软件开发指南
ECK20 核心板

成都亿佰特电子科技有限公司
Chengdu Ebyte Electronic Technology Co.,Ltd.

目录

免责申明和版权公告

本文中的信息，如有变更，恕不另行通知。 文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反 言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

文中所得测试数据均为亿佰特实验室测试所得，实际结果可能略有差异。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

最终解释权归成都亿佰特电子科技有限公司所有。

注 意 ：

由于产品版本升级或其他原因，本手册内容有可能变更。亿佰特电子科技有限公司保留在没有任何通知或者提示的情况下对本手册的内容进行修改的权利。本手册仅作为使用指导，成都亿佰特电子科技有限公司尽全力在本手册中提供准确的信息，但是成都亿佰特电子科技有限公司并不确保手册内容完全没有错误，本手册中的所有陈述、信息和建议也不构成任何明示或暗示的担保。

1. 开发环境准备

本章主要介绍基于 ECK20 开发板使用开发前需要的一些软硬件环境，包括必要的硬件配置，开发、烧录所需环境介绍。

通过阅读本章节，您将了解相关硬件工具，软件开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。

主机：i7-12700 + win11 22H2

虚拟机：Ubuntu 18.04

1.1. 硬件准备

硬件连接方式

硬件设备	核心板标注	备注
供电	U1A	
调试串口	U1B: uart1	波特率 115200，数据位 8，停止位 1，无奇偶校验，无硬件流控

1.1.1. 启动方式

参考硬件核心板原理图`iMX6U CFG`页面`FUSE MAP`与底板实际情况配置。

BMODE[1:0]	BOOT TYPE
00	Boot From Fuses
01	Serial Downloader
10	Internal Boot (Development)
11	Reserved

对于从 USB 下载而言，即`Serial Downloader`；一般通过配置 pin 脚状态从内部启动系统，也即`Internal Boot`，更多启动模式的详细情况请参考 NXP 官方手册《IMX6ULLRM.pdf》中 5.1 章节，该文件位于：`01_Documents/Datasheet/NXP/CPU`当中。下方列举了两种情况：

1. SD 卡一般为 BOOT_MODE1：ON；BOOT_MODE0：OFF；BOOT_CFG1[6]：ON。
2. USB 下载为 BOOT_MODE1：OFF；BOOT_MODE0：ON。

1.2. 虚拟机软件部署

本章节将介绍如何搭建 i.MX6UL 的开发环境。通过阅读本章节，您将了解相关硬件工具，软件开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。

1.2.1. 虚拟机配置

请自行根据使用的虚拟机软件配置虚拟机使用桥接的方式联网，并打开目录共享功能。

1.2.2. 配置编译环境

推荐 Ubuntu 18.04 64bit。

安装软件：

```
sudo apt update && sudo apt upgrade -y
sudo apt install make gcc g++ flex bison xz-utils libncurses5-dev device-tree-compiler netpbm bc libssl-dev lzop libncursesw5-dev -y
```

添加并验证交叉编译链：

```
mkdir -p ~/.local/share/
tar -xavf gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf.tar.xz -C ~/.local/share/
cd ~/.local/share/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin
./arm-linux-gnueabihf-gcc -v
```

创建工作目录：

```
mkdir ~/work
```

1.2.3. 开启虚拟机的 TFTP 服务

后面调试阶段往往会使用 TFTP 加载内核启动，而不是每次去烧写 flash 设备，因此要先安装并开启 Ubuntu 中的 TFTP 服务，使用如下命令安装 TFTP 服务：

```
sudo apt install tftpd-hpa -y
```

默认情况下，将分享/var/lib/tftpboot 目录下的文件，且只能下载，不能上传，如果想个性化配置，请参考官方文档：https://help.ubuntu.com/community/TFTP，本文档已经修改路径为/srv/tftp。

添加用户到 tftp 组并修改共享目录权限：

```
sudo usermod -a -G tftp $(whoami)
sudo chmod 775 /srv/tftp/
sudo chgrp tftp /srv/tftp/
```

如果没有执行上述操作，可能出现权限问题，非 root 权限下无法写入文件到共享目录。

1.2.4. 开启虚拟机的 NFS 服务

后面调试阶段往往需要用 NFS 共享文件或者直接通过 NFS 挂载文件系统，因此要先安装并开启 Ubuntu 中的 NFS 服务，使用如下命令安装 NFS 服务：

```
sudo apt install nfs-kernel-server -y
```

等待安装完成，安装完成以后设置需要导出的目录，此处直接使用 tftp 默认目录，用户可自行修改，也可使用顺手的文本编辑器修改/ext/exports 文件，配置方法参考 man 手册：

```
echo "/srv/tftp *(rw,sync,no_subtree_check,no_root_squash,fsid=root)" | sudo tee -a /etc/exports
man exports
```

1.2.5. 开启虚拟机的 SSH 服务

开启 Ubuntu 的 SSH 服务以后我们就可以在 Windwos 下使用 ssh 客户端软件登陆到 Ubuntu，比如使用 MobaXterm、VSCode，Ubuntu 下使用如下命令开启 SSH 服务：

```
sudo apt install openssh-server -y
```

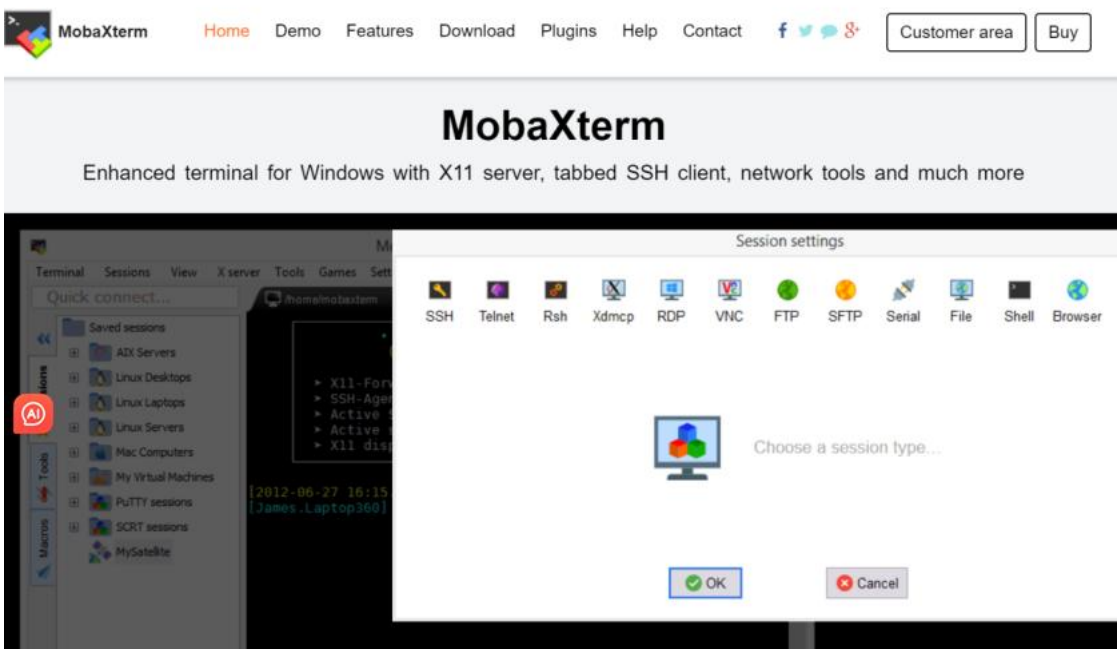
上述命令安装 ssh 服务，ssh 的配置文件为/etc/ssh/sshd_config，使用默认配置即可，更多配置参考 sshd_config 相关的 man 手册。

1.3. 主机软件安装

1.3.1. MobaXterm

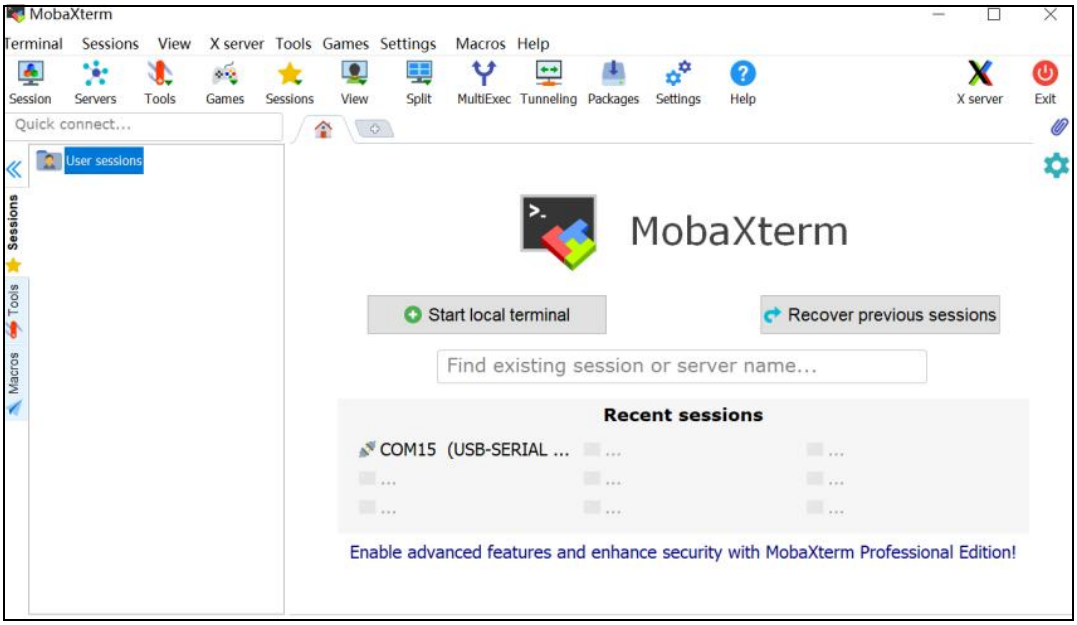
1.3.1.1. 软件下载安装

MobaXterm 是一款终端软件，功能强大而且免费，推荐使用此软件作为终端调试软件，MobaXterm 软件在其官网下载即可，地址为 https://mobaxterm.mobatek.net/，如图所示：

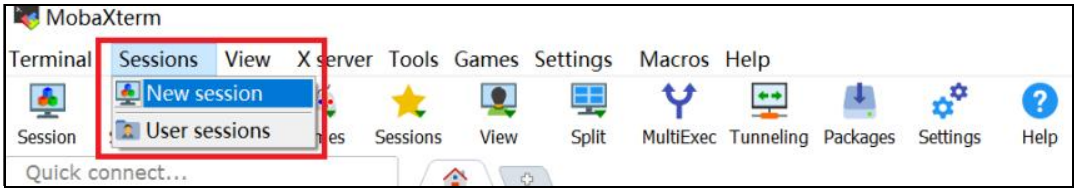


1.3.1.2. 软件使用

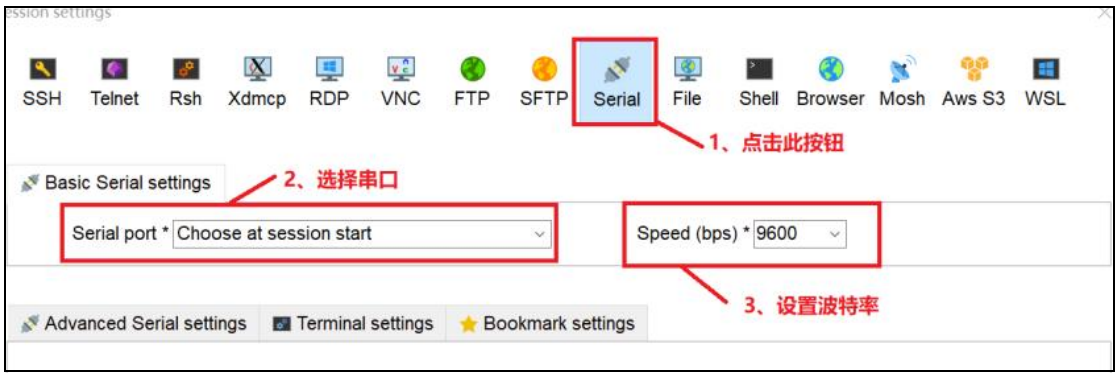
软件界面如图所示：



点击菜单栏中的“Sessions->New session”按钮，打开新建会话窗口，如图所示



建立 Serial 连接，也就是串口连接，因为我们使用 MobaXterm 的主要目的就是作为串口终端使用。点击图中的“Serial”按钮，打开串口设置界面，如图所示：

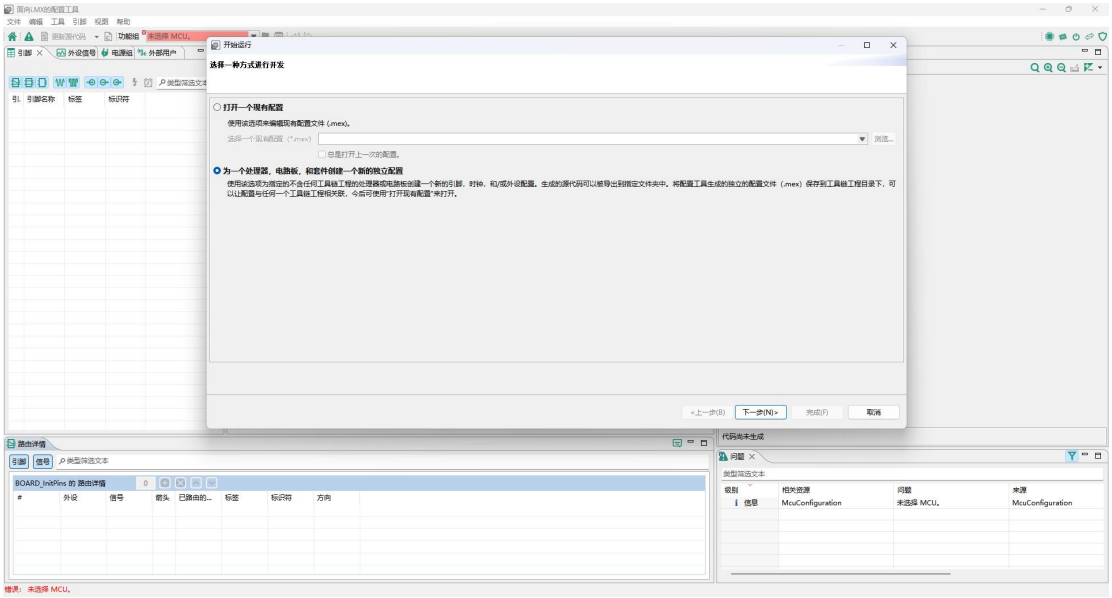


打开串口设置窗口以后先选择要设置的串口号，因此要先用串口线将开发板连接到电脑上，由于 imx6ull 默认使用的串口波特率为 115200，需要设置波特率为 115200。

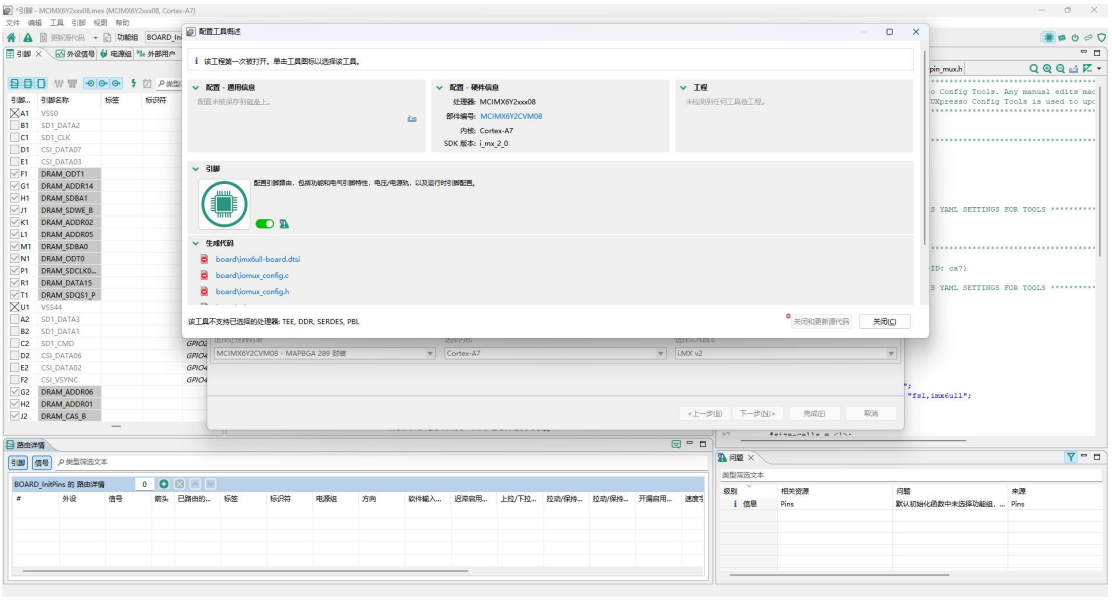
1.3.2. Config Tools for i.MX v15

这是 NXP 官方用于配置 i.MX 系列 CPU 的工具，主要用于配置设备树中的 iomuxc 分配。

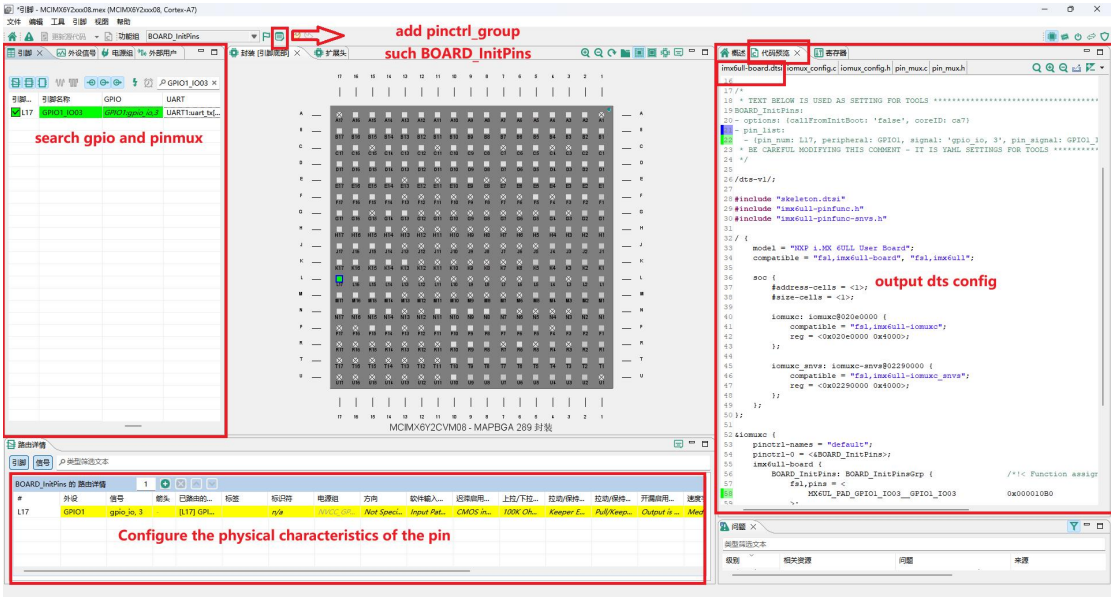
打开软件自动出现如下界面：



点击下一步后选择搜索 MCIMX6Y2xxx08，完成后显示如下：



关闭弹窗后，主要需要关注如下部分：



1.4. 数据传输

在前面章节我们已经配置好了虚拟机部分各种网络服务，现在只需要了解如何在客户端与开发板的使用即可。

1.4.1. TFTP 客户端

一般情况下，开发板只需要使用下载功能即可，此时可以简单的使用 curl 工具下载：

安装 curl：

```
apt install curl -y
```

使用 curl 下载文件(仅限 Linux 版 curl，windows 可选择使用 filezilla)：

```
curl -o /path/dist_file tftp://HOST/src_file
```

1. /path/dist_file：要下载到哪里，输出文件名字的名字与路径。

2. HOST：虚拟机的 ip 地址，需要注意这个地址是否可以访问。

3. src_file：文件目录以及名字，需要注意的是不需要添加配置中的"TFTP_DIRECTORY"部分，比如文件存放在`/srv/tftp/aaa/bbb/ccc`，src_file 值为"aaa/bbb/ccc"。

1.4.2. NFS 客户端

一般情况下，开发板只需要简单的使用 mount 命令即可挂载 NFS 文件系统，但是需要安装基本的支持：

```
apt install nfs-common -y
```

挂载 NFS：

```
mount -t nfs HOST:/path /mnt_point
```

1. HOST：虚拟机的 ip 地址，需要注意这个地址是否可以访问。

2. path：想要挂载的 NFS 服务器中的路径，比如/srv/tftp/aaa/bbb。

3. mnt_point：想要把 NFS 文件系统挂载到哪里，比如/mnt。

1.4.3. SSH

一般来说我们也可以使用 SSH 自带的 SFTP 与 SCP 功能进行文件下载，但直接使用命令上传下载的情况不多，更多使用第三方工具进行，如使用 MobaXterm 登录 ssh 后的 SFTP 功能，或者 VSCode 使用 remote-SSH 插件实现开发过程中的下载功能，使用方式都是鼠标放在需要下载的文件上右键，然后选择下载。

1.4.4. U 盘/SD 卡

对于 SD 卡，需要使用读卡器，实际用法和 U 盘一致。

1.4.4.1. 从虚拟机拷贝数据

需要根据使用的虚拟机软件先将 U 盘/SD 卡传递到虚拟机当中，一般来说会自动挂载，在文件管理器中可以直接访问，拷贝数据完毕后在文件管理器中卸载设备，然后从虚拟机中移除设备即可。

如果没有自动挂载，可以使用下方的两种指令手动挂载：

1. 方式一

```
udisksctl mount --block-device /dev/sdXY
```

2. 方式二

```
sudo mount /dev/sdXY /mnt_point
```

这种情况需要对 U 盘/SD 卡分区有一点了解，才可以指定使用 sdXY 中的 Y 值，X 值一般是'b'，但是也需要自行通过 lsblk 的返回信息判断情况。

使用方式一挂载，无需 root 权限即可访问，也不用指定路径，但是需要系统中有安装了`udisks2`这个程序包。

手动卸载对应如下命令：

1. 方式一

```
udisksctl unmount --block-device /dev/sdXY

udisksctl power-off --block-device /dev/sdXY
```

2. 方式二

```
sudo umount /mnt_point
```

1.4.4.2. 拷贝数据到开发板

对于开发板，一般会进行精简，不一定有 udisks2 包，也不一定有一个 GUI 界面，有界面也不一定支持自动挂载，往往需要从命令行执行操作。

1. 挂载

```
mount /dev/sdXY /mnt_point
```

2. 拷贝

```
cp /mnt_point/file /out_path/out_file

cp -ra /mnt_point/dir /out_path/out_dir
```

3. 取消挂载

```
umount /mnt_point
```

2. 如何烧录更新系统镜像

本章主要介绍基于 ECK20 核心板的烧录更新方式。

2.1. 官方工具烧录

本章节描述如何使用 NXP 官方工具 UUU（aka: MFGTools v3）实现 USB 快速烧录功能。UUU 软件资源已存放在目录： “04_Tools/uuu”。

使用官方工具烧录前需要的硬件设置

设置选项	设置内容
启动方式配置	配置启动方式为：`Serial Downloader`烧录模式
系统	官方工具不支持 win7，请使用 win 10 及以上的 windows 系统或 Linux 发行版
接线	供电，USB OTG 接口连接开发主机

1. 切换启动方式为 Serial Downloader
2. 打开一个命令行终端，进入 uuu 目录
3. 执行烧录指令等待返回，成功将显示绿色的 Done 且 Success 显示 1。

当用户编译生成自己的固件时，只需相应覆盖 files 目录中的文件即可，也可以修改 auto 文件指定新的固件路径名称。

烧录系统到 eMMC 中：

```
./uuu eck20_6Y2xA_flash_emmc.auto      # 邮票孔

./uuu eck20_6Y2xC_flash_emmc.auto      # BTB
```

烧录系统到 NAND 中：

```
./uuu eck20_6Y2xA_flash_nand.auto      # 邮票孔

./uuu eck20_6Y2xC_flash_nand.auto      # BTB
```

```
PS E:\SMB\imx6ULx\nxp\tools\burn_tool\uuu> .\uuu.exe -lsusb
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.5.165-0-g7347a80

Connected Known USB Devices
        Path     Chip    Pro     Vid     Pid     BcdVersion
        ==================================================
        1:6      MX6ULL  SDP:    0x15A2 0x0080   0x0001


PS E:\SMB\imx6ULx\nxp\tools\burn_tool\uuu> .\uuu.exe .\eck20_6Y2xA_flash_emmc.auto
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.5.165-0-g7347a80

Success 1    Failure 0


1:6     17/17 [Done                                ] FBK: done


PS E:\SMB\imx6ULx\nxp\tools\burn_tool\uuu>
```

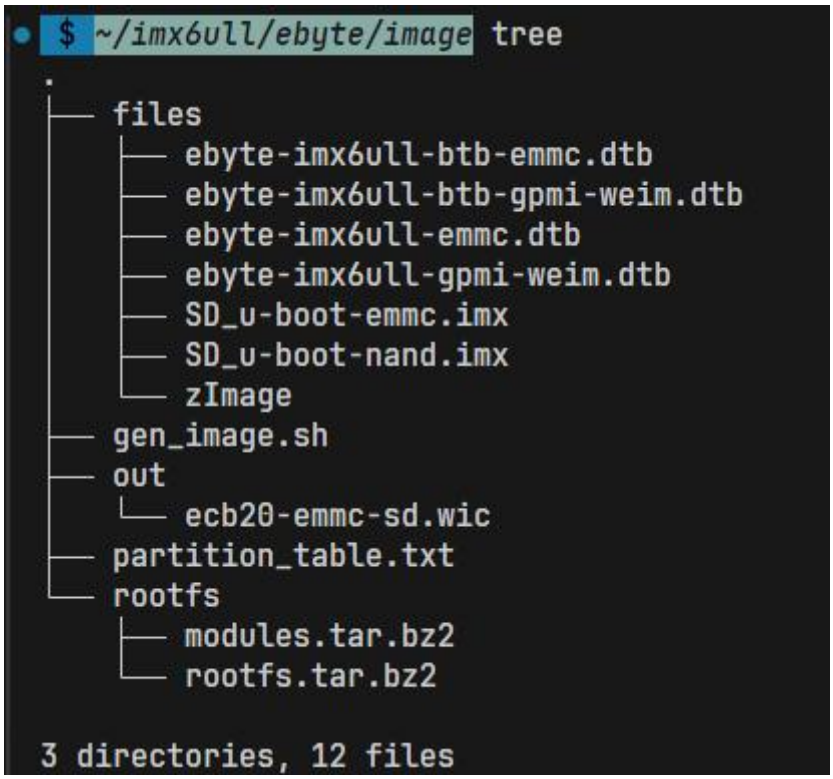
4. 切换启动方式为相应 Internal Boot 模式。

2.1.1. 注意事项

需要注意的是，在 USB 下载模式下，一定不能在上电之前连接 SD 卡，否则会尝试从 SD 卡启动，而不会进入下载模式，导致主机无法识别开发板。如果需要连接 SD 卡使用，请上电能够查询到 usb 设备后再次接入 SD 卡。

2.2. 制作 SD 卡启动卡

亿佰特已经提供了制作镜像与 SD 启动卡的脚本，可直接使用或参考，其位于 04_Tools/SD_Tools。



files 目录中存放所需要写入镜像或者 SD 卡的固件，rootfs 中存放所使用的 Ubuntu 镜像，该镜像可以参考后续章节制作。partition_table.txt 是划分 SD 卡或者镜像的分区表，除设备树文件可以任意更换名字，其他文件与上图名字位置不同时都需要修改脚本。

2.2.1. 三方工具镜像烧录

使用工具烧录，需要首先制作一个可烧录的镜像。按照 nxp 官方文档《i.MX Linux User's Guide.pdf》第 4.3 章节中对应的`image layout`表中的镜像布局要求制作，该文档位于 01_Documents/Datasheet/NXP/imx-yocto-LF5.10.9_1.0.0 中。

Table 1. Image layout

Start address (sectors)	Size (sectors)	Format	Description
0x400 bytes (2)	0x9FFC00 bytes (20478)	RAW	i.MX 6 and i.MX 7 U-Boot and reserved area
0x8400 (66)	0x9F7C00 (20414)	RAW	i.MX 8M Quad and i.MX 8M Mini imx-boot reserved area
0x8000 (64)	0x9F800 (20416)	RAW	i.MX 8QuadMax/8QuadXPlus/ 8M Nano/8M Plus/ 8DXL/8DualX
0xa00000 bytes (20480)	500 Mbytes (1024000)	FAT	Kernel Image and DTBs
0x25800000 bytes (1228800)	Remaining space	Ext3/Ext4	Rootfs

4.3.1 Preparing the card

从上图可以知道，对于 i.MX6ULL 而言，应该将 U-Boot 存放在 SD 卡或 eMMC 开头 0x400 bytes 的位置，也就是 1K 的偏移，可用大小为 10M。然后有 500M 的空间为 FAT 格式，存放内核与设备树，剩余空间为文件系统存放处。

但实际上内核与设备树完全无需 500M 这么大的空间，可自行根据实际情况划分即可，一般内核大小在 10M 左右。

2.2.1.1. 制作镜像

使用现有脚本制作 SD 卡镜像：

```
sudo ./gen_image.sh nand

sudo ./gen_image.sh emmc
```

注意：如果在命令最后增加一个参数`btb`，则会构建适用于 BTB 连接器的开发板。

```
me@me-virtual-machine:~/work/image$ ./gen_image.sh --help
Usage: sudo ./gen_image.sh <emmc|nand>
me@me-virtual-machine:~/work/image$ sudo ./gen_image.sh nand
Using NAND boot partition.
6+0 records in
6+0 records out
6442450944 bytes (6.4 GB, 6.0 GiB) copied, 4.22515 s, 1.5 GB/s
Checking that no-one is using this disk right now ... OK

Disk /dev/loop9: 6 GiB, 6442450944 bytes, 12582912 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

>>> Script header accepted.
>>> Script header accepted.
>>> Created a new DOS disklabel with disk identifier 0xa6c46d8a.
/dev/loop9p1: Created a new partition 1 of type 'EFI (FAT-12/16/32)' and of size 35 MiB.
/dev/loop9p2: Created a new partition 2 of type 'Linux' and of size 6 GiB.
/dev/loop9p3: Done.

New situation:
Disklabel type: dos
Disk identifier: 0xa6c46d8a

Device       Boot Start      End  Sectors Size Id Type
/dev/loop9p1 *    10240    81919    71680  35M ef EFI (FAT-12/16/32)
/dev/loop9p2      81920 12582911 12500992   6G 83 Linux

The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
mkfs.fat 4.1 (2017-01-24)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 1562624 4k blocks and 390912 inodes
Filesystem UUID: 7f3e2462-ffd7-4b76-afc7-580a7af2e349
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

waitting for copy rootfs...
waitting for copy kernel...
waitting for copy u-boot ./files/SD_u-boot-nand.imx...
1310+0 records in
1310+0 records out
670720 bytes (671 kB, 655 KiB) copied, 0.00360502 s, 186 MB/s
create image file: ./out/nand-ubuntu-sd.wic
me@me-virtual-machine:~/work/image$
```

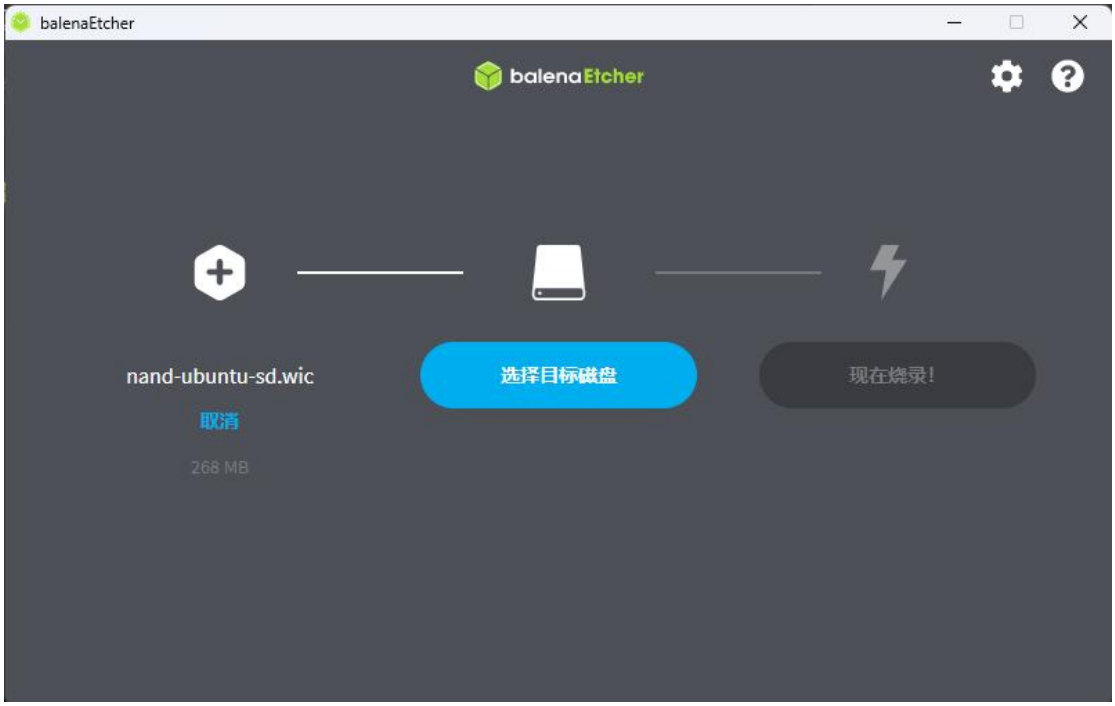
输出文件位于当前目录的 out 目录下。

```
me@me-virtual-machine:~/work/image$ ls
files  flsh_SD_Card.sh  gen_image.sh  out  partition_table.txt  rootfs
me@me-virtual-machine:~/work/image$ ls out/
emmc-ubuntu-sd.wic  nand-ubuntu-sd.wic
me@me-virtual-machine:~/work/image$
```

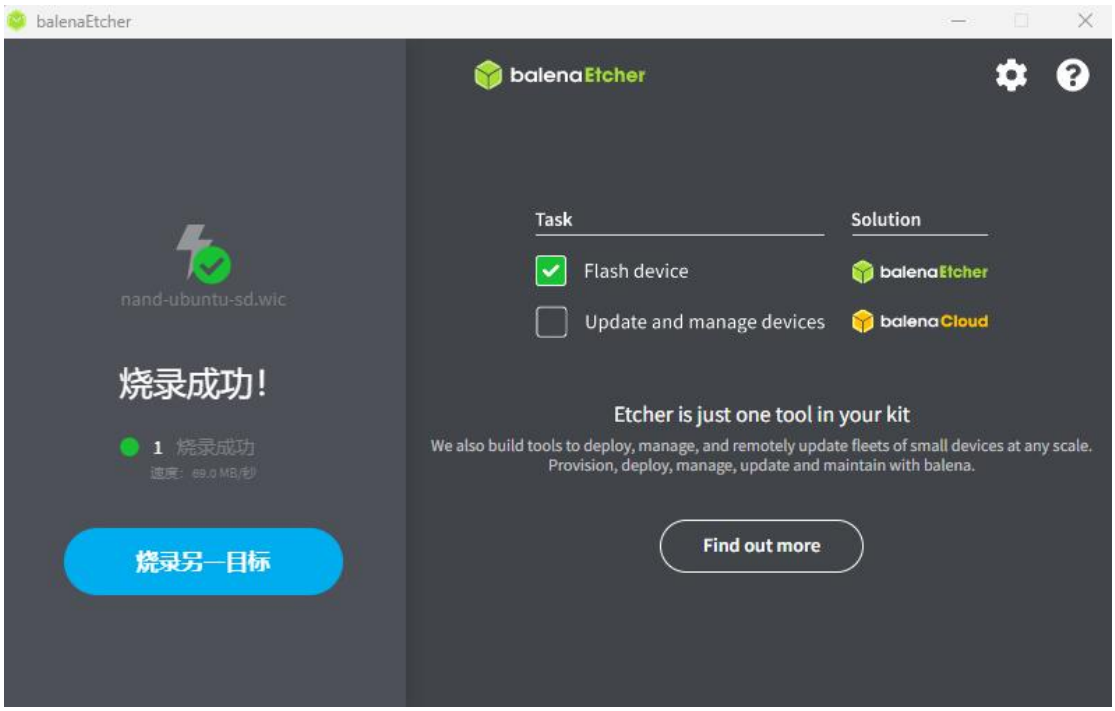
2.2.1.2. 烧录镜像

使用第三方的镜像烧录工具将上一小节中制作出来的镜像写入 SD 卡，在 04_Tools/中已经包含了一个烧写工具：balenaEtcher-Portable-1.18.11.exe。

选择从文件烧录，文件选择之前制作的 wic 文件。



选择目标磁盘，然后点击现在烧录，开始烧录并等待烧录完成后移除设备即可使用 SD 卡作为启动设备。



2.2.2. 直接制作启动卡

直接制作 SD 启动卡的方式类似三方工具镜像烧录中制作镜像的步骤，只不过从写入镜像文件变成了直接写入 SD 卡设备。

1. 使用如下命令查看设备信息：

```
lsblk
```

2. 执行烧写：

```
sudo ./flsh_SD_Card.sh /dev/sdb nand
```

2.3. 现有系统更新

在之前章节“官方工具烧录”和“三方工具烧录 SD 卡”中我们已经成功固化了 Linux 文件系统，系统也已经成功启动了。但有时候我们开发过程中，尝试自己编译制作 uboot、内核与文件系统，后续我们只需要更新 uboot、设备树和文件系统中的某一个，特别是在开发过程中可能要频繁更新，为了节省开发时间，我们可以采用单独更新的方法来更新我们需要更新的部分，而不必花更长的时间去重新制作一张 SD 系统启动卡或者使用脚本固化系统到 eMMC 或 NAND 中。

开发主机和开发板文件互传的方法可以参考本文档“数据传输”章节。

2.3.1. eMMC

如果用户使用的是 eMMC 的核心板，想要在已有的系统中更新固件。我们需要从 eMMC 启动或者从 SD 卡启动替换自己固件。这里简要总结下 eMMC 分区表。

eMMC 分区表

分区名	作用
/dev/mmcblk1boot0	boot 分区 0，存放 u-boot
/dev/mmcblk1boot1	boot 分区 1，默认不用
/dev/mmcblk1p1	fat32 格式，存放内核与设备树
/dev/mmcblk1p2	ext4 格式，存放根文件系统

2.3.1.1. U-boot

从 eMMC 启动系统，或者从 SD 卡启动系统来更新 uboot 到 eMMC 中。

1. 将编译生成的 u-boot-dtb.imx 文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下。

2. 解除 mmcblk1boot0 分区写保护

```
echo 0 > /sys/block/mmcblk1boot0/force_ro
```

3. 清空 mmcblk1boot0 分区旧数据

```
dd if=/dev/zero of=/dev/mmcblk1boot0
```

4. 写入 u-boot 数据到 mmcblk1boot0 分区

```
dd if=u-boot.imx of=/dev/mmcblk1boot0 bs=1k seek=1 conv=fsync
```

5. 打开 mmcblk1boot0 分区写保护

```
echo 1 > /sys/block/mmcblk1boot0/force_ro
```

6. 使能启动分区

```
mmc bootpart enable 1 1 /dev/mmcblk1
```

2.3.1.2. Kernel + 设备树

按照上面的分区表，可知内核与设备树存储在一个 fat32 格式的文件系统之中。

1. 将编译生成的内核与设备树文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下。

2. 挂载 mmcblk1p1 分区

```
mount /dev/mmcblk1p1 /boot
```

3. 拷贝内核与设备树到 mmcblk1p1 分区。

```
cp zImage /boot

cp *.dtb /boot

sync
```

4. 取消挂载分区

```
umount /boot
```

5. 重启系统

```
reboot
```

2.3.1.3. 根文件系统

一般来说只能使用 SD 卡启动来更新 eMMC 的文件系统，正在运行的一个系统不能把自己给格式化！

1. 通过 nfs/tftp/U 盘等方式拷贝制作好的根文件系统压缩包（rootfs.tar.bz2）到 eMMC/SD 卡系统中当前目录下。

2. 挂载 mmcblk1p2 分区

```
mount /dev/mmcblk1p2 /mnt
```

3. 清空 mmcblk1p2 分区旧数据

```
rm -rf /mnt/*
```

4. 解压新的根文件系统到 mmcblk1p2 分区

```
tar -xavf rootfs.tar.bz2 -C /mnt

sync
```

5. 取消挂载分区

```
umount /mnt
```

6. 重启系统

```
reboot
```

2.3.2. NAND

如果用户使用的是 NAND 的核心板，想要在已有的系统中更新固件。我们需要从 NAND 启动或者从 SD 卡启动替换自己固件。这里简要总结下 NAND 分区表。

NAND 分区表

分区名	作用
/dev/mtdblock0	boot 分区 0，存放 u-boot，10M
/dev/mtdblock1	Kernel 分区，存放 linux 内核，12M
/dev/mtdblock2	dtb 分区，存放 linux 设备树，2M
/dev/mtdblock3	ubi 分区，存放根文件系统，剩余全部空间

需要注意的是，如果从 NAND 启动，默认情况下其根分区是只读挂载的，无法拷贝文件到其系统之中，需要在 u-boot 中修改启动参数调整为可读写的方式挂载根分区。

```
setenv bootargs console=ttymxc0,115200 ubi.mtd=nandrootfs root=ubi0:rootfs rootfstype=ubifs rw ${mtdparts}
```

但是如果使用 nfs 或者 U 盘的方式，也可以无需把文件拷贝到 NAND 中，通过挂载的方式来绕过根分区的只读限制。

2.3.2.1. U-boot

由于 i.MX6ULL 自身 BootROM 的限制，需要使用 nxp 官方提供的 kobs-ng 软件进行 u-boot 更新。

1. 将编译生成的 u-boot-dtb.imx 文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下或者进入 U 盘/nfs 挂载的目录。

2. 擦除旧有的 u-boot

```
flash_erase /dev/mtd0 0 0
```

3. 重新写入更新的 u-boot

```
kobs-ng init -x -v --chip_0_device_path=/dev/mtd0 u-boot-dtb.imx

sync
```

4. 重启系统

```
reboot
```

2.3.2.2. Kernel

按照上文中分区情况可知，kernel 存放在 mtd1 分区，无文件系统存在。

1. 将编译生成的 zImage 文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下或者进入 U 盘/nfs 挂载的目录。

2. 擦除旧有的 kernel

```
flash_erase /dev/mtd1 0 0
```

3. 重新写入更新的 zImage

```
nandwrite -p /dev/mtd1 zImage

sync
```

4. 重启系统

```
reboot
```

2.3.2.3. dtb

按照上文中分区情况可知，设备树存放在 mtd2 分区，无文件系统存在。

1. 将编译生成的 zImage 文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下或者进入 U 盘/nfs 挂载的目录。

2. 擦除旧有的设备树

```
flash_erase /dev/mtd2 0 0
```

3. 重新写入更新的设备树

```
nandwrite -p /dev/mtd2 ebyte-imx6ull-gpmi-weim.dtb

sync
```

4. 重启系统

```
reboot
```

2.3.2.4. 根文件系统

一般来说只能使用 SD 卡启动来更新 NAND 的文件系统，正在运行的一个系统不能把自己给格式化！

1. 通过 nfs/tftp/U 盘等方式拷贝制作好的根文件系统压缩包（rootfs.tar.bz2）NAND/SD 卡系统中当前目录下或者进入 U 盘/nfs 挂载的目录。

2. 擦除旧有的文件系统

```
flash_erase /dev/mtd3 0 0
```

3. 重建 ubi 文件系统

```
ubiformat /dev/mtd3

ubiattach /dev/ubi_ctrl -m 3

ubimkvol /dev/ubi0 -Nrootfs -m
```

4. 挂载新的 ubi 文件系统

```
mount -t ubifs ubi0:rootfs /mnt/
```

5. 解压并拷贝更新的文件系统到 ubi 文件系统中

```
tar -xavf rootfs.tar.bz2 -C /mnt

sync
```

6. 重启系统

```
reboot
```

2.3.3. SD 卡

由于 SD 卡与 eMMC 都可归类到 MMC 设备，其使用方式与 eMMC 也类似。

2.3.3.1. U-boot

通过读卡器将 SD 卡传入虚拟机系统，在虚拟机中直接写入更新的 U-Boot 即可。

1. 进入存放 u-boot-dtb.imx 文件的目录下。

2. 查找 SD 卡设备

```
lsblk
```

```
root@ebyte-ubuntu:~# lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda         8:0    1 29.7G  0 disk
├─sda1      8:1    1   35M  0 part
└─sda2      8:2    1 29.7G  0 part
```

3. 写入 u-boot 数据到 SD 卡设备

```
sudo dd if=u-boot.imx of=/dev/sda bs=1k seek=1 conv=fsync

sync
```

2.3.3.2. Kernel + 设备树

通过读卡器将 SD 卡传入虚拟机系统，在虚拟机中直接拷贝文件到 EFI 分区即可。

1. 进入存放内核与设备树文件的目录下。

2. 查找 SD 卡设备

```
lsblk
```

3. 挂载 sda1 分区

```
sudo mount /dev/sda1 /mnt/
```

4. 拷贝内核与设备树到 sda1 分区。

```
cp zImage /mnt

cp *.dtb /mnt

sync
```

5. 取消挂载分区

```
umount /mnt
```

2.3.3.3. 根文件系统

通过读卡器将 SD 卡传入虚拟机系统，在虚拟机中直接拷贝文件到 root 分区即可。

1. 进入存放制作好的根文件系统压缩包（rootfs.tar.bz2）的目录下。

2. 查找 SD 卡设备

```
lsblk
```

3. 挂载 sda2 分区

```
sudo mount /dev/sda2 /mnt/
```

4. 更新文件系统到 sda2 分区。

```
sudo rm -rf /mnt/*

sudo tar -xavf rootfs.tar.bz2 -C /mnt

sync
```

5. 取消挂载分区

```
umount /mnt
```

3. 构建 u-boot

uboot 的全称是 Universal Boot Loader，uboot 是一个遵循 GPL 协议的开源软件，uboot 是一个裸机代码，可以看作是一个裸机综合例程。现在的 uboot 已经支持液晶屏、网络、USB 等高级功能。uboot 官网为 http://www.denx.de/wiki/U-Boot/。

3.1. 获取源码并编译

由于各家芯片原厂都在官方 uboot 的基础上进行了定制，一般情况下，使用芯片厂商提供的版本是一个更好的选择，亿佰特在芯片原厂提供的 uboot 基础上提供了基于自家核心板的使用示例，以便用户更快速的客制化。

在上面“配置编译环境”章节中，我们已经安装了编译所需的各种依赖包，也创建了工作目录`${HOME}/work`，下面解压本公司提供的源代码：

```
tar -xavf uboot.tar.xz -C ~/work

ls ~/work/uboot
```

查看目录情况如下：

```
me@me-virtual-machine:~$ ls work/uboot/
api    build.sh  config.mk  doc      env       include  lib          Makefile  README   tools
arch   cmd       configs    drivers  examples  Kbuild   Licenses     net       scripts
board  common    disk       dts      fs        Kconfig  MAINTAINERS  post      test
me@me-virtual-machine:~$
```

可以看到其中有一个 build.sh 脚本，该脚本由本公司提供，用于编译本公司提供的核心板示例代码，编译完毕后会在 uboot 的顶层目录出现一个 out 目录用于存放所有的输出文件。

```
me@me-virtual-machine:~/work/uboot$ ./build.sh --help
Usage: ./build.sh [n|e]
n: build for NAND boot
e: build for EMMC boot
no arg: just build, not configed
```

执行如下代码编译使用 NAND 的核心板：

```
./build n
```

执行如下代码编译使用 eMMC 的核心板：

```
./build e
```

有时候只是修改了部分与 uboot 配置不相关的文件，使用如下命令快速编译

```
./build
```

编译完毕后可以查看到如下情况。

```
  SHIPPED dts/dt.dtb
  FDTGREP dts/dt-spl.dtb
  CAT     u-boot-dtb.bin
  COPY    u-boot.dtb
  COPY    u-boot.bin
  CFGS    u-boot-dtb.cfgout
  MKIMAGE u-boot-dtb.imx
  CFGCHK  u-boot.cfg
make[1]: Leaving directory '/home/me/work/uboot/out'
me@me-virtual-machine:~/work/uboot$ ls
api    build.sh  compile_commands.json  disk     dts       fs       Kconfig   MAINTAINERS  out     scripts
arch   cmd       config.mk              doc      env       include  lib       Makefile     post    test
board  common    configs                drivers  examples  Kbuild   Licenses  net          README  tools
me@me-virtual-machine:~/work/uboot$ ls out/
arch    disk     examples  Makefile  System.map  u-boot.cfg          u-boot-dtb.cfgout   u-boot.map
board   drivers  fs        net       tools       u-boot.cfg.configs  u-boot-dtb.imx      u-boot-nodtb.bin
cmd     dts      include   scripts   u-boot      u-boot.dtb          u-boot-dtb.imx.log  u-boot.srec
common  env      lib       source    u-boot.bin  u-boot-dtb.bin      u-boot.lds          u-boot.sym
me@me-virtual-machine:~/work/uboot$
```

out 目录包含编译过程生成的所有文件，这些文件大部分对用户来说是不需要接触到的。一般我们只需要使用到`out/u-boot-dtb.imx`文件，就是编译 U-Boot 源码的最终产物，也就是 ECK20 核心板的示例 U-Boot 固件。

3.2. 编译用于从 SD 卡启动的 uboot

可以发现，build.sh 脚本只有编译为 NAND 或者 eMMC 的选项，这时候编译出来的 uboot 如果用于从 SD 卡启动，可能无法使用，特别是使用 NAND 的板子。

亿佰特已经针对这种情况做了定制修改，用户只需要启用从 SD 卡启动的选项即可。

首先使用自己习惯的文本编辑器打开并修改 build.sh 脚本，去除第 20 行的`#`号注释以启用 make menuconfig 功能：

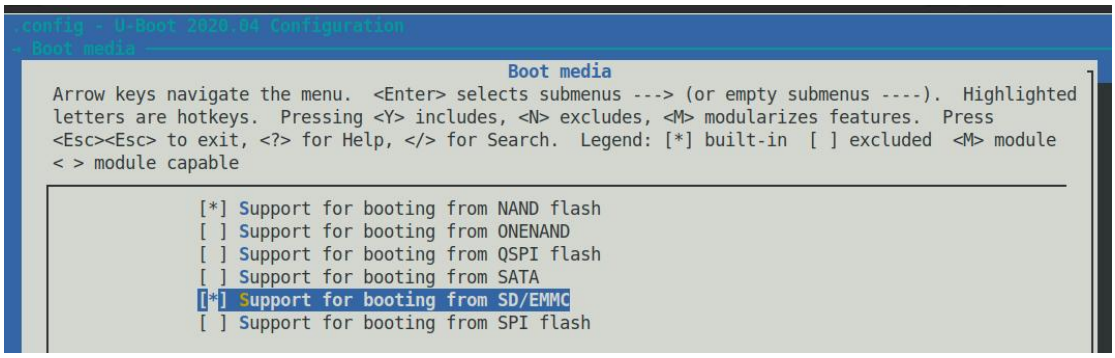
```
1 #!/bin/bash
2
3 if [ "$_" != "$0" ];   then
4         _exit="return"
5 else
6         _exit="exit"
7 fi
8
9 export PATH=${HOME}/.local/share/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/:${PATH}
10 export ARCH=arm
11 export CROSS_COMPILE=arm-linux-gnueabihf-
12
13 function init_config() {
14     # make distclean
15     rm -rf out
16     gen_compile_commands="bear"
17
18     make O=out "$1" || ${_exit}
19
20     # make O=out menuconfig || ${_exit}
21     # make O=out savedefconfig || ${_exit}
22 }
23
```

然后再次针对 eMMC 或者 NAND 的编译：

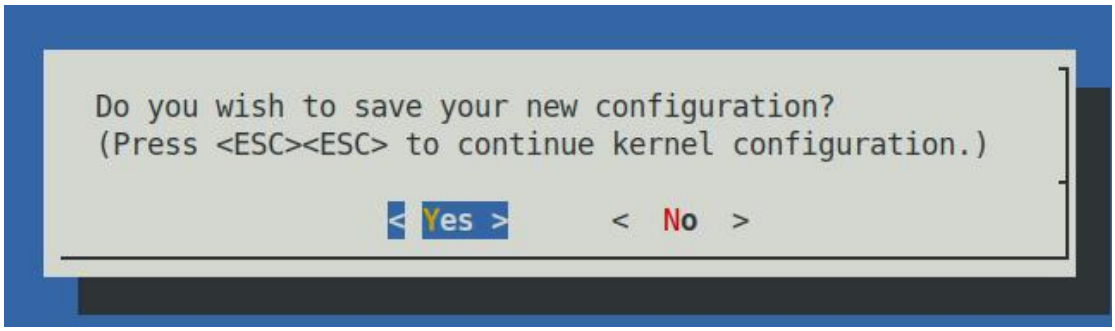
```
./build.sh e
```

```
./build.sh n
```

此时会进入配置界面，使能如下选项`Boot media -> support for booting from SD/EMMC，使能方式为使用方向键移动光标到该选项，然后敲击空格选中即可。



完成使能后移动光标到 Exit 上敲击回车返回，重复多次返回，直到如下界面，选择 Yes 正式退出，然后等待编译完成即可。



3.3. 设置 Uboot 从网络启动系统

从网络启动系统时，往往通过 tftp 下载内核与设备树，然后 nfs 挂载文件系统。从网络启动需要先设置网卡，并配置好服务端（虚拟机）的 nfs 服务和 tftp 服务。

```
setenv ethaddr 00:11:22:33:44:55

setenv ipaddr 192.168.0.136

setenv serverip 192.168.0.63

setenv image zImage

setenv fdt_file 'ebyte-imx6ull-emmc.dtb'

setenv rootpath /srv/tftp/ubuntu
```

1. ethaddr 用于配置 mac 地址，本公司提供的示例已经启用了 u-boot 阶段的随机 mac 地址，可以不用这一条配置。

2. ipaddr 用于配置开发板 ip 地址，该地址与 serverip 应位于同一个网段。

3. serverip 用于配置 tftp 服务器地址，该地址为虚拟机桥接模式所分配得到的地址。

4. image 为所需要引导启动的内核路径。

5. fdt_file 为开发板对应设备树路径。

6. rootpath 为 nfs 挂载文件系统所需的路径，存放有根文件系统。

设置环境变量添加网络启动选项：

```
setenv netargs 'setenv bootargs console=${console},${baudrate} root=/dev/nfs rw nfsroot=${serverip}:${rootpath},nfsvers=3,tcp ip=dhcp'

setenv netboot 'echo Booting from net ...; run netargs; tftp ${loadaddr} ${image}; tftp ${fdt_addr} ${fdt_file}; bootz ${loadaddr} - ${fdt_addr}'

saveenv
```

调用网络启动选项脚本，执行网络启动。

```
run netboot
```

4. 构建 kernel

由于各家芯片原厂都在官方 Linux 内核的基础上进行了定制，一般情况下，使用芯片厂商提供的版本是一个更好的选择，亿佰特在芯片原厂提供的 Linux 内核基础上提供了基于自家核心板的使用示例，以便用户更快速的客制化。

4.1. 获取源码

在上面“配置编译环境”章节中，我们已经安装了编译所需的各种依赖包，也创建了工作目录`${HOME}/work`，下面解压本公司提供的源代码：

```
tar -xavf kernel.tar.xz -C ~/work

ls ~/work/linux
```

查看目录情况如下：

```
me@me-virtual-machine:~/work/linux$ ls
arch      certs     crypto         fs       ipc      kernel    MAINTAINERS      mm       samples   sound  virt
block     COPYING   Documentation  include  Kbuild   lib       MAINTAINERS.NXP  net      scripts   tools
build.sh  CREDITS   drivers        init     Kconfig  LICENSES  Makefile         README   security  usr
me@me-virtual-machine:~/work/linux$
```

4.2. 编译内核

可以看到其中有一个 build.sh 脚本，该脚本由本公司提供，用于编译本公司提供的核心板示例代码，编译完毕后会在 linux 的顶层目录出现一个 out 目录用于存放所有的输出文件。

该脚本使用方式与 u-boot 提供的脚本类似，但是不再区分 eMMC，NAND 还是 SD 卡。不过依然支持在没有进行内核功能变动的情况下快速编译。

执行如下命令配置内核并编译：

```
./build.sh <any string>
```

此时只需要传递任意参数给脚本即可配置并编译内核。

执行如下命令快速编译：

```
./build.sh
```

编译完毕后可以查看到如下情况。

```
  INSTALL lib/crypto/libdes.ko
  INSTALL lib/libcrc32c.ko
  INSTALL net/8021q/8021q.ko
  INSTALL net/sunrpc/auth_gss/rpcsec_gss_krb5.ko
  INSTALL sound/usb/snd-usb-audio.ko
  INSTALL sound/core/snd-hwdep.ko
  INSTALL sound/usb/snd-usbmidi-lib.ko
  DEPMOD  5.10.9-g6dc7129aa68c-dirty
make[1]: Leaving directory '/home/me/work/linux/out'
/home/me/work/linux
me@me-virtual-machine:~/work/linux$ ls
arch                  COPYING        fs       Kconfig      MAINTAINERS.NXP  README    tools
block                 CREDITS        include  kernel       Makefile         samples   usr
build.sh              crypto         init     lib          mm               scripts   virt
certs                 Documentation  ipc      LICENSES     net              security
compile_commands.json drivers        Kbuild   MAINTAINERS  out              sound
me@me-virtual-machine:~/work/linux$ ls out/
arch    drivers  ipc       mm                      modules.tar.bz2  security    usr        vmlinux.symvers
block   fs       kernel    modules.builtin         Module.symvers   sound       virt
certs   include  lib       modules.builtin.modinfo  net              source      vmlinux
crypto  init     Makefile  modules.order           scripts          System.map  vmlinux.o
me@me-virtual-machine:~/work/linux$
```

out 目录包含编译过程生成的所有文件，这些文件大部分对用户来说是不需要接触到的。在一般情况下我们只需要使用到 linux 的内核文件：`out/arch/arm/boot/zImage`文件和设备树文件：`out/arch/arm/boot/dts/ebyte-imx6ull-*.dtb`以及放置在文件系统中的内核模块文件：`out/modules.tar.bz2`。设备树文件需要根据核心板与底板情况自行选择合对应文件。当前内核的模块压缩包，如果需要可以解压到根文件系统的`/lib`目录下。

5. 如何适配不同的硬件平台

用户需要对 CPU 的芯片手册，以及 ECK20 核心板的产品手册，底板管脚定义有比较详细的了解，以便于根据实际的功能对这些管脚进行正确的配置和使用。

5.1. u-boot

为了便于用户修改适配自己的底板，这里将亿佰特提供的 u-boot 示例修改添加部分列出：

u-boot 修改文件表

文件	说明
arch/arm/dts/Makefile	添加设备树，用于编译设备树
arch/arm/dts/ebyte-imx6ull-14x14-base.dtsi	ECK20 的基础配置
arch/arm/dts/ebyte-imx6ull-14x14-emmc.dts	ECK20 的 eMMC 板设备树
arch/arm/dts/ebyte-imx6ull-14x14-gpmi-weim.dts	ECK20 的 NAND 板设备树
arch/arm/dts/ebyte-imx6ull-14x14-u-boot.dtsi	ECK20 的 u-boot 阶段特殊定义相关设备树
arch/arm/mach-imx/mx6/Kconfig	添加开发板配置使能项，用于编译自己的开发板
board/ebyte/ebyte_imx6ull/Kconfig	开发板板级相关配置选项
board/ebyte/ebyte_imx6ull/MAINTAINERS	描述开发板相关文件
board/ebyte/ebyte_imx6ull/Makefile	用于编译开发板的 Makefile
board/ebyte/ebyte_imx6ull/ebyte_imx6ull.c	板级初始化文件
board/ebyte/ebyte_imx6ull/imximage-ddr256.cfg	DDR 相关配置文件，用于 NAND 板
board/ebyte/ebyte_imx6ull/imximage-ddr512.cfg	DDR 相关配置文件，用于 eMMC 板
include/configs/ebyte_imx6ull.h	板级配置文件
configs/ebyte_imx6ull_nand_defconfig	用于构建 NAND 板的默认 u-boot 配置
configs/ebyte_imx6ull_emmc_defconfig	用于构建 eMMC 板的默认 u-boot 配置
tools/logos/ebyte.bmp	用于显示的公司 logo

5.1.1. 设备树

对于 arch/arm/dts/ebyte-imx6ull-14x14-base.dtsi 文件，其它扩展的接口和设备可以对它们进行引用，如下所示（仅供参考）：

arch > arm > dts > ebyte-imx6ull-14x14-gpmi-weim.dts

```
// SPDX-License-Identifier: GPL-2.0
//
// Copyright (C) 2016 Freescale Semiconductor, Inc.

#include "ebyte-imx6ull-14x14-base.dtsi"

&gpmi {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpmi_nand_1>;
    status = "okay";
    nand-on-flash-bbt;
};

&iomuxc {
    pinctrl_gpmi_nand_1: gpmi-nand-1 {
        fsl,pins = <
            MX6UL_PAD_NAND_CLE__RAWNAND_CLE         0xb0b1
            MX6UL_PAD_NAND_ALE__RAWNAND_ALE         0xb0b1
            MX6UL_PAD_NAND_WP_B__RAWNAND_WP_B       0xb0b1
            MX6UL_PAD_NAND_READY_B__RAWNAND_READY_B 0xb000
            MX6UL_PAD_NAND_CE0_B__RAWNAND_CE0_B     0xb0b1
            MX6UL_PAD_NAND_CE1_B__RAWNAND_CE1_B     0xb0b1
            MX6UL_PAD_NAND_RE_B__RAWNAND_RE_B       0xb0b1
            MX6UL_PAD_NAND_WE_B__RAWNAND_WE_B       0xb0b1
            MX6UL_PAD_NAND_DATA00__RAWNAND_DATA00   0xb0b1
            MX6UL_PAD_NAND_DATA01__RAWNAND_DATA01   0xb0b1
            MX6UL_PAD_NAND_DATA02__RAWNAND_DATA02   0xb0b1
            MX6UL_PAD_NAND_DATA03__RAWNAND_DATA03   0xb0b1
            MX6UL_PAD_NAND_DATA04__RAWNAND_DATA04   0xb0b1
            MX6UL_PAD_NAND_DATA05__RAWNAND_DATA05   0xb0b1
            MX6UL_PAD_NAND_DATA06__RAWNAND_DATA06   0xb0b1
            MX6UL_PAD_NAND_DATA07__RAWNAND_DATA07   0xb0b1
        >;
    };
};

&usdhc2 {
    status = "disabled";
};

```

用户增加了新的设备树源文件之后，还需要在同目录下的 arch/arm/dts/Makefile 里添加设备树编译信息，这样就可以在编译内核的时候生成对应的设备树二进制文件。

```
dtb-$(CONFIG_MX6ULL) += \
    imx6ull-14x14-ddr3-val.dtb \
    imx6ull-14x14-ddr3-val-epdc.dtb \
    imx6ull-14x14-ddr3-val-emmc.dtb \
    imx6ull-14x14-ddr3-val-gpmi-weim.dtb \
    imx6ull-14x14-ddr3-val-tsc.dtb \
    imx6ull-14x14-evk.dtb \
    imx6ull-14x14-evk-emmc.dtb \
    imx6ull-14x14-evk-gpmi-weim.dtb \
    imx6ull-9x9-evk.dtb \
    imx6ull-colibri.dtb \
    imx6ull-phytec-segin-ff-rdk-emmc.dtb \
    imx6ull-dart-6ul.dtb \
    imx6ull-somlabs-visionsom.dtb \
    imx6ulz-14x14-evk.dtb \
    imx6ulz-14x14-evk-emmc.dtb \
    imx6ulz-14x14-evk-gpmi-weim.dtb \
    ebyte-imx6ull-14x14-emmc.dtb \
    ebyte-imx6ull-14x14-gpmi-weim.dtb
```

修改完成后即可增加设备驱动的板载描述，通过 3.1 章节的方法编译生成设备树 dtb 文件 ebyte-imx6ull-14x14-gpmi-weim.dtb 与 ebyte-imx6ull-14x14-emmc.dtb。

5.1.2. 板级文件

这一部分需要针对每一块底板进行单独的配置，主要是系统初始化阶段需要执行的工作。

虽然大部分配置情况都放在了设备树文件中，但是在初始化阶段依然需要关注一些暂时不支持在设备树中工作的引脚是否有冲突，是否符合实际底板情况，如在 ebyte_imx6ull.c 文件中，lcd 的背光控制引脚：

```
#ifdef CONFIG_DM_VIDEO
static iomux_v3_cfg_t const lcd_pads[] = {
    /* Use GPIO for Brightness adjustment, duty cycle = period. */
    MX6_PAD_GPIO1_IO08__GPIO1_IO08 | MUX_PAD_CTRL(NO_PAD_CTRL),
};

static int setup_lcd(void)
{
    enable_lcdif_clock(LCDIF1_BASE_ADDR, 1);

    imx_iomux_v3_setup_multiple_pads(lcd_pads, ARRAY_SIZE(lcd_pads));

#if 0
    /* Reset the touch screen */
    gpio_request(IMX_GPIO_NR(5, 9), "touch screen reset");
    gpio_direction_output(IMX_GPIO_NR(5, 9) , 0);
    udelay(500);
    gpio_direction_output(IMX_GPIO_NR(5, 9) , 1);
#endif

    /* Set Brightness to high */
    gpio_request(IMX_GPIO_NR(1, 8), "backlight");
    gpio_direction_output(IMX_GPIO_NR(1, 8) , 1);

    return 0;
}
#else
static inline int setup_lcd(void) { return 0; }
#endif
```

5.1.3. Logo

对于需要使用自定义 logo 的用户，只需要使用自制的 logo 文件替换`tools/logos/ebyte.bmp`即可。需要注意文件名字与`board/ebyte/ebyte_imx6ull/Kconfig`文件中`SYS_BOARD`或`SYS_VENDOR`选项配置相同，否则会回落使用 u-boot 默认 logo。

```
me@me-virtual-machine:~/work/uboot$ cat board/ebyte/ebyte_imx6ull/Kconfig
if TARGET_EBYTE_MX6ULL

config SYS_BOARD
        default "ebyte_imx6ull"

config SYS_VENDOR
        default "ebyte"

config SYS_CONFIG_NAME
        default "ebyte_imx6ull"

config SYS_TEXT_BASE
        default 0x87800000
endif
```

对于 deb 系的系统(比如 ubuntu)可以通过安装包 netpbm，然后使用如下命令创建可在 u-boot 阶段使用的 logo 文件：

```
jpegtopnm file.jpeg | ppmquant 256 | ppmtobmp -bpp 8 > file.bmp
```

5.1.4. u-boot 模块使能

亿佰特提供的示例中使能了部分设备驱动，如使能了 PHY 功能，LED 功能，指明了 DDR 配置文件路径等。用户应该根据底板实际使用的功能进行配置使能。

在调试阶段，如果每次都去修改 configs/*_defconfig 文件会较为麻烦，而且配置可能出现依赖相关问题，更推荐使用进入 menuconfig 手动选配功能。

5.1.4.1. 选配模块功能

1. 执行如下指令添加基本环境变量，用于执行 menuconfig 功能。

```
export PATH=${HOME}/.local/share/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/:${PATH}
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

2. 查看配置是否成功：

```
env | grep arm
```

返回如下：

```
me@me-virtual-machine:~/work/uboot$ env | grep arm
ARCH=arm
PATH=/home/me/.local/share/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-gnueabihf/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
CROSS_COMPILE=arm-linux-gnueabihf-
me@me-virtual-machine:~/work/uboot$
```

3. 进行初步配置，选择一个配置文件作为基础配置，这里推荐使用亿佰特提供的示例文件：

```
make O=out ebyte_imx6ull_nand_defconfig
make O=out ebyte_imx6ull_emmc_defconfig
```

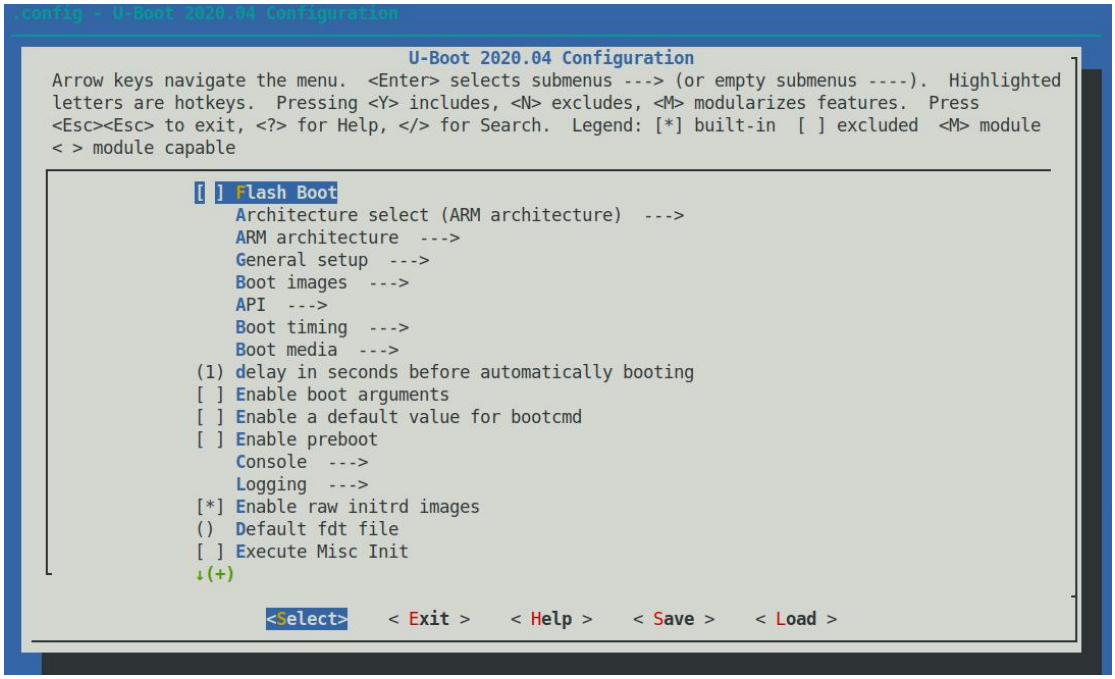
4. 然后执行如下命令进入 menuconfig 界面，其使用方法参考“构建 u-boot”章节中编译用于从 SD 卡启动的 uboot 小节。

```
make O=out menuconfig
```

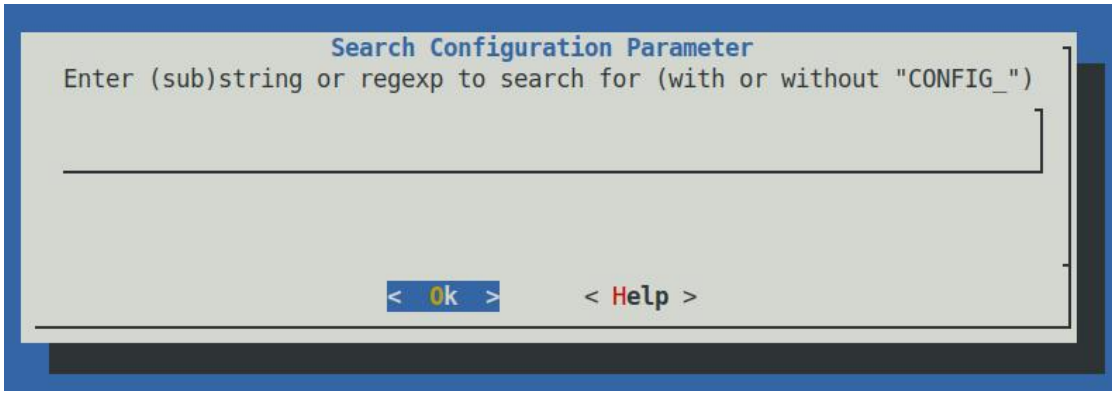
对于需要使能的功能名称，可以通过查看散落在各个目录下的 Kconfig 文件寻找。

对于已知功能名称，但不知道在哪一个 menu 下，可以在如下界面敲击"/"按键进入搜索

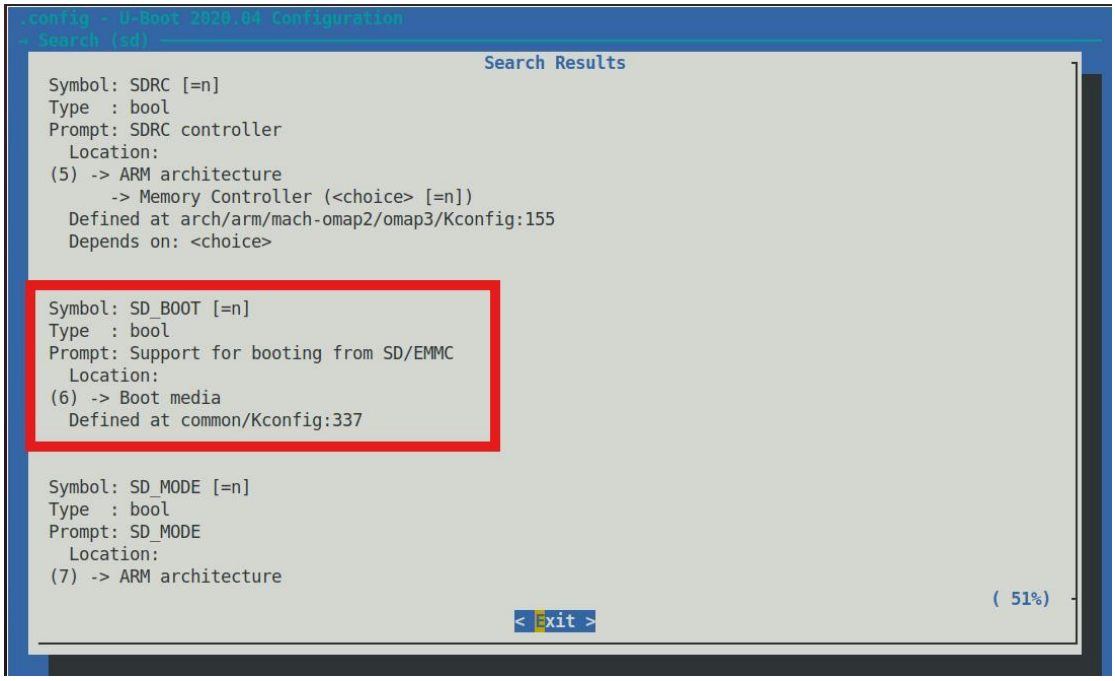
功能：



进入搜索后，输入需要寻找的功能项并回车确认：

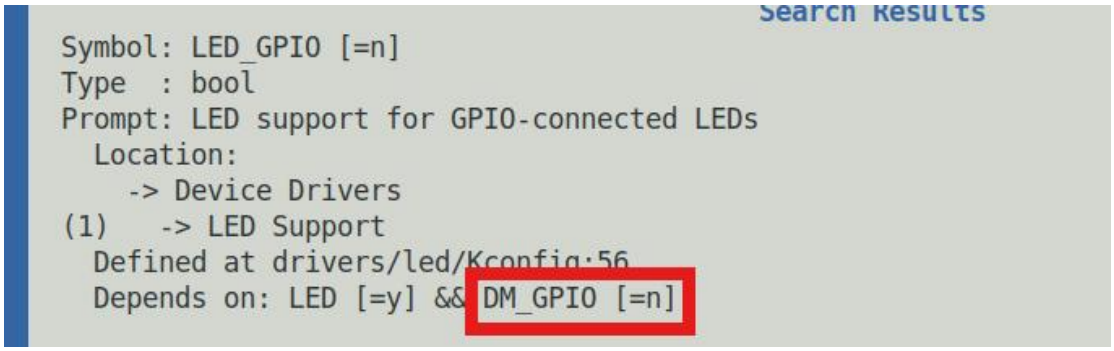


如此处搜索 SD 功能：

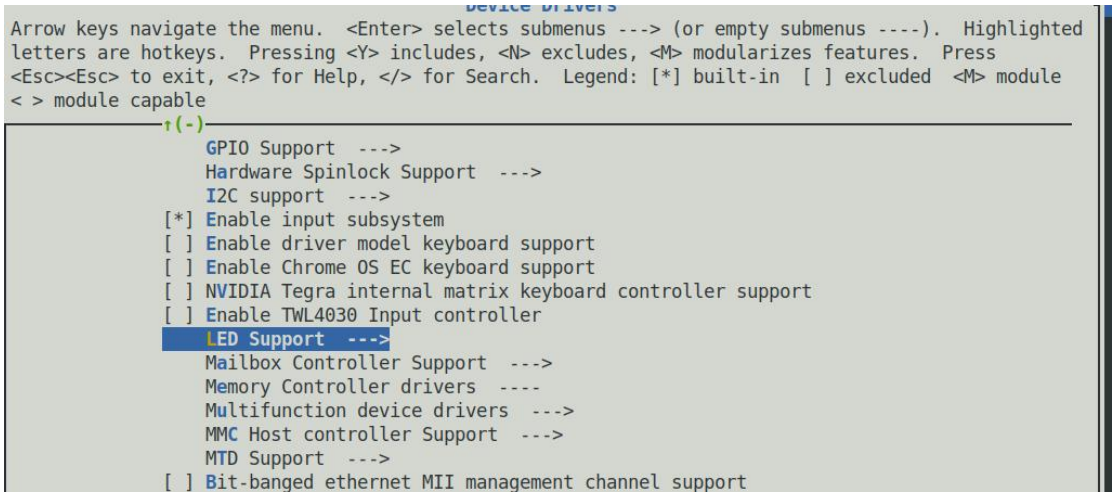


上下移动后会发现有大量选项，每一个选项前面都会有一个数字，当我们找到所需的选项后敲击前方数字 6 即可进入该选项所在 menu 子项。

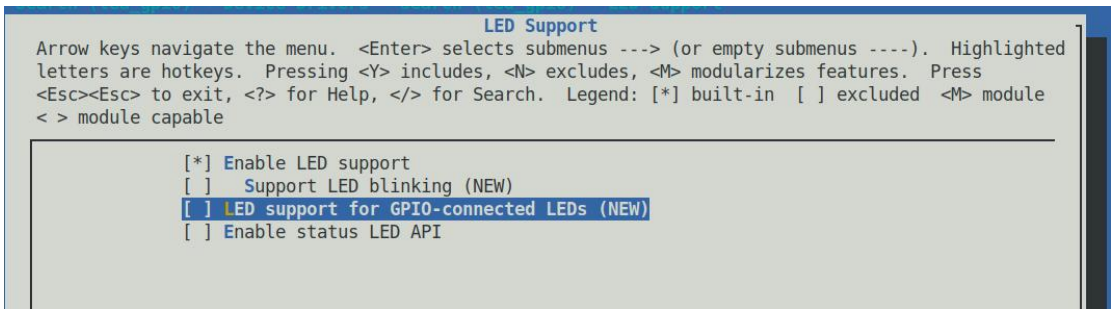
但是有时候会发现理论上应该会进入该选项所在位置时，却并没有该选项可选，此时需要回到搜索界面，确定该选项的依赖项是否满足，比如：



此时敲击 1 进入的界面如下：



当使能上面缺失的 DM_GPIO 功能后再次敲击按键 1 则是进入如下界面：



5.1.4.2. 保存配置

当完成所有模块功能选配后，执行如下指令保存配置：

```
make O=out savedefconfig
```

而后会在 out 目录生成文件 defconfig。该文件则可作为目标开发板的默认配置文件。

```
me@me-virtual-machine:~/work/uboot$ make O=out savedefconfig
make[1]: Entering directory '/home/me/work/uboot/out'
  GEN     ./Makefile
scripts/kconfig/conf  --savedefconfig=defconfig Kconfig
make[1]: Leaving directory '/home/me/work/uboot/out'
me@me-virtual-machine:~/work/uboot$ ls out/
arch       disk      fs        scripts     u-boot.bin          u-boot-dtb.cfgout   u-boot-nodtb.bin
board      drivers   include   source      u-boot.cfg          u-boot-dtb.imx      u-boot.srec
cmd        dts       lib       System.map  u-boot.cfg.configs  u-boot-dtb.imx.log  u-boot.sym
common     env       Makefile  tools       u-boot.dtb          u-boot.lds
defconfig  examples  net       u-boot      u-boot-dtb.bin      u-boot.map
me@me-virtual-machine:~/work/uboot$
```

5.2. Linux

为了便于用户修改适配自己的底板，这里将亿佰特提供的 kernel 示例修改添加部分列出：

Linux 修改文件表

文件	说明
arch/arm/boot/dts/Makefile	添加设备树，用于编译设备树
arch/arm/boot/dts/ebyte-imx6ull-base.dtsi	ECK20-6Y2XA 的基础配置

arch/arm/boot/dts/ebyte-imx6ull-gpmi-weim.dts	ECK20-6Y2XA 的 eMMC 板设备树
arch/arm/boot/dts/ebyte-imx6ull-emmc.dts	ECK20-6Y2XA 的 NAND 板设备树
arch/arm/boot/dts/ebyte-imx6ull-btb-base.dtsi	ECK20-6Y2XC 的基础配置
arch/arm/boot/dts/ebyte-imx6ull-btb-gpmi-weim.dts	ECK20-6Y2XC 的 eMMC 板设备树
arch/arm/boot/dts/ebyte-imx6ull-btb-emmc.dts	ECK20-6Y2XC 的 NAND 板设备树
arch/arm/configs/ebyte_imx6ull_defconfig	默认情况下内核功能配置文件
drivers/video/fbdev/core/fbcon.c	禁止 framebuffer 终端下光标闪烁
drivers/video/fbdev/core/fbmem.c	配置默认情况下光标居中显示
drivers/video/logo/logo_linux_clut224.ppm	用于显示的 logo

5.2.1. 设备树

对于 arch/arm/boot/dts/ebyte-imx6ull-base.dtsi 文件，其它扩展的接口和设备可以对它们进行引用，如下所示（仅供参考）：

arch > arm > boot > dts > ebyte-imx6ull-gpmi-weim.dts

```
// SPDX-License-Identifier: GPL-2.0
//
// Copyright (C) 2016 Freescale Semiconductor, Inc.

#include "ebyte-imx6ull-base.dtsi"

&gpmi {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpmi_nand_1>;
    status = "okay";
    nand-on-flash-bbt;
};

&iomuxc {
    pinctrl_gpmi_nand_1: gpmi-nand-1 {
        fsl,pins = <
            MX6UL_PAD_NAND_CLE__RAWNAND_CLE         0xb0b1
            MX6UL_PAD_NAND_ALE__RAWNAND_ALE         0xb0b1
            MX6UL_PAD_NAND_WP_B__RAWNAND_WP_B       0xb0b1
            MX6UL_PAD_NAND_READY_B__RAWNAND_READY_B 0xb000
            MX6UL_PAD_NAND_CE0_B__RAWNAND_CE0_B     0xb0b1
            MX6UL_PAD_NAND_CE1_B__RAWNAND_CE1_B     0xb0b1
            MX6UL_PAD_NAND_RE_B__RAWNAND_RE_B       0xb0b1
            MX6UL_PAD_NAND_WE_B__RAWNAND_WE_B       0xb0b1
            MX6UL_PAD_NAND_DATA00__RAWNAND_DATA00   0xb0b1
            MX6UL_PAD_NAND_DATA01__RAWNAND_DATA01   0xb0b1
            MX6UL_PAD_NAND_DATA02__RAWNAND_DATA02   0xb0b1
            MX6UL_PAD_NAND_DATA03__RAWNAND_DATA03   0xb0b1
            MX6UL_PAD_NAND_DATA04__RAWNAND_DATA04   0xb0b1
            MX6UL_PAD_NAND_DATA05__RAWNAND_DATA05   0xb0b1
            MX6UL_PAD_NAND_DATA06__RAWNAND_DATA06   0xb0b1
            MX6UL_PAD_NAND_DATA07__RAWNAND_DATA07   0xb0b1
        >;
    };
};

&qspi {
    status = "disabled";
};

&usdhc2 {
    status = "disabled";
};

```

用户增加了新的设备树源文件之后，还需要在同目录下的 arch/arm/dts/Makefile 里添加设备树编译信息，这样就可以在编译内核的时候生成对应的设备树二进制文件。

```
dtb-$(CONFIG_SOC_IMX6UL) += \
    imx6ul-liteboard.dtb \
    imx6ul-opos6uldev.dtb \
    imx6ul-pico-dwarf.dtb \
    imx6ul-pico-hobbit.dtb \
    imx6ul-pico-pi.dtb \
    imx6ul-phytec-segin-ff-rdk-nand.dtb \
    imx6ul-tx6ul-0010.dtb \
    imx6ul-tx6ul-0011.dtb \
    imx6ul-tx6ul-mainboard.dtb \
    imx6ull-14x14-evk.dtb \
    imx6ull-14x14-evk-emmc.dtb \
    imx6ull-14x14-evk-btwifi.dtb \
    imx6ull-14x14-evk-btwifi-oob.dtb \
    imx6ull-14x14-evk-gpmi-weim.dtb \
    imx6ull-9x9-evk.dtb \
    imx6ull-9x9-evk-ldo.dtb \
    imx6ull-9x9-evk-btwifi.dtb \
    imx6ull-9x9-evk-btwifi-oob.dtb \
    imx6ull-colibri-eval-v3.dtb \
    imx6ull-colibri-wifi-eval-v3.dtb \
    imx6ull-myir-mys-6ulx-eval.dtb \
    imx6ull-opos6uldev.dtb \
    imx6ull-phytec-segin-ff-rdk-nand.dtb \
    imx6ull-phytec-segin-ff-rdk-emmc.dtb \
    imx6ull-phytec-segin-lc-rdk-nand.dtb \
    ebyte-imx6ull-emmc.dtb \
    ebyte-imx6ull-gpmi-weim.dtb \
    imx6ulz-14x14-evk.dtb \
    imx6ulz-14x14-evk-btwifi.dtb \
    imx6ulz-14x14-evk-gpmi-weim.dtb \
    imx6ulz-14x14-evk-emmc.dtb
dtb-$(CONFIG_SOC_IMX7D) += \
```

增加完成后即可增加设备驱动的板载描述，通过第四章节的方法编译生成设备树 dtb 文件 ebyte-imx6ull-emmc.dtb 与 ebyte-imx6ull-gpmi-weim.dtb。

5.2.2. Logo

对于需要使用自定义 logo 的用户，只需要使用自制的 logo 文件替换 `drivers/video/logo/logo_linux_clut224.ppm`即可。

对于 deb 系的系统可以通过安装包 netpbm，然后使用如下命令创建可在 kernel 引导阶段使用的 logo 文件：

```
pngtopnm file.png | pnmquant 224 | pnmtoplainpnm > logo_linux_clut224.ppm
```

5.2.3. defconfig 配置

亿佰特提供的示例中使能了部分设备驱动，如使能了 PHY 功能以及音频的 wm8960 驱动等。用户应该根据底板实际使用的功能进行配置使能。

在调试阶段，如果每次都去修改 arch/arm/configs/*_defconfig 文件会较为麻烦，而且配

置可能出现依赖相关问题，更推荐使用进入 menuconfig 手动选配功能，这一部分参考本文中的 5.1.4 小节描述即可。

6. 制作文件系统

这是 Linux 系统移植的最后一步，根文件系统构建好以后就意味着我们已经拥有了一个完整的、可以运行的最小系统。以后我们就在这个最小系统上编写、测试 Linux 驱动，移植一些第三方组件，逐步的完善这个最小系统。最终得到一个功能完善、驱动齐全、相对完善的操作系统。

6.1. BuildRoot 构建根文件系统

Buildroot 是一个用于构建嵌入式 Linux 系统的工具集。它是一个开源项目，旨在帮助开发者创建最小化的、定制化的 Linux 系统。Buildroot 提供了一整套的自动化构建过程，能够从源码构建 Linux 内核、Bootloader（如 U-Boot）、根文件系统、以及其他依赖的库和应用程序。

6.1.1. 获取源码

目前已经在下载资料中随附适配过本司 ECB20 系列单板机的 buildroot 源码，其路径位于 04_Sources/buildroot.tar.xz，同时为方便用户快速构建，避免构建过程中的网络文件，也随附了构建最终镜像所需的大部分包对应的源码，其路径位于 04_Sources/buildroot_dl.tar.xz，请用户将其解压至 buildroot 根目录的 dl 子目录中。。

```
# 创建工作目录，解压 buildroot 源码
mkdir work && cd work
tar -xaf <your_path>/buildroot.tar.xz
# 进入 buildroot 目录，解压 dl 包（可选）
cd buildroot
tar -xaf <your_path>/buildroot_dl.tar.xz
```

解压完毕 buildroot 源码后有下面部分内容：

```
$ ~/src/buildroot ls
arch    build.sh    Config.in.legacy  DEVELOPERS  linux           package  system
board   CHANGES     configs           docs        Makefile        README   toolchain
boot    Config.in   COPYING           fs          Makefile.legacy support  utils
$ ~/src/buildroot
```

6.1.2. 编译源码

本司提供了一个`build.sh`的脚本，便于用户构建适配本司 ECB20 单板机的镜像。

```
$ ~/src/buildroot ./build.sh help
Usage: ./build.sh < **your_defconfig** | config | busybox | kernel | uboot | sdk >

        your_defconfig: which can found in '<buildroot>/configs/' directory
        config: configure the buildroot
        busybox: configure the busybox
        kernel: configure the kernel
        uboot: configure the uboot
        sdk: build the SDK
```

其中`config`参数用于对 buildroot 进行配置，用户可以不使用本司提供的初始配置自定义配置`configs/ebyte_imx6ulx_qt_defconfig`，但请至少确保自己的配置参考 NXP 官方的`configs/freescale_imx6ullevk_defconfig`。

注意：使用 BTB 连接器的用户，配置为`configs/ebyte_imx6ulx_qt_btb_defconfig`

完成初始配置后，执行如下命令构建 buildroot：

```
# 不使用 build.sh

make

# 使用 build.sh 需要修改 dl 目录名为.downloads

mv dl .downloads

./build.sh
```

```
>>>   Executing post-image script board/freescale/common/imx/post-image.sh
INFO: cmd: "mkdir -p "/home/me/imx6ull/ebyte/buildroot/output/build/genimage.tmp"" (stderr):
INFO: cmd: "rm -rf "/home/me/imx6ull/ebyte/buildroot/output/build/genimage.tmp"/*" (stderr):
INFO: cmd: "mkdir -p "/home/me/imx6ull/ebyte/buildroot/output/build/genimage.tmp"" (stderr):
INFO: cmd: "cp -a "/home/me/imx6ull/ebyte/buildroot/output/target" "/home/me/imx6ull/ebyte/buildroot/output/build/
INFO: cmd: "find '/home/me/imx6ull/ebyte/buildroot/output/build/genimage.tmp/root' -depth -type d -printf '%P\0' |
6ull/ebyte/buildroot/output/build/genimage.tmp/root/{}'" (stderr):
INFO: cmd: "mkdir -p "/home/me/imx6ull/ebyte/buildroot/output/images"" (stderr):
INFO: vfat(boot.vfat): cmd: "dd if=/dev/zero of="/home/me/imx6ull/ebyte/buildroot/output/images/boot.vfat" seek=16
INFO: vfat(boot.vfat): cmd: "mkdosfs  -n 'boot' '/home/me/imx6ull/ebyte/buildroot/output/images/boot.vfat'" (stder
mkfs.fat: Warning: lowercase labels might not work properly on some systems
INFO: vfat(boot.vfat): adding file 'ebyte-imx6ull-emmc.dtb' as 'ebyte-imx6ull-emmc.dtb' ...
INFO: vfat(boot.vfat): cmd: "MTOOLS_SKIP_CHECK=1 mcopy -sp -i '/home/me/imx6ull/ebyte/buildroot/output/images/boot
rr):
INFO: vfat(boot.vfat): adding file 'zImage' as 'zImage' ...
INFO: vfat(boot.vfat): cmd: "MTOOLS_SKIP_CHECK=1 mcopy -sp -i '/home/me/imx6ull/ebyte/buildroot/output/images/boot
INFO: hdimage(sdcard.img): adding partition 'u-boot' from 'u-boot-dtb.imx' ...
INFO: hdimage(sdcard.img): adding partition 'boot' (in MBR) from 'boot.vfat' ...
INFO: hdimage(sdcard.img): adding partition 'rootfs' (in MBR) from 'rootfs.ext2' ...
INFO: hdimage(sdcard.img): adding partition '[MBR]' ...
INFO: hdimage(sdcard.img): writing MBR
$ ~/imx6ull/ebyte/buildroot
```

如上所示编译完成，可以在 output/images 目录下看到编译的文件系统，并已经打包为适用于 ECB20 单板机的镜像文件：

```
$ ~/imx6ull/ebyte/buildroot ls output/images/
boot.vfat  ebyte-imx6ull-emmc.dtb  rootfs.ext2  rootfs.ext4  rootfs.tar  rootfs.tar.bz2  sdcard.img  u-boot-dtb.imx  zImage
$ ~/imx6ull/ebyte/buildroot
```

6.2. Yocto 构建根文件系统

由于芯片原厂在标准 Yocto 的基础上进行了定制，下面是使用芯片厂商提供的版本的基本流程，更多的使用过程详情参考 NXP 官方文档《i.MX_Yocto_Project_User's_Guide.pdf》，该文档位于`01_Documents/Datasheet/NXP/imx-yocto-LF5.10.9_1.0.0`中。

6.2.1. 获取源码

下载 repo 工具：

```
sudo apt install -u curl
mkdir -p ~/.local/bin
curl https://storage.googleapis.com/git-repo-downloads/repo -o ~/.local/bin/repo
chmod +x ~/.local/bin/repo
```

配置 repo 工具更新源使用镜像(国内可选)：

```
echo "export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo'" >> ~/.bashrc
```

使能相关环境变量：

```
source ~/.profile
```

进入到项目目录以后通过 NXP 官方 REPO 获取源码，查看目录情况如下：

```
me@me-virtual-machine:~/work/yocto$ ls -A
imx-setup-release.sh  README  README-IMXBSP  .repo  setup-environment  sources
me@me-virtual-machine:~/work/yocto$
```

6.2.2. 安装编译依赖

编译 Yocto 需要使用到超过编译内核与 u-boot 所需要的依赖程序。

```
sudo apt install chrpath diffstat gawk python3-distutils quilt
```

6.2.3. 初始化配置

使用提供的 imx-setup-release.sh 脚本，会创建一个工作空间，然后在此空间下构建镜像。执行脚本后会先要求阅读并同意版权声明后才会进入构建过目录。同时，脚本会默认创建并进入 build 目录。如果需要特定目录名称，可以使用-b 参数，如"-b out_dir"。

MACHINE 与 DISTRO 可选参数可以查看目录下的`README-IMXBSP`文件：

```
The following boards were tested in this release.

   * NXP i.MX 8MQuad EVK (imx8mqevk)
   * NXP i.MX 8MMini EVK (imx8mmevk)
   * NXP i.MX 8MNano EVK (imx8mnevk)
   * NXP i.MX 8MPlus EVK (imx8mpevk)
   * NXP i.MX 8QuadMax MEK (imx8qmmek)
   * NXP i.MX 8QuadXPlus MEK (imx8qxpmek)
   * NXP i.MX 8DualX MEK (imx8dxmek)
   * NXP i.MX 8DXL EVK (imdx8dxlevk)
   * NXP i.MX 7ULP EVK (imx7ulpevk)
   * NXP i.MX 7Dual SABRE Smart Device (imx7dsabresd)
   * NXP i.MX 6QuadPlus SABRE Device (imx6qpsabresd)
   * NXP i.MX 6QuadPlus SABRE Auto (imx6qpsabreauto)
   * NXP i.MX 6Quad SABRE Smart Device (imx6qsabresd)
   * NXP i.MX 6Quad SABRE Auto (imx6qsabreauto)
   * NXP i.MX 6DualLite SABRE Smart Device (imx6dlsabresd)
   * NXP i.MX 6DualLite SABRE Auto (imx6dlsabreauto)
   * NXP i.MX 6SOLO SABRE Smart Device (imx6solosabresd )
   * NXP i.MX 6SOLO SABRE Auto (imx6solosabreauto )
   * NXP i.MX 6SoloX SABRE Smart Device (imx6sxsabresd)
   * NXP i.MX 6SoloX SABRE Auto (imx6sxsabreauto)
   * NXP i.MX 6UltraLite EVK (imx6ulevk)
   * NXP i.MX 6ULL EVK (imx6ull14x14evk)
README-IMXBSP
```

```
Building Frame Buffer (FB)
--------------------------
  DISTRO=fsl-imx-fb MACHINE=imx6qsabresd source imx-setup-release.sh -b build-fb
  bitbake <image>

To run the QT5 examples use the following parameters:
<QT5 Example> -platform eglfs -plugin evdevtouch:/dev/input/event0

Building XWayland
--------------------------
  DISTRO=fsl-imx-xwayland MACHINE=imx6qsabresd source imx-setup-release.sh -b build-xwayland
  bitbake <image>

To run the QT5 examples use the following parameters:
<QT5 example> platform wayland-egl -plugin evdevtouch:/dev/input/event0 --fullscreen

Building Wayland-Weston (wayland)
--------------------------
  DISTRO=fsl-imx-wayland MACHINE=imx6qsabresd source imx-setup-release.sh -b build-wayland
  bitbake <image>
```

执行如下命令，编译 NXP 提供的示例，使用 framebuffer 作为显示功能：

```
MACHINE=imx6ull14x14evk DISTRO=fsl-imx-fb source ./imx-setup-release.sh -b build
```

阅读用户协议并同意：

```
LA_OPT_NXP_Software_License v19 February 2021

IMPORTANT.  Read the following NXP Software License Agreement ("Agreement")
completely. By selecting the "I Accept" button at the end of this page, or by
downloading, installing, or using the Licensed Software, you indicate that you
accept the terms of the Agreement and you acknowledge that you have the
authority, for yourself or on behalf of your company, to bind your company to
these terms. You may then download or install the file. In the event of a
conflict between the terms of this Agreement and any license terms and
conditions for NXP's proprietary software embedded anywhere in the Licensed
Software file, the terms of this Agreement shall control.  If a separate
license agreement for the Licensed Software has been signed by you and NXP,
then that agreement shall govern your use of the Licensed Software and shall
supersede this Agreement.

NXP SOFTWARE LICENSE AGREEMENT

This is a legal agreement between your employer, of which you are an authorized
representative, or, if you have no employer, you as an individual ("you" or
"Licensee"), and NXP B.V. ("NXP").  It concerns your rights to use the software
provided to you in binary or source code form and any accompanying written
materials (the "Licensed Software"). The Licensed Software may include any
updates or error corrections or documentation relating to the Licensed Software
provided to you by NXP under this Agreement. In consideration for NXP allowing
you to access the Licensed Software, you are agreeing to be bound by the terms
of this Agreement. If you do not agree to all of the terms of this Agreement,
do not download or install the Licensed Software. If you change your mind
later, stop using the Licensed Software and delete all copies of the Licensed
Software in your possession or control. Any copies of the Licensed Software
that you have already distributed, where permitted, and do not destroy will
continue to be governed by this Agreement. Your prior use will also continue to
be governed by this Agreement.

1.       DEFINITIONS

--More--(5%)[Press space to continue, 'q' to quit.]
```

6.2.4. 开始编译

6.2.4.1. 编译目标选择

在开始编译以前，需要先了解 Yocto 当前配置之下有哪些可以编译生成的项目，通过以下命令查看可选的编译目标：

```
bitbake -s
```

```
me@me-virtual-machine:~/work/yocto/build$ bitbake -s
NOTE: Your conf/bblayers.conf has been automatically updated.
Loading cache: 100% |                                                                | ETA:  --:--:--
Loaded 0 entries from dependency cache.
Parsing recipes: 100% |#################################################################| Time: 0:00:34
Parsing of 3281 .bb files complete (0 cached, 3281 parsed). 4848 targets, 578 skipped, 1 masked, 0 error
s.
NOTE: preferred version 11.0.0 of nativesdk-llvm not available
NOTE: versions of nativesdk-llvm available: 10.0.1
NOTE: preferred version 11.0.0 of llvm-native not available
NOTE: versions of llvm-native available: 10.0.1
NOTE: preferred version 11.0.0 of llvm not available
NOTE: versions of llvm available: 10.0.1
NOTE: preferred version 3.2.4.0 of opengl-es-cts not available
NOTE: versions of opengl-es-cts available: 3.2.6.1
Recipe Name                                    Latest Version         Preferred Version
===========                                    ==============         =================

a2jmidid                                                :9-r0
abseil-cpp                              :20190808+gitAUTOINC+aa844899c9-r0
ace                                                  :6.5.10-r0
ace-cloud-editor                        :02.07.17+gitAUTOINC+812e2c56ae-r0
acl                                                  :2.2.53-r0
acl-native                                           :2.2.53-r0
acpica                                              :20200717-r0
acpica-native                                       :20200717-r0
acpid                                                 :2.0.32-r0
acpitool                                               :0.5.1-r0
```

在上面如此多的项目中，我们主要需要关注的目标为这一部分：

```
me@me-virtual-machine:~/work/yocto/build$ bitbake -s | grep image
build-appliance-image                                :15.0.0-r0
core-image-base                                         :1.0-r0
core-image-clutter                                      :1.0-r0
core-image-full-cmdline                                 :1.0-r0
core-image-kernel-dev                                   :1.0-r0
core-image-minimal                                      :1.0-r0
core-image-minimal-dev                                  :1.0-r0
core-image-minimal-initramfs                            :1.0-r0
core-image-minimal-mtdutils                             :1.0-r0
core-image-sato                                         :1.0-r0
core-image-sato-dev                                     :1.0-r0
core-image-sato-ptest-fast                              :1.0-r0
core-image-sato-sdk                                     :1.0-r0
core-image-sato-sdk-ptest                               :1.0-r0
core-image-testmaster                                   :1.0-r0
fsl-image-gui                                           :1.0-r0
fsl-image-machine-test                                  :1.0-r0
fsl-image-mfgtool-initramfs                             :1.0-r0
fsl-image-multimedia                                    :1.0-r0
fsl-image-multimedia-full                               :1.0-r0
fsl-image-network-full-cmdline                          :1.0-r0
fsl-image-qt5                                           :1.0-r0
fsl-image-qt5-validation-imx                            :1.0-r0
fsl-image-validation-imx                                :1.0-r0
imagemagick                                       :7.0.10_25-r0
imagemagick-native                                :7.0.10_25-r0
imx-image-core                                          :1.0-r0
imx-image-full                                          :1.0-r0
imx-image-full-dev                                      :1.0-r0
imx-image-multimedia                                    :1.0-r0
imx-mkimage                                             :git-r0
libsdl-image                                         :1.2.12-r0
libsdl2-image                                         :2.0.5-r0
meta-filesystems-image                                  :1.0-r0
meta-filesystems-image-base                             :1.0-r0
```

其中编译目标`fsl-image-mfgtool-initramfs`是我们在烧录系统固件时需要使用到的initramfs 镜像。其余部分可以参考 nxp 官方文档《i.MX_Yocto_Project_User's_Guide.pdf》中第 5.2 章节：

imx-image-core	An i.MX image with i.MX test applications to be used for Wayland backends. This image is used by our daily core testing.	meta-imx/meta-sdk
fsl-image-machine-test	An FSL Community i.MX core image with console environment - no GUI interface.	meta-freescale-distro
imx-image-multimedia	Builds an i.MX image with a GUI without any Qt content.	meta-imx/meta-sdk
imx-image-full	Builds an opensource Qt 5 image with Machine Learning features. These images are only supported for i.MX SoC with hardware graphics. They are not supported on the i. MX 6UltraLite, i.MX 6UltraLiteLite, i.MX 6SLL, and i.MX 7Dual.	meta-imx/meta-sdk

6.2.4.2. 开始编译

在选定需要编译的目标以后，就是正式开始编译了。

```
bitbake imx-image-full
```

```
meta-freescale-distro = "HEAD:11be3f01962df8436c5c7b0d61cd3dbd1b872905"
meta-bsp
meta-sdk
meta-ml              = "HEAD:83d7642fe53f1cd3871cf2b2692b61459237cd64"
meta-nxp-demo-experience = "HEAD:60578a5dbbfc9abc360a12e2cef7b64bfa582904"
meta-browser         = "HEAD:ee3be3b5986a4aa0e73df2204a625ae1fe5df37e"
meta-rust            = "HEAD:53bfa324891966a2daf5d36dc13d4a43725aebed"
meta-clang           = "HEAD:61faae011fb95712064f2c58abe6293f0daeeab5"
meta-gnome
meta-networking
meta-filesystems     = "HEAD:ac4ccd2fbbb599d75ca4051911fcbaca39dbe6d7"
meta-qt5             = "HEAD:8d5672cc6ca327576a814d35dfb5d59ab24043cb"
meta-python2         = "HEAD:c43c29e57f16af4e77441b201855321fbd546661"

Initialising tasks: 100% |############################################################| Time: 0:00:04
Sstate summary: Wanted 2688 Found 0 Missed 2688 Current 6 (0% match, 0% complete)
NOTE: Executing Tasks
Currently  9 running tasks (86 of 6482)   1% |                                                 |
0: sqlite3-native-3_3.33.0-r0 do_fetch (pid 4966)  14% |#####                                 | 4.24M/s
1: pseudo-native-1.9.0+gitAUTOINC+cca0d7f15b-r0 do_fetch (pid 4967) |                        <=>      |
2: gmp-native-6.2.0-r0 do_unpack - 1s (pid 4985)
3: binutils-cross-arm-2.35-r0 do_fetch (pid 6628) |                                             <=>   |
4: autoconf-archive-native-2019.01.06-r0 do_fetch (pid 6858)   0% |                                 |
5: mpfr-native-4.1.0-r0 do_fetch (pid 7598)   0% |                                                 |
6: popt-native-1.18-r0 do_fetch (pid 7892)   0% |                                                  |
7: acl-native-2.2.53-r0 do_fetch (pid 8466)   0% |                                                 |
8: patch-native-2.7.6-r0 do_deploy_source_date_epoch - 0s (pid 8642)
```

如果中途断开编译并且退出终端以后，可执行如下命令重新建立配置：

```
source setup-environment <build_dir>
```

在建文件系统完成后，会在输出目录下有 manifest 文件，这个文件里包含了对应文件系统中已安装的软件包。Yocto 第一次构建会需要很长时间，取决于计算机的 CPU 核心数和硬件读写速度。Yocto 建议可以使用八核和 SSD 硬盘可以加速构建速度。第一次构建完成后会生成缓存，后面修改的构建，时间会减少很多。在构建完成后会在"tmp/deploy/images/imx6ull14x14evk/"目录下生成最终文件。

6.2.5. 加速编译

如上文所说，第一次编译完成以后，会生成编译缓存，其路径为：

`<build_dir>/sstate-cache`，我们也可以指定其不同的存放位置，如此一来在编译不同 DISTRO 时也可以共用一部分以提供加速。

想要实现这一功能，只需要修改`<build_dir>/conf/local.conf`文件，在其中添加一行：

```
SSTATE_DIR="${BSPDIR}/sstate-cache"
```

```
MACHINE ??= 'imx6ull14x14evk'
DISTRO ?= 'fsl-imx-fb'
PACKAGE_CLASSES ?= 'package_rpm'
EXTRA_IMAGE_FEATURES ?= "debug-tweaks"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
BB_DISKMON_DIRS ??= "\
    STOPTASKS,${TMPDIR},1G,100K \
    STOPTASKS,${DL_DIR},1G,100K \
    STOPTASKS,${SSTATE_DIR},1G,100K \
    STOPTASKS,/tmp,100M,100K \
    ABORT,${TMPDIR},100M,1K \
    ABORT,${DL_DIR},100M,1K \
    ABORT,${SSTATE_DIR},100M,1K \
    ABORT,/tmp,10M,1K"
PACKAGECONFIG_append_pn-qemu-system-native = " sdl"
CONF_VERSION = "1"

DL_DIR ?= "${BSPDIR}/downloads/"
SSTATE_DIR="${BSPDIR}/sstate-cache"
ACCEPT_FSL_EULA = "1"

# Switch to Debian packaging and include package-management in the image
PACKAGE_CLASSES = "package_deb"
EXTRA_IMAGE_FEATURES += "package-management"
~
~
~
~
~
conf/local.conf [+]
```

如此一来会将编译过程的缓存放在 Yocto 项目的根目录下的 sstate-cache 目录，按照本文的配置来说即：`~/work/yocto/sstate-cache`目录。

6.3. 使用 ubuntu-base 制作系统

ubuntu-base 是 Canonical 官方构建的 Ubuntu 最小文件系统，包含 apt 管理器，基础包大小通常只有几十兆，其背后有整个 ubuntu 软件源支持，ubuntu 软件一般稳定性比较好，基于 ubuntu-base 按需安装 Linux 软件，深度可定制等，常用于嵌入式 rootfs 构建。

6.3.1. 获取源码

官网地址：http://cdimage.ubuntu.com/ubuntu-base/releases/

获取 ubuntu-base-20.04.5-base-armhf.tar.gz 并解压。

```
mkdir -p ~/work/ubuntu

sudo tar -xavf ubuntu-base-20.04.5-base-armhf.tar.gz -C ~/work/ubuntu
```

6.3.2. 准备 chroot 环境

安装 qemu 模拟器

```
sudo apt-get install qemu-user-static

sudo cp /usr/bin/qemu-arm-static ~/work/ubuntu/usr/bin/
```

增加 DNS 配置，后期可直接使用网络更新包，可自定义 DNS 服务器，替代 8.8.8.8 即可

```
echo "nameserver 8.8.8.8" >> ~/work/ubuntu/etc/resolv.conf
```

将下述脚本拷贝到 ch-mount.sh 文件中，并改变权限为可执行。

```
#!/bin/bash


function mnt_chroot() {
        echo "Mounting chroot environment..."
        sudo mount -t proc proc "${1}"/proc
        sudo mount -t sysfs sysfs "${1}"/sys
        sudo mount -t tmpfs tmpfs "${1}"/tmp
        sudo mount -t devtmpfs devtmpfs "${1}"/dev
        sudo mount -t devpts devpts "${1}"/dev/pts


        echo "Entering chroot environment..."
        sudo chroot "${1}"
}


function umnt_chroot() {
        echo "Unmounting chroot environment..."
```

```
        sudo umount "${1}"/proc
        sudo umount "${1}"/sys
        sudo umount "${1}"/tmp
        sudo umount "${1}"/dev/pts
        sudo umount "${1}"/dev
}

function ch_mount() {
        if [ "$#" -ne 2 ]; then
                echo "Usage: $0 -m <chroot_dir>"
                echo "Usage: $0 -u <chroot_dir>"
                exit 1
        fi

        if [ ! -d "$2" ]; then
                echo "Mount point not exists."
                exit 1
        fi

        if [ "$1" == "-m" ]; then
                mnt_chroot "$2"
        elif [ "$1" == "-u" ]; then
                umnt_chroot "$2"
        else
                echo "Invalid argument."
                exit 1
        fi
}
```

```
ch_mount "$@"
```

6.3.3. 安装系统

挂载系统

```
./ch-mount.sh -m ubuntu
```

```
me@me-virtual-machine:~/work$ ./ch-mount.sh -m ubuntu/
Mounting chroot environment...
[sudo] password for me:
Entering chroot environment...
root@me-virtual-machine:/#
```

挂载成功即可配置 ubuntu 文件系统与安装一些必要的软件。

基础包安装：

```
# apt update

# apt install sudo

# apt install language-pack-en-base

# apt install vim

# apt install ssh

# apt install net-tools

# apt install ethtool

# apt install ifupdown

# apt install iputils-ping

# apt install rsyslog

# apt install htop
```

添加 log，用户调试 ubuntu 系统的调试

```
# touch /var/log/rsyslog

# chown syslog:adm /var/log/rsyslog

# chmod 666 /var/log/rsyslog

# systemctl unmask rsyslog
```

```
# systemctl enable rsyslog
```

安装网络和语言包支持

```
# apt-get install synaptic
# apt-get install network-manager network-manager-gnome
# apt-get install language-pack-zh-hant language-pack-zh-hans
# apt-get install rfkill
# apt install -y --force-yes --no-install-recommends fonts-wqy-microhei
# apt install -y --force-yes --no-install-recommends ttf-wqy-zenhei
```

Xfce4 桌面系统安装

```
# apt-get install xinit
# apt-get install xfce4
```

设置 root 密码

```
passwd root
```

创建一个用户名为：ebyte

```
adduser ebyte
passwd ebyte
```

添加用户 ebyte 到 sudo 组

```
usermod -a -G sudo ebyte
```

6.3.4. 卸载系统

以上步骤操作完成后即可退出系统。直接在系统中输入 exit 退出系统，并使用脚本来卸载。

```
root@me-virtual-machine:/# exit
exit
me@me-virtual-machine:~/work$ ./ch-mount.sh -u ubuntu/
Unmounting chroot environment...
me@me-virtual-machine:~/work$
```

7. 参考资料

❖Linux kernel 开源社区：https://www.kernel.org/

❖NXP 开发社区：https://community.nxp.com

❖IMX6ULL date-sheet

❖imx-yocto-LF5.10.9_1.0.0

8. 修订说明

修订说明表

版本	修改内容	修改时间	编制	校对	审批
V1.0	初稿	24-06-21	WYQ	LJQ	WFX
V1.1	增加 buildroot 章节 增加 ECB20-6Y28C 系列相关描述	25-03-05	WYQ	LJQ	WFX

9. 关于我们

销售热线：4000-330-990 公司电话：028-61543675
技术支持：support@cdebyte.com 官方网站：https://www.ebyte.com
公司地址：四川省成都市高新西区西区大道 199 号 B5 栋

成都亿佰特电子科技有限公司
Chengdu Ebyte Electronic Technology Co.,Ltd.