**GK710X**芯片系列

**Linux SDK开发环境**

用户手册

**湖南国科微电子有限公司**

2015-09-08

v1.0

# 修订历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version. | Date | Author | Approved By | Modification |
| 0.1 | 2015/3/14 | IPC | GOKE | 初始草稿 |
| 0.2 | 2015/5/20 | IPC | GOKE | 修改编译运行部分内容，统一命令格式 |
| 0.3 | 2015/6/13 | IPC | GOKE | 添加ctlserver运行前依赖库如何手动添加 |
| 0.4 | 2015/7/3 | IPC | GOKE | 文字错误修改 |
| 0.5 | 2015/8/4 | IPC | GOKE | 支持glibc、uClibc两种编译环境 |
| 0.6 | 2015/8/7 | IPC | GOKE | 增加关于内核和文件系统烧写的说明 |
| 0.7 | 2015/8/17 | IPC | GOKE | 增加 flex/bison 软件包安装说明 |
| 0.8 | 2015/8/22 | IPC | GOKE | 增加内核配置说明 |
| 0.9 | 2015/9/2 | IPC | GOKE | 增加文件系统类型介绍，使用超链接 |
| 1.0 | 2015/9/8 | IPC | GOKE | 增加用户如何基于SDK添加自己的硬件配置 |
| Note 1：Do please update this table each time you change this document. | | | | |

# 目录

[修订历史 2](#_Toc429498228)

[目录 3](#_Toc429498229)

[1前言 5](#_Toc429498230)

[1.1目的 5](#_Toc429498231)

[1.2目标读者 5](#_Toc429498232)

[1.3术语与缩写词 5](#_Toc429498233)

[1.4参考文档 5](#_Toc429498234)

[1.5参考软件及版本 5](#_Toc429498235)

[2嵌入式开发环境 6](#_Toc429498236)

[2.1硬件环境搭建 6](#_Toc429498237)

[2.2软件开发环境 6](#_Toc429498238)

[2.2.1 PC Windows软件环境 6](#_Toc429498239)

[2.2.2 PC Linux软件环境 7](#_Toc429498240)

[3 Linux SDK安装与配置 8](#_Toc429498241)

[3.1概述 8](#_Toc429498242)

[3.2 Linux SDK文件目录结构 8](#_Toc429498243)

[3.3 Linux SDK配置 9](#_Toc429498244)

[3.4 Linux SDK安装 12](#_Toc429498245)

[4 Linux SDK 编译与运行 13](#_Toc429498246)

[4.1编译环境 13](#_Toc429498247)

[4.2整体编译 13](#_Toc429498248)

[4.3内核配置选项 13](#_Toc429498249)

[4.4如何运行 14](#_Toc429498250)

[4.4.1 UBOOT 14](#_Toc429498251)

[4.4.2内核镜像 14](#_Toc429498252)

[4.4.3文件系统 14](#_Toc429498253)

[4.4.4运行ctlserver 14](#_Toc429498254)

[5 Linux 根文件系统 16](#_Toc429498255)

[5.1介绍 16](#_Toc429498256)

[5.2根文件系统配置 17](#_Toc429498257)

[5.3根文件系统安装 17](#_Toc429498258)

[5.4组件的安装与卸载 17](#_Toc429498259)

[5.5自定义的根文件系统组件 18](#_Toc429498260)

[5.5.1打包与命名 18](#_Toc429498261)

[5.5.2编写配置文件 18](#_Toc429498262)

[5.6文件系统类型介绍 19](#_Toc429498263)

[5.6.1闪存的特性与限制 19](#_Toc429498264)

[5.6.2 JFFS2 19](#_Toc429498265)

[5.6.3 SquashFS 21](#_Toc429498266)

[5.6.4 YAFFS2 22](#_Toc429498267)

[5.6.5 UBIFS 23](#_Toc429498268)

[5.6.6 NFS 26](#_Toc429498269)

[5.7制作根文件系统映像 26](#_Toc429498270)

[5.7.1文件系统配置 27](#_Toc429498271)

[5.7.2文件系统制作 27](#_Toc429498272)

[6 Flash分区与烧写 27](#_Toc429498273)

[6.1 Flash分区 27](#_Toc429498274)

[6.2 Flash 烧写 27](#_Toc429498275)

[6.2.1把内核zImage写入Flash 28](#_Toc429498276)

[6.2.2把根文件系统映像写入Flash 28](#_Toc429498277)

[6.2.3启动内核 28](#_Toc429498278)

[7附录 30](#_Toc429498279)

[7.1虚拟机上的NFS服务安装与配置 30](#_Toc429498280)

[7.1.1安装Ubuntu NFS 30](#_Toc429498281)

[7.1.2配置/etc/exports 30](#_Toc429498282)

[7.1.3启动NFS服务 30](#_Toc429498283)

[7.1.4测试NFS服务 30](#_Toc429498284)

[7.2如何基于SDK添加自己的硬件配置 31](#_Toc429498285)

[7.2.1 uboot部分修改 31](#_Toc429498286)

[7.2.2 kernel部分修改 32](#_Toc429498287)

# 前言

## 目的

本文档用于介绍基于国科微电子公司GK710X芯片系列的Linux SDK开发环境部署和使用说明。

## 目标读者

本文档的目标读者为采用 GK710X 系列芯片进行板端软件（包括IPCamera应用，外设驱动等）开发的开发人员。

## 术语与缩写词

无

## 参考文档

无

## 参考软件及版本

无

# 嵌入式开发环境

## 硬件环境搭建

基于GK710X的嵌入式开发的硬件环境如下图所示：



图 2-1-1

## 软件开发环境

出于开发过程的需要，我们需要在开发用PC上同时部署Windows和Linux。我们推荐在安装了Windows操作系统的PC上利用虚拟机部署Linux操作系统。

### PC Windows软件环境

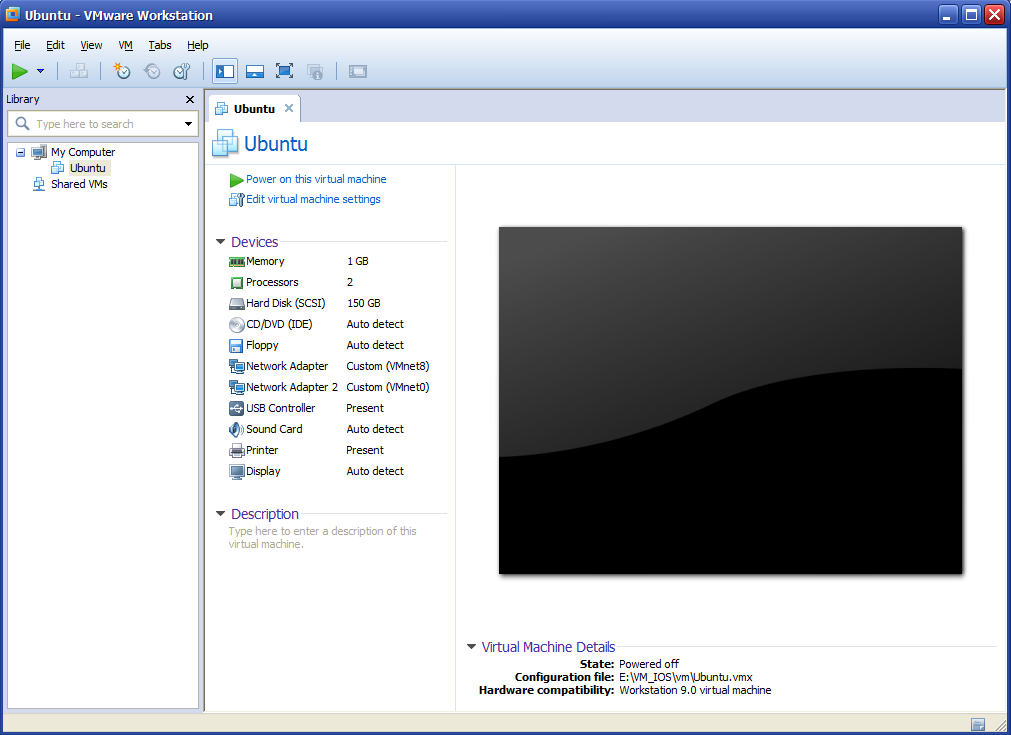
Windows操作系统版本建议使用 Windows 7 或者 Windows XP。

在Windows中，我们需要安装如下表所列的软件用于支持开发和调试：

|  |  |
| --- | --- |
| Windows软件 | 功能 |
| Putty/SecureCRT/超级终端 | 串口调试信息显示和命令输入 |
| TFTP Server | 内核/文件系统下载 |
| ARM RealView Debugger | 应用/驱动调试 |
| GOKE ImageTuning Tool | 图像调优 |
| GOKE浏览器插件 | 基于浏览器的视频播放 |
| VMware/Virtual Box | 运行Linux虚拟机 |

### PC Linux软件环境

在VMware上安装Linux开发环境:



PC Linux操作系统环境推荐使用如下发行版：

Ubuntu12.04 （32位）或更高版本

CentOS6.5 （32位）或更高版本

本文档所有操作示例均以Ubuntu12.04为例。

在Linux中，我们需要安装如下表所列的软件用于支持开发和调试：

|  |  |
| --- | --- |
| Linux软件 | 功能 |
| Samba | 用于Windows系统和Linux虚拟机之间的文件访问 |
| NFS Server | 用于调试时从NFS加载根文件系统 |
| GK7101 Linux SDK | 国科GK7101芯片嵌入式软件开发包 |
| flex  bison | 用于 boa web server 的编译  在Ubuntu环境下输入以下命令进行安装：  $ sudo apt-get install flex bison  在CentOS环境下输入以下名利进行安装：  *$ sudo yum install flex bison* |

备注: GK7101 Linux SDK 可以从国科发布的PDK中获取。

# Linux SDK安装与配置

## 概述

GK7101 Linux SDK以压缩包的形式发布，压缩包文件名为GK7101\_SDK\_V1.x.x.tar.bz2，需要从GK7101\_PDK\_V1.x.x.zip中解压得到。

## Linux SDK文件目录结构

把从3.1 所述的GK7101\_SDK\_v1.x.x.tar.bz2拷贝至Linux虚拟机环境中运行如下命令进行解压：

*$ tar xfGK7101\_SDK\_v1.x.x.tar.bz2*

解压后得到的文件目录结构如下图所示：



## Linux SDK配置

在GK7101 Linux SDK安装之前我们需要对其进行配置。配置的主要过程为对处于顶层目录的Config.mak文件进行修改。

*################################################################################*

*################################################################################*

*##*

*## Config.mak -- user specific development environment settings*

*##*

*## Version: $Id: Config.mak 3169 2015-05-05 06:09:37Z huangmin $*

*##*

*## (C) Copyright 2012-2015 byGoke Microelectronics Co.,Ltd*

*##*

*## Copy this file intto Config.mak.<username> to protect your private settings*

*## from being overwritten the next time you get an update.*

*## The make environment will first check for file "Config.mak.<username>",*

*## it will use the file "Config.mak" only if the user specific configuration*

*## file does not exist. <username> should be the string reported by*

*## "echo %USERNAME%" if you are using the standard windows shell, Cygwin*

*## users should call "echo $USERNAME" instead.*

*##*

*################################################################################*

*################################################################################*

*################################################################################*

*#*

*# target/host/decoder/system settings*

*#*

*# mandatory setting*

*#*

*# none of these setting should be modified.*

*#*

*##################################################*

*HOST ?= linux*

*TARGET ?= arm11-gcc*

*DECODER ?= GK7101*

*SYSTEM ?= linux*

*################################################################################*

*#*

*# board settings*

*#*

*# mandatory setting*

*#*

*# Available value for BOARD:*

*#*

*# GK7101\_FPGA : -- Once this board selected, the [linux/kernel/fpga] will be*

*# compiled and installed during the make process*

*# -- Once this board selected, u-boot will compiled.*

*#*

*# GK7101\_EVB : -- Once this board selected, the [linux/kernel/drv] will be*

*# compiled and installed during the make process*

*# -- Once this board selected, u-boot will compiled.*

*#*

*#*

*# Note for BOARD:*

*# - Need run (make deploy-env) to make it effect after changing this setting.*

*#*

*################################################################################*

*BOARD ?= GK7101\_EVB*

*################################################################################*

*#*

*# linux specific [kernel/toolchain/clib] settings*

*#*

*# mandatory setting*

*#*

*# Available values for KERNEL\_VER:*

*#*

*# 3.4.43 : 3.4.43 for EVB*

*# 3.4.43-f : 3.4.43 for FPGA*

*#*

*# Available values for GCC\_VER:*

*#*

*# 4.6.1*

*#*

*# Available values for C\_LIBRARY:*

*#*

*# glibc*

*# uClibc*

*#*

*################################################################################*

*KERNEL\_VER ?= 3.4.43*

*GCC\_VER ?= 4.6.1*

*C\_LIBRARY ?= glibc*

*################################################################################*

*#*

*# path settings*

*#*

*# mandatory setting*

*#*

*# SDK\_TOP:*

*# - Description: specify the absolute path directory where this file reside in.*

*# you must specify the value of this configuration before any*

*# 'make' operation.*

*# - Values : MUST be current path directory.*

*#*

*# KERNEL\_TOP:*

*# - Description: specify the abolute path directory where kernel reside in.*

*# generally you don't need change this manually.*

*# - Values : decided by the $(KERNEL\_VER). Don't change.*

*#*

*# UBOOT\_TOP:*

*# - Description: specify the abolute path directory where u-boot reside in.*

*# generally you don't need change this manually.*

*# - Values : decided by the $(SDK\_TOP). Don't change.*

*#*

*################################################################################*

*SDK\_TOP ?=*

*KERNEL\_TOP ?= $(SDK\_TOP)/linux/kernel/$(KERNEL\_VER)*

*UBOOT\_TOP ?= $(SDK\_TOP)/boot/u-boot*

*################################################################################*

*#*

*# toolchain parent directroy settings (optional)*

*#*

*# This macro can be used to override the default directory where the toolchain*

*# reside in.*

*#*

*# By default the toolchain will be installed into /opt/goke/*

*# Note:*

*# Thisconfig CAN NOT be assigned with $(PWD), you can use $(SDK\_TOP) instead.*

*#*

*################################################################################*

*TOOLCHAIN\_PARENT\_DIRECTORY ?= /opt/goke*

*################################################################################*

*#*

*# rootfs parent directroy settings (optional)*

*#*

*# This macro can be used to override the default directory where the rootfs*

*# reside in.*

*#*

*# By default the rootfs will be installed into /opt/goke/*

*# Note:*

*# Thisconfig CAN NOT be assigned with $(PWD), you can use $(SDK\_TOP) instead.*

*#*

*################################################################################*

*ROOTFS\_PARENT\_DIRECTORY ?= /opt/goke*

*################################################################################*

*#*

*# ISP3A library settings (optional)*

*#*

*# This macro can be used to override the default ISP3A library settings*

*#*

*# Note: if you change the ISP3A\_LIBRARY setting, you need cd into subsystem/isp3a*

*# then execute 'make clean' & 'make' to compile and install the selected lib.*

*# Also you need to rebuild the application(like ctlserver) to make it effect.*

*#*

*# Available values:*

*# goke : using goke ISP3A library for application*

*#*

*################################################################################*

*ISP3A\_LIBRARY ?= goke*

*################################################################################*

*#*

*# Target memory definitions*

*#*

*#*

*# Available values:*

*#*

*# DDR\_MEMORY\_512M config the system with 512M bits memory*

*#*

*# DDR\_MEMORY\_1G config the system with 1G bits memory*

*# Note: on FPGA board, this configuration support only 720P*

*#*

*################################################################################*

*DDR\_MEMORY\_SIZE ?= DDR\_MEMORY\_1G*

*################################################################################*

*#*

*# Install and Release Configuration (optional)*

*#*

*# Here you can specific your private installation and release directories.*

*# The macros $(RELEASE\_MAJOR) and $(RELEASE\_MINOR) can be used to create*

*# release specific sub directories.*

*#*

*################################################################################*

*INSTALL\_DIR ?= $(TOPDIR)/install*

*################################################################################*

*#*

*# Global make arguments (optional)*

*#*

*# Here you can specific your private make arguments to be passed to every*

*# make call, such as:*

*# REPORT = 1 : to enable command line reporting*

*#*

*################################################################################*

*REPORT ?= 0*

*################################################################################*

*#*

*# Development Config (GOKE Internal Only)*

*#*

*################################################################################*

*-include $(SDK\_TOP)/Config.mak.dev*

目前Config.mak文件中需要进行修改配置的选项为如下几项：

***BOARD***

该选项为选择对应的参考硬件，可选的配置如下：

* GK7101\_EVB

***C\_LIBRARY***:

该选项可选的配置如下：

* glibc
* uClibc (可生成更小的文件系统image)

***SDK\_TOP***:

该选项用于指定SDK当前在Linux虚拟机中的所处位置的绝对路径。

***TOOLCHAIN\_PARENT\_DIRECTORY***:

该选项用于指定交叉编译工具链的安装路径，默认为/opt/goke

***ROOTFS\_PARENT\_DIRECTORY***:

该选项用于指定根文件系统安装的路径，默认为/opt/goke

## Linux SDK安装

完成章节3.3 所述的配置后，可在$(SDK\_TOP)目录下运行如下命令进行交叉编译器和根文件系统的安装：

*$ make deploy*

该安装需要sudo权限，在安装过程中需要用户输入sudo密码。

建议用户不要直接使用root账户进行安装。

在安装完成后，进行如下操作（或者重新登录），以使SDK相关环境变量生效：

*$source env/build\_env.sh*

# Linux SDK 编译与运行

## 编译环境

目前Linux SDK存在两种编译环境，基于glibc的编译环境与基于uClibc的编译环境。当前SDK使用何种编译环境取决于Config.mak中的变量C\_LIBRARY，C\_LIBRARY可赋值为glibc或uClibc来设置编译环境。需要注意的是修改C\_LIBRARY后，必须运行*source env/build\_env.sh*

或者重新登录使新的编译环境相关的环境变量生效，以免继续使用老的环境变量造成编译环境冲突。

## 整体编译

在Linux SDK安装完成后，可以进入之前在Config.mak中配置的$SDK\_TOP目录，直接运行make命令，可对整个SDK进行编译，此时，Uboot，内核，驱动及应用都将自动编译。

编译完成后，相关的Demo应用和配置文件会自动安装到$ROOTFS\_TOP定义的根文件系统目录下。

## 内核配置选项

对于GK710X 平台，LinuxSDK 的内核提供了两个默认配置：

gk710x\_evb\_mini\_defconfig

gk710x\_evb\_full\_defconfig

该两个配置对应于GK7101 EVB 1G DDR BOARD。

其中gk7101\_evb\_mini\_defconfig为最小内核配置，gk7101\_evb\_full\_defconfig 在 mini 配置的基础上选择了所有GK710X支持的外设驱动。

对于GK7101的各种RB板，需要在linux/kernel/3.4.43目录下运行make menuconfig进行如下配置：

*进入 [System Type] --> [GK710X board select (GK7101 Evaluation Board)]*

*选择对应的RB板*

*保存内核配置选项*

对于基于GK7102的板，需要在linux/kernel/3.4.43目录下运行make menuconfig进行如下配置：

*进入 [System Type] -->[GK710X chip select(GK7101)]*

*选择GK7102*

*保存内核配置选项并重新编译*

注意：对于GK7102的平台，必须要在$(SDK\_TOP)/Config.mak中把DDR\_MEMORY\_SIZE设置为DDR\_MEMORY\_512M。

## 如何运行

### UBOOT

从$SDK\_TOP/boot/u-boot/中得到u-boot.bin. 通过FlashBurner工具进行加头得到uboot.bin进行烧录。详见《GK710X flash烧写工具用户手册.pdf》或者用烧录器来烧写uboot.bin

进入uboot，配置启动环境:

*\*网络设置\**

*setenvgatewayip 192.168.1.1*

*setenvipaddr 192.168.1.200*

*setenvserverip 192.168.1.11*

*setenvnetmask 255.255.255.0*

*setenvloadaddr0xC1000000*

*\*NFS启动方式\**

*setenvbootargs 'console=ttySGK0 root=/dev/nfsrwnfsroot=192.168.1.11:/opt/goke/rootfs*

*ip=192.168.1.200::192.168.1.1:255.255.255.0:goke:eth0:off init=/linuxrcmem=46mphytype=1'*

*setenvbootcmd 'tftp ${loadaddr} zImage; bootm'*

### 内核镜像

从$SDK\_TOP/linux/kernel/3.4.43/arch/arm/boot/中拷贝zImage到TFTP服务器上

### 文件系统

由于内核默认配置为NFS（详见[5.6.6](#_NFS)）挂载方式，因此配置/opt/goke/rootfs\_glibc(rootfs\_uClibc)为共享路径。(详见[7.1](#_虚拟机上的NFS服务安装与配置))

虚拟机IP地址需要和Uboot中配置的地址一致。这里配置为192.168.1.11。

### 运行ctlserver

目前ctlserver需要依赖freetype-2.4.11，zlib-1.2.7，libstdc++这3个开源库，因此需要我们在software目录下手动添加这3个库到rootfs中，操作如下:

*$make add-freetype-2.4.11*

*$make add-zlib-1.2.7*

*$make add-libstdc++*

打开电源，内核启动并挂载NFS服务器，在串口看到登陆提示，可以通过 root 登陆。

执行如下脚本运行ctlserver.

*$run\_app.sh –sensor imx222*（如果sensor是ov2710, 则红色部分替换为 ov2710）

PC端通过Internet Explorer访问<http://192.168.1.200>

通过访问IP Camera端的webserver ，进入主页面获取帮助信息。

# Linux 根文件系统

## 介绍

Linux系统启动时第一个必须挂载的文件系统就是根文件系统，它包含Linux启动时所必须的目录和文件。

为方便用户生成制作和定制化根文件系统，GK7101 Linux SDK内置了一系列的工具、预编译组件和命令：

**根文件系统组件**

为了更加方便和灵活的使用根文件系统，将根文件系统拆分为一系列组件。部分组件是文件系统必需的，如：busybox，这类组件是必需的，而大部分组件都是一些开源的库或者工具，如：gdb、zlib，这类组件可以选择安装。

**文件系统制作工具**

用于将根文件系统制作成各种类型的文件系统，如: ubi，jffs2。

**预编译库**

程序的编译时会使用其中的头文件和依赖库。

**文件系统配置文件**

根文件系统安装，根文件系统组件默认设置（根文件系统安装时会安装默认支持的组件），文件系统制作以及其他配置所依赖的文件。

**文件系统模块命令**

安装根文件系统和制作文件系统的命令，全部的命令和其功能如下：

* make deploy-rootfs

安装根文件系统

* make rootfs-list

列出所有的可选择安装的根文件系统组件的全名

* make roots-info

查看当前安装的根文件系统组件

* make add-组件全名

安装指定的根文件系统组件（同时安装其依赖组件）

* make remove-组件全名

卸载指定的根文件系统组件（删除安装组件时产生的文件，不会删除安装组件时产生的目录，也不会删除其依赖库，根据提示手动删除目录）

* make roots-image

制作文件系统

## 根文件系统配置

通过文件系统配置文件rootfs.mak（software/linux/rootfs/rootfs.mak）可以选择性安装根文件系统所需要的组件。

例如：

############################################################################

# --gdb

############################################################################

GDB\_7\_5\_1\_SUPPORT := 0

GDB\_7\_5\_1\_DEPEND := ZLIB\_1\_2\_7\_SUPPORT NCURSES\_5\_9\_SUPPORT

如果需要安装gdb，则做如下设置：

############################################################################

# --gdb

############################################################################

GDB\_7\_5\_1\_SUPPORT := 1

GDB\_7\_5\_1\_DEPEND := ZLIB\_1\_2\_7\_SUPPORT NCURSES\_5\_9\_SUPPORT

这样在安装根文件系统的时候就会将gdb以及它所依赖的zlib，ncurses组件一并装入新建的根文件系统。

注意：配置文件中已经默认选择的组件一般情况下是必需并能满足整个方案的需求，如果没有特别需求可以直接使用组件的默认配置。

## 根文件系统安装

配置好根文件系统后（见5.2），在software目录下面执行根文件系统安装命令：

*$ make deploy-rootfs*

根文件系统rootfs\_glibc(rootfs\_uClibc)默认安装路径为/opt/goke/

## 组件的安装与卸载

在文件系统已经生成的情况下，若开发过程中需要安装或卸载根文件系统组件可以使用命令来完成，命令需要用到组件的全名，执行命令make rootfs-list会列出所有可配置组件的全名，以gdb的安装与卸载为例：

安装gdb命令如下：

*$ make add-gdb-7.5.1*

运行此命令后可从终端看到如下的输出：

*-->install zlib-1.2.7*

*-->install ncurses-5.9*

*-->install gdb-7.5.1*

注意：安装gdb时会安装其依赖组件zilb和ncurses。

卸载gdb命令如下：

*$ make remove-gdb-7.5.1*

运行此命令后可从终端看到如下的输出：

-->uninstall gdb-7.5.1

注意：

* 卸载时不会卸载其依赖组件
* 卸载时只会删除组件中的文件而不会删除组件中的目录，需要根据提示手动删除残余目录，如删除freetype时会有如下提示：

*$ make remove-freetype-2.4.11*

*-->uninstall freetype-2.4.11...*

*The following may be legacy directory:*

*/opt/goke/rootfs/usr/share/fonts*

## 自定义的根文件系统组件

在开发过程中有可能需要用到新的开源库或工具，这时可以选择将其制作成根文件系统组件，以gdb为例：

### 打包与命名

**打包**：将编译好的gdb按其它组件的方式进行打包，压缩类型为bzip2。

**命名**：以(.tar.bz2)结尾，符合只能包含“.”、“-”、“\_”。

这里将gdb组件命名为gdb-7.5.1.tar.bz2，然后放入目录software/linux/rootfs/arm-goke-linux-gnueabi（依赖的C库为glibc）或software/linux/rootfs/arm-goke-linux-uclibcgnueabi（依赖的C库为uClibc）。

### 编写配置文件

变量命名：将组件名中的小写字母转为大写字母，符合都由下划线“\_”替代，加入后缀\_SUPPORT表示是否支持此组件，加入后缀\_DEPEND表示此组件依赖的组件。

gdb组件依赖组件有zlib（zlib-1.2.7）与ncurses（ncurses-5.9），需要在配置文件rootfs.mak中写入如下内容：

############################################################################

# --gdb

############################################################################

GDB\_7\_5\_1\_SUPPORT := 0

GDB\_7\_5\_1\_DEPEND := ZLIB\_1\_2\_7\_SUPPORT NCURSES\_5\_9\_SUPPORT

## 文件系统类型介绍

### 闪存的特性与限制

为了更好的理解与选择合适的文件系统，这里针对NOR flash 和 NAND flash这两种类型的闪存进行简单的介绍。

NOR flash特性与限制：

* 读/写操作的基本单位是字节
* 写操作前一般需要进行擦除操作，擦除的最小单位是sector
* 每一位(bit)可以被写操作置成逻辑0，擦写所完成的功能就是把擦写块内的每一位都置成逻辑 1。
* 读速度比nand flash快一些，写速度比nand flash慢很多
* 擦写块的最大可擦写次数在十万次左右

NAND flash特性与限制：

* 读/写操作的基本单位是page
* 写操作前一般需要进行擦除操作，擦除的最小单位是block
* 每一位(bit)可以被写操作置成逻辑0，擦写所完成的功能就是把擦写块内的每一位都置成逻辑 1。
* 读速度比nor flash慢一些，写速度比nor flash快很多，擦除速度远比nor flash快
* 擦写块的最大可擦写次数在百万次左右
* 生产和使用的过程中都有可能出现坏块
* page分为main area和spare area，main area用于存放数据，spare area用于存放坏块标记、ECC值、芯片信息、文件信息。

### JFFS2

#### JFFS2文件系统简介

JFFS2 是日志结构(log-structured)的可读写文件系统，它会直接在MTD设备上实现日志结构，将文件系统数据和原数据(meta-data)以节点的形式在闪存上顺序存储。JFFS2在闪存上使用非常广泛，普遍应用在嵌入式系统中。

优点：

* 可读写
* 数据压缩
* 掉电保护
* 损耗均衡wear-leveling

缺点：

* JFFS2在挂载（mount）时会扫描整个flash 分区的内容来建立文件结构,会消耗很多时间，并且分区增大挂载时间会变长。
* 由于垃圾回收的原因，文件系统在已满或接近满时运行速度大大放慢
* 由于采用概率的方法解决损耗均衡，较随意，很难损耗均衡的确定性。

#### JFFS2文件系统制作

* **方式一（推荐）**

详见[5.7](#_制作根文件系统Image)

* **方式二**

**Nandflash：**

*$ ./mkfs.jffs2 -p -n -l -e 0x20000 -r . /rottfs -o jffs2.img*

**Norflash：**

*$ ./mkfs.jffs2 -p -l -e 0x20000 -r ./rottfs -o jffs2.img*

-l, 创建小端（little-endian）文件系统。

-p, 指定文件系统的大小并用0xFF填充，可以使用要使用的MTD分区的大小，

如果没有指定大小，则只填充最后一个block。

-e, 擦除块（eraseblock）的大小

-r, 指定所要建立文件系统的目录

-o, 指定文件系统映像

-n, 不在每个擦除块中添加擦除标记（cleanmarker）

注意：nand flash的cleanmarker位置在nand flash的OOB area，制作nandflash的jffs2 image时加上-n以避免在每个擦除块前面加上擦除标记。

#### JFFS2文件系统烧写

* **方式一（推荐）**

以下操作在开发板NFS根文件系统下完成

**Nandflash：**

*$ flash\_erase-j /dev/mtdx 0 0*

*$nandwrite-p /dev/mtdx/jffs2.img*

**Norflash：**

*$ flash\_erase/dev/mtdx 0 0*

*$flashcp-v /jffs2.img /dev/mtdx*

* **方式二**

详见[6.2.2](#_把根文件系统image写入Flash)

#### JFFS2内核配置

File systems

---> Miscellaneous filesystems

<\*>Journalling Flash file system v2 (JFFS2) support

#### JFFS2内核启动参数

console=ttySGK0,115200 root=/dev/mtdblockxrootfstype=jffs2 init=/linuxrcmem=46m

### SquashFS

#### SquashFS文件系统简介

SquashFS是基于Linux使用的高度压缩的只读文件系统，适用于存储介质有限的嵌入式系统。

优点：

* 高压缩率，数据（data）,节点（inode）,目录（directories）都被压缩，SquashFS文件系统块的默认大小为128K，最大支持1M。
* 低开销，节点占用的空间非常小，每个块（blocks）的开销降到最低。
* 存储了全部的uid/gids（32bits）和文件创建时间。
* 理论上文件和文件系统都能支持到2^64 bytes。
* 响应速度快

缺点：

* 由于不支持写，常常不能单独满足用户需求。

#### SquashFS文件系统制作

* **方式一（推荐）**

详见[5.7](#_制作根文件系统Image)

* **方式二**

*$ ./mksquashfs./rootfssquashfs.img-b 512K*

-b, 指定块大小，默认大小为128K

-comp, 指定压缩格式，默认是gzip

注意：1，增大文件系统块的大小可以增大压缩率，但也会增大一些内存的开销

2，使用xz（-comp）压缩可以增大压缩率，但会增大一些内存的开销以及降低文件系统响应速度

#### SquashFS文件系统烧写

* **方式一（推荐）**

以下操作在开发板NFS根文件系统下完成

*$ flash\_erase/dev/mtdx 0 0*

*$flashcp-v /squashfs.img/dev/mtdx*

* **方式二**

详见[6.2.2](#_把根文件系统image写入Flash)

#### SquashFS内核配置

File systems

---> Miscellaneous filesystems

<\*>SquashFS 4.0 – Squashed file system support

……

[\*]Include support for XZ compressed file systems

……

注意：[]Include support for xxx compressed file systems是选择支持的压缩格式，

选上制作squashfs文件系统使用的压缩格式即可

#### SquashFS内核启动参数

console=ttySGK0,115200 root=/dev/mtdblockxrootfstype=squashfsinit=/linuxrcmem=46m

### YAFFS2

#### YAFFS2文件系统简介

YAFFS2是为NAND Flash 设计的嵌入式文件系统。

优点：

* 可读写
* 运行速度较jffs2快
* 采用日志结构，供掉电保护和损耗均衡功能
* 使用了NAND Flash oob(Out Of Band) area来存储文件信息及ECC信息，挂载时只会去扫描oob area，因此挂载速度非常快
* 内存开销低

缺点：

* 没有采用压缩格式，YAFFS2文件系统会很大

#### YAFFS2文件系统制作

* **方式一（推荐）**

详见[5.7](#_制作根文件系统Image)

* **方式二**

*$ ./mkyaffs2image./rootfs yaffs2.img ECC\_TYPE*

ECC\_TYPE,指NAND Flash ECC layout类型

#### YAFFS2文件系统烧写

* **方式一（推荐）**

以下操作在开发板NFS根文件系统下完成

*$ flash\_erase/dev/mtdx 0 0*

*$flashcp-v /yaffs2.img/dev/mtdx*

* **方式二**

详见[6.2.2](#_把根文件系统image写入Flash)

#### YAFFS2内核配置

File systems

---> Miscellaneous filesystems

<\*>yaffs2 file system support

#### YAFFS2内核启动参数

console=ttySGK0,115200 root=/dev/mtdblockxrootfstype=yaffs2 init=/linuxrcmem=46m

### UBIFS

#### UBIFS文件系统简介

UBIFS是Nokia工程师在塞格德大学帮助下开发的新的Flash 文件系统。UBIFS可以认为是下一代的JFFS2文件系统。但是JFFS2是运行在MTD设备之上，而UBIFS只能工作在UBI volume之上。UBIFS涉及了以下三个子系统：

1. MTD 子系统，为flash芯片提供对了统一的访问接口， MTD子系统提供了MTD devices(如: /dev/mtd0)的概念，MTD devices基本上可被认为是原始的 flash。

2. UBI subsystem，为flash设备提供了损耗均衡和卷（volume）管理；UBI工作在MTD设备之上，提供了UBI volume的概念；UBI volume建立在MTD devices之上并且为上层的文件系统解决了许多MTD devices存在的问题（如：损耗和坏块管理）。

3. UBIFS文件系统，工作于UBIvolume之上

优点：

* 可扩展性，对于flash设备的大小有着很好的扩展性，可以很好的运行在GB级的flashe设备;
* 挂载速度快，UBIFS在mount时不需要扫描整个文件系统，挂载时间只是毫秒级且不依赖于flash的大小，但是UBI的初始化时间却依赖flash的大小
* 快速I/O，支持write-back,这减少了I/O的数目并获得很好的性能，很大程度的提高了文件体系的吞吐量
* 数据压缩
* 损耗均衡
* 可读写
* 日志文件系统，容忍突然崩溃和unclean重启

缺点：

* 在突然掉电的情况下，不能提供JFFS2那样可靠的保证

#### UBIFS文件系统制作

* **方式一（推荐）**

详见[5.7](#_制作根文件系统Image)

* **方式二**

Ubifs:

*$ mkfs.ubifs -r ./rootfs -e 0XFF80-m 0x1 -c 2048-o ubifs.img*

-e, 逻辑擦除块的大小

-m, 最小的I/O单元的大小

-c, 最大的逻辑擦除块的数量

-r, 指定所要建立文件系统的目录

-o, 指定文件系统映像

ubi:

$ viubinize-rootfs.cfg

[ubifs]

mode=ubi

image=ubifs.img

vol\_id=0

vol\_type=dynamic

vol\_name=rootfs

vol\_flags=autoresize

vol\_alignment=1

注意：image需要指定为Ubifs的名字，这里没有指定vol\_size，因此制作ubi.img时volume会定为ubifs.img的大小，但挂载ubifs时，ubi volume会根据分区的大小做出相应的调整。

*$ubinize-o ubi.img-p 0x10000 -m 0x1-s 0x1 ubinize-rootfs.cfg*

-p, 物理擦除块的大小

-m, 最小的I/O单元的大小

-o, 指定文件系统映像

-s, 默认与最小的I/O单元的大小相同（若nand flash支持sub page，可指定为sub page的大小）

#### UBIFS文件系统烧写

* **方式一（推荐）**

以下操作在开发板NFS根文件系统下完成

*$flash\_erase/dev/mtdx 0 0*

*$ubiformat/dev/mtdx-fubi.img*

* **方式二**

详见[6.2.2](#_把根文件系统image写入Flash)

注意：请使用ubi.img烧写到flash，ubifs.img不能直接烧写到flash

#### UBIFS内核配置

Device Drivers --->

<\*> Memory Technology Device (MTD) support --->

<\*> Enable UBI - Unsorted block images --->

File systems --->

[\*] Miscellaneous filesystems --->

<\*> UBIFS file system support

#### UBIFS内核启动参数

console=ttySGK0,115200ubi.mtd=3 root=ubi0:rootfs rootfstype=ubifsrwinit=/linuxrcmem=46m

注意：ubi.mtd=3 root=ubi0:rootfs表示将mtd3(attach as ubi0)作为根文件系统所在分区挂载

### NFS

#### NFS文件系统简介

NFS（Network File System，网络文件系统）是一种分布式的文件系统。用户可以挂载远端的文件系统（如linux服务器下的文件系统目录）到本地文件系统的目录上（或直接挂载为本地根文件系统），这样用户就可以通过访问挂载到本地的目录结构来访问远端服务器的目录结构。

嵌入式系统使用NFS作为初期开发和调试的文件系统，避免了每次修改都需要把文件系统映像重新烧入flash得操作，大量的节省了时间。

#### NFS内核配置

File systems --->

[\*] Network File Systems --->

<\*> NFS client support

[\*] NFS client support for NFS version 3

……

[\*] NFS client support for NFS version 4

……

[\*] Root file system on NFS

#### NFS远端服务器配置

详见[7.1](#_虚拟机上的NFS服务安装与配置)

#### NFS内核启动参数

console=ttySGK0 root=/dev/nfsrwnfsroot=192.168.0.100:/rootfsip=192.168.0.102::192.168.0.1:255.255.0.0:goke:eth0:offinit=/linuxrcmem=46m

root=/dev/nfs nfsroot=192.168.1.151:/opt/goke/rootfs\_glibc,proto=tcp,nfsvers=3,nolock ip=192.168.1.200:192.168.1.151:192.168.1.1:255.255.255.0:"gk8602":eth0 mac=3C:97:0E:22:E1:14 phytype=0 sensor=OV2710 wifi=RTL8188 console=ttySGK0,115200 mem=56m video=gk\_fb0:720x480,720x480,8,0 . product=WIFI\_DVR rtc=0

## 制作根文件系统映像

文件系统是在存储设备上组织和存放文件的方法。文件系统模块支持的文件系统有UBIFS、SquashFS、YAFFS2、JFFS2。

### 文件系统配置

通过文件系统配置文件rootfs.mak可以设置需要制作的文件系统，配置方法如下：

NAND\_FLASH := nand flash型号（如：HY27UF082G2A）

FILESYSTEM\_TYPE := 文件系统类型（如：ubifs）

### 文件系统制作

配置好文件系统后（见5.6.2），在Linux SDK顶层目录下面执行如下命令：

*$ make roots-image*

该命令执行完成后，生成的文件系统存放在$SDK\_TOP/linux/rootfs目录下。

注意：若制作的为UBIFS，请使用ubi.img烧写到flash，ubifs.img不能直接烧写到flash

# Flash分区与烧写

## Flash分区

GK7101 RB 板上为16M Nor Flash，我们默认的内核配置把该 Flash 分为4个分区：

|  |  |  |
| --- | --- | --- |
| MTD Partition | Flash Offset | Size(Bytes) |
| uboot | 0x0000 0000 | 0x0004 0000 |
| ubootenv | 0x0004 0000 | 0x0001 0000 |
| kernel | 0x0005 0000 | 0x0020 0000 |
| rootfs | 0x0025 0000 | 0x00DB 0000 |

以下的Flash烧写过程我们以该配置为例。

## Flash 烧写

我们可以使用u-boot中的tftp命令和sf命令来进行内核和根文件系统image的烧写。

其中tftp命令用于从服务器上把内核image和文件系统image下载到板端内存中，sf命令用于对SFlash的烧写。

### 把内核zImage写入Flash

把交叉编译服务器上编译好的内核zImage拷贝到TFTP Server目录下之后，我们就可以在u-boot中按如下的操作把内核zImage写入Flash的指定位置：

*GK7101#tftp 0xC1000000 zImage*

*GK7101#sf probe 0 0*

*GK7101#sf erase 0x00050000 0x200000*

*GK7101#sf write 0xC1000000 0x00050000 0x200000*

### 把根文件系统映像写入Flash

把文件系统映像（以ubi.img为例）拷贝到TFTP Server目录下之后，我们就可以在u-boot中按如下的操作把根文件系统映像写入文件系统分区：

*GK7101#tftp 0xC1000000 ubi.img*

*GK7101#sf probe 0 0*

*GK7101#sferase0x2500000xdb0000*

*GK7101#sf write 0xC1000000 0x2500000xdb0000*

注意：

* 烧写前请擦除整个文件系统分区
* 请使用ubi.img烧写到flash，ubifs.img不能直接烧写到flash

### 启动内核

以UBIFS为例，内核自动启动挂载UBIFS根文件系统，需要对u-boot的环境变量进行如下设置：

*\*SFLASH UBI启动方式\**

*使用Flash分区中的内核镜像进行启动*

*setenvbootcmd'sfprobe;sf read 0xc1000000 0x50000 0x200000; bootm'*

*使用Flash分区中的UBI系统作为根文件系统(其中IP地址和phytype需根据实际情况进行修改)*

*setenvbootargs 'console=ttySGK0,115200 noinitrdmem=49m rwubi.mtd=3 rootfstype=ubifs root=ubi0:rootfs init=/linuxrcip=192.168.100.101::192.168.100.1:255.255.255.0:goke:eth0:off phytype=1'*

启动参数参考（仅提供文件系统相关参数作为参考，其他参数根据实际情况添加和修改）：

[JFFS2内核启动参数](#_JFFS2内核启动参数)

[SquashFS内核启动参数](#_SquashFS内核启动参数)

[YAFFS2内核启动参数](#_YAFFS2内核启动参数)

[UBIFS内核启动参数](#_UBIFS内核启动参数)

[NFS内核启动参数](#_NFS内核启动参数)

# 附录

## 虚拟机上的NFS服务安装与配置

### 安装Ubuntu NFS

　Ubuntu上默认是没有安装NFS服务器的，因此我们首先安装NFS服务器端：

*$sudo apt-get installnfs-kernel-server*

### 配置/etc/exports

　Ubuntu NFS允许挂载的目录及权限在文件**/etc/exports**中进行了定义。

*$sudovim /etc/exports* 在exports文件中添加红色部分:

*# /etc/exports: the access control list for filesystems which may be exported*

*# to NFS clients. See exports(5).*

*#*

*# Example for NFSv2 and NFSv3:*

*# /srv/homes hostname1(rw,sync,no\_subtree\_check) hostname2(ro,sync,no\_subtree\_check)*

*#*

*# Example for NFSv4:*

*# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt,no\_subtree\_check)*

*# /srv/nfs4/homes gss/krb5i(rw,sync,no\_subtree\_check)*

*/opt/goke/rootfs\_glibc \*(rw,sync,no\_root\_squash,no\_subtree\_check)*

*/opt/goke/rootfs\_uClibc \*(rw,sync,no\_root\_squash,no\_subtree\_check)*

### 启动NFS服务

*$sudo /etc/init.d/portmap restart*

*$sudo /etc/init.d/nfs-kernel-server restart*

### 测试NFS服务

　此时可以运行以下命令来显示一下共享出来的目录：

*$showmount -e*

## 如何基于SDK添加自己的硬件配置

比如需要添加对 GK7102\_MYBD\_SC1035\_V1硬件的支持, 方法如下:

### uboot部分修改

打开 boot/u-boot/include/configs/gk7101.h

// note: add a new board

// 1. add a gpio map file at arch\arm\include\asm\arch-gk7101

// 2. add a board define at include\configs\gk7101.h

/\*

\* GK7101 board

\*/

[ADD]

#define CONFIG\_MACH\_GK7102\_MYBD\_SC1035\_V1

/\*

\* GK7101 board specific data

\*/

[ADD]

#ifdef CONFIG\_MACH\_GK7102\_MYBD\_SC1035\_V1

#define CONFIG\_IDENT\_STRING "For MYBDGK7102SC1035v1.00 "

#define CONFIG\_PHY\_USE\_AO\_MCLK

#define CONFIG\_CPU\_GK7102

#endif

进入boot/u-boot/arch/arm/include/asm/arch-gk7101/ 目录

创建gpio\_gk7102\_mybd\_sc1035\_v1.h文件，配置硬件相关的GPIO。

打开gpio.h文件,

[ADD]

#if defined(CONFIG\_MACH\_GK7102\_MYBD\_SC1035\_V1)

#include "gpio\_gk7102\_mybd\_sc1035\_v1.h"

#endif

### kernel部分修改

打开linux/kernel/3.4.43/arch/arm/mach-gk7101/Kconfig

choice

prompt "GK710X board select"

default MACH\_GK7101\_EVB

help

Select board base on gk7101 Soc

config MACH\_GK7101\_EVB

depends on MACH\_GK7101

bool "GK7101 Evaluation Board"

help

Select Goke GK7101 Evaluation Board

[ADD]

configMACH\_GK7102\_MYBD\_SC1035\_V1

depends on MACH\_GK7102

bool "GK7102My Board"

help

Select My GK7102 + SC1035 Board

进入linux/kernel/3.4.43/arch/arm/mach-gk7101/include/mach/ 目录

创建gpio\_gk7102\_mybd\_sc1035\_v1.h文件，配置硬件相关的GPIO。

打开gpio.h文件,

#if defined(CONFIG\_MACH\_GK7102\_RB\_BG0703\_V1\_00)

#include "gpio\_rb\_bg0703.h"

#endif

[ADD]

#if defined(CONFIG\_MACH\_GK7102\_MYBD\_SC1035\_V1)

#include "gpio\_gk7102\_mybd\_sc1035\_v1.h "

#endif

打开linux/kernel/3.4.43/arch/arm/mach-gk7101/mach-gk7101.c

#if defined(CONFIG\_MACH\_GK7102\_RB\_BG0703\_V1\_00)

MACHINE\_START(GK7102\_RB\_BG0703\_V1\_00, "Goke GK7102 RB\_BG0703 board V1.00")

#endif

[ADD]

#if defined(CONFIG\_MACH\_GK7102\_MYBD\_SC1035\_V1)

MACHINE\_START(GK7102\_MYBD\_SC1035\_V1,”My GK7102\_SC1035 V1 Board”)

#endif

打开linux/kernel/3.4.43/arch/arm/tools/mach-types

gk7101\_fpga MACH\_GK7101\_FPGA GK7101\_FPGA 3988

gk7101\_evb MACH\_GK7101\_EVB GK7101\_EVB 3988

gk7101\_rb MACH\_GK7101\_RB\_V1\_00 GK7101\_RB\_V1\_00 3988

gk7101\_rb MACH\_GK7101\_RB\_V1\_10 GK7101\_RB\_V1\_10 3988

gk7101\_rb\_imx222 MACH\_GK7101\_RB\_IMX222\_V1\_00 GK7101\_RB\_IMX222\_V1\_00 3988

gk7101\_rb\_ov2710 MACH\_GK7101\_RB\_OV2710\_V1\_00 GK7101\_RB\_OV2710\_V1\_00 3988

gk7101\_rb\_ov2710 MACH\_GK7101\_RB\_OV2710\_V2\_00 GK7101\_RB\_OV2710\_V2\_00 3988

gk7101\_rb\_ov2710 MACH\_GK7101\_RB\_OV2710\_V2\_10 GK7101\_RB\_OV2710\_V2\_10 3988

gk7101\_rb\_ps1210 MACH\_GK7101\_RB\_PS1210\_V1\_00 GK7101\_RB\_PS1210\_V1\_00 3988

gk7101\_rb\_ar0230 MACH\_GK7101\_RB\_AR0230\_V1\_00 GK7101\_RB\_AR0230\_V1\_00 3988

gk7102\_rb\_imx225 MACH\_GK7102\_RB\_IMX225\_V1\_00 GK7102\_RB\_IMX225\_V1\_00 3988

gk7102\_rb\_ov9712 MACH\_GK7102\_RB\_OV9712\_V1\_00 GK7102\_RB\_OV9712\_V1\_00 3988

gk7102\_rb\_ov9750 MACH\_GK7102\_RB\_OV9750\_V1\_00 GK7102\_RB\_OV9750\_V1\_00 3988

gk7102\_rb\_ar0130 MACH\_GK7102\_RB\_AR0130\_V1\_00 GK7102\_RB\_AR0130\_V1\_00 3988

gk7102\_rb\_jxh42 MACH\_GK7102\_RB\_JXH42\_V1\_00 GK7102\_RB\_JXH42\_V1\_00 3988

gk7102\_rb\_bg0703 MACH\_GK7102\_RB\_BG0703\_V1\_00 GK7102\_RB\_BG0703\_V1\_00 3988

[ADD]

gk7102\_mybd\_sc1035\_v1 MACH\_GK7102\_MYBD\_SC1035\_V1GK7102\_MYBD\_SC1035\_V13988