# EasyLinux平台编译系统设计

|  |  |  |
| --- | --- | --- |
| 历史/状态 | 作者 | 备注 |
| V1.0/正式 | bingo（523182570@qq.com） | 创建文档 |
|  |  |  |

目录

[EasyLinux平台编译系统设计 1](#_Toc465457083)

[一、 项目简介 3](#_Toc465457084)

[二、EasyLinux平台编译系统架构 3](#_Toc465457085)

[2.1Buildroot 3](#_Toc465457086)

[2.2EasyLinux平台目录结构 3](#_Toc465457087)

[2.2.1EasyLinux目录结构 3](#_Toc465457088)

[2.2.2Buildroot/easyLinux编译目录 4](#_Toc465457089)

[2.2.3EasyLinux/src目录结构 5](#_Toc465457090)

[2.2.4库和头文件的引用 5](#_Toc465457091)

[2.3板级适配 6](#_Toc465457092)

[2.3.1配置适配 6](#_Toc465457093)

[2.3.2编译适配 6](#_Toc465457094)

[2.4EasyLinux平台编译方法 7](#_Toc465457095)

[2.5添加一个Package 8](#_Toc465457096)

[2.6文件系统编译和定制 11](#_Toc465457097)

[2.7升级镜像制作 13](#_Toc465457098)

# 项目简介

嵌入式系统的开发过程较为复杂，编译，裁剪，定制等如果没有一套规范的流程将会难于管理和控制。本项目的目的是设计一个嵌入式Linux编译系统，实现代码的编译，定制和裁剪。Bootloader, 内核，驱动，文件系统，升级镜像等都可以自动化编译，打包。

# 二、EasyLinux平台编译系统架构

## 2.1Buildroot

Buildroot是一个非常优秀的开源嵌入式编译系统，其本身已经非常完善。但其默认是在线编译的，即从网上下载源码包进行编译。对企业而言，本地编译可方便进行版本控制。EasyLinux平台除了工具软件之外，其他的软件包如内核，bootloader,app等都是采用本地编译的方式。此外，嵌入式系统模块之间的耦合还是比较大，如头文件，库文件的引用等，需要设计一个目录结构编译时方便地对这些文件进行引用。

## 2.2EasyLinux平台目录结构

### 2.2.1EasyLinux目录结构

easyLinux

├── archive

│   └── gt2440

├── boot

│   └── u-boot-2015.01

├── buildroot

│   ├── arch

│   ├── board

│   ├── boot

│   ├── build

│   ├── CHANGES

│   ├── Config.in

│   ├── Config.in.legacy

│   ├── configs

│   ├── COPYING

│   ├── dl

│   ├── docs

│   ├── easylinux

│   ├── easylinux\_patch\_clean.sh

│   ├── easylinux\_patch.sh

│   ├── ext

│   ├── fs

│   ├── linux

│   ├── Makefile

│   ├── Makefile.legacy

│   ├── package

│   ├── README

│   ├── support

│   ├── system

│   └── toolchain

├── kernel

│   └── linux-3.18.6

├── LICENSE

├── README.md

├── sdk

└── src

├── application

└── platform

EasyLinux平台有archive、boot、buildroot、kernel、sdk、src六个一级目录，每个目录的设计如下：

Archive:存放src目录下编译生成的库文件，以机型为子目录存放，如archive/gt2440.

Boot:这个目录下存放bootloader源码，如uboot。

Buildroot:这个目录下添加了我们自己的目录easylinux，用于编译easylinux平台特有的软件包。

Kernel:存放内核。

Sdk:存放芯片商SDK

Src:存放项目源代码

### 2.2.2Buildroot/easyLinux编译目录

easylinux

├── Config.in

├── core

│   ├── Config.in

│   └── core.mk

├── easylinux.mk

├── procmgr

│   ├── Config.in

│   └── procmgr.mk

└── watcher

   ├── Config.in

└── watcher.mk

EasyLinux编译目录中定义src目录下的源码包的编译规则。

### 2.2.3EasyLinux/src目录结构

src

├── application

│   ├── adapter

│   ├── app

│   ├── drivers

│   ├── include

│   └── lib

└── platform

├── adapter

├── app

│   ├── core

│   │   ├── CMakeLists.txt

│   │   ├── include

│   │   └── src

│   ├── procmgr

│   └── watcher

├── drivers

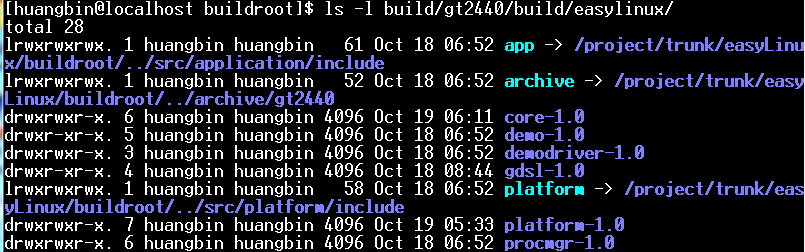
├── include

└── lib

Src目录下存放我们自己开发的软件包源码，包括应用层App和内核驱动，所有软件包都以cmake组织。

### 2.2.4库和头文件的引用

Buildroot中的package编译时会把源码拷贝到$(BUILD\_DIR)目录下进行编译，为了便于管理，我们把easylinux的package拷贝到$(BUILD\_DIR)/easylinux目录下进行编译。编译产生的库文件存放到easylinux/archive中。通过在$(BUILD\_DIR)/easylinux目录下创建软连接arvhive,plat分别指向easylinux/arvhive中的库文件目录和src/platform/include下的头文件目录。则编译软件包时可通过../arvhive和../plat目录引用头文件和库文件。



## 2.3板级适配

为了使不同的开发板和芯片可以共用一套编译系统，需要进行一定的适配。

### 2.3.1配置适配

buildroot/board/samsung

├── common

│   ├── busybox.config

│   ├── linux.config

│   ├── uboot.config

│   ├── uboot.mk

│   └── uClibc-0.9.33.config

└── yoka

└── uboot.mk

在vendor/board目录下存放各自的配置文件。

Xxx\_defconfig文件中通过BR2\_LINUX\_KERNEL\_CUSTOM\_CONFIG\_FILE等变量可以指定配置文件的路径，可以为不同的板指定不同的配置文件。

### 2.3.2编译适配

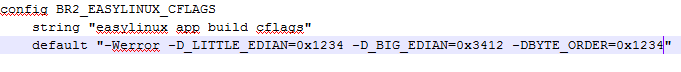
1. mk文件中引入板级定制mk,以uboot为例：

|  |
| --- |
| #include board common mk files if any  -include $(BR2\_BOARD\_COMMON\_DIR)/uboot.mk  #include board specify mk files if any  -include $(BR2\_BOARD\_CUSTOM\_DIR)/uboot.mk  #include the mk file to fix the pkgdir in package/pkg-utils.mk  include $(TOPDIR)/boot/uboot/uboot-last.mk  $(eval $(generic-package)) |

在$(eval $(generic-package))之前插入上面的规则，则可以在板级的uboot.mk中重新设置一些环境变量，达到不同的板可以有不同的编译参数，编译路径的目的。

1. 使用全局编译参数

Easylinux/Config.in中添加全局编译参数配置项，在所有package的mk文件中添加进他们的CFLAGS中。



如在core.mk中

|  |
| --- |
| CORE\_CFLAGS += $(BR2\_EASYLINUX\_CFLAGS)  #CORE\_CFLAGS +=  CORE\_CONF\_OPTS += -DCMAKE\_C\_FLAGS="$(CORE\_CFLAGS)" |

在core\_main.c中

|  |
| --- |
| #if BYTE\_ORDER == \_LITTLE\_EDIAN  Do something  #else  Do something  #endif |

1. 在模块的编译中定义宏

如在core.mk中

|  |
| --- |
| #如果是yoka  Ifeq ($(BR2\_EASYLINUX\_PROJECT\_NAME),yoka)  CORE\_CFLAGS += -DEASYLINUX\_YOKA  endif |

## 2.4EasyLinux平台编译方法

编译所有的操作都在easylinux/buildroot目录下

2.4.1打入配置和编译

|  |
| --- |
| make O=build\_dir xxx\_defconfig  其中build\_dir为想要输出到的目录  Xxx\_defconfig为configs目录下的机型配置文件  make O=build\_dir {target}  target是可选的，如果输入了target则只编译这个目标，否则编译配置的所有软件包。  make O=build\_dir target-dirclean  target是目标，清理特定的软件包  make O=build\_dir easylinux-clean  这个命令是easylinux平台的，用于清理easylinux平台的所有软件包。Buildroot全部编译需要较长的时间，这个命令可用于只编译easylinux软件包。 |

2.4.2定制

|  |
| --- |
| make O=build\_dir menuconfig  make O=build\_dir savedefconfig  定制package,保存到xxx\_defconfig文件中  make O=build\_dir linux-menuconfig  make O=build\_dir linux-update-config  定制linux内核并保存  make O=build\_dir busybox-menuconfig  make O=build\_dir busybox-update-config  定制busybox并保存  make O=build\_dir uboot-menuconfig  make O=build\_dir uboot-update-config  定制uboot并保存 |

## 2.5添加一个Package

2.5.1添加app

以添加应用程序procmgr为例,procmgr是系统启动第一个用户进程，其他所有进程都是他的子进程并由他管理。

|  |
| --- |
| 在src/platform/app相应目录下添加源代码,在源码下添加CMakeLists.txt文件  cmake\_minimum\_required(VERSION 3.0)  #project name  PROJECT(procmgr)  #head file path  INCLUDE\_DIRECTORIES(  ./include  ../platform  )  #库路径，实际指向easylinux/archive/gt2440  SET(STATIC\_LIB\_PATH ../archive)  SET(DYNAMIC\_LIB\_PATH ../archive/usr/lib)  LINK\_DIRECTORIES(  ${STATIC\_LIB\_PATH}  ${DYNAMIC\_LIB\_PATH})  #source directory  AUX\_SOURCE\_DIRECTORY(src PROCMGR\_FILES)  #set environment variable  #SET(CMAKE\_INSTALL\_PREFIX /easylinux)  SEPARATE\_ARGUMENTS(LINK UNIX\_COMMAND ${LINK\_LIB})  #add executable file  ADD\_EXECUTABLE(procmgr ${PROCMGR\_FILES})  #add link library  TARGET\_LINK\_LIBRARIES(procmgr pthread ${LINK})  install(TARGETS procmgr RUNTIME DESTINATION bin)  2.在buildroot/easylinux/procmgr目录下添加Config.in  config BR2\_EASYLINUX\_PROCMGR  bool "easylinux app procmgr"  default n  3.在buildroot/easylinux/procmgr目录下添加procmgr.mk  ################################################################################  #  # procmgr  #  ################################################################################  PROCMGR\_VERSION = 1.0  PROCMGR\_SITE = $(TOPDIR)/../src/platform/app/procmgr  PROCMGR\_SITE\_METHOD = local  PROCMGR\_INSTALL\_STAGING = NO  PROCMGR\_INSTALL\_TARGET = YES  #PROCMGR\_CONF\_OPTS +=  #PROCMGR\_DEPENDENCIES +=  #这里拷贝数据到文件系统,用我们的脚本替换掉buildroot生成的脚本  define PROCMGR\_COPY\_DATA  chmod 777 $(TARGET\_DIR)/etc/init.d -R  rm -rf $(TARGET\_DIR)/etc/init.d/\*  chmod 777 $(PROCMGR\_SITE)/etc -R  cp -rf $(PROCMGR\_SITE)/etc/inittab $(TARGET\_DIR)/etc  cp -rf $(PROCMGR\_SITE)/etc/start\_system.sh $(TARGET\_DIR)/etc  cp -rf $(PROCMGR\_SITE)/etc/init.d/\* $(TARGET\_DIR)/etc/init.d/  endef  PROCMGR\_POST\_INSTALL\_TARGET\_HOOKS += PROCMGR\_COPY\_DATA  PROCMGR\_DEPENDENCIES += platform  PROCMGR\_LIBS += libplatform.so  PROCMGR\_CFLAGS += $(BR2\_EASYLINUX\_CFLAGS)  PROCMGR\_CFLAGS += -Wall -fno-builtin  PROCMGR\_CONF\_OPTS += -DLINK\_LIB="$(PROCMGR\_LIBS)"  PROCMGR\_CONF\_OPTS += -DCMAKE\_C\_FLAGS="$(PROCMGR\_CFLAGS)"  $(eval $(cmake-package)) |

2.5.2添加内核驱动

|  |
| --- |
| 1.在src/app相应目录下添加源代码,在源码下添加Makefile文件  obj-m = demodriver.o  2.在buildroot/easylinux/demodriver目录下添加Config.in  3.在buildroot/easylinux/demodriver目录下添加demodriver.mk  ################################################################################  #  # demodriver  #  ################################################################################  DEMODRIVER\_VERSION = 1.0  DEMODRIVER\_SITE = $(TOPDIR)/../src/application/drivers/demodriver  DEMODRIVER\_SITE\_METHOD = local  DEMODRIVER\_INSTALL\_STAGING = NO  DEMODRIVER\_INSTALL\_TARGET = YES  #DEMODRIVER\_CONFIG\_SCRIPTS = DEMODRIVER-config  #DEMODRIVER\_DEPENDENCIES = host-libaaa libbbb  define DEMODRIVER\_BUILD\_CMDS  $(TARGET\_MAKE\_ENV) $(MAKE) $(LINUX\_MAKE\_FLAGS) -C $(LINUX\_DIR) M=$(@D) modules  Endef  define DEMODRIVER\_INSTALL\_TARGET\_CMDS  cp $(@D)/\*.ko $(TARGET\_DIR)/easylinux/lib/modules  endef  $(eval $(generic-package)) |

2.5.3添加动态库

|  |
| --- |
| 1.在src/app相应目录下添加源代码,在源码下添加CMakeLists.txt文件  cmake\_minimum\_required(VERSION 3.0)  #project name  PROJECT(platform)  #head file path  INCLUDE\_DIRECTORIES(  ../platform  )  #source directory  AUX\_SOURCE\_DIRECTORY(demo DEMO\_FILES)  #set environment variable  #SET(CMAKE\_INSTALL\_PREFIX /easylinux)  #add executable file  ADD\_LIBRARY(platform SHARED ${DEMO\_FILES})  #add link library  #TARGET\_LINK\_LIBRARIES(core m)  install(TARGETS platform LIBRARY DESTINATION lib)  2.在buildroot/easylinux/platform目录下添加Config.in  3.在buildroot/easylinux/platform目录下添加platform.mk  ################################################################################  #  # PLATFORM  #  ################################################################################  PLATFORM\_VERSION = 1.0  PLATFORM\_SITE = $(TOPDIR)/../src/platform/lib  PLATFORM\_SITE\_METHOD = local  PLATFORM\_INSTALL\_STAGING = NO  PLATFORM\_INSTALL\_TARGET = YES  PLATFORM\_INSTALL\_TARGET\_OPTS = DESTDIR=$(BR2\_EASYLINUX\_ARCHIVE\_DIR) install  #PLATFORM\_CONF\_OPTS +=  #PLATFORM\_DEPENDENCIES +=  PLATFORM\_CFLAGS += $(BR2\_EASYLINUX\_CFLAGS)  #PLATFORM\_CFLAGS +=  PLATFORM\_CONF\_OPTS += -DCMAKE\_C\_FLAGS="$(PLATFORM\_CFLAGS)"  $(eval $(cmake-package)) |

## 2.6文件系统编译和定制

目前easylinux平台设计为3个文件系统，根文件系统initramfs，用户镜像文件系统usrimage.jffs2和用户配置文件系统usrconf.jffs2。initramfs是最小根文件系统，和内核编译到一起挂载到内存。Usrimage.jffs2用来存放我们自己的可执行文件和库以及一些相应的数据。Usrconf.jffs用来保存配置，升级的时候可以保留这个分区，从而保存用户配置。

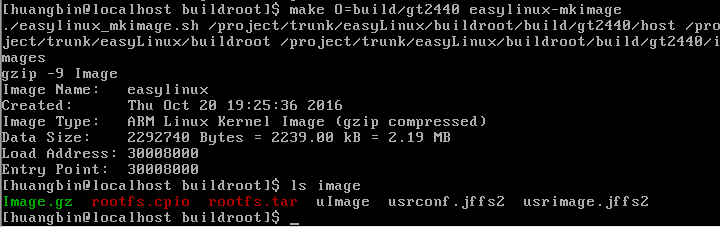
Initramfs基于cpio制作，我们在cpio.mk中对根文件系统进行定制

|  |
| --- |
| define ROOTFS\_CPIO\_CMD  #删掉不需要的文件  rm -rf $(TARGET\_DIR)/usr/bin/top && \  rm -rf $(TARGET\_DIR)/usr/bin/unzip && \  rm -rf $(TARGET\_DIR)/usr/bin/wget && \  #先拷出easylinux文件夹，这些数据放入usrimage.jffs2中  cp -arf $(BR2\_EASYLINUX\_ARCHIVE\_DIR)/usr/lib/\* $(TARGET\_DIR)/easylinux/lib && \  mkdir -p $(TARGET\_DIR)/../tmptarget && \  cp -rf $(TARGET\_DIR)/easylinux $(TARGET\_DIR)/../tmptarget && \  rm -rf $(TARGET\_DIR)/easylinux && \  #制作cpio镜像  cd $(TARGET\_DIR) && find . | cpio --quiet -o -H newc > $@ && \  cp -rf $(TARGET\_DIR)/../tmptarget/easylinux $(TARGET\_DIR)  endef |

制作usrimage.jffs2和usrconf.jffs2

|  |
| --- |
| define ROOTFS\_JFFS2\_CMD  mkdir -p $(TARGET\_DIR)/../tmptarget/usrconf && \  mkdir -p $(TARGET\_DIR)/../tmptarget/usrconf/conf && \  mkdir -p $(TARGET\_DIR)/../tmptarget/usrconf/log && \  mkdir -p $(TARGET\_DIR)/../tmptarget/usrconf/key && \  $(MKFS\_JFFS2) $(JFFS2\_OPTS) –d \  $(TARGET\_DIR)/../tmptarget/usrconf \  $(BINARIES\_DIR)/usrconf.jffs2 && \  $(MKFS\_JFFS2) $(JFFS2\_OPTS) –d\  $(TARGET\_DIR)/../tmptarget/easylinux –o\  $(BINARIES\_DIR)/usrimage.jffs2  endef |

编译系统后即可得到3个文件系统镜像



启动脚本start\_system.sh制作

|  |
| --- |
| #!/bin/sh  echo "start system,please wait..."  #更改printk打印级别  echo 4 > /proc/sys/kernel/printk  mkdir -p /mnt/easylinux  mkdir -p /mnt/usrconf  #挂载usrimage.jffs2  mount -t jffs2 /dev/mtdblock3 /mnt/easylinux  #挂载usrconf.jffs2  mount -t jffs2 /dev/mtdblock4 /mnt/usrconf  mkdir -p /easylinux  #将usrimage.jffs2也挂载到ramfs中  mount -t ramfs none /easylinux  cp -rf /mnt/easylinux/\* /easylinux  chmod 0644 /easylinux/lib/\*.so  #设置环境变量  export PATH=$PATH:/easylinux/app/bin  export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/easylinux/lib  #启动我们的第一个进程，其他进程都由他派生  /usr/bin/procmgr  echo "something error,we should not go here" |

Start\_system.sh放在src/platform/app/procmgr/etc中，编译procmgr时，和etc下的其他定制文件一起拷贝到根文件系统中。定制inittab文件

|  |
| --- |
| # now run any rc scripts  ::sysinit:/etc/init.d/rcS > /dev/null  # Put a getty on the serial port  #console::respawn:/sbin/getty -L console 0 vt100 # GENERIC\_SERIAL  #这里启动我们的脚本  console::sysinit:/etc/start\_system.sh > /dev/null  # Stuff to do for the 3-finger salute  ::ctrlaltdel:/sbin/reboot |

## 2.7升级镜像制作

Flash分区和文件系统设计见《EasyLinux平台文件系统设计》

有两种镜像类型，一种是up.bin，用于系统升级，仅包含有效数据和头部信息。一种是flash.bin，用于工厂生产时烧写，包含数据和分区之间填充的空洞。

Easylinux-mkimage目标用于制作镜像文件

|  |
| --- |
| make O=build/gt2440 easylinux-mkimage |

生成的镜像在buildroot/image/img目录下，buildroot/image/source目录下是用于制作的源文件。其中，gt2440\_all包含uboot,kernel,usrimage等所有分区数据，exclude\_uboot去掉了uboot,

Exclude\_uboot\_kernel去掉了uboot和kernel。升级的时候可根据需要只升级部分区域。

