mini2440.git是该项目是qemu.git项目的一个分支。

描述：三星s3c2440和arm920t两款芯片支持mini2440开发板

主页：<http://code.google.com/p/mini2440/>

作者联系方式： [buserror@gmail.com](mailto:buserror@gmail.com)

URL：

git://repo.or.cz/qemu/mini2440.git

<http://repo.or.cz/qemu/mini2440.git>

Push URL：

ssh://repo.or.cz/qemu/mini2440.git

<https://repo.or.cz/qemu/mini2440.git>

关键词：arm920t dm9000 micro2440 mini2440 qemu s3c2410 s3c2440 s4c2410

# 0 主机环境信息

|  |
| --- |
| Linux version 4.10.0-42-generic (buildd@lgw01-amd64-051) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.5) ) #46~16.04.1-Ubuntu SMP Mon Dec 4 15:57:59 UTC 2017 |

# 1. 交叉编译工具

安装：

sudo apt-get install gcc-arm-linux-gnueabi

这样安装后，交叉编译工具是最新版的，在编译较低内核版本时，发现遇到问题，所以，这儿可以尽量安装较低版本的交叉编译工具，比如4.7。当然，如果已经安装了最新版的交叉编译工具，可以按照后面一节，把版本降低即可解决。

# 2 qemu-mini2440

## 2.1下载qemu for mini2440

git clone git://repo.or.cz/qemu/mini2440.git qemu

如果感觉速度慢，直接打包下载

http://repo.or.cz/w/qemu/mini2440.git/snapshot/HEAD.tar.gz

## 2.2 编译qemu for mini2440

解压后，进入源代码的根目录中，

# ./configure --target-list=arm-softmmu --prefix=/usr/local

# make -j4

## 2.3 编译qemu for mini2440时，遇到的问题

（1）undefined reference to symbol 'timer\_settime@@GLIBC\_2.2

错误内容：

/usr/bin/ld: ../qemu-timer.o: undefined reference to symbol 'timer\_settime@@GLIBC\_2.2'

/usr/bin/ld: note: 'timer\_settime@@GLIBC\_2.2' is defined in DSO /lib/i386-linux-gnu/librt.so.1 so try adding it to the linker command line

/lib/i386-linux-gnu/librt.so.1: could not read symbols: 无效的操作

collect2: 错误： ld 返回 1

make[1]: \*\*\* [qemu-system-arm] 错误 1

make: \*\*\* [subdir-arm-softmmu] 错误 2

搜了半天，后来在一个论坛里找到一个方法，修复了这个问题，但是我目前还没弄明白问题的所在，先把解决方案放这儿备用吧。

论坛的地址是：http://forum.gns3.net/topic6494.html

但是怕地址失效，就把关键的地方简介的说下：

主要是里面提供了一个patch，给configure文件打完补丁之后就解决了，patch里面的内容是：

|  |
| --- |
| --- configure.orig 2013-06-13 10:56:41.516276000 +0100  +++ configure 2013-06-13 11:20:26.010368566 +0100  @@ -1362,7 +1362,7 @@  cat > $TMPC <<EOF  #include  #include  -int main(void) { clockid\_t id; return clock\_gettime(id, NULL); }  +int main(void) { struct itimerspec v; timer\_t t; return timer\_gettime (t, &v); }  EOF    rt=no |

可以直接把上面的内容拷贝到自己创建的空文件中，文件名命名为configure.patch，但是我觉得其他的名字应该也可以。或者直接下载configure.patch文件，下载地址是：<http://forum.gns3.net/download/file.php?id=1990&sid=2a58777d1298f176a295e8f7baacd630>。然后执行下面的指令

patch configure configure.patch

之后重新configure后再make就没有问题了。

（2）fatal error: SDL.h: 没有那个文件或目录

错误内容：

|  |
| --- |
| The error log from compiling the libSDL test is:  /tmp/qemu-conf--3170-.c:1:17: fatal error: SDL.h: 没有那个文件或目录  compilation terminated.  ERROR: QEMU requires SDL or Cocoa for graphical output  To build QEMU without graphical output configure with --disable-gfx-check  Note that this will disable all output from the virtual graphics card  except through VNC or curses. |

那么就可以使用以下语句修复：

sudo apt-get install libsdl-dev

再使用 make -j4 时， 即可编译成功。

# 3. u-boot for mini2440

## 3.1下载u-boot for mini2440

git clone git://repo.or.cz/w/u-boot-openmoko/mini2440.git uboot

或者打包下载

http://repo.or.cz/w/u-boot-openmoko/mini2440.git/snapshot/HEAD.tar.gz

## 3.2 编译u-boot for mini2440

（注意 采用打包下载的时候这几个包的文件名可能相同，注意区分）解压后，配置Makefile文件，打开Makefile文件，CROSS\_COMPILE变量赋值，即自己所使用的交叉编译工具链，比如我的是arm -linux-gnueabi-，保存退出，输入

# make mini2440\_config

# make -j4

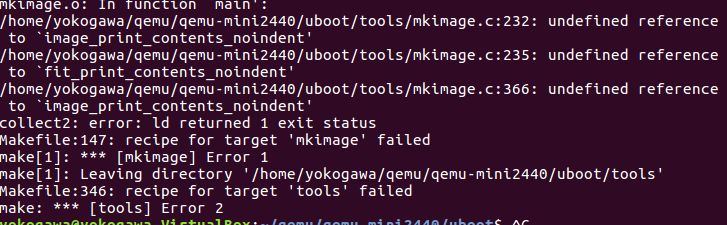
稍等两分钟，即在当前目录下生成名为 u-boot.bin 的文件，注意如果想在之后使用u-boot 的nfs下载文件功能，需要修改代码中的一部分，将net/nfs.c文件中的

NFS\_TIMEOUT = 2UL 修改为 NFS\_TIMEOUT = 20000UL 否则会造成nfs文件下载失败，如果不使用nfs下载功能，不改也可。

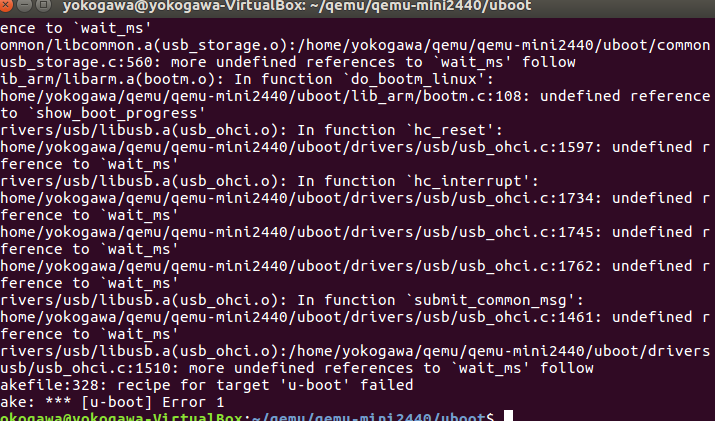
## 3.3 编译uboot时遇到的错误

1.mkimage编译错误，看下面两幅图片：

undefined reference to `image\_print\_contents\_noindent`



undefined reference to `show\_boot\_progress`:

发现上述出现错误的地方都是inline函数，而且声明和定义在不同的文件，且前面都有inline关键字。觉得问题可能出在inline关键字的使用上，于是上网搜索资料后，发现一篇文章，分析的挺好，《[C语言inline详细讲解](http://www.cnblogs.com/xkfz007/articles/2370640.html)》其中一段话，是这样说的：

本文介绍了GCC和C99标准中inline使用上的不同之处。inline属性在使用的时候，要注意以下两点：**inline关键字在GCC参考文档中仅有对其使用在函数定义（Definition）上的描述，而没有提到其是否能用于函数声明（Declare）**。

**从 inline的作用来看，其放置于函数声明中应当也是毫无作用的：inline只会影响函数在translation unit（可以简单理解为C源码文件）内的编译行为，只要超出了这个范围inline属性就没有任何作用了。所以inline关键字不应该出现在函数声明 中，没有任何作用不说，有时还可能造成编译错误（在包含了sys/compiler.h的情况下，声明中出现inline关键字的部分通常无法编译通 过）；**

**inline关键字仅仅是建议编译器做内联展开处理，而不是强制。在gcc编译器中，如果编译优化设置为-O0，即使是inline函数也不会被内联展开，除非设置了强制内联（\_\_attribute\_\_((always\_inline))）属性。**

所以，将上面出错的地方的定义处，把inline关键字去掉，然后重新编译，通过。

# 4. linux kernel for mini2440

## 4.1 下载linux kernel for mini2440

git clone git://repo.or.cz/linux-2.6/mini2440.git kernel

## 4.2 编译linux kernel for mini2440

$ make ARCH=arm mini2440\_defconfig

$ make -j4 ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- uImage

## 4.3 编译linux kernel遇到的问题

问题1：fatal error: linux/compiler-gcc5.h: No such file or directory

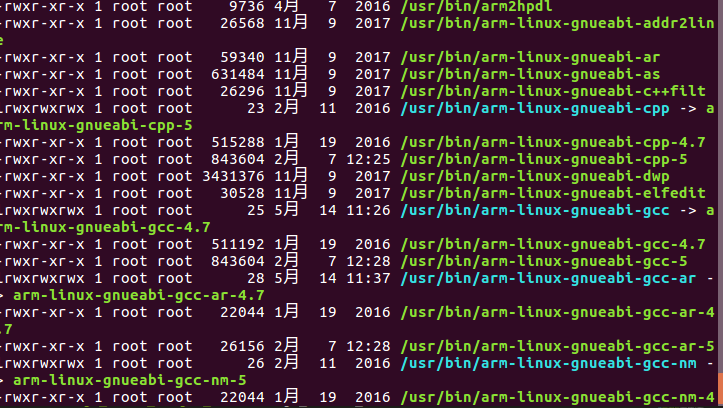
原因：

应该是使用的arm交叉编译工具中需要这个头文件，因为已经安装的arm交叉编译工具版本较高，但是linux内核源代码在编写的时候还没有这个头文件。据此推断，需要降低交叉编译工具的版本。

方法：

|  |
| --- |
| sudo apt install gcc-4.7-arm-linux-gnueabi |

执行完命令后，我们查看主机/usr/bin目录下关于交叉编译工具的信息如下图所示：



我们从图中可以看到，我们实际使用的arm-linux-gnueabi-gcc等工具就是一个符号链接文件，实际指向对应版本的文件，比如，我现在的主机上：

|  |
| --- |
| /usr/bin/arm-linux-gnueabi-gcc -> arm-linux-gnueabi-gcc-4.7 |

出现上面的问题时，我的arm-linux-gnueabi-gcc实际指向的是

/usr/bin/arm-linux-gnueabi-gcc-5，

所以在这里，删除原有的链接文件，新建降低版本后的链接文件，命令如下：

ln -s src dest

比如：

ln -s arm-linux-gnueabi-gcc-4.7 arm-linux-gnueabi-gcc

这样之后，执行：

arm-linux-gnueabi-gcc –version

之后，就可以看到版本信息：

|  |
| --- |
| arm-linux-gnueabi-gcc (Ubuntu/Linaro 4.7.4-3ubuntu12) 4.7.4  Copyright (C) 2012 Free Software Foundation, Inc.  This is free software; see the source for copying conditions. There is NO  warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. |

结果：重新编译内核源代码就可以了。

问题2：在Ubuntu 64位环境下编译Liunx内核出现 Can't use 'defined(@array)' 错误的解决办法

内容：

Can't use 'defined(@array)' (Maybe you should just omit the defined()?) at kernel/timeconst.pl line 373.

原因：错误中已经给出了解决方法，就是省略掉defined()就可以了。

结果：重新编译成功。

# 5. rootfs for mini2440

# 6. 启动内核

## 6.1 Nand启动内核

在flashimg文件下，新建一个脚本文件方便启动，文件名

qemu-system-arm-mini2440-nand-start.sh

文件内容

|  |
| --- |
| #!/bin/bash  /**qemu源码所在目录**/mini2440/arm-softmmu/qemu-system-arm -M mini2440 -serial stdio -mtdblock nand.bin -usbdevice mouse |

这里使用绝对路径的原因是因为，我的usr/local/bin目录下已经存在其他版本的qemu-system-arm。

没有挂载根文件系统rootfs的结果如下：

|  |
| --- |
| ini2440-nand-start.sh  mini2440\_init: Boot mode: NAND  S3C: CLK=240 HCLK=240 PCLK=240 UCLK=57  QEMU: ee24c08\_init  DM9000: INIT QEMU MAC : 52:54:00:12:34:56  QEMU mini2440\_reset: loaded default u-boot from NAND  QEMU mini2440\_reset: loaded override u-boot (size 39a00)  S3C: CLK=240 HCLK=60 PCLK=30 UCLK=57  S3C: CLK=240 HCLK=60 PCLK=30 UCLK=48  S3C: CLK=405 HCLK=101 PCLK=50 UCLK=48  U-Boot 1.3.2-mini2440 (May 13 2018 - 22:02:27)  I2C: ready  DRAM: 64 MB  NOR Flash not found. Use hardware switch and 'flinit'  Flash: 0 kB  NAND: Bad block table not found for chip 0  Bad block table not found for chip 0  64 MiB  \*\*\* Warning - bad CRC or NAND, using default environment  USB: S3C2410 USB Deviced  In: serial  Out: serial  Err: serial  MAC: 08:08:11:18:12:27  Hit any key to stop autoboot: 0  MINI2440 # |

上面是Uboot启动结果。启动之后，还需要配置一下（其中mini2440=3tb 是为了使屏幕分辨率成为320\*240，横向显示。默认是240\*320，是竖着的）

|  |
| --- |
| MINI2440 # nboot kernel  MINI2440 #  setenv bootargs root=/dev/mtdblock3 rootfstype=jffs2 console=ttySAC0,115200 mini2440=3tb  MINI2440 # saveenv  MINI2440 # bootm |

启动界面如下，报错是因为我们还没有挂载rootfs。



### 6.1.1 Nand初始化

U-boot启动了，并且会显示一些Warning信息：

|  |
| --- |
| NAND: Bad block table not found for chip 0  Bad block table not found for chip 0  64 MiB  \*\*\* Warning - bad CRC or NAND, using default environment |

别担心，显示这些信息很正常，因为我们的NAND文件还没初始化呢。接下来初始化：

|  |
| --- |
| MINI2440 # nand scrub  MINI2440 # nand createbbt  MINI2440 # reset |

## 6.2 NFS启动内核

在上面的介绍中，我们使用mini2440-qemu启动了uboot，但是此时的qemu并没有网络功能，那么也就不能使用u-boot通过NFS或者TFTP来加载内核或者文件系统，为了方便开发，接下来，我们为qemu配置网络功能。

qemu支持多种网络链接方式，其中最常用的就是桥接(bridge)。 这需要依赖内核的tun/tap模块的支持。

另外需要注意的是，如果主机使用虚拟机，必须将虚拟机网卡设置为桥接方式(bridge)。

### 6.2.1 安装必要的工具包

1.qemu支持多种网络链接方式，其中最常用的就是桥接(bridge)。这需要依赖内核的tun/tap模块的支持。

|  |
| --- |
| $ sudo apt-get install uml-utilities  $ sudo apt-get install bridge-utils |

首先，查看/dev/net/tun这个文件存不存在。使用下面的命令，

|  |
| --- |
| $ ls -l /dev/net/tun |

显示结果：

|  |
| --- |
| crw-rw-rwT 1 root root 10, 200 Jul 26 08:28 /dev/net/tun |

如果上面的文件存在，这表明内核已经支持开启了tun支持，ubuntu12.04中，这个功能默认已经开启。如果这个文件不存在，则需要加载tun模块，并创建/dev/net/tun文件。

2. openvpn

创建虚拟网络接口tap0的工具

|  |
| --- |
| $ sudo apt-get install openvpn |

### 6.2.2 搭建NFS服务

1. 假设你用的操作系统为ubuntu，首先安装 nfs服务器

|  |
| --- |
| # sudo apt-get install nfs-kernel-server |

2. 创建nfsroot目录

之前在~/buildroot-2012.05/output/images/这个目录里，除了生成rootfs.jffs2文件外，还生成了rootfs.tar文件，是根文件系统的压缩文件包。我们在/srv文件夹下新建一个nfsroot文件夹以储存解压的根文件系统：

|  |
| --- |
| $ sudo mkdir /srv/nfsroot/  $ cd /srv/nfsroot/  $ sudo tar xvf ~/buildroot-2012.05/output/images/rootfs.tar |

3. 修改/etc/exports 文件，添加一行，使NFS能认识/srv/nfsroot/这个目录：

|  |
| --- |
| /srv/nfsroot/ 192.168.42.0/24(rw,sync,no\_root\_squash,no\_subtree\_check) |

将黄色区域修改为“\*”，代表所有的IP都接受。

参数说明：

（1）rw/ro 该目录分享的权限是可读写(read-write)或只读(read-only)，但最终能不能读写，还是与文件系统的 rwx 及身份有关。

（2）sync/async sync代表数据会同步写入到内存与硬盘中，async则代表数据会先暂存于内存当中，而非直接写入硬盘！

（3）no\_root\_squash/root\_squash 客户端使用NFS文件系统的账号若为root时，系统该如何判断这个账号的身份？预设的情况下，客户端root的身份会由root\_squash的设定压缩成nfsnobody，如此对服务器的系统会较有保障。但如果你想要开放客户端使用root身份来操作服务器的文件系统，那么这里就得要开 no\_root\_squash才行！

（4）all\_squash 不论登入 NFS 的使用者身份为何， 他的身份都会被压缩成为匿名用户，通常也就是 nobody(nfsnobody) 啦！

（5）anonuid/anongid anon 意指 anonymous (匿名者) 前面关于 \*\_squash 提到的匿名用户的 UID 设定值，通常为 nobody(nfsnobody)，但是你可以自行设定这个 UID 的值！当然，这个 UID 必需要存在于你的 /etc/passwd 当中！ anonuid 指的是 UID 而 anongid 则是群组的 GID 啰。

使其生效：

|  |
| --- |
| $ sudo exportfs -r |

### 6.2.3 修改网络配置文件

修改 /etc/network/interfaces，文件如下：

|  |
| --- |
| # interfaces(5) file used by ifup(8) and ifdown(8)  auto lo  iface lo inet loopback  auto enp0s3  iface enp0s3 inet static  address 10.48.100.240  netmask 255.255.255.0  gateway 10.48.100.1  #auto br0  #iface br0 inet dhcp  # bridge\_ports eth0 |

根据自己主机实际情况修改。

### 6.2.4 创建qemu-ifup和qemu-ifdown

1. 创建脚本，建立桥接并启动tap网络接口。下面的脚本文件qemu-ifup。

|  |
| --- |
| #!/bin/sh  #  # script to bring up the tun device in QEMU in bridged mode  # first parameter is name of tap device (e.g. tap0)  #  # some constants specific to the local host - change to suit your host  #  ETH0IPADDR=192.168.0.3  MASK=255.255.255.0  GATEWAY=192.168.0.1  BROADCAST=192.168.0.255  #  # First take eth0 down, then bring it up with IP address 0.0.0.0  #  /sbin/ifdown eth0  /sbin/ifconfig eth0 0.0.0.0 promisc up  #  # Bring up the tap device (name specified as first argument, by QEMU)  #  /usr/sbin/openvpn --mktun --dev $1 --user `id -un`（不需要删除）  /sbin/ifconfig $1 0.0.0.0 promisc up  #  # create the bridge between eth0 and the tap device  #  /usr/sbin/brctl addbr br0  /usr/sbin/brctl addif br0 eth0  /usr/sbin/brctl addif br0 $1  #  # only a single bridge so loops are not possible, turn off spanning tree protocol  #  /usr/sbin/brctl stp br0 off  #  # Bring up the bridge with ETH0IPADDR and add the default route  #  /sbin/ifconfig br0 $ETH0IPADDR netmask $MASK broadcast $BROADCAST  /sbin/route add default gw $GATEWAY  #  # stop firewall - comment this out if you don't use Firestarter  #  /sbin/service firestarter stop |

上面的代码中红色区域根据自己主机的实际情况修改，黄色区域需要安装，灰色区域不需要刻意删除。

2. 在QEMU退出后，还需要一个脚本来重置主机网络。 为了保持一致，我们将其称为qemu-ifdown。

|  |
| --- |
| #!/bin/sh  #  # Script to bring down and delete bridge br0 when QEMU exits  #  # Bring down eth0 and br0  #  /sbin/ifdown eth0  /sbin/ifdown br0  /sbin/ifconfig br0 down  #  # Delete the bridge  #  /usr/sbin/brctl delbr br0  #  # bring up eth0 in "normal" mode  #  /sbin/ifconfig eth0 -promisc  /sbin/ifup eth0  #  # delete the tap device  #  /usr/sbin/openvpn --rmtun --dev $1  #  # start firewall again  #  /sbin/service firestarter start |

同qemu-ifup文件一样，根据自己的实际情况修改。

### 6.2.5 创建NFS启动脚本

文件名称：qemu-system-arm-mini2440-nfs-start.sh。

|  |
| --- |
| #!/bin/bash  sudo sh qemu-ifup tap0  /home/yokogawa/qemu/qemu-mini2440/mini2440/arm-softmmu/qemu-system-arm \  -M mini2440 \  -serial stdio \  -mtdblock nand.bin \  -kernel uImage \  -net nic -net tap,ifname=tap0,script=no,downscript=no \  -usb -usbdevice keyboard -usbdevice mouse  sudo sh qemu-ifdown tap0 |

### 6.2.6 U-boot引导

U-boot启动后，设置引导参数（其中mini2440=3tb 是为了使屏幕分辨率成为320\*240，横向显示。默认是240\*320，是竖着的）

|  |
| --- |
| MINI2440 #  setenv bootargs root=/dev/nfs rw nfsroot=10.48.100.246:/srv/nfsroot/  ip=10.48.100.247:10.48.100.240:10.48.100.1:255.255.255.0 console=ttySAC0,115200 mini2440=3tb  MINI2440 # bootm |

说明：

黄色区域，主机IP地址，也就是上面创建的br0的地址，NFS服务器地址

红色区域，第一字段是目标板IP，也就是mini2440

第二字段是主机IP，boot时使用的服务器地址

第三字段是网关

第四字段是子网掩码

### 6.2.7 Nand启动，添加网卡

启动时，设置引导参数为：

|  |
| --- |
| MINI2440 # nboot kernel  MINI2440 #  setenv bootargs root=/dev/mtdblock3 rootfstype=jffs2  ip=10.48.100.247:10.48.100.240:10.48.100.1:255.255.255.0 console=ttySAC0,115200 mini2440=3tb  MINI2440 # saveenv  MINI2440 # bootm |

# 7 Flashimg工具

## 7.1 关于Flashimg

Flashimg是一个强大的工具，是一个由网友FabriceJouhaud 开发的软件，可以很快捷地生成NAND或NOR镜像文件。在不了解flash内部组成和操作原理的情况下，这个软件就可以生成你所想要的大小的镜像文件。

## 7.2 使用flashimg

1. 首先下载flashimg：

git clone git://gitorious.org/flashimg/flashimg.git

2. 在linux系统编译安装：

|  |
| --- |
| $ ./autogen.sh  $ ./configure  $ make  $ sudo make install |

编译成功后，出现一个可执行文件flashimg。

查看版本：

flashimg -v

详细信息：

|  |
| --- |
| flashimg version 0.2 |

要生成NAND或NOR镜像文件，可以先把之前生成的2个文件：u-boot.bin, uImage 拷贝到flashimg（新建）文件夹下，然后把编译flashimg源码目录下的uboot.part也拷贝过来，执行下面的命令，生成NAND或NOR镜像文件：

$ flashimg –s 64M -t nand -f nand.bin -p uboot.part -w boot,u-boot.bin -w kernel,uImage -w root,rootfs.jffs2 -z 512

$ flashimg -s 2M -t nor-f nor.bin -p uboot.part -w boot,u-boot.bin -w kernel,uImage -w root,rootfs.jffs2

其中，

-s 镜像文件的大小

-t 类型指定nand或nor

-f 镜像文件

-p 指定名字，偏移和大小

-w 指定生成镜像的源文件

-z 页大小

# 8. Buildroot

一款超好用的开源软件，一次帮我们把交叉编译工具链和引导程序（bootloader），内核镜像和根文件系统镜像等都编译好。

首先我们安装wget这个小工具，便于下载Buildroot：

$ sudo apt-get install wget

接着下载Buildroot：

$ cd ~

$ wget http://buildroot.org/downloads/buildroot-2012.05.tar.gz

$ tar zxvf buildroot-2012.05.tar.gz

在让Buildroot为我们生成三个镜像前，需要对其进行一些配置。Mini2440开发板的基本所需配置已经集成在Buildroot里了（众多defconfig文件中的一个），省了不少功夫。在此基础上，我们只需再进行少许配置，就可以让Buildroot开工了。

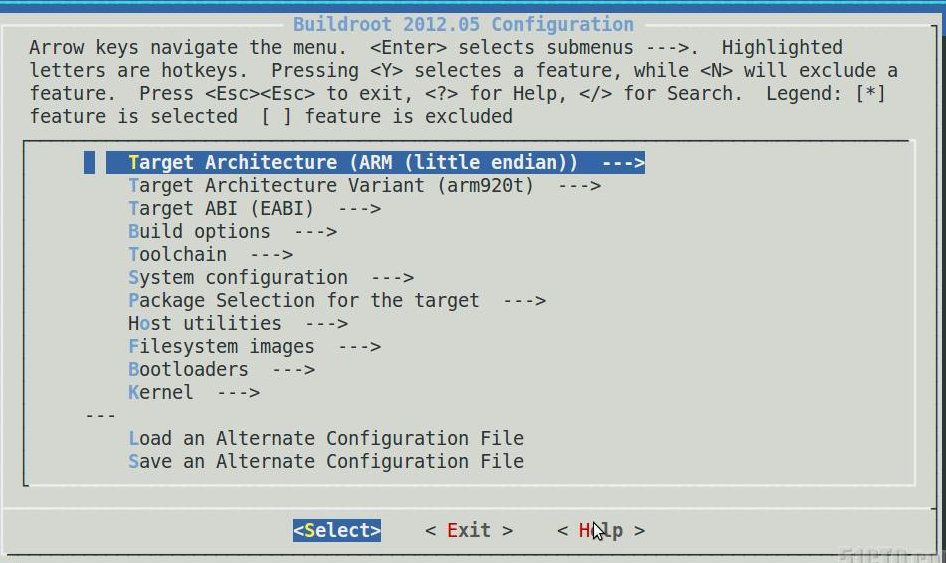
我们选取mini2440\_defconfig来生成我们的 .config 文件，再用 make menuconfig 进一步配置：

$ cd buildroot-2012.05

$ make mini2440\_defconfig

$ make menuconfig

一个方便的配置窗口会跟着出现，是基于Kconfig的配置机制：



以下列出比较重要的几个编译选项，需要改动的几项用红色表示：

* Toolchains ： 在Toolchains目录里，须要选取Linux内核版本的适配的编译链版本。默认是设置为 3.3.x。后面的Kernel(内核版本)我使用的是3.3.7，所以没问题，这里不用改也可以。当然也可以使用诸如 3.0.x这样的版本，只要跟之后的Kernel版本适配就好了。
* Bootloaders ： 默认是U-boot， 也可以选用其他版本的Bootloader。本文就用默认的U-boot。
* Kernel ： 默认是 3.0.4，因为之前Toolchains目录选择的是 3.3.x版本，所以这里须要改选，比如 3.3.7版本。
* Package Selection for the target ： Busybox已经包含在里面了，我们也可以选择更多的项目，比如 Qt， EFL， directfb之类的图形库。在 Graphic libraries and application子目录里选择。
* Filesystem images： 本文中使用的是 jffs2格式的根文件系统（也是Buildroot默认的），因为之后要制作NAND镜像，所以需要配置成一个每页512字节和16字节的ECC的NAND类型。在 Flash Type这项，改为 NAND flash with 512B Page and 16 KB erasesize 。

这些都配置好之后，就可以请出我们最喜欢的命令了 -- make ：

$ make

如果中间出现错误，一般是没装必要的东西，照错误提示安装就好，然后接着 make。 视电脑速度和网速而定，我的情况是大概30分钟后 make 完成。

如果一切顺利， make 结束后，在 buildroot-2012.05目录的 output/images/ 子目录下可以找到生成的三个文件：

- u-boot.bin ： bootloader的镜像

- uImage ： u-boot格式的（用mkimage命令生成的）Linux内核镜像

- rootfs.jffs2 ： jffs2格式的根文件系统镜像

使用请参照[7.2节](#_7.2_使用flashimg)。

在Ubuntu16.04上编译buildroot，费劲周折，好在总算完成。经验教训就是尽量选择低版本的linux发行版作为主机，要不然，太多依赖性，C标准的问题。

# 9. C语言问题-gcc

## 9.1 默认标准是GNU11

GCC defaults to -std=gnu11 instead of -std=gnu89. This brings several changes that users should be aware of. The following paragraphs describe some of these changes and suggest how to deal with them.

Some users might prefer to stay with gnu89, in which case we suggest to use the -std=gnu89 command-line option, perhaps by putting it in override CFLAGS or similarly in Makefiles.

To ease the migration process, GCC offers two new warning options, -Wc90-c99-compat and -Wc99-c11-compat. The former warns about features not present in ISO C90, but present in ISO C99. The latter warns about features not present in ISO C99, but present in ISO C11. See the GCC manual for more info.

## 9.2 inline函数的不同

虽然-std=gnu89和-std=gnu11这两种标准，都包含了inline内联函数的语义。但是，C99标准里，如果具有外部链接的函数声明具有inline函数说明符，那么它必须在同一个转译单元里进行定义。因此，现在的GCC看见下面的代码，会发出warnings：

inline int foo (void);

这个例子会给出下面的信息：

f.c:1:12: warning: inline function 'foo' declared but never defined inline int foo (void);

此外，extern inline和inline之间有区别：

C99 inline: 产生的函数不具有外部可见功能。如果在这个转移单元中引用该函数，在另一个转移单元必须存在外部定义；与GNU89大部分时候相同，但是不同的是，允许重定义。

C99 extern inline: 产生的函数具有外部可见功能；与GNU89 inline相同；

GNU89 inline: 与C99 extern inline相同。

GNU89 extern inline: 产生的函数不具有外部可见功能；不等价于C99，因为不允许重定义。

（幸运的是，static inline在C99和GNU89是相同的。）

换句话说，ISO C99要求只有一个C源文件具有内联函数的可调用拷贝。 考虑以下程序：

|  |
| --- |
| inline int foo (void)  {  return 42;  }  int main (void)  {  return foo ();  } |

The program above will not link with the C99 inline semantics, because no out-of-line function foo is generated. To fix this, either mark the function foo as extern, or add the following declaration:

上面的程序不会使用C99 inline内联语义进行链接，因为没有外部函数foo产生。为了修改这个程序，既可以标记该函数为extern，或者添加下面的声明：

extern inline int foo (void);

这确保了外部可见函数被省略。为了加强CNU89的inline语义，既可以使用-fgnu89-inline命令行选项，也可以使用gnu\_inline属性。例如，

|  |
| --- |
| \_\_attribute\_\_ ((gnu\_inline)) inline int  foo (void)  {  return 42;  } |

使用GNU89 extern inline的程序在新的标准下编译失败，因为发生多重定义错误：

|  |
| --- |
| extern inline int foo (void)  {  return 42;  }  int foo (void)  {  return 23;  }  int main (void)  {  return foo ();  } |