

[首页](#)[新闻](#)[博问](#)[专区](#)[闪存](#)[班级](#)[代码改变世界](#)[注册](#)[登录](#)[LXR](#) | [KVM](#) | [PM](#) | [Time](#) | [Interrupt](#) | [Systems Performance](#) | [Bootup Optimization](#)

## Arnold Lu@南京

联系方式: arnoldlu@qq.com

随笔 - 255, 文章 - 0, 评论 - 26, 阅读 - 113万

### 导航

[博客园](#)[首页](#)[新随笔](#)[联系](#)[订阅](#) [管理](#)

<	2021年4月						>
日	一	二	三	四	五	六	
28	29	30	31	1	2	3	
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	1	
2	3	4	5	6	7	8	

### 公告

昵称: ArnoldLu

园龄: 4年3个月

粉丝: 172

关注: 0

[+加关注](#)

### 搜索

[找找看](#)

## buildroot使用介绍

**buildroot**是Linux平台上一个构建嵌入式Linux系统的框架。整个Buildroot是由Makefile脚本和Kconfig配置文件构成的。你可以和编译Linux内核一样, 通过buildroot配置, menuconfig修改, 编译出一个完整的可以直接烧写到机器上运行的Linux系统软件(包含boot、kernel、rootfs以及rootfs中的各种库和应用程序)。

使用buildroot搭建基于qemu的虚拟开发平台, 参考《[通过buildroot+qemu搭建ARM-Linux虚拟开发环境](#)》。

## 1. buildroot入门

首先如何使用buildroot, 1.选择一个defconfig; 2.根据需要配置buildroot; 3.编译buildroot; 4.在qemu或者目标板上运行buildroot构建的系统。

### 1.1 buildroot目录介绍

进入buildroot首先映入眼帘的是一系列目录, 简要介绍如下:



```
.
├── arch: 存放CPU架构相关的配置脚本, 如arm/mips/x86, 这些CPU相关的配置, 在制作工具链时, 编译uboot和kernel时
├── board
├── boot
├── CHANGES
└── Config.in
```

## 常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

## 最新随笔

1.在CPU Hit/Miss时, Cache和Main Memory之间交互策略  
2.ARM Trusted Firmware分析——镜像签名/加密/生成、解析/解密/验签  
3.ARM Trusted Firmware分析——固件升级(FWU)  
4.ARM Trusted Firmware分析——TBBR、TBB、FIP-TBB  
5.ARM CoreSight Debug and Trace  
6.ARM Trusted Firmware分析——编译选项  
7.QEMU搭建虚拟化开发环境(QEMU 5.2.0)  
8.ARMv8-A QEMU运行OP-TEE/ATF环境搭建  
9.ARM工具链选择参考  
10.ARM PSCI在ATF和Linux kernel中的实现

## 积分与排名

积分 - 473554  
排名 - 886

## 随笔分类 (242)

Android相关学习总结(3)  
ARM Ecosystem(9)

```
├─ Config.in.legacy
├─ configs: 放置开发板的一些配置参数.
├─ COPYING
├─ DEVELOPERS
├─ dl: 存放下载的源代码及应用软件的压缩包.
├─ docs: 存放相关的参考文档.
├─ fs: 放各种文件系统的源代码.
├─ linux: 存放着Linux kernel的自动构建脚本.
├─ Makefile
├─ Makefile.legacy
├─ output: 是编译出来的输出文件夹.
│   └─ build: 存放解压后的各种软件包编译完成后的现场.
│   └─ host: 存放着制作好的编译工具链, 如gcc、arm-linux-gcc等工具.
│   └─ images: 存放着编译好的uboot.bin, zImage, rootfs等镜像文件, 可烧写到板子里, 让linux系统跑起来.
│   └─ staging
│   └─ target: 用来制作rootfs文件系统, 里面放着Linux系统基本的目录结构, 以及编译好的应用库和bin可执行文件.
├─ package: 下面放着应用软件的配置文件, 每个应用软件的配置文件有Config.in和soft_name.mk.
├─ README
├─ support
├─ system
└─ toolchain
```



## 1.2 buildroot配置

通过make xxx\_defconfig来选择一个defconfig, 这个文件在config目录下。

然后通过make menuconfig进行配置。



Target options --->选择目标板架构特性。  
Build options --->配置编译选项。

ARM Trusted Firmware(5)  
 Linux Debug(26)  
 Linux/UNIX系统编程手册(16)  
 Linux并发与同步专题(5)  
 Linux电源管理(14)  
 Linux进程管理(6)  
 Linux内存管理(32)  
 Linux时间子系统(28)  
 Linux相关学习总结(53)  
 Linux虚拟化KVM(7)  
 Linux中断子系统(14)  
 Python(3)  
 Zephyr(6)  
 性能优化(15)

## 随笔档案 (254)

2021年3月(1)  
 2021年2月(2)  
 2021年1月(7)  
 2020年12月(6)  
 2020年11月(5)  
 2020年10月(3)  
 2020年9月(2)  
 2020年8月(3)  
 2020年7月(2)  
 2020年6月(2)  
 2020年5月(1)  
 2020年4月(1)  
 2020年3月(3)  
 2020年2月(2)  
 2020年1月(3)  
 2019年12月(3)  
 2019年11月(1)  
 2019年10月(3)  
 2019年9月(5)  
 2019年8月(1)  
 2019年7月(9)

Toolchain ---> 配置交叉工具链, 使用buildroot工具链还是外部提供。  
 System configuration --->  
 Kernel --->  
 Target packages --->  
 Filesystem images --->  
 Bootloaders --->  
 Host utilities --->  
 Legacy config options --->



## 1.3 make命令使用

通过make help可以看到buildroot下make的使用细节, 包括对package、uclibc、busybox、linux以及文档生成等配置。



### Cleaning:

clean	- delete all files created by build
distclean	- delete all non-source files (including .config)

### Build:

all	- make world
toolchain	- build toolchain

### Configuration:

menuconfig	- interactive curses-based configurator-----
savedefconfig	- Save current config to BR2_DEFCONFIG (minimal config)-----

### Package-specific:-----

<pkg>	- Build and install <pkg> and all its dependencies-----
<pkg>-source	- Only download the source files for <pkg>
<pkg>-extract	- Extract <pkg> sources
<pkg>-patch	- Apply patches to <pkg>

2019年6月(2)  
 2019年5月(5)  
 2019年4月(5)  
 2019年3月(3)  
 2019年2月(2)  
 2019年1月(6)  
 2018年12月(1)  
 2018年11月(4)  
 2018年10月(2)  
 2018年9月(4)  
 2018年8月(4)  
 2018年7月(8)  
 2018年6月(2)  
 2018年5月(7)  
 2018年4月(4)  
 2018年3月(2)  
 2018年2月(9)  
 2018年1月(7)  
 2017年12月(8)  
 更多

## 最新评论

1. Re:bootrom/spl/u-boot/linu  
 逐级加载是如何实现的?  
 CSKY 中天微? ? ?  
 --心火合滨
2. Re:Linux时间子系统之  
 (一) : 时间的基本概念  
 博主, 地球自转变快了  
 --一世流离
3. Re:Linux中断管理  
 (1)Linux中断管理机制  
 好文  
 --梧桐素
4. Re:《TrustZone for  
 Armv8-A》阅读笔记

## buildroot使用介绍 - ArnoldLu - 博客园

<pkg>-depends	- Build <pkg>'s dependencies
<pkg>-configure	- Build <pkg> up to the configure step
<pkg>-build	- Build <pkg> up to the build step
<pkg>-show-depends	- List packages on which <pkg> depends
<pkg>-show-rdepends	- List packages which have <pkg> as a dependency
<pkg>-graph-depends	- Generate a graph of <pkg>'s dependencies
<pkg>-graph-rdepends	- Generate a graph of <pkg>'s reverse dependencies
<pkg>-dirclean	- Remove <pkg> build directory-----
<pkg>-reconfigure	- Restart the build from the configure step
<pkg>-rebuild	- Restart the build from the build step-----

busybox:

busybox-menuconfig	- Run BusyBox menuconfig
--------------------	--------------------------

uclibc:

uclibc-menuconfig	- Run uClibc menuconfig
-------------------	-------------------------

linux:

linux-menuconfig	- Run Linux kernel menuconfig-----
linux-savedefconfig	- Run Linux kernel savedefconfig
linux-update-defconfig	- Save the Linux configuration to the path specified by BR2_LINUX_KERNEL_CUSTOM_CONFIG_FILE

Documentation:

manual	- build manual in all formats
manual-pdf	- build manual in PDF
graph-build	- generate graphs of the build times-----
graph-depends	- generate graph of the dependency tree
graph-size	- generate stats of the filesystem size



MPU和TZASC的区别是什么呢？二者可以任选其一保护内存吗？

--楚慕

#### 5. Re:Linux中断管理

硬件中断号的映射是在系统启动过程中接卸DTS，然后经由

irq\_domain\_alloc\_irqs()完成映射到Linux中断号 硬件中断号的映射是在系统启动过程中解析DTS，然后经由irq\_domai...

--yi~ya~yo~

#### 阅读排行榜

1. 系统级性能分析工具perf的介绍与使用(114074)
2. 调试器GDB的基本使用方法(77541)
3. Linux CPU占用率监控工具小结(50474)
4. buildroot使用介绍(46807)
5. NB-IoT协议及其PSM(25367)
6. 数据分析之---Python可视化工具(20042)
7. Linux时间子系统之三: jiffies(19868)
8. Linux中断管理 (1)Linux中断管理机制(18637)
9. 非法指令(Illegal Instruction)问题定位(17554)
10. 开源HTTP解析器---http-parser和fast-http(17142)

#### 评论排行榜

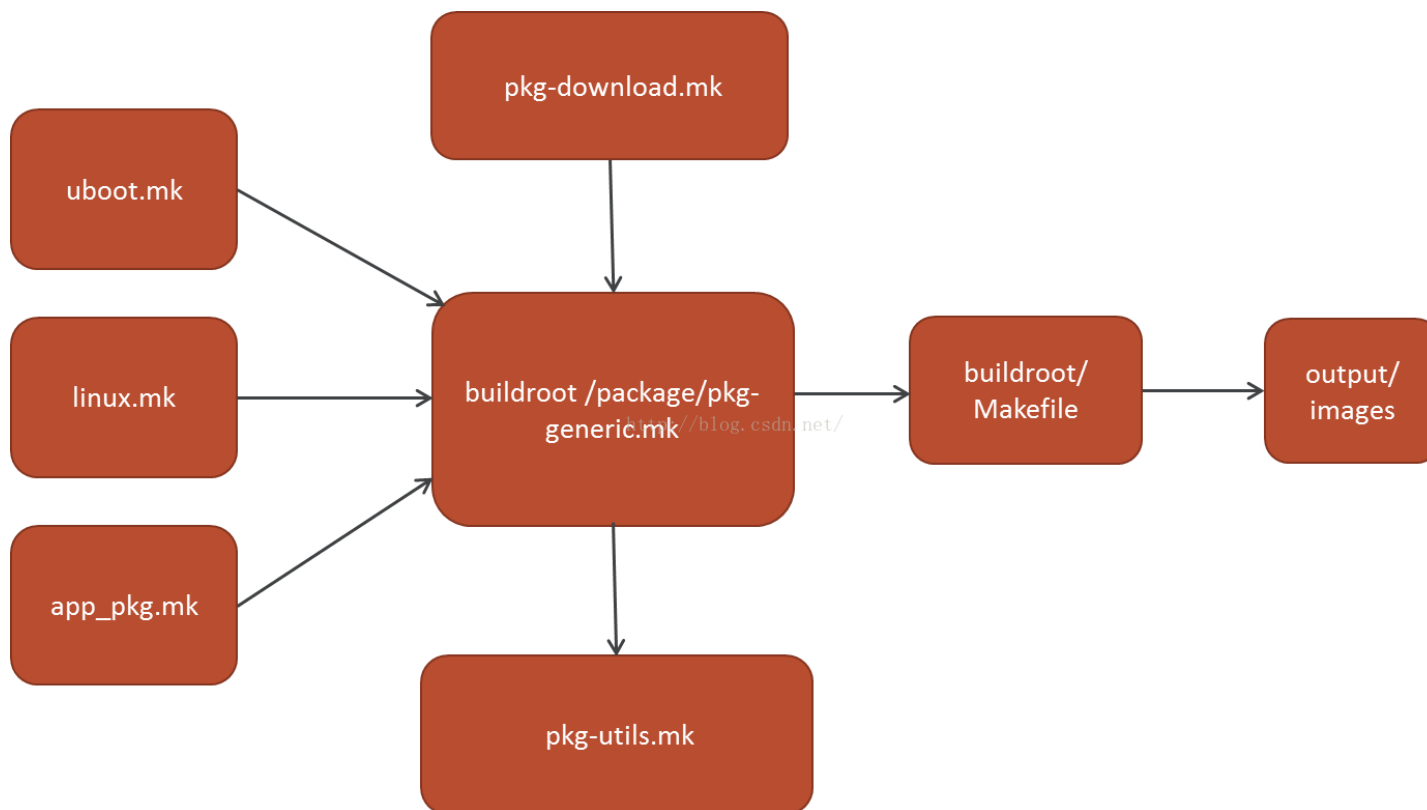
1. buildroot使用介绍(3)
2. Linux内存管理 (3)内核内存的布局图(2)

## 2. buildroot框架

Buildroot提供了函数框架和变量命令框架(下一篇文章将介绍细节)，采用它的框架编写的app\_pkg.mk这种Makefile格式的自动构建脚本，将被package/pkg-generic.mk 这个核心脚本展开填充到buildroot主目录下的Makefile中去。

最后make all执行Buildroot主目录下的Makefile，生成你想要的image。 package/pkg-generic.mk中通过调用同目录下的pkg-download.mk、pkg-utils.mk文件，已经帮你自动实现了下载、解压、依赖包下载编译等一系列机械化的流程。

你只要需要按照格式写Makefile脚app\_pkg.mk，填充下载地址，链接依赖库的名字等一些特有的构建细节即可。总而言之，Buildroot本身提供构建流程的框架，开发者按照格式写脚本，提供必要的构建细节，配置整个系统，最后自动构建出你的系统。



- 3. 开源HTTP解析器---http-parser和fast-http(2)
- 4. 《TrustZone for Armv8-A》阅读笔记(1)
- 5. bootrom/spl/uboot/linux 逐级加载是如何实现的? (1)
- 6. Linux Thermal Framework 分析及实施(1)
- 7. coredump配置、产生、分析以及分析示例(1)
- 8. sigsuspend()阻塞：异步信号SIGIO为什么会被截胡? (1)
- 9. Linux CPU占用率监控工具小结(1)
- 10. Linux中断管理 (3)workqueue工作队列(1)

推荐排行榜

- 1. 系统级性能分析工具perf的介绍与使用(6)
- 2. 调试器GDB的基本使用方法(5)
- 3. Linux内存管理专题(4)
- 4. /proc/<pid>/maps简要分析(3)
- 5. Linux内存管理 (1)物理内存初始化(3)

对buildroot的配置通过Config.in串联起来，起点在根目录Config.in中。

配置选项	Config.in位置	
Target options	arch/Config.in	
Build options	Config.in	
Toolchain	toolchain/Config.in	
System configuration	system/Config.in	
Kernel	linux/Config.in	
Target packages	package/Config.in	
Target packages-> Busybox		
Filesystem images	fs/Config.in	
Bootloaders	boot/Config.in	
Host utilities	package/Config.in.host	
Legacy config options	Config.in.legacy	

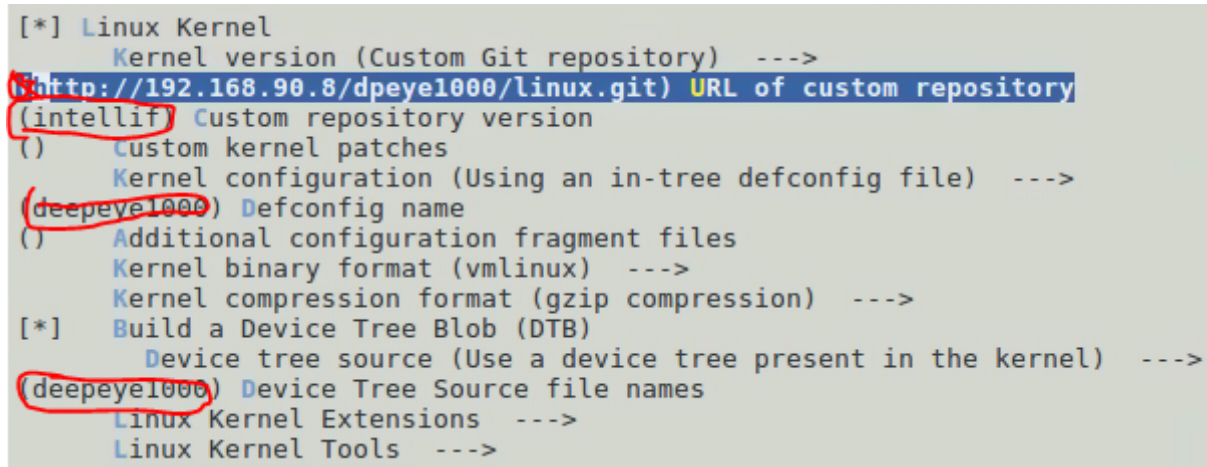
### 3. 配置Linux Kernel

对Linux内核的配置包括两部分：通过make menuconfig进入Kernel对内核进行选择，通过make linux-menuconfig对内核内部进行配置。

### 3.1 选择Linux内核版本

如下“Kernel version”选择内核的版本、“Defconfig name”选择内核config文件、“Kernel binary format”选择内核格式、“Device tree source file names”选择DT文件，

在“Linux Kernel Tools”中选择内核自带的工具，比如perf。



```
[*] Linux Kernel
    Kernel version (Custom Git repository) --->
    (http://192.168.90.8/dpeye1000/linux.git) URL of custom repository
    (intellif) Custom repository version
    () Custom kernel patches
    Kernel configuration (Using an in-tree defconfig file) --->
    (depeye1000) Defconfig name
    () Additional configuration fragment files
    Kernel binary format (vmlinux) --->
    Kernel compression format (gzip compression) --->
[*] Build a Device Tree Blob (DTB)
    Device tree source (Use a device tree present in the kernel) --->
    (depeye1000) Device Tree Source file names
    Linux Kernel Extensions --->
    Linux Kernel Tools --->
```

可以选择“Custom Git repository”来指定自己的Git库，在“Custom repository version”中指定branch名称。

选择“Using an in-tree defconfig file”，在“Defconfig name”中输入defconfig名称，注意不需要末尾\_defconfig。

选择“Use a device tree present in the kernel”，在“Device Tree Source file names”中输入dts名称，不需要.dts扩展名。

#### 3.1.1 Kernel binary format

可以选择vmlinux或者uImage。

ulImage是uboot专用的映像文件，它是在zImage之前加上一个长度为64字节的“头”，说明这个内核的版本、加载位置、生成时间、大小等信息；其0x40之后与zImage没区别。

zImage是ARM Linux常用的一种压缩映像文件，ulImage是U-boot专用的映像文件，它是在zImage之前加上一个长度为0x40的“头”，说明这个映像文件的类型、加载位置、生成时间、大小等信息。

vmlinux编译出来的最原始的内核elf文件，未压缩。

zImage是vmlinux经过objcopy gzip压缩后的文件，objcopy实现由vmlinux的elf文件拷贝成纯二进制数据文件。

ulImage是U-boot专用的映像文件，它是在zImage之前加上一个长度为0x40的tag。

选择vmlinux和ulImage的区别在于：

```
PATH="/bin..." BR_BINARIES_DIR=/home/.../output/images /usr/bin/make -j9 HOSTCC="/usr/bin/gcc"
```

如果是vmlinux，在结尾就是vmlinux。

## 3.2 对Kernel进行配置

通过make linux-menuconfig可以对内核内部细节进行配置。

让Linux内核带符号表：

```
# CONFIG_COMPILE_TEST is not set  
CONFIG_DEBUG_INFO=y
```

## 4. 配置文件系统APP

对目标板文件系统内容进行配置主要通过make menuconfig进入Target packages进行。



```
[*] BusyBox
(package/busybox/busybox.config) BusyBox configuration file to use?
() Additional BusyBox configuration fragment files
[ ] Show packages that are also provided by busybox
[ ] Install the watchdog daemon startup script
Audio and video applications --->
Compressors and decompressors --->
Debugging, profiling and benchmark --->
Development tools --->
Filesystem and flash utilities --->
Fonts, cursors, icons, sounds and themes --->
Games --->
Graphic libraries and applications (graphic/text) --->
Hardware handling --->
Interpreter languages and scripting --->
Libraries --->
Mail --->
Miscellaneous --->
Networking applications --->
Package managers --->
Real-Time --->
Security --->
Shell and utilities --->
System tools --->
Text editors and viewers --->
```

在Filesystem images中配置文件系统采用的格式，以及是否使用RAM fs。

```

[ ] cramfs root filesystem
[*] ext2/3/4 root filesystem
    ext2/3/4 variant (ext4) --->
()   filesystem label
(100M) exact size
(0)   exact number of inodes (leave at 0 for auto calculation)
(5)   reserved blocks percentage
(-O ^64bit) additional mke2fs options
[ ] Compression method (no compression) --->
[*] initial RAM filesystem linked into linux kernel
[ ] jffs2 root filesystem
[ ] romfs root filesystem
[ ] squashfs root filesystem
[*] tar the root filesystem
    Compression method (no compression) --->
()   other random options to pass to tar
[ ] ubi image containing an ubifs root filesystem
[ ] ubifs root filesystem
[ ] yaffs2 root filesystem

```

## 4.1 ramfs

如果选中“initial RAM filesystem linked into linux kernel”，那么文件系统会集成到vmlinux中。

如不选中，则vmlinux中只包括内核，文件系统会以其他形式提供，比如rootfs.cpio。

如果定义了BR2\_TARGET\_ROOTFS\_INITRAMFS，那么在编译的末期需要重新编译内核，将rootfs.cpio加入到vmlinux中。

fs/initramfs/initramfs.mk中：



```

rootfs-initramfs: linux-rebuild-with-initramfs

rootfs-initramfs-show-depends:
    @echo rootfs-cpio

.PHONY: rootfs-initramfs rootfs-initramfs-show-depends

```

```
ifeq ($(BR2_TARGET_ROOTFS_INITRAMFS),y)
TARGETS_ROOTFS += rootfs-initramfs
endif
```



在linux/linux.mk中:



```
.PHONY: linux-rebuild-with-initramfs
linux-rebuild-with-initramfs: $(LINUX_DIR)/.stamp_target_installed
linux-rebuild-with-initramfs: $(LINUX_DIR)/.stamp_images_installed
linux-rebuild-with-initramfs: rootfs-cpio
linux-rebuild-with-initramfs:
    @$(call MESSAGE, "Rebuilding kernel with initramfs")
    # Build the kernel.
    $(LINUX_MAKE_ENV) $(MAKE) $(LINUX_MAKE_FLAGS) -C $(LINUX_DIR) $(LINUX_TARGET_NAME)
    $(LINUX_APPEND_DTB)
    # Copy the kernel image(s) to its(their) final destination
    $(call LINUX_INSTALL_IMAGE, $(BINARIES_DIR))
    # If there is a .ub file copy it to the final destination
    test ! -f $(LINUX_IMAGE_PATH).ub || cp $(LINUX_IMAGE_PATH).ub $(BINARIES_DIR)
```



在打开initramfs的情况下, 重新将rootfs.cpio编译进内核vmlinu中。

然后将ulmage之类的文件拷贝到BINARIES\_DIR中。

## 5. 添加自己的APP

要添加自己的本地APP, 首先在package/Config.in中添加指向新增APP目录的Config.in;

然后在package中新增目录helloworld, 并在里面添加Config.in和helloworld.mk;

最后添加对应的helloworld目录。

## 5.1 添加package/Config.in入口

系统在make menuconfig的时候就可以找到对应的APP的Config.in。



```
diff --git a/package/Config.in b/package/Config.in
index 43d75a9..6ef9fad 100644
--- a/package/Config.in
+++ b/package/Config.in
@@ -1868,5 +1868,8 @@ menu "Text editors and viewers"
     source "package/uemacs/Config.in"
     source "package/vim/Config.in"
 endmenu
+menu "Private package"
+    source "package/helloworld/Config.in"
+endmenu
```



如果在make menuconfig的时候选中helloworld，在make savedefconfig的时候就会打开  
**BR2\_PACKAGE\_HELLOWORLD=y**。

## 5.2 配置APP对应的Config.in和mk文件

helloworld/Config.in文件，通过make menuconfig可以对helloworld进行选择。

只有在BR2\_PACKAGE\_HELLOWORLD=y条件下，才会调用helloworld.mk进行编译。

```
config BR2_PACKAGE_HELLOWORLD
    bool "helloworld"
    help
        This is a demo to add local app.
```

buildroot编译helloworld所需要的设置**helloworld.mk**，包括源码位置、安装目录、权限设置等。

下面的HELLOWORLD的开头也是必须的。



```
#####  
#  
# helloworld  
#  
#####  
  
HELLOWORLD_VERSION:= 1.0.0  
HELLOWORLD_SITE:= $(CURDIR)/work/helloworld  
HELLOWORLD_SITE_METHOD:=local  
HELLOWORLD_INSTALL_TARGET:=YES  
  
define HELLOWORLD_BUILD_CMDS  
    $(MAKE) CC="$(TARGET_CC)" LD="$(TARGET_LD)" -C $(@D) all  
endef  
  
define HELLOWORLD_INSTALL_TARGET_CMDS  
    $(INSTALL) -D -m 0755 $(@D)/helloworld $(TARGET_DIR)/bin  
endef  
  
define HELLOWORLD_PERMISSIONS  
    /bin/helloworld f 4755 0 0 - - - -  
endef  
  
$(eval $(generic-package))
```



如果源码在git上，需要如下设置：

```
DMA_TEST_VERSION:=master-----仓库分支名称
DMA_TEST_SITE:=http://.../dma.git-----仓库git地址
DMA_TEST_SITE_METHOD:=git-----获取源码的方式
```

**\_VERSION**结尾的变量是源码的版本号；**\_SITE\_METHOD**结尾的变量是源码下载方法；**\_SITE**结尾变量是源码下载地址。

**\_BUILD\_CMDS**结尾的变量会在buildroot框架编译的时候执行，用于给源码的Makefile传递编译选项和链接选项，调用源码的Makefile。

**\_INSTALL\_TARGET\_CMDS**结尾的变量是在编译完之后，自动安装执行，一般是让buildroot把编译出来的bin或lib拷贝到指定目录。

**\$(eval\$(generic-package))** 最核心的就是这个东西了，一定不能够漏了，不然源码不会被编译，这个函数就是把整个.mk构建脚本，通过Buildroot框架的方式，展开到Buildroot/目录下的Makefile中，生成的构建目标(构建目标是什么，还记得Makefile中的定义吗？)。

## 5.3 编写APP源码

简单的编写一个helloworld.c文件：



```
#include <stdio.h>

void main(void)
{
    printf("Hello world.\n");
}
```



然后编写Makefile文件：



```
CPPFLAGS +=
LDLIBS +=

all: helloworld

analyzestack: helloworld.o
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LDLIBS)

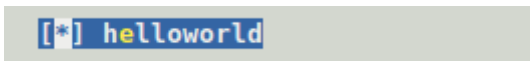
clean:
    rm -f *.o helloworld

.PHONY: all clean
```




## 5.4 通过make menuconfig选中APP

通过Target packages -> Private package进入，选中helloworld。



然后make savedefconfig，对helloworld的配置就会保存到qemu\_arm\_vexpress\_defconfig中。



## 5.5 编译APP

可以和整个平台一起编译APP；或者make helloworld单独编译。

这两个文件在选中此APP之后，都会被拷贝到output/build/helloworld-1.0.0文件夹中。

然后生成的bin文件拷贝到output/target/bin/helloworld，这个文件会打包到文件系统中。

如果需要清空相应的源文件，通过make helloworld-dirclean。

## 5.6 运行APP

在shell中输入helloworld, 可以得到如下结果。

```
# which helloworld
/bin/helloworld
# helloworld
Hello world.
```

添加APP工作完成。

## 6. uboot配置

使用uboot作为bootloader, 需要进行一些配置。

在选中U-boot作为bootloader之后, 会弹出一系列相关配置。

“U-Boot board name” 配置configs的defconfig名称。

“U-Boot Version” 选择Custom Git repository, 然后在 “URL of custom repository” 中选择自己的git地址, 并在 “Custom repository version” 中选择git的分支。

在 “U-Boot binary format” 中选择想要输出的image格式, 比如u-boot.img或者u-image.bin。

还可以选择 “Intall U-Boot SPL binary image”, 选择合适的SPL。

```
[ ] Barebox
[*] U-Boot
    Build system (Legacy) --->
    ( ) U-Boot board name
    U-Boot Version (Custom Git repository) --->
    ( ) URL of custom repository
    (master) Custom repository version
    () Custom U-Boot patches
    [ ] U-Boot needs dtc
    [ ] U-Boot needs pylibfdt
    [ ] U-Boot needs OpenSSL
    [ ] U-Boot binary format --->
    [*] Install U-Boot SPL binary image
    (spl/u-boot-spl-bh.bin) U-Boot SPL/TPL binary image name(s)
    [ ] Environment image ----
    [ ] Generate a U-Boot boot script
```



## 7. Finalizing target

在buildroot编译的末期，需要对编译结果进行一些检查或者其他操作。

buildroot预留了两个接口：

BR2\_ROOTFS\_OVERLAY - 指向一个目录，此目录下的所有文件将会覆盖到output/target下。比如一些配置文件，或者预编译的库等可以在此阶段处理。

BR2\_ROOTFS\_POST\_BUILD\_SCRIPT - 一个脚本，更加复杂的对文件进行删除、重命名、strip等功能。

BR2\_ROOTFS\_POST\_IMAGE\_SCRIPT - 对最终生成的images进行打包处理等。

### 7.1 FS Overlay

有些应用或者配置不通过编译，直接采取拷贝的方式集成到rootfs中，可以设置 “System configuration” -> “Root filesystem overlay directories” 。

设置的目录中的内容，会对output/target进行覆盖。

相关处理在Makefile中如下：

```
@$(foreach d, $(call qstrip,$(BR2_ROOTFS_OVERLAY)), \  
    $(call MESSAGE,"Copying overlay $(d)"); \  
    rsync -a --ignore-times --keep-dirlinks $(RSYNC_VCS_EXCLUSIONS) \  
        --chmod=u=rwX,go=rX --exclude .empty --exclude '*~' \  
        $(d) / $(TARGET_DIR)$(sep))
```

### 7.2 post build

除了fs overlay这种方式，buildroot还提供了一个脚本进行更加复杂的处理。

可以进行文件删除、重命名，甚至对带调试信息的文件进行strip等。

```
@$(foreach s, $(call qstrip,$(BR2_ROOTFS_POST_BUILD_SCRIPT)), \  
    $(call MESSAGE,"Executing post-build script $(s)"); \  
    $(s)
```

```
$(EXTRA_ENV) $(s) $(TARGET_DIR) $(call qstrip,$(BR2_ROOTFS_POST_SCRIPT_ARGS))$(sep))
```

一个post\_build.sh范例，对一系列文件进行删除和strip操作：



```
#!/bin/sh
#set -x
set +o errexit

cp -a ${BINARIES_DIR}/deepeye1000e_hk.dtb ${BINARIES_DIR}/deepeye1000.dtb

#Strip files in tbc_lists.txt. tbc means 'to be stripped'.
STRIP=${HOST_DIR}/bin/csky-abiv2-linux-strip

for file in `cat ${BR2_EXTERNAL_INTELLIF_PATH}/board/deepeye1000e_hk/tbs_lists.txt`
do
    if [ -e ${TARGET_DIR}${file} ]; then
        echo Strip ${file}.
        ${STRIP} ${TARGET_DIR}${file}
    else
        echo Not found ${file}.
    fi
done

#Delete files in tbd_lists.txt. tbd means 'to be deleted'
for file in `cat ${BR2_EXTERNAL_INTELLIF_PATH}/board/deepeye1000e_hk/tbd_lists.txt`
do
    if [ -e ${TARGET_DIR}${file} ]; then
        echo Delete ${file}.
        rm ${TARGET_DIR}${file}
    else
        echo Not found ${file}.
    fi
done
```

done

```
${BR2_EXTERNAL_INTELLIF_PATH}/board/common/post_build.sh
```



## 7.2 post image

post image在post build之后，更倾向于生成完整的release文件。包括进行一些images打包、debug文件打包等等。

```
.PHONY: target-post-image
target-post-image: $(TARGETS_ROOTFS) target-finalize
    @$(foreach s, $(call qstrip,$(BR2_ROOTFS_POST_IMAGE_SCRIPT)), \
        $(call MESSAGE,"Executing post-image script $(s)"); \
        $(EXTRA_ENV) $(s) $(BINARIES_DIR) $(call qstrip,$(BR2_ROOTFS_POST_SCRIPT_ARGS)) $(sep))
```

一个范例如下，对images文件进行打包操作。



```
#!/bin/sh
set -x -e

IMG_DIR=output/images
DEBUG_DIR=${IMG_DIR}/debug
KERNEL_DIR=output/build/linux-master

ROOTFS_CPIO=${IMG_DIR}/rootfs.cpio
KERNEL_IMAGE=${IMG_DIR}/uImage
SPL_IMAGE=${IMG_DIR}/u-boot-spl-bh.bin
UBOOT_IMAGE=${IMG_DIR}/u-boot.bin

IMG_TAR=images.tar.gz
DEBUG_TAR=debug.tar.gz
```

```
IMG_MD5=images.md5

rm -f ${IMG_TAR} ${DEBUG_TAR} ${IMG_MD5}

mkdir -p ${DEBUG_DIR}
cp -a ${KERNEL_DIR}/vmlinux ${KERNEL_DIR}/System.map ${ROOTFS_CPIO} ${DEBUG_DIR}/

tar -czf ${IMG_TAR} ${KERNEL_IMAGE} ${SPL_IMAGE} ${UBOOT_IMAGE}
tar -czf ${DEBUG_TAR} -C ${IMG_DIR} debug/

md5sum ${IMG_TAR} > ${IMG_MD5}
```



## 8. buildroot编译性能

buildroot还提供了一些命令，用于分析buildroot编译过程中耗时、依赖关系、文件系统尺寸等等。

通过make help发现相关命令：



Documentation:

manual	- build manual in all formats
manual-html	- build manual in HTML
manual-split-html	- build manual in split HTML
manual-pdf	- build manual in PDF
manual-text	- build manual in text
manual-epub	- build manual in ePub
graph-build	- generate graphs of the build times
graph-depends	- generate graph of the dependency tree
graph-size	- generate stats of the filesystem size
list-defconfigs	- list all defconfigs (pre-configured minimal systems)



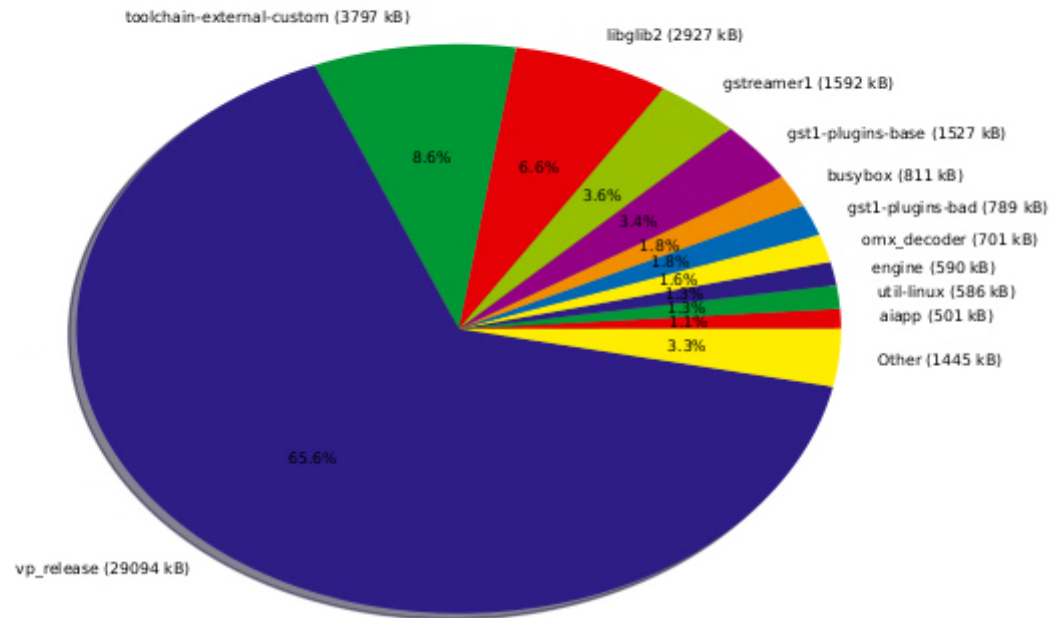




<https://www.cnblogs.com/arnoldlu/p/9553995.html>

# Filesystem size per package

Total filesystem size: 44367 kB



另外更有参考意义的是file-size-stats.csv和package-size-stats.csv文件。

通过file和package两个视角，更加详细的了解整个rootfs空间都被那些文件占用。

File name	Package name	File size	Package size	File size in package (%)	File size in system (%)
usr/lib/libgio-2.0.so.0.5400.2	libglib2	1444892	2927434	49.4	3.3
lib/libstdc++.so.6.0.22	toolchain-external-custom	1397196	3797216	36.8	3.1
lib/libc-2.28.9000.so	toolchain-external-custom	1268500	3797216	33.4	2.9
usr/lib/libglib-2.0.so.0.5400.2	libglib2	994124	2927434	34	2.2
usr/lib/libgstreamer-1.0.so.0.1402.0	gstreamer1	831884	1592624	52.2	1.9
bin/busybox	busybox	806596	811543	99.4	1.8
usr/lib/libifsdk.so	engine	590628	590628	100	1.3
root/app/AiApp	aiapp	497268	501904	99.1	1.1
lib/libm-2.28.9000.so	toolchain-external-custom	461384	3797216	12.2	1
usr/lib/libgstvideo-1.0.so.0.1402.0	gst1-plugins-base	363116	1527936	23.8	0.8
usr/lib/bellagio/libOMX.hantro.G1.video.decoder.so	omx_decoder	310664	701020	44.3	0.7
lib/libmount.so.1.1.0	util-linux	292240	586828	49.8	0.7
usr/lib/libgobject-2.0.so.0.5400.2	libglib2	278996	2927434	9.5	0.6
lib/libblkid.so.1.1.0	util-linux	271952	586828	46.3	0.6
usr/lib/libgstaudio-1.0.so.0.1402.0	gst1-plugins-base	258460	1527936	16.9	0.6
usr/lib/libgstbase-1.0.so.0.1402.0	gstreamer1	241772	1592624	15.2	0.5

《[buildroot认知](#)》

联系方式:arnoldlu@qq.com

分类: [Linux相关学习总结](#)

好文要顶

关注我

收藏该文



ArnoldLu

关注 - 0

粉丝 - 172

[+加关注](#)

2

0

« 上一篇: [通过buildroot+qemu搭建ARM-Linux虚拟开发环境](#)» 下一篇: [Linux kprobe调试技术使用](#)

posted on 2018-09-25 08:35 ArnoldLu 阅读(46810) 评论(3) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页

【推荐】大型组态、工控、仿真、CAD\GIS 50万行VC++源码免费下载!

【推荐】HMS Core Discovery 有奖直播--探索天谕手游的幻想世界

【推荐】限时秒杀! 国云大数据魔镜, 企业级云分析平台



### 园子动态:

- 致园友们的一封检讨书：都是我们的错
- 数据库实例 CPU 100% 引发全站故障
- 发起一个开源项目：博客引擎 fluss

### 最新新闻:

- 国家市场监督管理总局对阿里巴巴集团行政处罚决定书全文
- 亚马逊工会投票失利后 RWDSU负责人称“重新投票”的可能性很大
- 华为云人事架构一周两次大调整，徐直军任董事长、余承东为CEO
- Pixel Watch渲染图曝光 Apple Watch的最强竞争对手终于现身
- OneWeb和SpaceX卫星上周避免了一次潜在的在轨碰撞
- » 更多新闻...

Powered by:

博客园

Copyright © 2021 ArnoldLu

Powered by .NET 5.0 on Kubernetes