目录

[1 E9V2 系统烧写 2](#_Toc2631262)

[1.1 使用官方工具烧写 2](#_Toc2631263)

[1.1.1烧写工具：Mfgtools-NEW-EMMC\_V6 2](#_Toc2631264)

[1.1.2 mfgtools\_V3.1使用 2](#_Toc2631265)

[1.2 配置内核支持nfs 4](#_Toc2631266)

[1.3 重新编译内核 4](#_Toc2631267)

[1.4 nfs挂载网络文件夹 5](#_Toc2631268)

[1.5 uboot网络设置 5](#_Toc2631269)

[1.6 ubutu 开启tftp服务 7](#_Toc2631270)

[1.7 通过tftp更新emmc内核 7](#_Toc2631271)

[1.8 如何uboot从哪里开始执行 8](#_Toc2631272)

[1.9 uboot如何下载内核 8](#_Toc2631273)

[1.10 EMMC分区信息 8](#_Toc2631274)

[2 yocto 方式编译内核 9](#_Toc2631275)

[2.1 免翻墙获取repo工具的方式 9](#_Toc2631276)

[2.2 安装配置yocto 10](#_Toc2631277)

[2.1.1安装软件包 10](#_Toc2631278)

[2.1.2安装u-boot-tools 10](#_Toc2631279)

[2.1.3设置repo() 10](#_Toc2631280)

[2.1.4设置git环境 10](#_Toc2631281)

[2.1.5 repo同步 11](#_Toc2631282)

[2.1.6 编译yocto工程 11](#_Toc2631283)

[2.1.7编译镜像文件 11](#_Toc2631284)

[2.1.8编译qt5的交叉编译工具链 12](#_Toc2631285)

[2.1.9设置开机自动运行脚本 13](#_Toc2631286)

[2.1.10 授予非root用户权限 13](#_Toc2631287)

[2.2 源文件位置 14](#_Toc2631288)

[2.2.1dts设备树位置 14](#_Toc2631289)

[2.2.2根文件系统位置 14](#_Toc2631290)

[2.2.3 uboot所在位置 14](#_Toc2631291)

[2.2.4 内核所在位置 14](#_Toc2631292)

[2.2.5 映像文件位置 14](#_Toc2631293)

[2.3 单独编译内核 14](#_Toc2631294)

[2.4 使用编译出来的文件系统 14](#_Toc2631295)

[2.5在IMX6q处理器上面使用QT5 16](#_Toc2631296)

[2.5.1 设置环境变量 16](#_Toc2631297)

[3 uboot流程分析 16](#_Toc2631298)

[4移植 18](#_Toc2631299)

[4.1 移植uboot 18](#_Toc2631300)

[4.1.1 移植uboot启动内核 18](#_Toc2631301)

[IOMUX 概述 25](#_Toc2631302)

[GPIO 27](#_Toc2631303)

# 1 E9V2 系统烧写

## 1.1 使用官方工具烧写

### 1.1.1烧写工具：Mfgtools-NEW-EMMC\_V6

如果是4.15内核需要使用mfgtools\_V3.1

烧写文件：uboot.bin;uImage;roorfs.tgz。

烧写配置：

硬件：拨码开关设为usb启动，值为1001。网口方向为上，向上拨码为1。

软件：镜像文件改名为uboot.bin;uImage;roorfs.tgz。

存放目录Mfgtools-NEW-EMMC\_V6\Profiles\MX6Q Linux Update\OS Firmware\files\linux

cfg配置如下：不用改动。

[profiles]

chip = MX6Q Linux Update

[platform]

board = SabreSD

[LIST]

name =Linux-SabreSD-eMMC

烧写：

打开mfgtool ，检测到硬件设备，点击start,进度条变为绿色成功。先点击down,后点击exit。烧写成功

### 1.1.2 mfgtools\_V3.1使用

**烧写.tar.bz2格式文件配置**

cfg 文件配置如下

[profiles]

chip = Linux

[platform]

board = sabresd

[LIST]

name = emmc-linux

[variable]

uboot=u-boot.imx

dtb=imx6q-sabresd.dtb

kernel=zImage

linux\_system=rootfs.tar.bz2

ucl2文件配置

查找 <LIST name="eMMC-linux"

<!-- burn rootfs -->

<CMD state="Updater" type="push" body="$ mkfs.ext4 -E nodiscard /dev/mmcblk3p2">Formatting rootfs partition</CMD>

<CMD state="Updater" type="push" body="$ mkdir -p /mnt/mmcblk3p2"/>

<CMD state="Updater" type="push" body="$ mount -t ext4 /dev/mmcblk3p2 /mnt/mmcblk3p2"/>

<CMD state="Updater" type="push" body="pipe tar -jxv -C /mnt/mmcblk3p2" file="files/linux/rootfs.tar.bz2" ifdev="MX6SL MX6D MX6Q MX6SX">Sending and writting rootfs</CMD>

<CMD state="Updater" type="push" body="frf">Finishing rootfs write</CMD>

<CMD state="Updater" type="push" body="$ umount /mnt/mmcblk3p2">Unmounting rootfs partition</CMD>

<CMD state="Updater" type="push" body="$ echo Update Complete!">Done</CMD>

</LIST>

1.1.3 如何只烧写uboot

cfg 文件配置如下

[profiles]

chip = Linux

[platform]

board = sabresd

[LIST]

name = emmc-linux

[variable]

uboot=u-boot.imx

#dtb=imx6q-sabresd.dtb

#kernel=zImage

#linux\_system=rootfs.tar.bz2

ucl2文件配置

只保留烧写uboot命令，其他全部删除

<!-- burn uboot -->

<CMD state="Updater" type="push" body="$ dd if=/dev/zero of=/dev/mmcblk3 bs=1k seek=384 conv=fsync count=129">clear u-boot arg</CMD>

<!-- access boot partition -->

<CMD state="Updater" type="push" body="$ echo 0 > /sys/block/mmcblk3boot0/force\_ro">access boot partition 1</CMD>

<CMD state="Updater" type="push" body="send" file="files/linux/u-boot.imx" ifdev="MX6Q">Sending u-boot.bin</CMD>

<CMD state="Updater" type="push" body="$ dd if=$FILE of=/dev/mmcblk3boot0 bs=512 seek=2">write U-Boot to sd card</CMD>

<CMD state="Updater" type="push" body="$ echo 1 > /sys/block/mmcblk3boot0/force\_ro"> re-enable read-only access </CMD>

<CMD state="Updater" type="push" body="$ mmc bootpart enable 1 1 /dev/mmcblk3">enable boot partion 1 to boot</C

….删除

## 配置内核支持nfs

1、配置网络部分，主要是使能CONFIG\_IP\_PNP以在2中能够看到Root file system on NFS选项  
Networking support   
Networking options   
TCP/IP networking  
[\*]IP: kernel level autoconfiguration  
[\*] IP: DHCP support  
[\*] IP: BOOTP support  
  
2、配置开启nfs服务  
File systems --->   
Network File Systems --->  
<\*> NFS client support   
[\*] NFS client support for NFS version 3

[\*] NFS client support for the NFSv3 ACL protocol extension   
[\*] NFS client support for NFS version 4 (EXPERIMENTAL)   
[\*] NFS client support for NFSv4.1 (DEVELOPER ONLY)   
[\*] Root file system on NFS

<\*> NFS server support

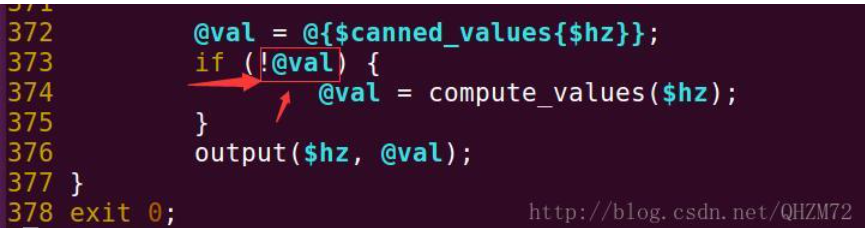
## 1.3 重新编译内核

编译器使用gcc4.6-linaro

修改内核build.sh编译器路径为编译器实际安装路径，注意大小写。采用四核编译make -j4.

编译内核错误提示2：recipe for target 'kernel/timeconst.h' failed

查找资料后发现：因为在新版本的(Linux) 上编译老版本的内核版本出现的错误（新版主机linux内核是4.10，被编译版本的内核是3.0的，好像是这么回事！），解决办法是：将kernel/timeconst.pl中第373行的defined()去掉，去掉后如下



## 1.4 nfs挂载网络文件夹

1、使用老的ubuntu14，32位 系统挂载nfs

mount -t nfs -o nolock,vers=2 192.168.1.111:/home/chao/work /mnt

挂载成功

2、使用新的ubuntu18，64位 系统挂载nfs 需要如下操作：

设置Ubuntu18.04系统中的nfs服务支持协议2，修改nfs配置文件

sudo gedit /etc/default/nfs-kernel-server,在文件末尾加入一句：RPCNFSDOPTS="--nfs-version 2,3,4 --debug --syslog"。

原因是从Ubuntu17.04开始，nfs默认只支持协议3和协议4，而kernel中默认支持协议2，所以才会出现挂载失败的情况。

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

mount -t nfs -o nolock,vers=2 192.168.1.118:/home/chao/work /mnt

## 1.5 uboot网络设置

##### param setting #####

[1]setting nfs args

[2]setting tftp args

[3]setting display args

[4]setting default boot

[s]save setting

[q]quit

Enter your selection: 1

--------setting nfs args--------

Enter the PC IP address:(xxx.xxx.xxx.xxx)

192.168.1.118

Enter the IMX6 IP address:(xxx.xxx.xxx.xxx)

192.168.1.6

Enter the IMX6 MAC address:(xx:xx:xx:xx:xx:xx)

12:32:43:34:21:43

Enter NFS directory:(eg: /opt/EmbedSky/root\_nfs)

/home/chao/work

save setting ? (y/n) :

y

Saving Environment to MMC...

Writing to MMC(3)... done

##### param setting #####

[1]setting nfs args

[2]setting tftp args

[3]setting display args

[4]setting default boot

[s]save setting

[q]quit

Enter your selection: 2

--------setting tftp args--------

Enter the TFTP Server(PC) IP address:(xxx.xxx.xxx.xxx)

192.168.1.118

Enter the IMX6 IP address:(xxx.xxx.xxx.xxx)

192.168.1.7

Enter the GateWay IP:(xxx.xxx.xxx.xxx)

192.168.1.1

Enter the bootloader (u-boot or bootimage) image name:

u-boot.bin

Enter the LOGO image name:

Enter the kernel image name:

uImage

Enter the root image name:

rootfs

save setting ? (y/n) :

y

Saving Environment to MMC...

Writing to MMC(3)... done

##### param setting #####

[1]setting nfs args

[2]setting tftp args

[3]setting display args

[4]setting default boot

[s]save setting

[q]quit

Enter your selection: s

Saving Environment to MMC...

Writing to MMC(3)... done

MX6Q SABRESD U-Boot > ping 192.168.1.118

PHY indentify @ 0x0 = 0x001cc915

mx6\_rgmii\_rework fin to rework network!

FEC: Link is Up 796d

Using FEC0 device

host 192.168.1.118 is alive

ping 虚拟机成功，主机不成功。此时需要关闭主机防火墙。

## 1.6 ubutu 开启tftp服务

**虚拟机开启nfs服务**

1、sudo apt-get install nfs-kernel-server

2、gedit /etc/exports最后添加：

/home/chao/nfs \*(rw,sync,no\_root\_squash)// /home/chao/work允许挂载nfs的目录

3、重启服务

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

sudo /etc/init.d/rpcbind restart

tftp下载需要nfs服务支持，从nfs文件夹内寻找内核映像文件

1、安装tftp服务

sudo apt-get install tftp tftpd openbsd-inetd

2、sudo chmod 777 nfs/ -R //nfs目录

3、修改配置

gedit /etc/inetd.conf //root用户

tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /home/chao/nfs

4、重启服务

/etc/init.d/openbsd-inetd restart

in.tftpd -l /home/chao/nfs

5、测试

echo 'ad' > /home/chao/nfs/hello  
tftp 127.0.0.1  
tftp>get hello

6、将文件系统中/etc/init.d/rcS中的“net\_set &”   
修改为“#net\_set &”这句是设置开发板ip用的，会出问题

## 1.7 通过tftp更新emmc内核

启动的时候，停在uboot计时那里，插上网线

1.按一下q键退出天嵌uboot菜单，进入到uboot命令模式

2.配置好uboot的ipaddr和serverip，并确定E9在uboot下能ping通电脑主机

3.打开tftp32小工具软件，指定好uImage文件目录

4.执行以下命令

mmc dev 3 //选中emmc设备

mmc erase 0x800 0x3000 //擦除内核

tftp 0x10800000 uImage //下载内核到RAM中

mmc write 0x10800000 0x800 0x3000 //将下载到ram中的内核，烧到emmc

## 1.8 如何uboot从哪里开始执行

先编译uboot，然后搜索uboot.lds

uboot.lds连接文件指示启动执行流程

.text :

{

board/freescale/mx6q\_sabresd/flash\_header.module (.text.flasheader)

cpu/arm\_cortexa8/start.o

board/freescale/mx6q\_sabresd/libmx6q\_sabresd.a (.text)

lib\_arm/libarm.a (.text)

net/libnet.a (.text)

drivers/mtd/libmtd.a (.text)

drivers/mmc/libmmc.a (.text)

. = DEFINED(env\_offset) ? env\_offset : .;

common/env\_embedded.o(.text)

\*(.text)

}

## 1.9 uboot如何下载内核

1]boot from emmc

[2]boot from sdcard

[3]boot from tftp

[5]download from sdcard

[0]setting boot args

[q]exit to command mode

Enter your selection: 1

mmc3(part 0) is current device

MMC read: dev # 3, block # 2048, count 12288 ... 12288 blocks read: OK

根据uboot打印出来的菜单，搜索menu命令：

U\_BOOT\_CMD(

menu, 3, 0, do\_menu,

)

do\_menu

menu\_shell()//菜单在这里定义

run\_command("run bootcmd\_mmc\_linux", 0);

cmdtp = find\_cmd(argv[0]) //查找菜单对应的命令

(cmdtp->cmd) (cmdtp, flag, argc, argv)//执行命令

搜索is current device找到对应的函数是do\_mmcops

## 1.10 EMMC分区信息

mmc0: new high speed DDR MMC card at address 0001

mmcblk0: mmc0:0001 008G92 7.28 GiB

mmcblk0boot0: mmc0:0001 008G92 partition 1 4.00 MiB

mmcblk0boot1: mmc0:0001 008G92 partition 2 4.00 MiB

mmcblk0: p1

mmcblk0boot1: unknown partition table

mmcblk0boot0: unknown partition table

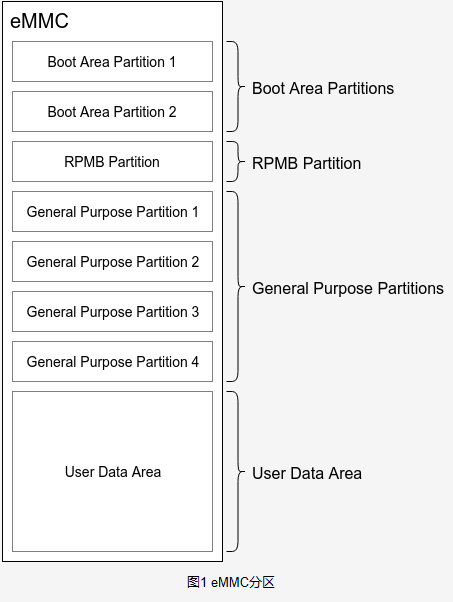
eMMC 标准中，将内部的 Flash Memory 划分为 4 类区域，分别是：

User Data:区域用于存储数据；

boot: 区域用于启动；

replay protected memory block:区域用于存放受保护的数据

general purpose:分区



setenv mxcfb0 video=mxcfb0:dev=lcd,TQ-TFT\_1024600

# 2 yocto 方式编译内核

## 2.1 免翻墙获取repo工具的方式

通过清华大学镜像源clone repo代码

git clone https://gerrit-google.tuna.tsinghua.edu.cn/git-repo1

本地会同步下一个名为git-repo的工程，工程里包含最新的repo工具，执行repo init就可以直接本地的repo，此时不会再向google的服务器同步最新的repo工具

执行repo init

[git-repo路径]/repo init -u https://android.googlesource.com/platform/manifest

PS：

我还在清华大学镜像网站上看到另外一种方式：

mkdir ~/bin

PATH=~/bin:$PATH

curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

chmod a+x ~/bin/repo

update:

上面的方式是将repo添加到环境变量，但是repo运行时，仍然会去google服务器请求最新版本，解决方式是在本地环境变量中指定repo服务器地址

repo的运行过程中会尝试访问官方的git源更新自己，如果想使用tuna的镜像源进行更新，可以将如下内容>复制到你的~/.bashrc里

export REPO\_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/'

并重启终端模拟器。

## 2.2 安装配置yocto

### 2.1.1安装软件包

$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libsdl1.2-dev

$ sudo apt-get install libsdl1.2-dev xterm sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 help2man make gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl lzop asciidoc

### 2.1.2安装u-boot-tools

$ sudo apt-get install u-boot-tools

### 2.1.3设置repo()

$ mkdir ~/bin

$ cd ~/bin

$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo

$ chmod a+x repo

$ export PATH=~/bin:$PATH

这里需要从谷歌下载文件，有梯子的同学需要注意了，如果你的梯子只支持http代理（如lantern），这里用命令是下载不下来的，你需要手工把链接粘贴到浏览器里下载下来再粘贴到~/bin目录下。

这是因为curl wget等命令是通过socket下载的，比http更底层，当你的代理只是在http层代理的时候是无效的

### 2.1.4设置git环境

$ git config --global user.name "Your Name"

$ git config --global user.email "Your Email"

$ git config --list

### 2.1.5 repo同步

$ mkdir ~/fsl-release-bsp

$ cd ~/fsl-release-bsp

$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth

$ repo sync

然后等待同步完成，如果同步过程出错，可以用ctrl+c中断执行，并重新执行repo sync直到同步成功。

### 2.1.6 编译yocto工程

$ DISTRO=<distro name> MACHINE=<machine name> source fsl- setup-release.sh -b <build dir>

DISTRO有如下四种设置：

fsl-imx-x11

fsl-imx-wayland

fsl-imx-xwayland

fsl-imx-fb

我选择fsl-imx-x11

MACHINE同样有多种配置：

imx6qpsabreauto

imx6qpsabresd

imx6ulevk

imx6ull14x14evk

imx6ull9x9evk

imx6dlsabreauto

imx6dlsabresd

imx6qsabreauto

imx6qsabresd

imx6slevk

imx6solosabreauto

imx6solosabresd

imx6sxsabresd

imx6sxsabreauto

imx7dsabresd

我选择imx6qsabresd。因此我执行的最终命令如下：

$ DISTRO=fsl-imx-x11 MACHINE=imx6qsabresd source fsl-setup-release.sh -b build

### 2.1.7编译镜像文件

$ bitbake fsl-image-qt5

同样的bitbake可以编译以下多种镜像：  
core-image-minimal  
core-image-base  
core-image-sato  
fsl-image-machine-test  
fsl-image-gui  
fsl-image-qt5  
编译fsl-image-qt5的时间非常长，在网速飞快的情况下也需两天，如果编译中途出错，可以ctrl+c中断后重新执行命令：

$ bitbake fsl-image-qt5

如果编译过程中无意关掉了终端，那么重新打开终端进入fsl-release-bsp目录后只需要执行以下命令就可以重新进入编译环境：

$ source setup-environment build

编译结束后会占用80G左右空间，虚拟机默认配置空间会不足

### 2.1.8编译qt5的交叉编译工具链

$ bitbake meta-toolchain-qt5

编译完成后会在/opt/yocto/fsl-release-bsp/build/tmp/deploy/sdk目录下  
生成文件fsl-imx-fb-glibc-i686-meta-toolchain-qt5-cortexa9hf-neon-toolchain-4.1.15-2.0.0.sh。

执行如下命令安装qt5的交叉编译工具链：

$cd ~/fsl-release-bsp/build/tmp/deploy/sdk $ ./fsl-imx-x11-glibc-x86\_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-4.1.15-2.1.0.sh

........

SDK has been successfully set up and is ready to be used.

Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.

$ . /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi

执行如下命令安装GCC的交叉编译工具链：

安装会提示目录冲突，不用管，直接安装

$ source setup-environment build

$ bitbake meta-toolchain

$ cd ~/fsl-release-bsp/build/tmp/deploy/sdk

$ ./fsl-imx-x11-glibc-x86\_64-meta-toolchain-cortexa9hf-neon-toolcha-4.1.15-2.1.0.sh

一路选择默认设置，最终将在/opt/fsl-imx-x11/4.1.15-2.1.0/目录下生成我们所需要的工具链。  
SDK has been successfully set up and is ready to be used.  
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.  
$ . /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi

至此，我们所需要的所有文件均已生成，包含了uboot，kernel，rootfs以及交叉编译qt应用程序的交叉编译工具。相对应的软件版本如下：  
uboot：v2016.03  
kernel：v4.1.15  
qt版本：qt5.6.2  
当然最重要的是yocto编译生成的rootfs中包括了imx6所需要的各种库，特别是有我们所需要的能完整的支持imx6的gpu性能的qt5.6.

### 2.1.9设置开机自动运行脚本

$cd /opt/fsl-imx-x11/4.1.15-2.1.0

$chmod a+x environment-setup-cortexa9hf-neon-poky-linux-gnueabi

$./environment-setup-cortexa9hf-neon-poky-linux-gnueabi

vim ~/.bashrc

在最后一行添加

source /opt/fsl-imx-x11/4.1.15-2.1.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi这样开机自动会把工具链加入环境变量中

### 2.1.10 授予非root用户权限

$cd /opt/fsl-imx-x11/4.1.15-2.1.0/sysroots/x86\_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi

$ls -l

$cd ../

$chown chao arm-poky-linux-gnueabi/ -R

$ls arm-poky-linux-gnueabi/ -l

作者：Lazy\_Caaat  
链接：https://www.jianshu.com/p/f6e0debb5e1f  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

## 2.2 源文件位置

### 2.2.1dts设备树位置

/home/chao/fsl-release-bsp/build/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/4.1.15-r0/git/arch/arm/boot/dts/imx6q-sabresd.dts

### 2.2.2根文件系统位置

/home/chao/fsl-release-bsp/build/tmp/deploy/images/imx6qsabresd

根文件系统是个镜像文件需要挂载到一个目录才能看见里面的文件

执行如下命令挂载

chao@ubuntu:~$ cd /home/chao/fsl-release-bsp/build/tmp/deploy/images/imx6qsabresd

chao@ubuntu:~/fsl-release-bsp/build/tmp/deploy/images/imx6qsabresd$ sudo mount -o loop -t ext4 fsl-image-qt5-imx6qsabresd.ext4 /home/chao/work/rootfs

在/home/chao/work/rootfs文件夹里面可以看到生成的文件系统内容

执行如下命令生成myrootfs.tgz

chao@ubuntu:~/work$ sudo tar cfz rootfs.tgz ./rootfs

[sudo] password for chao:

chao@ubuntu:~/work$ ls

myrootfs.tgz Qtarm rootfs

### 2.2.3 uboot所在位置

/home/chao/fsl-release-bsp/build/tmp/work/imx6qsabresd-poky-linux-gnueabi/u-boot-imx /2016.03-r0/git

### 2.2.4 内核所在位置

/home/chao/fsl-release-bsp /build/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/4.1.15-r0/git

### 2.2.5 映像文件位置

编译出来的image在/home/chao/fsl-releases-bsp/imx6q-x11/tmp/deploy/images/目录下  
  包括的u-boot zImage rootfs

## 2.3 单独编译内核

source setup-environment build

bitbake -c menuconfig -v linux-imx //配置内核

## 2.4 使用编译出来的文件系统

1、因为内核串口名字为ttySAC0，需要修改文件系统内容rc\_mxc.S添加行

if grep -sq ttymxc0 /proc/cmdline; then

/sbin/getty -L ttymxc0 115200 vt100

elif grep -sq ttySAC0 /proc/cmdline; then

echo "found ttySAC0, set /dev/ttySAC0 as debugcom"

/sbin/getty -n -L ttySAC0 115200 vt100

2、cd rootfs/

压　缩：tar -jcv -f rootfs.tar.bz2 ./\* -R

查　询：tar -jtv -f rootfs.tar.bz2  
解压缩：tar -jxv -f rootfs.tar.bz2 -C 要解压的目录

3、配置uboot启动参数，yocto生成的文件系统init是在/sbin/init目录下的.执行下面命令：

setenv mmcargs setenv bootargs console=ttySAC0,115200 init=/sbin/init root=/dev/mmcblk3p2 rootwait rw video=mxcfb0:dev=lcd,TQ-TFT\_1024600 video=mxcfb1:off video=mxcfb2:off video=mxcfb3:off vmalloc=256M androidboot.console=ttySAC0 consoleblank=0 androidboot.hardware=freescale cma=384M

saveenv 命令保存参数

注意显示参数设置每一个fb都是用空格区分的，一个fb的设置之间不能有空格

4、boot启动内核，登陆用户名为root

5、etc/profile里面添加如下

export DISPLAY=:0.0

否则执行QT程序时提示错误QXcbConnection: Could not connect to display

6、开机执行自己的应用程序

桌面应用配置文件都在/usr/share/applications下面

例如：qtimxplayer.desktop

添加自己的应用mydrawing

第一步添加图标，把图标放到如下目录

/usr/share/pixmaps

名字为mydrawing.png

第二步添加配置文件

cp qtimxplayer.desktop mydrawing.desktop

vi mydrawing.desktop

配置内容如下：注意qt程序代码必须使用绝对路径，不能使用相对路径。

[Desktop Entry]

Name=mydrawing

#Exec=/usr/share/qt5/examples/multimedia/imxplayer/imxplayer

Exec=/mydrawing/mydrawing

Icon=mydrawing

Terminal=false

Type=Application

X-MB-SingleInstance=true

Comment=i.MX Player

StartupNotify=false

启动系统，发现桌面多出一个mydring的图标

cat /sys/class/graphsics/fb0/mode 查看设置好的显示分辨率

## 2.5在IMX6q处理器上面使用QT5

### 2.5.1 设置环境变量

1、使用yocto编译环境执行qmake -v 时出现如下错误提示

qmake: could not exec ‘/usr/lib/x86\_64-linux-gnu/qt4/bin/qmake’

原因是qt指向的qt安装位置不正确.需要设置qt环境变量。设置如下：

cd /usr/lib/x86\_64-linux-gnu/qt-default/qtchooser

sudo vi default.conf

添加如下行：

/opt/fsl-imx-x11/4.1.15-2.1.0/sysroots/x86\_64-pokysdk-linux/usr/bin/qt5

2、设置arm交叉编译工具链环境变量

chao@ubuntu:/$ sudo su

root@ubuntu:/# vim /etc/environment

添加如下:

/opt/fsl-imx-x11/4.1.15-2.1.0/sysroots/x86\_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi

root@ubuntu:/# source /etc/environment

root@ubuntu:/# arm-poky-linux-gnueabi-gcc -v

3、 添加库路径

在chao用户名下执行

export PATH=$PATH:/opt/fsl-imx-x11/4.1.15-2.1.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi/usr/lib

4、编译APP的时候出现:oe-device-extra.pri: No such file or directory但又可以编译出app且可以在平台上运行，没什么影响，但我还是看不惯，一定要把它解决掉：参考解决方案：https://github.com/meta-qt5/meta-qt5/commit/9aa870eecf6dc7a87678393bd55b97e21033ab48直接到出错的目录下：touch oe-device-extra.pri 即可解决这个问题。

### 2.6 使用交叉编译器

$CC -o test test.c

第三方makefile 中使用,配置编译器

CROSS-COMPILE=${CC} //makefile中变量必须用大括号括起来

# 3 uboot流程分析

如果有多个文件里面有同一个函数，可以编译一下uboot，查看编译过程使用的是哪个文件。

arch/arm/cpu/armv7/start.s

reset:

bl cpu\_init\_crit

ENTRY(cpu\_init\_crit)

b lowlevel\_init @ go setup pll,mux,memory

arch\arm\cpu\armv7\ lowlevel\_init.S

ENTRY(lowlevel\_init)

bl s\_init

arch\arm\cpu\armv7\mx6\soc.c

void s\_init(void)

arch/arm/cpu/armv7/start.s

bl \_main

crt0.S (arch\arm\lib) line 58 : ENTRY(\_main)

ENTRY(\_main)

bl board\_init\_f

common/board\_f.c

board\_init\_f

initcall\_run\_list(init\_sequence\_f

env\_init,//环境变量初始化

crt0.S (arch\arm\lib)// board\_init\_f返回

b relocate\_code

relocate.S (arch\arm\lib) line 66 : ENTRY(relocate\_code)

ENTRY(relocate\_code)

ENDPROC(relocate\_code)

crt0.S (arch\arm\lib)// board\_init\_f返回

ldr pc, =board\_init\_r /\* this is auto-relocated! \*/

common\board\_r.c

board\_init\_r(gd\_t \*id, ulong dest\_addr)

initcall\_run\_list(init\_sequence\_r)//执行里面的一系列函数

initr\_mmc, //初始化MMC

run\_main\_loop //最后一个函数

common\main.c

void main\_loop(void)

s = bootdelay\_process();//uboot读秒，等待用户按键

autoboot\_command(s); //用户没有按键，执行环境参数命令

common/autoboot.c

autoboot\_command(const char \*s)

run\_command\_list(s, -1, 0); //传递过来的命令流s会在这里被解析执行

s = getenv("menucmd");

run\_command\_list(s, -1, 0);

common/cli.c

int run\_command\_list(const char \*cmd, int len, int flag)

rcode = parse\_string\_outer(buff, FLAG\_PARSE\_SEMICOLON);

int parse\_string\_outer(const char \*s, int flag)

parse\_stream\_outer(&input, flag);

code = run\_list(ctx.list\_head);

rcode = run\_list\_real(pi);

rcode = run\_pipe\_real(pi);

cmd\_process(flag,child->argc, child->argv,&flag\_repeat, NULL);

cmdtp = find\_cmd(argv[0]);

cmd\_call(cmdtp, flag, argc, argv)

命令最终在这里被执行，以上一系列过程 将收到的指令通过一系列字符处理然后加入一个执行列表，然后执行这个列表。这些命令的的具体实现大家可以 执行 find -name ./common/cmd\*.c这些文件里定义了命令的具体实现。比如我们mmc read xx xx命令，在common\cmd\_mmc.c :842中，大家可以具体去看看,其实读秒过后的，系统自动执行了一系列环境变量（）中保存的命令，执行命令这一套的通用的，只是命令的来源不一样，一个是用户输入的，一个是从环境命令中读取的

# 4移植

## 4.1 移植uboot

### 4.1.1 移植uboot启动内核

1> 复制uboot代码

$cp fsl-release-bsp/build/tmp/work/imx6qsabresd-poky-linux-gnueabi/u-boot-imx/2016.03-r0/git mylinux/u-boot-imx-201603 -rfd

cd mylinux/u-boot-imx-201603

2>添加单板

$mkdir board/EmbedSky

$cp -R board/freescale/common board/EmbedSky/common

$cp -R board/freescale/mx6sabresd/ board/EmbedSky/mx6q\_tqe9/

$cp include/configs/mx6sabresd.h include/configs/mx6q\_tqe9.h

$cp configs/mx6qsabresd\_defconfig configs/mx6q\_tqe9\_defconfig

$mv board/EmbedSky/mx6q\_tqe9/mx6sabresd.c board/EmbedSky/mx6q\_tqe9/mx6q\_tqe9.c

3> 修改内存配置文件

将天嵌公司提供的 mx6q\_4x\_mt41j128.cfg替换mx6q\_tqe9目录下原来的内存配置文件

名字必须是mx6q\_4x\_mt41j128.cfg

4> 修改配置文件

修改mx6q\_tqe9\_defconfig如下：

CONFIG\_SYS\_EXTRA\_OPTIONS="IMX\_CONFIG=board/EmbedSky/mx6q\_tqe9/mx6q\_4x\_mt41j128.cfg,MX6Q"

CONFIG\_ARM=y

CONFIG\_TARGET\_MX6Q\_TQE9=y

CONFIG\_SYS\_MALLOC\_F=y

CONFIG\_SYS\_MALLOC\_F\_LEN=0x400

CONFIG\_DM=y

CONFIG\_DM\_THERMAL=y

把board/EmbedSky/mx6q\_tqe9/Makefile中的下面一行

obj-y  := mx6sabresd.o

改成如下一行  
obj-y  := mx6q\_tqe9.o

修改board/EmbedSky/mx6q\_tqe9/Kconfig(红色部分)

if TARGET\_MX6Q\_TQE9

config SYS\_BOARD

default "mx6q\_tqe9"

config SYS\_SOC

default "mx6"

config SYS\_VENDOR

default "EmbedSky"

config SYS\_CONFIG\_NAME

default "mx6q\_tqe9"

endif

修改arch/arm/Kconfig，增加TARGET\_MX6Q\_TQE9的配置

config ARCH\_MX6

bool "Freescale MX6"

select CPU\_V7

config TARGET\_MX6Q\_TQE9

bool "Support mx6q\_tqe9"

select CPU\_V7

select SUPPORT\_SPL

文件后面添加

source "board/EmbedSky/mx6q\_tqe9/Kconfig"

5>创建编译脚本build.sh:

#!/bin/bash

export ARCH=arm

export CROSS\_COMPILE=arm-poky-linux-gnueabi-

make distclean

make mx6q\_tqe9\_defconfig

make

然后修改执行权限

chmod 777 build.sh

执行./build.sh应该可以生成uboot.imx

6> 修改硬件配置文件

修改串口

E9的串口终端IO跟sabresd不太一样，使用了SD3\_DAT7、SD3\_DAT分别作UART1\_TXD、UART1\_RXD，所以这里需要做相应的改变。

修改board/EmbedSky/mx6q\_tqe9/mx6q\_tqe9.c ：

static iomux\_v3\_cfg\_t const uart1\_pads[] = {

MX6\_PAD\_SD3\_DAT7\_\_UART1\_TX\_DATA | MUX\_PAD\_CTRL(UART\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT6\_\_UART1\_RX\_DATA | MUX\_PAD\_CTRL(UART\_PAD\_CTRL),

};

修改sd卡接口

由于MX6\_PAD\_SD3\_DAT7、MX6\_PAD\_SD3\_DAT6已经被USDHC3占用，继续修改board/EmbedSky/mx6q\_tqe9/mx6q\_tqe9.c ：

static iomux\_v3\_cfg\_t const usdhc3\_pads[] = {

MX6\_PAD\_SD3\_CLK\_\_SD3\_CLK | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_CMD\_\_SD3\_CMD | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT0\_\_SD3\_DATA0 | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT1\_\_SD3\_DATA1 | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT2\_\_SD3\_DATA2 | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT3\_\_SD3\_DATA3 | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT4\_\_SD3\_DATA4 | MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL),

MX6\_PAD\_SD3\_DAT5\_\_SD3\_DATA5|MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL), //MX6\_PAD\_SD3\_DAT6\_\_SD3\_DATA6|MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL), //MX6\_PAD\_SD3\_DAT7\_\_SD3\_DATA7|MUX\_PAD\_CTRL(USDHC\_PAD\_CTRL), //MX6\_PAD\_NANDF\_D0\_\_GPIO2\_IO00 | MUX\_PAD\_CTRL(NO\_PAD\_CTRL), /\* CD \*/

};

E9的sd卡槽使用的是USDHC2控制器，并且与sabresd不同的是该SD CARD电路的CD引脚使用的是GPIO\_4，继续修改board/EmbedSky/mx6q\_tqe9/mx6q\_tqe9.c ：

usdhc2\_pads引脚配置更改：

// MX6\_PAD\_NANDF\_D2\_\_GPIO2\_IO02 | MUX\_PAD\_CTRL(NO\_PAD\_CTRL), /\* CD \*/

MX6\_PAD\_GPIO\_4\_\_GPIO1\_IO04 | MUX\_PAD\_CTRL(NO\_PAD\_CTRL), /\* CD \*/

sd卡槽cd引脚宏定义：

//#define USDHC2\_CD\_GPIO IMX\_GPIO\_NR(2, 2)

#define USDHC2\_CD\_GPIO IMX\_GPIO\_NR(1, 4)

7>编译烧写uboot.imx运行开发[板报](https://www.baidu.com/s?wd=%E6%9D%BF%E6%8A%A5&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)错：Can't find PMIC:PFUZE100

include/configs/mx6\_common.h去掉宏定义CONFIG\_LDO\_BYPASS\_CHECK

/\* LDO Bypass \*/

#ifndef CONFIG\_MX6SLL

/\*#define CONFIG\_LDO\_BYPASS\_CHECK\*/

#endif

修改了include/configs/mx6sabre\_common.h，屏蔽了pmic的宏定义

/\* PMIC \*/

/\*

#define CONFIG\_POWER

#define CONFIG\_POWER\_I2C

#define CONFIG\_POWER\_PFUZE100

#define CONFIG\_POWER\_PFUZE100\_I2C\_ADDR 0x08

\*/

修改了board/freescale/mx6sabresd/mx6sabresd.c(我的板子为mx6q\_tqe9.c)，屏蔽了power\_init\_board(void)函数

#if 0

int power\_init\_board(void)

#endif

这里命令行提示符为"=> "，可以将之改为正式些的形式，修改include/config\_fallbacks.h 行79:

#define CONFIG\_SYS\_PROMPT       "TQE9>> "

最后修改下board/EmbedSky/mx6q\_tqe9.c ：

int checkboard(void)  
{  
puts("Board: MX6Q\_TQE9\n");  
return 0;  
}

8> 编译下载发现卡在starting kernel…,修改环境变量

方法是模仿天嵌配置修改，打印启动参数，看看那些是必须修改的，然后将参数改为天嵌文件一致

修改mx6q\_tqe9.h

对比天嵌mx6sabresd.h，修改如下：

添加下面两行

#include <asm/arch/imx-regs.h>

#include <asm/imx-common/gpio.h>

添加宏定义

#define CONFIG\_EMBEDSKY\_E9

#define CONFIG\_MMC

#define CONFIG\_CMD\_MMC

#define CONFIG\_GENERIC\_MMC

#define CONFIG\_BOUNCE\_BUFFER

#define CONFIG\_CMD\_EXT2

#define CONFIG\_CMD\_EXT4

#define CONFIG\_CMD\_EXT4\_WRITE

#define CONFIG\_CMD\_FAT

#define CONFIG\_FAT\_WRITE

#define CONFIG\_FAT\_CP

#define CONFIG\_DOS\_PARTITION

#define CONFIG\_GET\_DDR\_TARGET\_DELAY

#define CONFIG\_SUPPORT\_EMMC\_BOOT /\* eMMC specific \*/

修改

#define CONFIG\_MACH\_TYPE 3980

#define CONFIG\_MXC\_UART\_BASE UART1\_BASE

#define CONFIG\_CONSOLE\_DEV "ttySAC0"

修改

#ifdef CONFIG\_EMBEDSKY\_E9

/\*#define CONFIG\_MMCROOT "/dev/mmcblk0p2"\*/ /\* SDHC4 \*/

#define CONFIG\_MMCROOT "/dev/mmcblk3p2"

#else

#define CONFIG\_MMCROOT "/dev/mmcblk2p2" /\* SDHC3 \*/

#endif /\* CONFIG\_EMBEDSKY\_E9 \*/

修改

#elif defined(CONFIG\_MX6Q)

#define CONFIG\_DEFAULT\_FDT\_FILE "imx6q-sabresd.dtb"

#define PHYS\_SDRAM\_SIZE (1u \* 1024 \* 1024 \* 1024)

修改

#ifdef CONFIG\_EMBEDSKY\_E9

#define CONFIG\_SYS\_MMC\_ENV\_DEV 0 /\* SDHC4 \*/

#else

#define CONFIG\_SYS\_MMC\_ENV\_DEV 1 /\* SDHC3 \*/

#endif /\* CONFIG\_EMBEDSKY\_E9 \*/

修改内核启动选项include/configs/mx6sabre\_common.h

添加

#define CONFIG\_LOADADDR 0x12000000

#define CONFIG\_SYS\_TEXT\_BASE 0x17800000

#define CONFIG\_BOOTLINUX "fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} zImage;" "fatload mmc ${mmcdev}:${mmcpart} ${fdt\_addr} imx6q-sabresd.dtb;" "run mmcargs;" "bootz ${loadaddr} - ${fdt\_addr}"

修改

"splashimage=0x15000000\0" \

"fdt\_file=" CONFIG\_DEFAULT\_FDT\_FILE "\0" \

"boot\_linux="CONFIG\_BOOTLINUX "\0" \

"splashpos=m,m\0" \

"mmcargs=setenv bootargs console=${console},${baudrate} init=/sbin/init ${smp} " \

"root=${mmcroot} ${mxcfb0} ${mxcfb1} ${mxcfb2} ${mxcfb3} vmalloc=256M androidboot.console=ttySAC0 consoleblank=0 androidboot.hardware=freescale cma=384M\0" \

mmcboot=echo Booting from mmc ...; run mmcargs; if run loadimage; then if run loadfdt; then booti ${loadaddr} - ${fdt\_addr}; else echo ERROR: Cannot load the DT;fi; else echo ERROR: Cannot load the kernel; fi;

CONFIG\_BOOTCOMMAND 就是uboot中用于引导内核的命令，注释掉前面设置

#define CONFIG\_BOOTCOMMAND "run boot\_linux"

如果启动时没有boot\_linux，命令行设置环境变量

setenv boot\_linux "fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} zImage;" "fatload mmc ${mmcdev}:${mmcpart} ${fdt\_addr} imx6q-sabresd.dtb;" "run mmcargs;" "bootz ${loadaddr} - ${fdt\_addr}"

现在可以启动内核到文件系统了

4.1.2 移植网卡

1 、uboot网卡初始化整体流程:

该版Uboot中网卡的启动过程如下board\_init\_f>>board\_init\_r>>initr\_net>>eth\_initialize.（net/eth.c）

在eth\_initialize中会执行miiphy\_init(),phy\_init(),phy\_env\_init()以及board\_eth\_init四个函数。

.网卡具体初始化:

miiphy\_init():初始化mii\_devs链表，该列表中是一系列象征mac控制器，每个miid\_dev表示一条mdio总线，可挂载多个phy设备。

phy\_init():根据宏定义(例如CONFIG\_PHY\_REALTEK、CONFIG\_PHY\_ATHEROS等)初始化相应一个或多个系列网卡，其中就有我们项目中使用的realtek(RTL8211E)系列网卡以及sabresd公版所使用的aheros(AR8031)系列网卡。

phy\_env\_init():初始化关于phy的一些uboot环境变量，这里我们不需要关心。

board\_eth\_init():网卡的板级初始化。

3.网卡板级具体初始化:

board\_eth\_init()中会执行setup\_iomux\_enet(),setup\_pcie(),cpu\_eth\_init()三个函数。

其中setup\_iomux\_enet()函数配置网卡的rgmii接口，一般不用动这部分代码。

setup\_pcie()配置板子pcie总线,具体这里也不用管。

cpu\_eth\_init()函数会做些网卡的芯片级初始化。

4.网卡芯片级具体初始化:

cpu\_eth\_init()只是调用了fecmx\_initialize()函数，该函数中则只调用了fecmxc\_initialize\_multi(bd,-1,CONFIG\_FEC\_MXC\_PHYADDR,IMX\_FEC\_BASE)函数，该函数原型为

Int fecmxc\_initialize\_multi(bd\_t \*bd.int phy\_id,uinit32\_t addr);

作用是初始化挂载在器件地址为phy\_id的mdio上的所有phy芯片，具体实现颇复杂但也比较有意思，有兴趣的可以自行研究下，这里我们知道这些就已经足够了，如此看来似乎phy\_id这个参数名如果改为phy\_addr会更容易理解些。

到这里我们也就已经获得了移植RTL8211E网卡所需要的所有信息，这里做个总结:

由步骤2得知为了注册RTL8211E的驱动，需要定义CONFIG\_PHY\_REALTEK宏，为了探测到MDIO上我们的phy芯片(RTL8211E),需要用调用fecmxc\_initialize\_multi(bd,-1,CONFIG\_FEC\_MXC\_PHYADDR,IMX\_FEC\_BASE)，这里CONFIG\_FEC\_MXC\_PHYADDR就是我们的网卡芯片的器件地址，可通过电路图得知我们的地址为5，也就是说需要把CONFIG\_FEC\_MXC\_PHYADDR宏定义为数值5。

综上所述，为了移植我们的网卡芯片我们只需要两个步骤:

1.注掉include/configs/mx6sabre\_common.h中的

/\* #define CONFIG\_PHY\_ATHEROS \*/

并追加(貌似不需要修改)

#define CONFIG\_PHY\_REALTEK

2.把include/configs/mx6sabre\_common.h中的

#define CONFIG\_FEC\_MXC\_PHYADDR 1

改为(貌似不需要修改)

#define CONFIG\_FEC\_MXC\_PHYADDR 5

注意:在2016.03版本uboot中如果启动uboot提示找不到FEC PHY需要在drivers/net/phy/phy.c下做如下修改

int \_\_weak get\_phy\_id(struct mii\_dev \*bus, int addr, int devad, u32 \*phy\_id)

{

int phy\_reg;

mdelay(10);//加入睡眠10ms

...

}

drivers/net/phy/realtek.cs是rtl8211在uboot下的驱动

3、修改网络配置参数

=> setenv ipaddr 192.168.1.112

=> setenv gatewayip 192.168.1.1

=>setenv serverip 192.168.1.118

=>setenv netmask 255.255.255.0

=> setenv ethaddr ea:d8:c5:cd:46:3f

=> ping 192.168.1.118

Using FEC device

host 192.168.1.118 is alive

4、配置支持nfs和tftp

"tftpimage=zImage\0" \

"boot\_nfs="CONFIG\_BOOT\_NFS "\0" \

"nfsserverip="\_\_stringify(CONFIG\_SERVERIP)"\0" \

"mxcfb0=video=mxcfb0:dev=lcd,CLAA-WVGA\0" \

"mxcfb1=video=mxcfb1:off\0" \

"mxcfb2=video=mxcfb2:off\0" \

"mxcfb3=video=mxcfb3:off\0" \

"logoimgname=logo.bmp\0"

"nfs\_root=setenv mmcroot /dev/nfs rootwait rw nfsroot=${nfsserverip}:${nfsroot},v3,tcp\0" \

"nfsroot=/home/chao/nfs\0" \

添加如下

"loadfdt=fatload mmc ${mmcdev}:${mmcpart} ${fdt\_addr} ${fdt\_file}\0" \

"tftp\_loadimage=tftpfat ${tftpimage} mmc ${mmcdev}:${mmcpart} ${image}\0" \

"tftp\_loadfdt=tftpfat ${fdt\_file} mmc ${mmcdev}:${mmcpart} imx6q-sabresd.dtb\0" \

"tftp\_loadlogo=tftpfat ${logoimgname} mmc ${mmcdev}:${mmcpart} logo.bmp\0" \

命令行设置，记得加双引号

setenv tftp\_loadimage "tftpfat ${tftpimage} mmc ${mmcdev}:${mmcpart} ${image}"

setenv tftp\_loadfdt "tftpfat ${fdt\_file} mmc ${mmcdev}:${mmcpart} imx6q-sabresd.dtb"

4.1.3 添加菜单项

复制天嵌common目录里面的cmd\_menu和cmd\_tftp到我们uboot的common目录

# We always have this since drivers/ddr/fs/interactive.c needs it

obj-y += cli\_simple.o

obj-y += cli.o

obj-y += cli\_readline.o

obj-y += command.o

obj-y += s\_record.o

obj-y += xyzModem.o

obj-y += cmd\_menu.o

obj-y += cmd\_tftp.o

编译出现错误：找不到console\_buffer

在include/common.h中添加

int run\_command\_list(const char \*cmd, int len, int flag);

extern char console\_buffer[];

'BootFile' undeclared

net.h (include) line 429 :添加

extern char BootFile[128];

undefined reference to `TQ\_readline'

在common/cli\_readlin.c中添加

int cli\_readline(const char \*const prompt)

{

/\*

\* If console\_buffer isn't 0-length the user will be prompted to modify

\* it instead of entering it from scratch as desired.

\*/

console\_buffer[0] = '\0';

return cli\_readline\_into\_buffer(prompt, console\_buffer, 0);

}

int TQ\_readline(char \*prompt)

{

/\*

\* If console\_buffer isn't 0-length the user will be prompted to modify

\* it instead of entering it from scratch as desired.

\*/

console\_buffer[0] = '\0';

if(prompt) {

strcpy(console\_buffer,prompt);

}

prompt = NULL;

return cli\_readline\_into\_buffer(prompt, console\_buffer, 0);

}

common/cmd\_menu.c:59: undefined reference to `eth\_random\_addr'

发现在net.h中该函数名字已经改变为

net\_random\_ethaddr

修改cmd\_menu.c59行eth\_random\_addr((uint8\_t \*)addr);为net\_random\_ethaddr

include/net.h:795: undefined reference to `rand\_r'

common.h (include) line 764 : unsigned int rand\_r(unsigned int \*seedp);

undefined reference to `NetLoop'

已经变为net\_loop()

IOMUXC模块

## IOMUX 概述

#### IOMUX Hardware Operation：

配置各个引脚的复用功能，可配置成功能引脚，也可配置成普通IO。

中断引脚应该配置成普通IO口，并且设置为输入io引脚。

SW\_MUX\_CTL-Selects the primary or alternate function of a pin. Also enables  
loopback mode when applicable.  
• SW\_SELECT\_INPUT-Controls pin input path. This register is only required when  
multiple pads drive the same internal port.  
• SW\_PAD\_CTL-Control pad slew rate, driver strength, pull-up/down resistance, and  
so on.

#### IOMUX Source Code Structure：

|  |  |
| --- | --- |
| pinctrl-imx.c | i.MX pinctrl core driver |
| pinctrl-imx6q.c | i.MX 6Quad/DualLite pinctrl driver |

IOMUX Programming Interface：  
• imx-pinctrl.txt in Documentation/devicetree/bindings/pinctrl/fsl  
• imx6q-pinctrl.txt in Documentation/devicetree/bindings/pinctrl/fsl  
 IOMUX Controller (IOMUXC)

Every module requires a specific pad setting (such as pull up, keeper, and so on), and for  
each pad, there are up to 8 muxing options (called ALT modes).  
MX6Q\_PAD\_KEY\_COL1\_\_GPIO\_4\_8, //GPIO\_VOL-

(((u64)( 0x0200) << 0 | \ //MUX\_CTRL\_OFS\_SHIFT

((u64)( 5) << 36 | \ //MUX\_MODE\_SHIFT

((u64)( 0x05D0) << 12 \ //MUX\_PAD\_CTRL\_OFS\_SHIFT

((u64)( 0) << 41 | \ // MUX\_PAD\_CTRL\_SHIFT

((u64)( 0x0000) << 24 | \ //MUX\_SEL\_INPUT\_OFS\_SHIFT

((u64)(0) << 59| \ // MUX\_SEL\_INPUT\_SHIFT

(u64)(1<<17) << 41 //MUX\_PAD\_CTRL\_SHIFT

#define MX6Q\_IOMUXC\_BASE\_ADDR (0x02000000 + 0x80000 + 0x60000)=20E0000

\_\_init mx6\_map\_io(void)

mxc\_iomux\_v3\_init(IO\_ADDRESS(MX6Q\_IOMUXC\_BASE\_ADDR));

base = iomux\_v3\_base;// 20E0000

int mxc\_iomux\_v3\_setup\_pad(iomux\_v3\_cfg\_t pad)

{

u32 mux\_ctrl\_ofs = (pad & MUX\_CTRL\_OFS\_MASK) >> MUX\_CTRL\_OFS\_SHIFT;

u32 mux\_mode = (pad & MUX\_MODE\_MASK) >> MUX\_MODE\_SHIFT;

u32 sel\_input\_ofs = (pad & MUX\_SEL\_INPUT\_OFS\_MASK) >> MUX\_SEL\_INPUT\_OFS\_SHIFT;

u32 sel\_input = (pad & MUX\_SEL\_INPUT\_MASK) >> MUX\_SEL\_INPUT\_SHIFT;

u32 pad\_ctrl\_ofs = (pad & MUX\_PAD\_CTRL\_OFS\_MASK) >> MUX\_PAD\_CTRL\_OFS\_SHIFT;

u32 pad\_ctrl = (pad & MUX\_PAD\_CTRL\_MASK) >> MUX\_PAD\_CTRL\_SHIFT;

if (mux\_ctrl\_ofs)

\_\_raw\_writel(mux\_mode, base + mux\_ctrl\_ofs);//5,即ALT5，GPIO[8] of instance: gpio4

if (sel\_input\_ofs)

\_\_raw\_writel(sel\_input, base + sel\_input\_ofs);

if (!(pad\_ctrl & NO\_PAD\_CTRL) && pad\_ctrl\_ofs)// NO\_PAD\_CTRL意思是不需要配置改寄存器

*\_\_raw\_writel(pad\_ctrl, base + pad\_ctrl\_ofs);*

return 0;

}

## GPIO

配置普通IO是输入还是输出，读写IO口电平。

论坛里的关于“客户自动运行自己的程序”不太适合手里的资源，我用的镜像是他们最新发布的，琢磨过程如下、完整启动后，用“ps”查看运行中的进程，如下  
  PID TTY          TIME CMD  
  910 ttySAC0  00:00:00 login  
  911 ttySAC0  00:00:00 sh  
  912 ttySAC0  00:00:00 touchinteractio  
  916 ttySAC0  00:00:00 ps  
看起来默认启动的程序是touchinteractio  
2、cd / 切到根目录下 查找程序出现的地方（grep "touchinteractio" \* -nr）  
etc/embedsky\_conf:34:/test\_file/touchinteraction &  
home/root/.bash\_history:168:cd touchinteraction/  
home/root/.bash\_history:173:./touchinteraction   
home/root/.bash\_history:175:./touchinteraction   
home/root/.bash\_history:177:./touchinteraction   
3、修改etc/embedsky\_conf 34行为自己的应用程序就可以啦  
  
启动过程大同小异，顺藤摸瓜反推一下