [perf1]:
```

```
All available flash:
```

```
0. default
1. nor
```

```
Choice [default]:
```

- 步骤 2，编译整个 SDK

```
./build.sh
```

- 步骤 3，打包固件

```
./build.sh pack //打包固件
```

如果 pack 成功，会在 longan 的 out 目录下生一个 img 镜像，该镜像就是最终生成的固件，如 T509 demo 板编译 linux 最终生成的镜像为 t509_linux_demo_uart0.img，将该固件通过 PhoenixSuit 烧录到开发板上，上电，系统就可以起来。具体烧录方式可以参考下一节。

如果对打包的固件有所要求，如需要卡打印，以及生成安全固件等等，可以参照以下指令

```
./build.sh pack_debug //打包卡打印固件，android方案无需此打包
./build.sh pack_debug_secure //打包安全卡打印固件，android方案无需此打包
./build.sh pack_secure //打包安全固件，android方案无需此打包
```


注意如果需要打包成安全固件，需要先生成安全密钥，再执行打包操作。跳转到 build 目录，执行 createkeys 文件，然后选择相应的平台：

```
longan/build$ ./createkeys
All valid Sunxi ic:
 0. a100
 1. a133
 2. a50
 3. a63
 4. a64
 5. f113
 6. f133
 7. h3
 8. h6
 9. h616
10. r328
11. r328-s2
12. r328-s3
13. r329
14. r528
15. t3
16. t507
17. t7
18. t8
19. tv303
20. v316
21. v5
22. v533
23. v536
24. v831
25. v833
Please select a ic[a100]:
```

2.5.2 SDK 固件的烧录

全志平台统一采用 PhoenixSuite 软件进行烧录。界面如下



图 2-1: PhoenixSuit

点击浏览按钮，选中刚才 pack 生成的固件。按住开发板的 FEL/UBOOT 键，然后上电，既可以进入烧录界面，在弹出界面中默认选择是，即可进入烧录，当显示固件烧写成功的时候，表明该固件已经烧录到板子里面了。

如果板子上已有 SDK 环境的话，也可以在控制台执行 `reboot efex` 进入烧录或者在上电的时候按住键盘 2 进行烧录。

2.5.3 SDK 的调试

固件烧录到板子后，通过串口线连接到电脑的串口，打开串口软件，推荐使用 SecureCRT，给板子上电，串口会打印系统启动的 log 信息。如果固件可用，可以正常的进入控制台，调试 linux 系统。

3 longan 配置说明

3.1 编译配置规则

当./build.sh config 完成之后，编译系统会生成一个方案的编译规则，其保存在在 longan 根目录中的.buildconfig 文件里面。下面是一个规则の説明。

```
LICHEE_PLATFORM //编译平台
LICHEE_LINUX_DEV //编译的方案
LICHEE_IC // 编译的IC
LICHEE_BOARD //编译的板级配置
LICHEE_FLASH //编译的flash配置
LICHEE_CHIP //编译的芯片代号名称
LICHEE_KERN_VER //编译的内核版本
LICHEE_KERN_DEFCONF //编译内核的默认配置
LICHEE_BUILDING_SYSTEM //编译的构造系统
LICHEE_BR_VER //buildroot的版本
LICHEE_BR_DEFCONF //buildroot的默认配置
LICHEE_BR_RAMFS_CONF //buildroot的ramfs默认配置
LICHEE_BRANDY_VER //uboot的版本
LICHEE_BRANDY_DEFCONF //brandy的默认配置
LICHEE_CROSS_COMPILER //编译使用的交叉编译链
LICHEE_CHIP_CONFIG_DIR //编译系统的IC配置目录
LICHEE_OUT_DIR //编译的输出目录
```

上面只列出了部分重要的配置，详细的配置可以查看该文件。

而某个方案的大部分编译规则都是由 BoardConfig.mk，其存放在每个方案的板级配置目录中，同一个板型的不同系统拥有不用的 BoardConfig.mk 配置，例如 r328 bsp 验证系统和 Linux SDK 系统拥有各自的 BoardConfig.mk 配置。

3.2 修改 longan 平台配置

3.2.1 修改编译规则 BoardConfig.mk

修改 BoardConfig.mk 可以达到修改编译规则的目的，可以用来选择编译工具链，内核所用的默认配置，使用的根文件系统等等，对于上面.buildconfig 文件里面的配置选项，都可以在 Board-Confli.mk 里面进行配置，下面是一个示例

```
LICHEE_ARCH:=arm //芯片架构 arm or arm64
LICHEE_PRODUCT:=drum //芯片产品代号
LICHEE_BRANDY_VER:=2.0 //brandy版本
```

```
LICHEE_KERN_VER:=4.9 //kernel版本
LICHEE_KERN_DEFCONF:=sun50iw10p1_defconfig //kernel defconfig名称
LICHEE_COMPILER_TAR:=gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz //kernel编译
工具链
```

3.2.2 相关文件覆盖规则

3.2.2.1 BoardConfig.mk/env.cfg/sys_partition.fex 覆盖规则

在 device/config/chips/\$(LICHEE_IC)/configs 目录下，目录结构如下：

```
.
├── default
│   ├── BoardConfig.mk
│   ├── BoardConfig_nor.mk
│   ├── boot_package.cfg
│   ├── boot_package_nor.cfg
│   ├── diskfs.fex
│   ├── dragon_toc_android.cfg
│   ├── dragon_toc.cfg
│   ├── env_burn.cfg
│   ├── env.cfg
│   ├── env_dragon.cfg
│   ├── env_ubifs.cfg
│   ├── image_android.cfg
│   ├── image.cfg
│   ├── image_dragonboard.cfg
│   ├── image_linux.cfg
│   ├── image_longan.cfg
│   ├── parameter.fex
│   ├── sys_partition.fex
│   ├── sys_partition_nor.fex
│   ├── sys_partition_ubi.fex
│   ├── sysrecovery.fex
│   ├── verity_block.fex
│   └── version_base.mk
├── demo
│   ├── android
│   │   ├── BoardConfig.mk
│   │   └── sys_partition.fex
│   ├── board.dts
│   ├── dragonboard
│   │   ├── BoardConfig.mk
│   │   ├── sys_partition_dragonboard.fex
│   │   └── test_config.fex
│   └── sys_config.fex
├── demo2.0
│   ├── android
│   │   ├── BoardConfig.mk
│   │   └── sys_partition.fex
│   ├── board.dts
│   ├── dragonboard
│   │   ├── BoardConfig.mk
│   │   ├── sys_partition_dragonboard.fex
│   │   └── test_config.fex
```

```
├── longan
│   ├── BoardConfig.mk
│   ├── sw-description
│   ├── sw-subimgs.cfg
│   └── sys_partition.fex
├── sys_config.fex
├── sys_partition_nor.fex
└── systemd
    ├── BoardConfig.mk
    └── sys_partition.fex
demo2.0_car
├── android
│   ├── BoardConfig.mk
│   └── sys_partition.fex
├── board.dts
├── dragonboard
│   ├── BoardConfig.mk
│   ├── sys_partition_dragonboard.fex
│   └── test_config.fex
├── longan
│   ├── BoardConfig.mk
│   └── sys_partition.fex
└── sys_config.fex
demo2.0_td100
├── android
│   ├── BoardConfig.mk
│   └── sys_partition.fex
├── board.dts
├── dragonboard
│   ├── BoardConfig.mk
│   ├── sys_partition_dragonboard.fex
│   └── test_config.fex
└── sys_config.fex
demo_rawnand_ubi
├── android
│   └── sys_partition.fex
├── BoardConfig.mk
├── board.dts
├── bsp
│   ├── env.cfg
│   └── sys_partition.fex
├── longan
│   ├── env.cfg
│   └── sys_partition.fex
└── sys_config.fex
evb
├── android
│   ├── BoardConfig.mk
│   └── sys_partition.fex
├── board.dts
├── dragonboard
│   ├── BoardConfig.mk
│   ├── sys_partition_dragonboard.fex
│   └── test_config.fex
├── dtbo
│   ├── board1.dts
│   ├── dtboimg.cfg
│   ├── edp_1080p.dts
│   ├── gmac.dts
│   └── hdmi_switch_vbyone.dts
```

```

├── lvsd_1080p.dts
├── rgb_to_vga.dts
├── longan
│   ├── BoardConfig.mk
│   └── sys_partition.fex
├── sys_config.fex
├── fpga
│   └── sys_config.fex
├── ft
│   ├── android
│   │   └── sys_partition.fex
│   ├── board.dts
│   └── sys_config.fex
├── qa
│   ├── android
│   │   ├── BoardConfig.mk
│   │   └── sys_partition.fex
│   ├── board.dts
│   ├── dragonboard
│   │   ├── BoardConfig.mk
│   │   ├── sys_partition_dragonboard.fex
│   │   └── test_config.fex
│   └── sys_config.fex
└── ver_v1_0
    ├── android
    │   └── sys_partition.fex
    ├── board.dts
    └── sys_config.fex

```

在执行./build.sh config 阶段，如果是 Linux 系统需要配置环境变量 LICHEE_LINUX_DEV(bsp/longan/sata/dragonboard)、LICHEE_BOARD(demo2.0 等)、LICHEE_FLASH(default 和 nor)，如果是 Android，则 LICHEE_LINUX_DEV 一般为空值，LICHEE_FLASH 一般是 default 或空值

完整 BoardConfig.mk、env.cfg、sys_partition*.fex 分布路径如下：

```

├── default
│   ├── BoardConfig_android.mk //android方案使用
│   ├── BoardConfig.mk //linux方案使用
│   └── BoardConfig_nor.mk //nor方案使用，会覆盖同级别目录的BoardConfig.mk
├── perf1
│   ├── env.cfg //非nor方案使用
│   ├── env_nor.cfg //nor方案使用，会覆盖同级别目录的env.cfg
│   ├── sys_partition.fex //非nor方案使用
│   └── sys_partition_nor.fex //nor方案使用，会覆盖同级别目录的sys_partition.fex
├── perf1
│   ├── android
│   │   ├── BoardConfig.mk
│   │   ├── env.cfg
│   │   └── sys_partition.fex
│   ├── bsp
│   │   ├── BoardConfig.mk
│   │   ├── BoardConfig_nor.mk
│   │   ├── env.cfg
│   │   ├── env_nor.cfg
│   │   ├── sys_partition.fex
│   │   └── sys_partition_nor.fex

```

```

├── dragonboard
│   ├── BoardConfig.mk
│   ├── BoardConfig_nor.mk
│   ├── env.cfg
│   ├── env_nor.cfg
│   ├── sys_partition.fex
│   └── sys_partition_nor.fex
├── longan
│   ├── BoardConfig.mk
│   ├── BoardConfig_nor.mk
│   ├── env.cfg
│   ├── env_nor.cfg
│   ├── sys_partition.fex
│   └── sys_partition_nor.fex
└── sata
    ├── BoardConfig.mk
    ├── BoardConfig_nor.mk
    ├── env.cfg
    ├── env_nor.cfg
    ├── sys_partition.fex
    └── sys_partition_nor.fex

```

覆盖规格：

1. bsp、longan、android 等系统方案，属于并行目录，不会相互覆盖
2. bsp、longan 等系统方案的 BoardConfig.mk、env.cfg、sys_partition.fex 会覆盖 default 目录的 BoardConfig.mk、env.cfg、sys_partition.fex
3. 同级别目录的，BoardConfig_nor.mk、env_nor.cfg、sys_partition_nor.fex 会覆盖 BoardConfig.mk、env.cfg、sys_partition.fex
4. 当 bsp、longan 等系统方案没有配置 BoardConfig.mk、env.cfg、sys_partition.fex，会使用 default 目录的 BoardConfig.mk、env.cfg、sys_partition.fex
5. 注意，BoardConfig.mk 是包含关系，由系统方案的 BoardConfig.mk 往上层包含，如果系统方案和上层有相同属性，才会覆盖，否则视为增加
6. 为了兼容以前的 env.cfg、sys_partition.fex，可能有些板型的 env.cfg 和 sys_partition.fex 在板型根目录下存在，如 (perf1/)，但目前构建系统不推荐这样使用，不予支持

3.2.3 kernel 更换交叉编译工具链

- 步骤 1：拷贝交叉编译工具链到 build/toolchain 目录中。
- 步骤 2：修改编译配置，修改 device/config 目录下 BoardConfig.mk 将 LICHEE_COMPILER_TAR 修改为需要更换的工具链名字。

也可以直接修改 build 目录下的 mkcmd.sh，打开该文件，找到下面的代码：

```

cross_compiler=(
'linux-3.4 arm gcc-linaro.tar.bz2 target_arm.tar.bz2' #kernel arm kernel-gcc rootfs
gcc
'linux-3.4 arm64 gcc-linaro-aarch64.tar.xz target_arm64.tar.bz2'
'linux-3.10 arm gcc-linaro-arm.tar.xz target_arm.tar.bz2'

```



```
'linux-3.10 arm64 gcc-linaro-aarch64.tar.xz target_arm64.tar.bz2'
'linux-4.4 arm gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz target-arm-
linaro-5.3.tar.bz2'
'linux-4.4 arm64 gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz target-
arm64-linaro-5.3.tar.bz2'
'linux-4.9 arm gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz target-arm-
linaro-5.3.tar.bz2'
'linux-4.9 arm64 gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz target-
arm64-linaro-5.3.tar.bz2'
'linux-5.4 arm gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz target-arm-
linaro-5.3.tar.bz2'
'linux-5.4 arm64 gcc-linaro-5.3.1-2016.05-x86_64_aarch64-linux-gnu.tar.xz target-arm64-
linaro-5.3.tar.bz2'
'linux-5.4 riscv riscv64-linux-x86_64-20200528.tar.xz target_riscv.tar.bz2'
)
```

该结构体里面的第三列用来指定编译工具了，可以直接修改里面交叉编译链的名称。当然，该优先级要比修改 BoardConfig.mk 的低。

3.2.4 修改 linux 的根文件系统

- 步骤 1：拷贝相应的 rootfs 文件到 longan 目录下的/device/config/rootfs_tar 目录下，如果已有相应的 rootfs，可以不用拷贝
- 步骤 2：修改编译配置，修改 device/config 目录下 BoardConfig.mk 将 LICHEE_ROOTFS 修改为需要更换的根文件系统名字。

也可以直接修改 build 目录下的 mkcmd.sh，打开该文件，找到 cross_compiler 这个结构体，第四列就是 ROOTFS 文件的名字。当然，该优先级要比修改 BoardConfig.mk 的低。

这里的 ROOTFS 推荐使用 buildroot 或者 busybox 来编译。



如果需要将该文件系统编译到固件里面，需要到 longan 的 out/{IC}/{board}/{project} 下面把 rootf_def 删掉。

3.2.5 修改 kernel 的 defconfig

linux 目录位于 longan 的 kernel 目录下，如 kernel/linux-4.9，而 linux 的默认配置文件则位于内核目录下的 arch/arm64(arm)/configs 下，的默认配置为 sun50iw9p1smp_defconfig，如果需要选择其他的 defconfig 文件，可以到 BoardConfig.mk 下面把下面这个参数定义上。

```
LICHEE_KERN_DEFCONF:=sun50iw10p1_defconfig //kernel defconfig名称
```

如果需要修改相应的 defconfig 配置，再选中完相应的 deconfig 之后进入到内核目录，执行 make ARCH=arm64(32 位系统用 arm) menuconfig 进入配置界面。

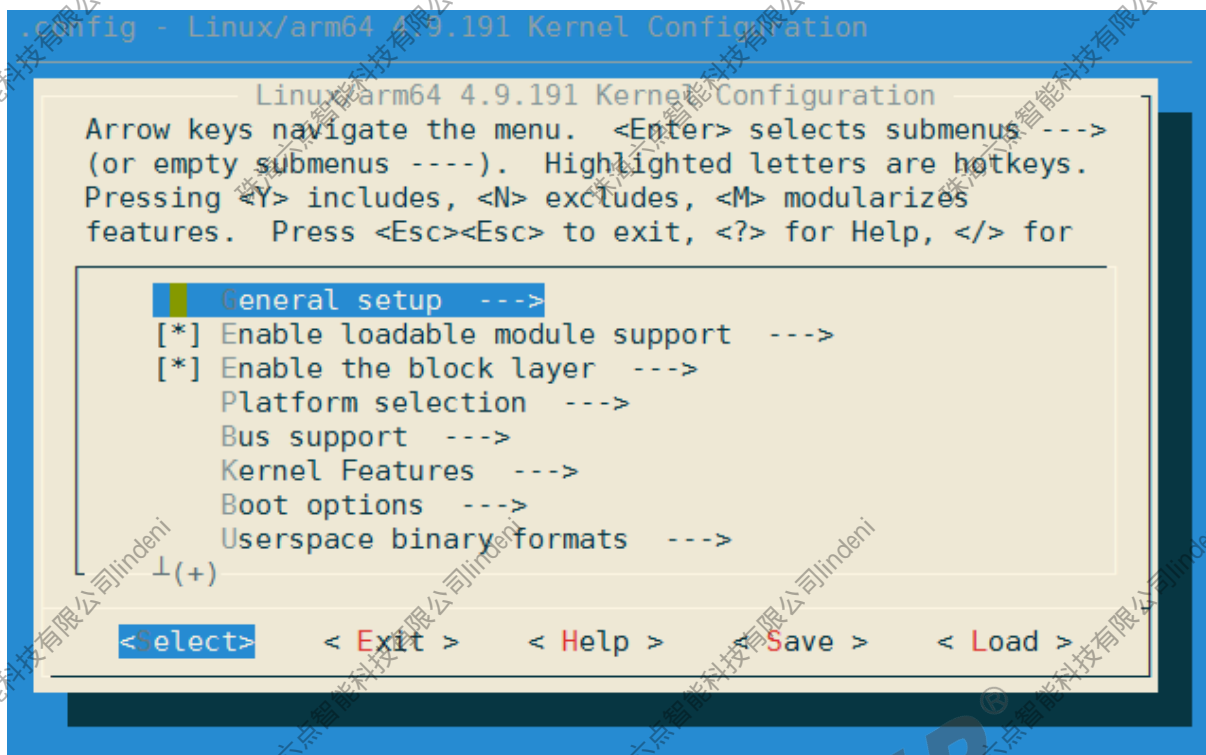


图 3-1: menuconfig

选中相应配置后选择 save 保存，然后选择 Exit 退出，相应的配置会保存在内核目录的.config 文件里面。

如果需要保存该.config 配置到相应的 defconfig 里面，可以到 longan 根目录下执行以下指令

```
./build.sh saveconfig
```

3.2.6 修改 linux 的设备树文件

linux 的设备树文件存放的位置有三个地方，分别是：

1. 64 位 arm 位于内核目录下的 arch/arm64/boot/dts/sunxi 目录下，32 位 arm 位于内核目录下的 arch/arm/boot/dts 下面，设备树文件以 dtsi 命名
2. 位于 device/config/chips/{IC}/configs/{board}/{kernel} 目录下的 board.dts
3. 位于 device/config/chips/{IC}/configs/{board}/目录下的 sys_config.fex 在 linux-5.4 中已被摒弃

📖 说明

sys_config.fex 在 linux-5.4 中已被摒弃

这几个的优先级为 3 最高，2 次之，1 最低，优先级高的会替换优先级低的配置。其中 1,2 的配置方式与标准 linux 的用法一样。而 sys_config.fex 由于采用了全志平台的编译工具，用法跟标准的 linux 用法不同，下面介绍 sys_config 的用法。

ys config.fex, 用于配置系统参数 (作用与 linux 内核的设备树), 可以按照具体的平台来配置, 具体的用法如下:

```
[name]
param = description
```

下面是一个示例:

```
[product]
version = "100"
machine = "perf1"
```

其定义了一个 product 节点, 里面有 version 和 machine 参数, 其会覆盖设备树中相关的 product 节点, 如若没有, 则会添加。

3.2.7 支持编译 buildroot

Longan BSP 方案, 一般由于编译 Buildroot 时间比较长, 因此没有支持编译 buildroot, 而是选择已经做好的文件系统包来使用, 但这样不方便进行文件系统定制化, 为了方便文件系统定制, 需要支持 Buildroot 构建文件系统, BSP 方案下, buildroot 的配置在 BoardConfig*.mk 如下所示:

```
LICHEE_BUILDING_SYSTEM
LICHEE_BR_VER
LICHEE_BR_DEFCONF
LICHEE_BR_RAMFS_CONF
```

在 BSP 方案中, BoardConfig.mk 一般不会配置上面的几个环境变量, 表示不使用 buildroot 来构建文件系统, 当需要使用 buildroot 来构建文件系统时, 需要在 BoardConfig.mk 文件增加这些环境变量的配置, 介绍如下:

```
LICHEE_BUILDING_SYSTEM:=buildroot //构建系统, 目前只支持buildroot
LICHEE_BR_VER:=201902 //buildroot版本, 目前支持201902和
201611
LICHEE_BR_DEFCONF:=sun50iw9p1_longan_defconfig //buildroot构建根文件系统时, 使用的配置
文件
LICHEE_BR_RAMFS_CONF:=sun50iw9p1_longan_defconfig //buildroot编译ramfs时使用的配置文件,
没有编译ramfs, 可以不配置
```

在 buildroot, 对应的 buildroot 版本, 需要提供构建文件系统的 buildroot 的配置文件, 如 sun50iw9p1_longan_defconfig

```
buildroot-201902/configs/sun50iw9p1_longan_defconfig
```

满足这些配置后, 按照 longan 的编译过程, 就可以对 buildroot 进行编译了。

3.2.8 向文件系统增加文件

3.2.8.1 方式一

- 1. 查看 BoardConfig*.mk 确认使用的文件系统压缩包，如 nor 方案的配置文件 BoardConfig_nor.mk, 看 LICHEE_ROOTFS:=target-arm-fast-linaro-5.3.tar.bz2, 或者在顶层路径 `cat .buildconfig | grep "LICHEE_ROOTFS"`
- 2. 到 `device/config/rootfs_tar/` 路径下, 先 `mkdir rootfs_tar`, 建立一个目录, 把压缩包的内容解压到 `rootfs_tar` 下, 具体命令如下:

```
mkdir rootfs_tar;  
tar -jxvf target-arm-fast-linaro-5.3.tar.bz2 -C rootfs_tar;
```

- 3. 拷贝相关文件到文件系统对应的路径下
- 4. 重新制作文件系统压缩包, 命令如下:

```
cd rootfs_tar;  
tar -jcvf target-arm-fast-linaro-5.3.tar.bz2 ./*  
cp target-arm-fast-linaro-5.3.tar.bz2 ../
```

- 4. 在 `sdk` 根路径下, 执行 `./build.distclean`, 清除所有生成的文件, 然后重新编译 `sdk` 即可

3.2.8.2 方式二

- 1. 在 `sdk` 根路径下, 先整体编译 `sdk`
- 2. 把相关文件拷贝到 `rootfs_def` 目录下相关的路径下, 命令如下:

```
//向文件系统增加一个可执行的二进制文件  
cp read-io ./out/v833/perf1/bsp/rootfs_def/usr/bin/
```

- 3. 重新编译打包

注: 使用该方式, 如果执行 `./build.distclean`, 会重新生成一个文件系统包, 之前 `cp` 到 `rootfs_def` 都会丢失

3.2.8.3 方式三

- 1. 直接通过 `sd` 卡, 或者 `adb` 等方式, 向小机端推送文件

4 linux 使用指南

4.1 U-boot 调试指南

在内核启动过程中，看见出现“Hit any key to stop autoboot”的时候按下 ssss 即可进入 boot 命令行。

该操作在 device/config/chips/{IC}/configs/default 目录下修改 env.cfg 文件的 bootdelay 参数为大于 0 时可用。

可以双击 Tab 键获取 boot 支持的命令，u-boot 常用指令有以下几个：

```
getenv: 获取环境变量
setenv: 设置环境变量
print: 打印环境变量
fdt print: 打印相关设备树参数
fdt set: 设置相关的设备树参数
boot: 启动内核
reset: 重新启动内核
```

4.2 Kernel 调试指南

4.2.1 打印/设置寄存器

全志平台实现了 sunxi_dump 机制，可以到/sys/class/sunxi_dump 目录下打印/设置相关的寄存器。部分命名如下所示：

```
echo 0x02001000,0x02001200 > dump //打印0x02001000到0x02001200这段寄存器数据
cat dump
echo 0x02500100 0x02000000 > wrte //设备0x02500100寄存器数据为0x02000000
```

4.2.2 挂载 debugfs

debugfs 系统可以提供一些调试信息，具体可以在控制台里面执行以下指令挂载 debugfs。

```
mount -t debugfs none /sys/kernel/debug
```

4.2.3 电脑与主机端传输文件

可以从电脑端往主机端发送文件，具体可以在控制台里面执行以下指令：

```
rz
```

如果想要直接把相应的文件/程序放到根文件系统里面，可以直接把文件放到 `longan` 的 `out/{IC}/{board}/{project}` 下面把 `rootf_def` 对应的目录下面，重新执行编译打包编译即可。




著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

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。