Software Development Guide for ECK20 Core Board



Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

Table of Contents

Disclaimer and Copyright Notice 2
1. Development Environment Preparation 3
1.1. Hardware Preparation 3
1.2. Host software installation 6
1.3. Data Transfer 9
2. How to burn and update the system image 12
2.1. Official tool burning 12
2.2. Make an SD card boot card 13
2.3. Existing system updates 17
3. Build u-boot 24
3.1. Get the source code and compile 25
3.2. Compile uboot for booting from SD card 26
3.3. Set Uboot to start the system from the network 28
4. Building the kernel 29
4.1. Get the source code 29
4.2. Compile the kernel 29
5. How to adapt to different hardware platforms 30
5.1. u-boot 31
5.2. Linux 39
6. Making a file system 42
6.1. BuildRoot builds the root file system 42
6.2. Yocto build root file system 44
6.3. Use ubuntu-base to create a system 51
7. References 55
8. Revision Notes 55
9. About us 56

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Disclaimer and Copyright Notice

If any information in this document changes, we will not notify you separately. The document is provided "as is" without any warranties, including any warranties of merchantability, fitness for a specific purpose, or non-infringement, and any guarantees mentioned in proposals, specifications, or samples elsewhere. This document does not assume any responsibility, including liability for patent infringement arising from the use of information in this document. No intellectual property license is granted by this document in any form, explicit or implicit.

The test data mentioned in this document was obtained from the ebyte Laboratory and may vary slightly from actual results.

All trademark names, trademarks, and registered trademarks mentioned in this document belong to their respective owners.

The final interpretation right belongs to Chengdu ebyte Electronic Technology Co., Ltd.

Attention: Due to product version upgrades or other reasons, the content of this manual may be modified. Chengdu ebyte Electronic Technology Co., Ltd. reserves the right to modify the content of this manual without any notice or prompt. This manual is provided as a user guide. Chengdu ebyte Electronic Technology Co., Ltd. makes every effort to ensure the accuracy of information in this manual, but does not guarantee that the manual is completely error-free. All statements, information, and recommendations in this manual do not constitute any explicit or implicit warranties.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1. Development Environment Preparation

This chapter mainly introduces the necessary software and hardware environments for using the ECK20 development board before development, including essential hardware configurations, and the introduction of development and burning environments.

By reading this chapter, you will understand the relevant hardware tools, software development and debugging tools' installation and use, and be able to quickly set up the relevant development environment for subsequent development and debugging.

Host: i7-12700 + Win11 22H2Virtual machine: Uhuntu 18.04

1.1. Hardware Preparation

Hardware Connection Method

Hardware Equipment	Core Board Label	Notes
Power Supply	U1A	
Debug Serial Port	U1B: uart1	Baud rate 115200, data bits 8, stop bit 1, no parity, no hardware flow control

1.1.1. Startup Method

Refer to the hardware core board schematic diagram 'iMX6U CFG' page 'FUSE MAP' and the actual situation of the bottom plate configuration.

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

For USB Downloading, That is Serial Downloader ; generally, the system starts internally by configuring the pin foot state, also known as Internal Boot. For more details about boot modes, please refer to the NXP official manual 《 IMX6ULLRM.pdf 》 Section 5.1, located at:01_Documents/Datasheet/NXP/CPU.

The following two scenarios are listed:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1. SD cards are generally set as: BOOT_MODE1: ON; BOOT_MODE0: OFF; BOOT_CFG1[6]: ON.
2. USB downloading is set as:BOOT_MODE1: OFF; BOOT_MODE0: ON.

1.1.2.Virtual Machine Software Deployment

This chapter will introduce how to set up the development environment for iMX6UL.By reading this chapter, you will understand the relevant hardware tools, the installation and use of software development and debugging tools. And you can quickly set up the relevant development environment, preparing for the subsequent development and testing.

1.1.3.Virtual Machine Configuration

Please configure the virtual machine to use bridged networking according to the virtual machine software you are using, and enable directory sharing functionality.

1.1.4.Configure the Compilation Environment

Recommended: Ubunto 18.04 64bit.

Install software::

```
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
```

Add and Validate Cross-Compilation Toolchain:

```
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
```

Create a working directory:

```
mkdir ~/work
```

1.1.5.Enable TFTP Service on Virtual Machine

During the debugging phase, TFTP is often used to load the kernel for booting instead of burning flash devices repeatedly. Therefore, we need to install and enable the TFTP

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

service in Ubuntu first. Use the following command to install the tftpd-hpa service:

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

By default, files in the /var/lib/tftpboot directory will be shared, and only downloads are allowed, not uploads. If you want to customize the configuration, please refer to the official documentation: https://helpublicyttu.com/community/TFTP.This document has modified the path to /srv/tftp.

Add users to the tftp group and modify the permissions of the shared directory:

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

If the above operations are not performed, permission issues may arise, and non-root users may not be able to write files to shared directories.

1.1.6. Enable NFS Service on Virtual Machine

During the debugging phase, NFS shared files or direct NFS-mounted file systems are often required. Therefore, it is necessary to install and enable the NFS service in Ubuntu first. Use the following commands to install the NFS service:

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

Wait for the installation to complete. After installation, set the directory to be exported. Here, directly use the default tftp directory. Users can modify it themselves, or use a convenient text editor to modify the /ext/exports file. Configuration methods refer to the man manual:

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

1.1.7. Enable the SSH service of the virtual machine

After enabling Ubuntu's SSH service, we can use ssh client software to log in to Ubuntu

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

under Windwos, such as using MobaXterm, VSCode, and use the following command to enable SSH service under Ubuntu:

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

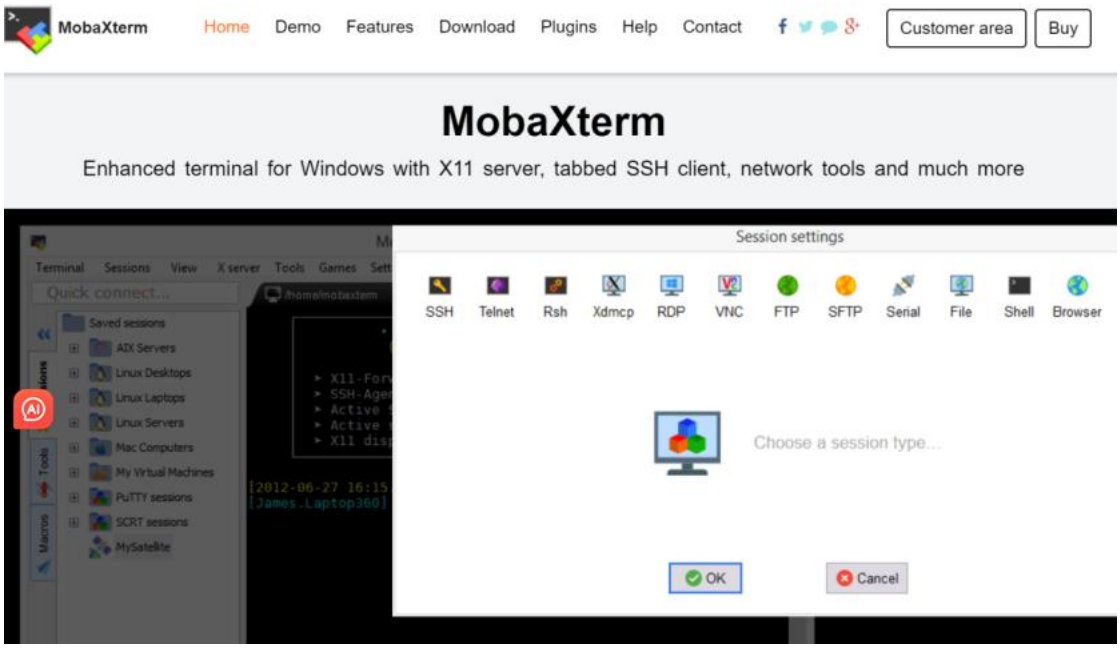
The above command installs the ssh service. The ssh configuration file is /etc/ssh/sshd_config. Use the default configuration. For more configuration, refer to the man manual related to sshd_config.

1.2. Host software installation

1.2.1. MobaXterm

1.2.1.1. Software download and installation

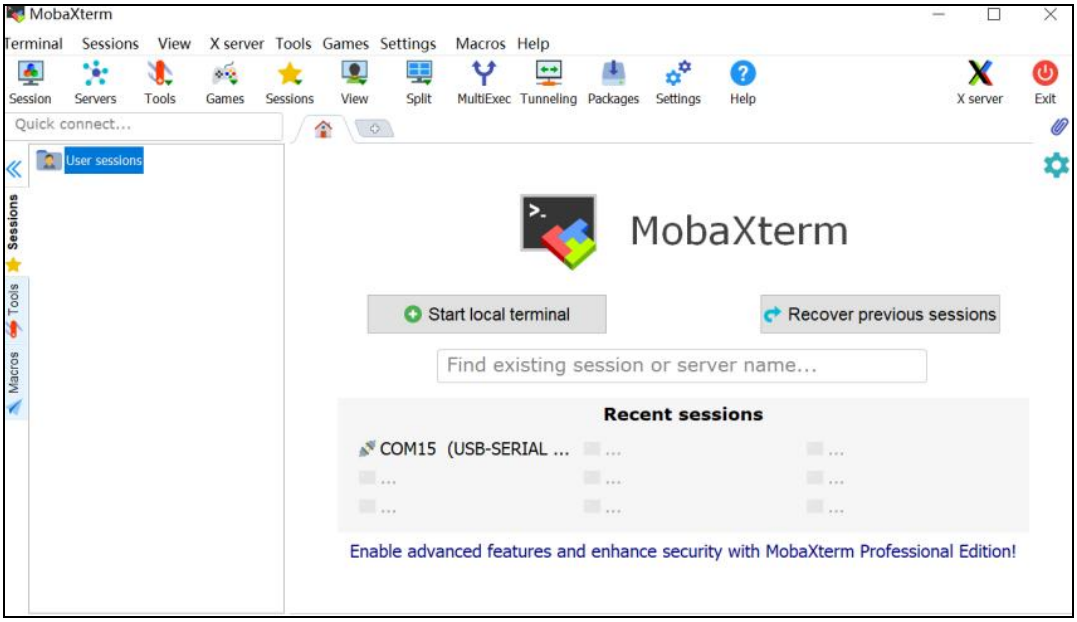
MobaXterm is a terminal software with powerful functions and is free. It is recommended to use this software as terminal debugging software. MobaXterm software can be downloaded from its official website athttps://mobaxterm.mobatek.net/, as shown in the figure:



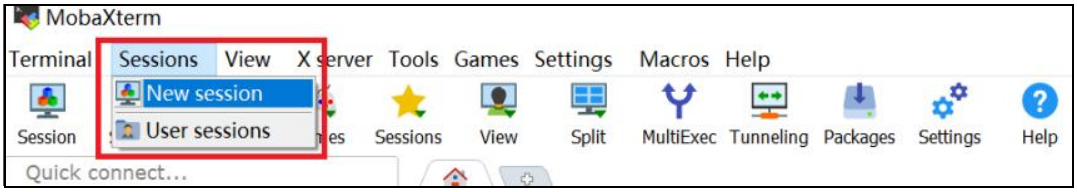
1.2.1.2. Software Usage

The software interface is shown in the figure:

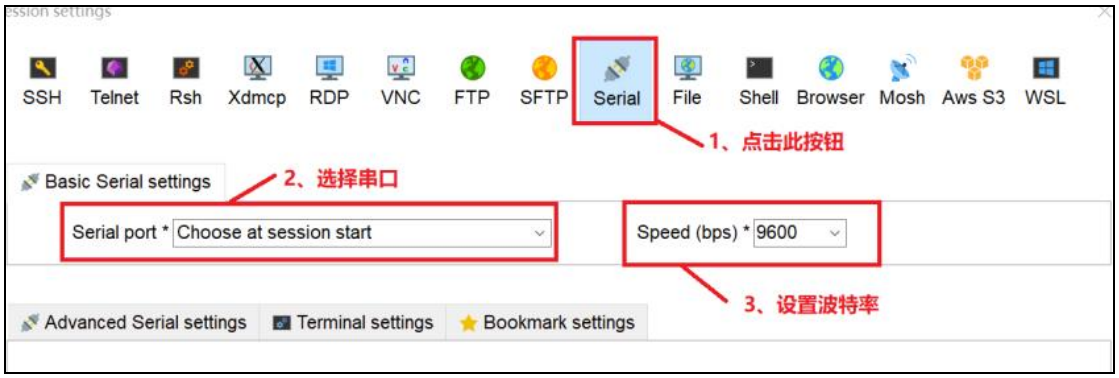
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



Click the "Sessions->New session" button in the menu bar to open the new session window, as shown in the figure



Establish a Serial connection, that is, a serial port connection, because the main purpose of using MobaXterm is to use it as a serial port terminal. Click the "Serial" button in the figure to open the serial port setting interface, as shown in the figure:



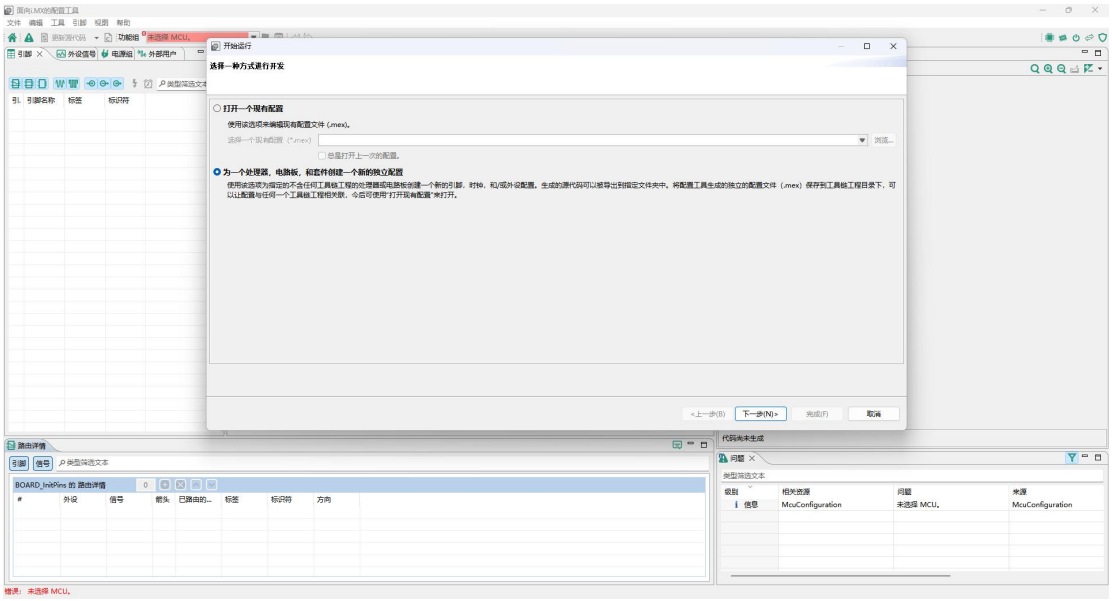
After opening the serial port setting window, first select the serial port number to be set. Therefore, you must first connect the development board to the computer with a serial port cable. Since the default serial port baud rate used by imx6ull is 115200, you need to set the baud rate to 115200.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

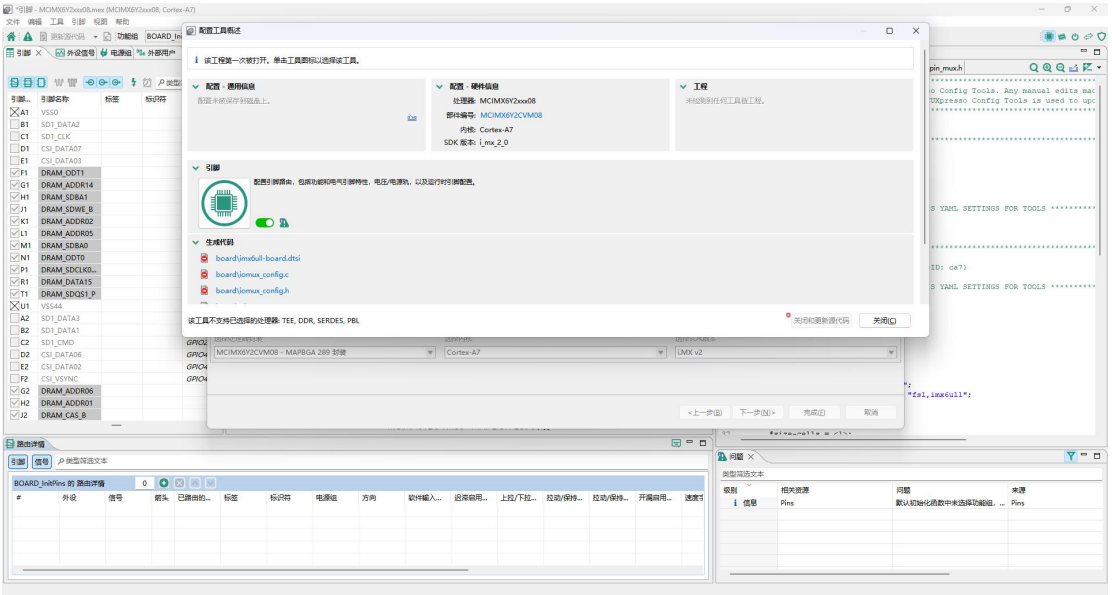
1.2.2. Config Tools for i.MX v15

This is the official tool used by NXP to configure the i.MX series CPU, mainly used to configure the iomuxc allocation in the device tree.

When you open the software, the following interface will appear automatically:

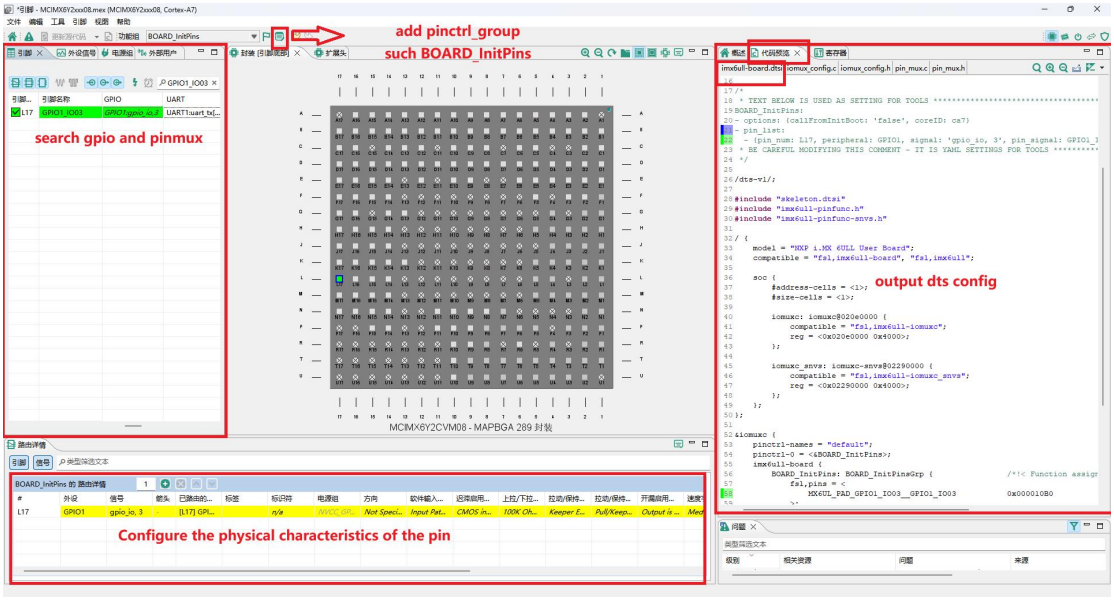


Click Next and select Search MCIMX6Y2xxx08. After completion, the following is displayed:



After closing the pop-up window, you need to pay attention to the following parts:

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



1.3. Data Transfer

In the previous chapters, we have configured various network services for the virtual machine. Now we just need to understand how to use them on the client and development board.

1.3.1. TFTP Client

In general, the development board only needs to use the download function. In this case, you can simply use the curl tool to download:

Install curl:

```
apt install curl -y
```

Use curl to download the file (only for Linux version of curl, you can choose to use filezilla for Windows):

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

1. /path/dist_file: Where to download, the name and path of the output file.

2. HOST: The IP address of the virtual machine. You need to pay attention to whether this address is accessible.

3. src_file: file directory and name. Note that you do not need to add the "TFTP_DIRECTORY" part in the configuration. For example, if the file is stored in

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

`/srv/tftp/aaa/bbb/ccc`, the src_file value is "aaa/bbb/ccc".

1.3.2. NFS Client

In general, the development board only needs to use the mount command to mount the NFS file system, but basic support needs to be installed:

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

Mount NFS:

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

1. HOST: The IP address of the virtual machine. You need to pay attention to whether this address is accessible.

2. path: The path on the NFS server to be mounted, such as /srv/tftp/aaa/bbb.

3. mnt_point: Where you want to mount the NFS file system, such as /mnt.

1.3.3. SSH

Generally speaking, we can also use the SFTP and SCP functions that come with SSH to download files, but there are not many cases of directly using commands to upload and download. More third-party tools are used, such as the SFTP function after using MobaXterm to log in to ssh, or VSCode using the remote-SSH plug-in to implement the download function during the development process. The way to use it is to right-click the file to be downloaded, and then select Download.

1.3.4. USB flash drive/SD card

For SD cards, you need to use a card reader, and the actual usage is the same as that of a USB flash drive.

1.3.4.1. Copying data from a virtual machine

You need to transfer the USB flash drive/SD card to the virtual machine first according to the virtual machine software you are using. Generally, it will be mounted automatically and you can directly access it in the file manager. After copying the data, uninstall the device in the file manager and then remove the device from the virtual machine.

If it is not mounted automatically, you can use the following two commands to mount

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

it manually:

1. Method 1

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

2. Method 2

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

In this case, you need to have some understanding of the USB disk/SD card partitions before you can specify the Y value in sdXY. The X value is generally 'b', but you also need to judge the situation yourself based on the information returned by lsblk.

Use method 1 to mount. You can access it without root privileges and without specifying a path, but you need to have the `udisks2` package installed in your system.

Manual uninstallation corresponds to the following command:

1. Method 1

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

2. Method 2

```
sudo umount /mnt_point
```

1.3.4.2. Copy data to the development board

For development boards, they are generally streamlined and may not have a udisks2 package or a GUI interface. Even if there is an interface, it may not support automatic mounting and operations often need to be performed from the command line.

1. Mount

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

2. copy

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

3. Unmount

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
umount /mnt_point
```

2. How to burn and update the system image

This chapter mainly introduces the burning and updating method based on the ECK20 core board.

2.1. Official tool burning

This section describes how to use the NXP official tool UUU (aka: MFGTools v3) to implement the USB fast burning function. The UUU software resources are stored in the directory: "04_Tools/uuu".

Hardware settings required before burning using official tools

Setting options	Setting content
Startup mode configuration	Configure the startup mode to: `Serial Downloader` burning mode
system	The official tool does not support Windows 7. Please use Windows 10 and above or Linux distributions.
wiring	Power supply, USB OTG interface connected to the development host

1. Switch the startup mode to Serial Downloader

2. Open a command line terminal and enter the uuu directory

3. Execute the programming command and wait for the return. If successful, the green Done will be displayed and Success will be displayed as 1.

When users compile and generate their own firmware, they only need to overwrite the files in the files directory accordingly, or they can modify the auto file to specify a new firmware path name.

Burn the system to eMMC:

```
./uuu eck20_6Y2xA_flash_emmc.auto # Stamp hole

./uuu eck20_6Y2xC_flash_emmc.auto # BTB
```

Burn the system to NAND:

```
./uuu eck20_6Y2xA_flash_nand.auto # Stamp hole

./uuu eck20_6Y2xC_flash_nand.auto # BTB
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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. Switch the boot mode to the corresponding Internal Boot mode.

2.1.1. Precautions

It should be noted that in USB download mode, you must not connect the SD card before powering on, otherwise it will try to boot from the SD card instead of entering download mode, causing the host to fail to recognize the development board. If you need to connect the SD card for use, please connect the SD card again after the USB device can be queried after powering on.

2.2. Make an SD card boot card

Ebyte has provided scripts for making images and SD boot cards, which can be used directly or for reference. They are located in 04_Tools/SD_Tools.

```
$ ~/imx6ull/ebyte/image tree
.
├── files
│   ├── ebyte-imx6ull-btb-emmc.dtb
│   ├── ebyte-imx6ull-btb-gpmi-weim.dtb
│   ├── ebyte-imx6ull-emmc.dtb
│   ├── ebyte-imx6ull-gpmi-weim.dtb
│   ├── SD_u-boot-emmc.imx
│   ├── SD_u-boot-nand.imx
│   └── zImage
├── gen_image.sh
├── out
│   └── ecb20-emmc-sd.wic
├── partition_table.txt
└── rootfs
    ├── modules.tar.bz2
    └── rootfs.tar.bz2

3 directories, 12 files
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The files directory stores the firmware that needs to be written to the image or SD card, and the rootfs stores the Ubuntu image used. The image can be made by referring to the subsequent chapters. partition_table.txt is the partition table for the SD card or image. Except for the device tree file, the name can be changed arbitrarily. If the name and position of other files are different from the above figure, the script needs to be modified.

2.2.1. Third-party tool image burning

To use the tool to burn, you need to first create a burnable image. Follow the image layout requirements in the corresponding `image layout` table in Chapter 4.3 of the NXP official document "i.MX Linux User's Guide.pdf", which is located in 01_Documents/Datasheet/NXP/imx-yocto-LF5.10.9_1.0.0.

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

From the above figure, we can know that for i.MX6ULL, U-Boot should be stored at the beginning of SD card or eMMC at 0x400 bytes, which is 1K offset, and the available size is 10M. Then there is 500M of space in FAT format to store the kernel and device tree, and the remaining space is for the file system storage.

But in fact, the kernel and device tree do not need such a large space of 500M. You can divide it according to the actual situation. Generally, the kernel size is about 10M.

2.2.1.1. Make an image

Use the existing script to create an SD card image:

```
sudo ./gen_image.sh nand

sudo ./gen_image.sh emmc
```

Note: If you add a parameter `btb` to the end of the command, it will build the board

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

for the BTB connector.

```
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$
```

The output files are located in the out directory of the current directory.

```
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$
```

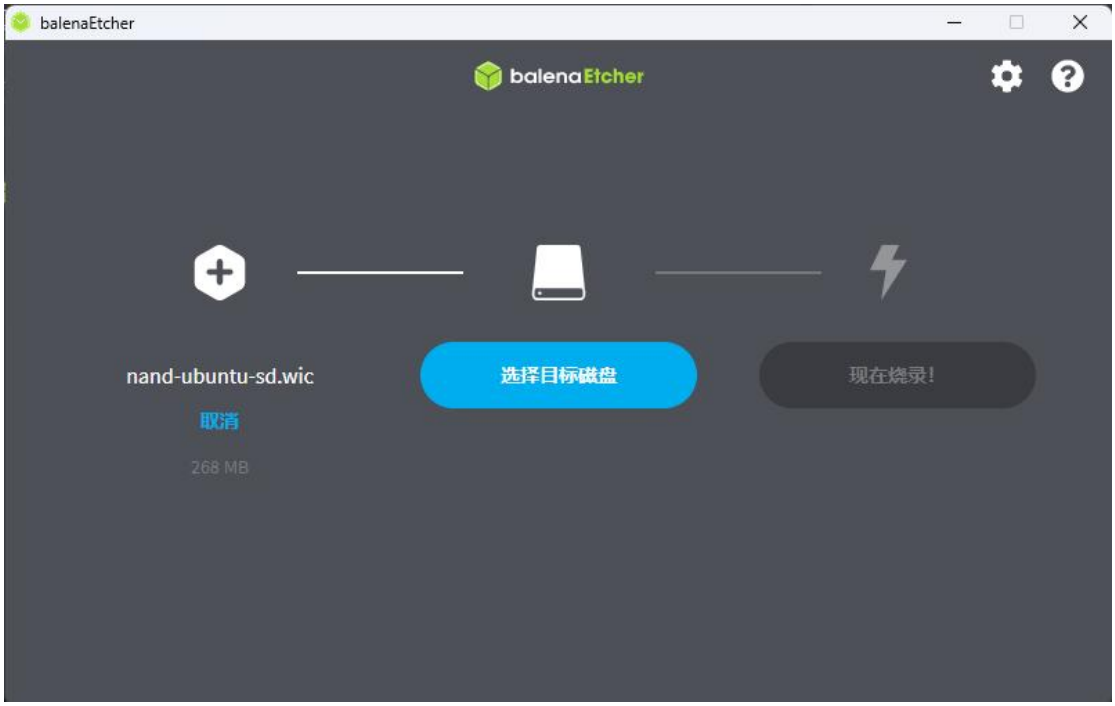

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.2.1.2. Burn the image

Use a third-party image burning tool to write the image created in the previous section to the SD card. A burning tool is already included in 04_Tools/: balenaEtcher-Portable-1.18.11.exe.



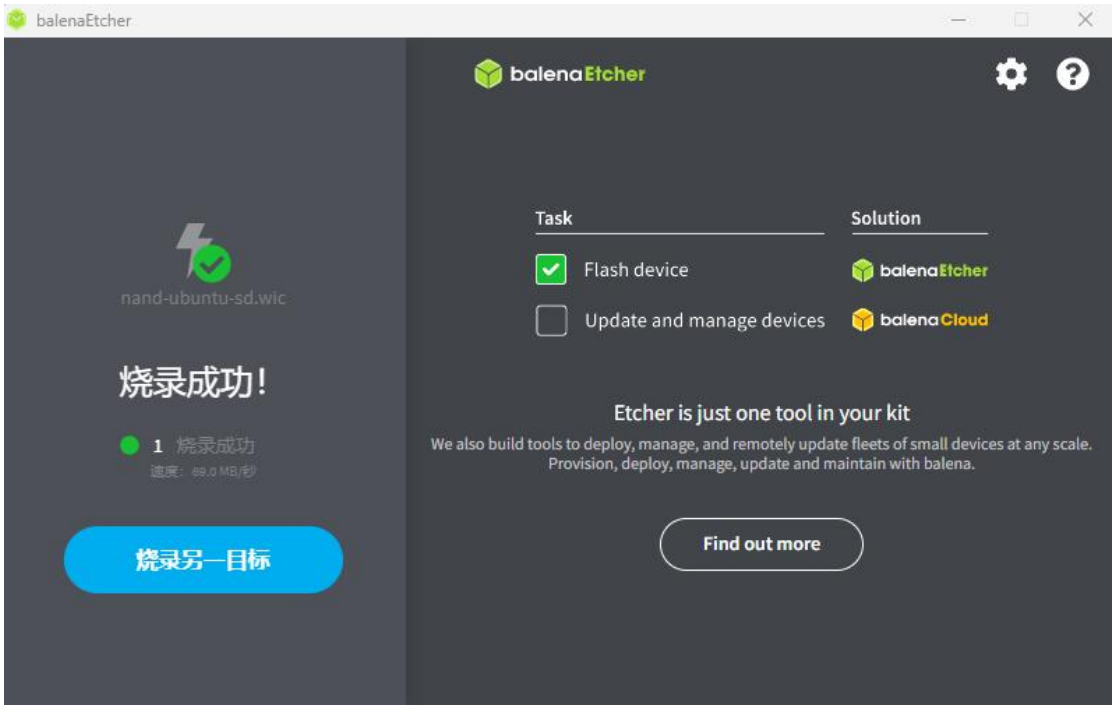
Select Burn from File and select the wic file created previously.



Select the target disk and click Burn Now to start burning. After the burning is

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

completed, remove the device and use the SD card as the boot device.



2.2.2. Directly create a startup card

The method of directly making an SD boot card is similar to the steps of making an image in the third-party tool image burning, except that the image file is written directly into the SD card device.

1. Use the following command to view device information:

```
lsblk
```

2. Execute the flashing:

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

2.3. Existing system updates

In the previous chapters "Official Tool Burning" and "Third-party Tool Burning SD Card", we have successfully solidified the Linux file system and the system has been successfully started. But sometimes during the development process, we try to compile and make uboot, kernel and file system by ourselves. Later, we only need to update one of uboot, device tree and file system. Especially during the development process, frequent updates may be required. In order to save development time, we can use a separate update method to

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

update the part we need to update, instead of spending more time to re-make an SD system startup card or use scripts to solidify the system to eMMC or NAND.

For the method of transferring files between the development host and the development board, please refer to this document "Data Transfer"chapter.

2.3.1. eMMC

If the user is using an eMMC core board and wants to update the firmware in an existing system, we need to boot from eMMC or from an SD card to replace our own firmware. Here is a brief summary of the eMMC partition table.

eMMC partition table

Partition name	effect
/dev/mmcblk1boot0	Boot partition 0, stores u-boot
/dev/mmcblk1boot1	Boot partition 1, not used by default
/dev/mmcblk1p1	fat32 format, stores kernel and device tree
/dev/mmcblk1p2	ext4 format, storing the root file system

2.3.1.1. U-boot

Boot the system from eMMC, or boot the system from SD card to update uboot to eMMC.

1. Copy the compiled u-boot-dtb.imx file to the current directory of the eMMC/SD card system through nfs/tftp/U disk and other methods.

2. Remove the write protection of mmcblk1boot0 partition

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

3. Clear the old data of mmcblk1boot0 partition

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

4. Write u-boot data to mmcblk1boot0 partition

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

5. Enable mmcblk1boot0 partition write protection

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

6. Enable boot partition

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

2.3.1.2. Kernel + Device Tree

According to the partition table above, the kernel and device tree are stored in a fat32 file system.

1. Copy the compiled kernel and device tree files to the current directory of the eMMC/SD card system through nfs/tftp/U disk and other methods.

2. Mount the mmcblk1p1 partition

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

3. Copy the kernel and device tree to the mmcblk1p1 partition.

```
cp zImage /boot

cp *.dtb /boot

sync
```

4. Unmount the partition

```
umount /boot
```

5. Restart the system

```
reboot
```

2.3.1.3. Root file system

Generally speaking, onlyUse SD card boot to update the eMMC file system. A running system cannot format itself!

1. Copy the prepared root file system compressed package (rootfs.tar.bz2) to the current directory of the eMMC/SD card system through nfs/tftp/U disk.

2. Mount the mmcblk1p2 partition

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

3. Clear old data in mmcblk1p2 partition

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


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4. Unzip the new root file system to the mmcblk1p2 partition

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

sync
```

5. Unmount the partition

```
umount /mnt
```

6. Restart the system

```
reboot
```

2.3.2. NAND

If the user is using a NAND core board and wants to update the firmware in an existing system, we need to boot from NAND or boot from an SD card to replace our own firmware. Here is a brief summary of the NAND partition table.

NAND Partition Table

Partition name	effect
/dev/mtdblock0	Boot partition 0, stores u-boot, 10M
/dev/mtdblock1	Kernel partition, stores Linux kernel, 12M
/dev/mtdblock2	dtb partition, stores Linux device tree, 2M
/dev/mtdblock3	ubi partition, store the root file system, all remaining space

It should be noted that if you boot from NAND, the root partition is mounted read-only by default, and files cannot be copied to the system. You need to modify the boot parameters in u-boot to mount the root partition in a read-write manner.

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

However, if you use nfs or a USB flash drive, you do not need to copy the files to NAND and can bypass the read-only restriction of the root partition by mounting it.

2.3.2.1. U-boot

Due to the limitation of i.MX6ULL's own BootROM, you need to use the official NXPkobs-ng softwareUpdate u-boot.

1. Copy the compiled u-boot-dtb.imx file to the current directory of the eMMC/SD

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

card system through nfs/tftp/U disk or enter the directory mounted by U disk/nfs.

2. Erase old u-boot

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

3. Re-write the updated u-boot

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

4. Restart the system

```
reboot
```

2.3.2.2. Kernel

According to the partitioning information above, the kernel is stored in the mtd1 partition and there is no file system.

1. Copy the compiled zImage file to the current directory of the eMMC/SD card system through nfs/tftp/U disk or enter the directory mounted by U disk/nfs.

2. Erase the old kernel

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

3. Rewrite the updated zImage

```
nandwrite -p /dev/mtd1 zImage
sync
```

4. Restart the system

```
reboot
```

2.3.2.3. dtb

According to the partitioning situation above, the device tree is stored in the mtd2 partition and no file system exists.

1. Copy the compiled zImage file to the current directory of the eMMC/SD card system through nfs/tftp/U disk or enter the directory mounted by U disk/nfs.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2. Erase the old device tree

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

3. Rewrite the updated device tree

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

4. Restart the system

```
reboot
```

2.3.2.4. Root file system

Generally speaking, onlyUse SD card boot to update NAND file system. A running system cannot format itself!

1. Use nfs/tftp/U disk to copy the prepared root file system compressed package (rootfs.tar.bz2) to the current directory of the NAND/SD card system or enter the directory mounted by U disk/nfs.

2. Erase the old file system

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

3. Rebuild the ubi file system

```
ubiformat /dev/mtd3
ubiattach /dev/ubi_ctrl -m 3
ubimkvol /dev/ubi0 -Nrootfs -m
```

4. Mount the new ubi file system

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

5. Unzip and copy the updated file system to the ubi file system

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

6. Restart the system

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
reboot
```

2.3.3. SD Card

Since both SD cards and eMMC can be classified as MMC devices, their usage is similar to that of eMMC.

2.3.3.1. U-boot

Transfer the SD card to the virtual machine system through the card reader, and directly write the updated U-Boot in the virtual machine.

1. Enter the directory where the u-boot-dtb.imx file is stored.

2. Find SD card device

```
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. Write u-boot data to SD card device

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

sync
```

2.3.3.2. Kernel + Device Tree

Transfer the SD card to the virtual machine system through the card reader, and directly copy the files to the EFI partition in the virtual machine.

1. Enter the directory where the kernel and device tree files are stored.

2. Find SD card device

```
lsblk
```

3. Mount the sda1 partition

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

4. Copy the kernel and device tree to the sda1 partition.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
cp zImage /mnt
cp *.dtb /mnt
sync
```

5. Unmount the partition

```
umount /mnt
```

2.3.3.3. Root file system

Transfer the SD card to the virtual machine system through the card reader, and directly copy the files to the root partition in the virtual machine.

1. Enter the directory where the prepared root file system compressed package (rootfs.tar.bz2) is stored.

2. Find SD card device

```
lsblk
```

3. Mount sda2 partition

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

4. Update the file system to the sda2 partition.

```
sudo rm -rf /mnt/*
sudo tar -xavf rootfs.tar.bz2 -C /mnt
sync
```

5. Unmount the partition

```
umount /mnt
```

3. Build u-boot

The full name of uboot is Universal Boot Loader. uboot is an open source software that complies with the GPL agreement. uboot is a bare metal code and can be regarded as a bare metal comprehensive routine. Now uboot already supports advanced functions such as LCD screen, network, USB, etc.http://www.denx.de/wiki/U-Boot/.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.1. Get the source code and compile

Since each chip manufacturer has customized the official uboot, it is generally a better choice to use the version provided by the chip manufacturer. Based on the uboot provided by the chip manufacturer, Ebyte provides usage examples based on its own core board to facilitate users to customize more quickly.

In the "Configure the Compilation Environment" section above, we have installed various dependency packages required for compilation and created a working directory `${HOME}/work`. Now unzip the source code provided by our company:

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

ls ~/work/uboot
```

Check the directory as follows:

```
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:~$
```

You can see that there is a build.sh script, which is provided by our company and is used to compile the core board sample code provided by our company. After compilation, an out directory will appear in the top-level directory of uboot to store all output files.

```
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
```

Execute the following code to compile the core board using NAND:

```
./build n
```

Execute the following code to compile the core board using eMMC:

```
./build e
```

Sometimes only some files not related to uboot configuration are modified. Use the following command to quickly compile

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
./build
```

After compiling, you can see the following situation.

```
  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$
```

The out directory contains all the files generated by the compilation process. Most of these files are not needed by users. Generally, we only need to use the `out/u-boot-dtb.imx` file, which is the final product of compiling the U-Boot source code, that is, the sample U-Boot firmware of the ECK20 core board.

3.2. Compile uboot for booting from SD card

It can be found that the build.sh script only has the option to compile to NAND or eMMC. If the uboot compiled at this time is used to boot from an SD card, it may not be used, especially for boards using NAND.

Ebyte has made customized modifications for this situation, and users only need to enable the option to boot from the SD card.

First, use your favorite text editor to open and modify the build.sh script, remove the `#` comment on line 20 and

Enable the make menuconfig feature:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
#!/bin/bash

if [ "$_" != "$0" ];    then
        _exit="return"
else
        _exit="exit"
fi

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-

function init_config() {
    # make distclean
    rm -rf out
    gen_compile_commands="bear"

    make O=out "$1" || ${_exit}

    # make O=out menuconfig || ${_exit}
    # make O=out savedefconfig || ${_exit}
}

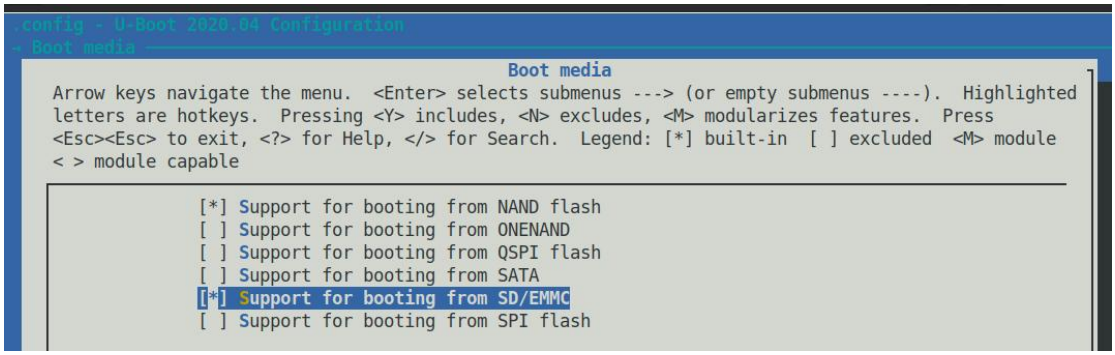
```

Then compile again for eMMC or NAND:

```
./build.sh e

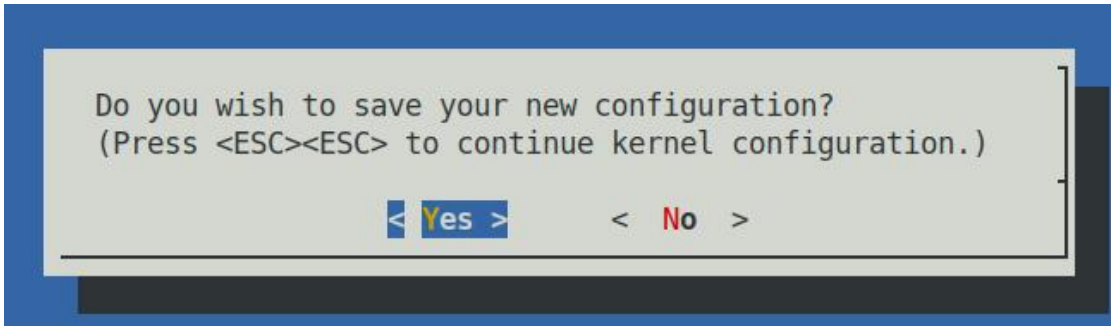
./build.sh n
```

At this time, you will enter the configuration interface and enable the following option `Boot media -> support for booting from SD/EMMC. To enable it, use the arrow keys to move the cursor to this option, and then press the space bar to select it.



After enabling, move the cursor to Exit and press Enter to return. Repeat the return multiple times until the following interface appears. Select Yes to officially exit, and then wait for the compilation to complete.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



3.3. Set Uboot to start the system from the network

When booting the system from the network, the kernel and device tree are often downloaded through tftp, and then the file system is mounted by nfs. To boot from the network, you need to set up the network card first, and configure the nfs service and tftp service of the server (virtual machine).

```
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 is used to configure the MAC address. The example provided by our company has enabled the random MAC address in the u-boot stage, so this configuration is not required.

2. Ipaddr is used to configure the IP address of the development board. This address and serverip should be in the same network segment.

3. serverip is used to configure the tftp server address, which is the address allocated by the virtual machine bridge mode.

4. image is the kernel path that needs to be booted.

5. fdt_file is the device tree path corresponding to the development board.

6. rootpath is the path required by nfs to mount the file system, where the root file system is stored.

Set environment variables to add network startup options:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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
```

Call the network boot option script to perform network boot.

```
run netboot
```

4. Building the kernel

Since each chip manufacturer has customized the official Linux kernel, it is generally a better choice to use the version provided by the chip manufacturer. Ebyte provides usage examples based on its own core board based on the Linux kernel provided by the chip manufacturer, so that users can customize more quickly.

4.1. Get the source code

In the "Configure the Compilation Environment" section above, we have installed various dependency packages required for compilation and created a working directory `${HOME}/work`. Now unzip the source code provided by our company:

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

ls ~/work/linux
```

Check the directory as follows:

```
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. Compile the kernel

You can see that there is a build.sh script, which is provided by our company and is used to compile the core board sample code provided by our company. After the compilation is completed, an out directory will appear in the top-level directory of Linux to store all output files.

The script is used in a similar way to the script provided by u-boot, but it no longer

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

distinguishes between eMMC, NAND or SD card. However, it still supports fast compilation without kernel function changes.

Execute the following commands to configure the kernel and compile:

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

At this point, you only need to pass any parameters to the script to configure and compile the kernel.

Execute the following command to quickly compile:

```
./build.sh
```

After compiling, you can see the following situation.

```
  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$
```

The out directory contains all the files generated by the compilation process, most of which are unnecessary for users. In general, we only need to use the Linux kernel file: `out/arch/arm/boot/zImage` file and the device tree file: `out/arch/arm/boot/dts/ebyte-imx6ull-*.dtb` and the kernel module file placed in the file system: `out/modules.tar.bz2`. The device tree file needs to be selected according to the core board and the baseboard. The module compressed package of the current kernel can be unzipped to the `/lib` directory of the root file system if necessary.

5. How to adapt to different hardware platforms

Users need to have a detailed understanding of the CPU chip manual, the ECK20 core board product manual, and the baseboard pin definitions in order to correctly configure and use these pins according to their actual functions.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1. u-boot

In order to facilitate users to modify and adapt their own baseboards, the modified and added parts of the u-boot example provided by Ebyte are listed here:

u-boot modify file table

document	illustrate
arch/arm/dts/Makefile	Add device tree for compiling device tree
arch/arm/dts/ebyte-imx6ull-14x14-base.dtsi	Basic configuration of ECK20
arch/arm/dts/ebyte-imx6ull-14x14-emmc.dts	eMMC board device tree for ECK20
arch/arm/dts/ebyte-imx6ull-14x14-gpmi-weim.dts	NAND board device tree for ECK20
arch/arm/dts/ebyte-imx6ull-14x14-u-boot.dtsi	ECK20's u-boot stage special definition related device tree
arch/arm/mach-imx/mx6/Kconfig	Add development board configuration enable item for compiling your own development board
board/ebyte/ebyte_imx6ull/Kconfig	Development board-level configuration options
board/ebyte/ebyte_imx6ull/MAINTAINERS	Describe the development board related files
board/ebyte/ebyte_imx6ull/Makefile	Makefile for compiling the development board
board/ebyte/ebyte_imx6ull/ebyte_imx6ull.c	Board initialization file
board/ebyte/ebyte_imx6ull/imximage-ddr256.cfg	DDR related configuration files, for NAND boards
board/ebyte/ebyte_imx6ull/imximage-ddr512.cfg	DDR related configuration files, for eMMC boards
include/configs/ebyte_imx6ull.h	Board Configuration File
configs/ebyte_imx6ull_nand_defconfig	Default u-boot configuration for building NAND boards
configs/ebyte_imx6ull_emmc_defconfig	Default u-boot configuration for building eMMC boards
tools/logos/ebyte.bmp	Company logo for display

5.1.1. Device Tree

For the arch/arm/dts/ebyte-imx6ull-14x14-base.dtsi file, other extended interfaces and devices can reference them as follows (for reference only):

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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";
};

```

After the user adds a new device tree source file, it is also necessary to add the device tree compilation information in the arch/arm/dts/Makefile in the same directory, so that the corresponding device tree binary file can be generated when compiling the kernel.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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
```

After the modification is completed, you can add the onboard description of the device driver, and compile and generate the device tree dtb files ebyte-imx6ull-14x14-gpmi-weim.dtb and ebyte-imx6ull-14x14-emmc.dtb through the method in Section 3.1.

5.1.2. Board level files

This part requires separate configuration for each baseboard, and is mainly the work that needs to be performed during the system initialization phase.

Although most of the configurations are placed in the device tree file, during the initialization phase, you still need to pay attention to whether some pins that are not currently supported in the device tree have conflicts and whether they conform to the actual board conditions. For example, in the ebyte_imx6ull.c file, the LCD backlight control pins:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
#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

For users who need to use a custom logo, just replace `tools/logos/ebyte.bmp` with your own logo file. It is important to note that the file name should be the same as the `SYS_BOARD` or `SYS_VENDOR` option configuration in the `board/ebyte/ebyte_imx6ull/Kconfig` file, otherwise it will fall back to the u-boot default logo.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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
```

For deb systems (such as Ubuntu), you can install the netpbm package and then use the following command to create a logo file that can be used in the u-boot stage:

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

5.1.4. u-boot module enabled

In the examples provided by Ebyte, some device drivers are enabled, such as PHY function, LED function, and DDR configuration file path. Users should configure and enable according to the functions actually used by the baseboard.

During the debugging phase, it would be troublesome to modify the configs/*_defconfig files every time, and the configuration may have dependency issues. It is recommended to use the manual configuration function in menuconfig.

5.1.4.1. Optional module function

1. Execute the following command to add basic environment variables for executing the menuconfig function.

```
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. Check whether the configuration is successful:

```
env | grep arm
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The return is as follows:

```
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/loca
l/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. Perform preliminary configuration and select a configuration file as the basic configuration. Here we recommend using the sample file provided by Ebyte:

```
make O=out ebyte_imx6ull_nand_defconfig

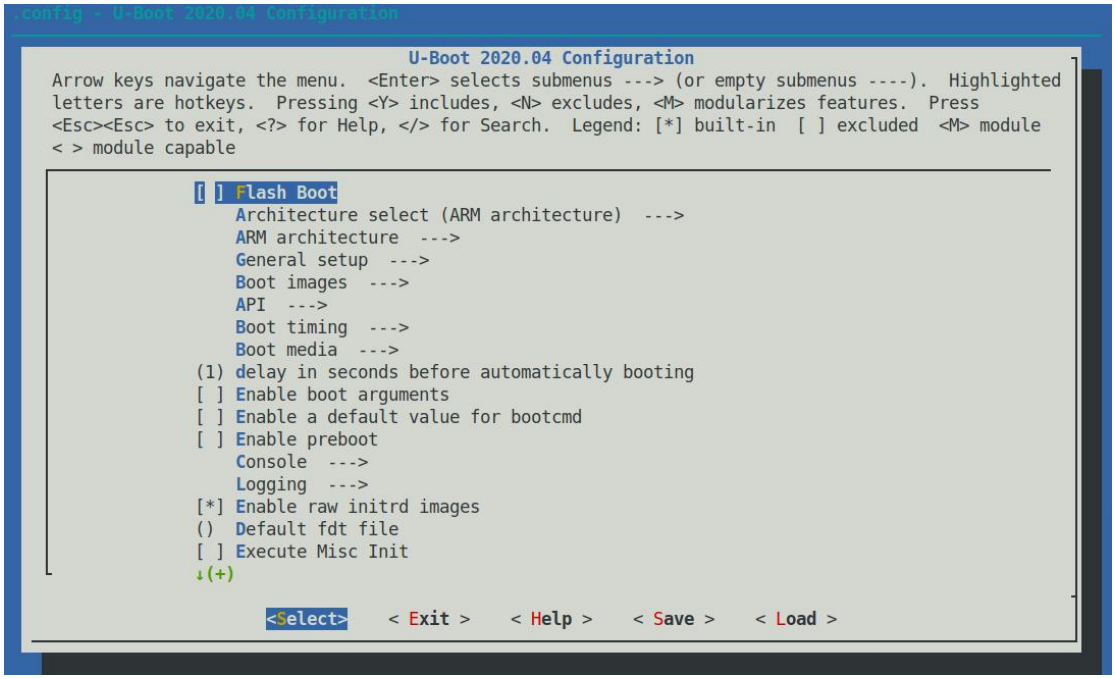
make O=out ebyte_imx6ull_emmc_defconfig
```

4. Then execute the following command to enter the menuconfig interface. For usage, refer to the section "Building u-boot" in the chapter "Compiling u-boot for booting from SD card".

```
make O=out menuconfig
```

For the function names that need to be enabled, you can find them by viewing the Kconfig files scattered in various directories.

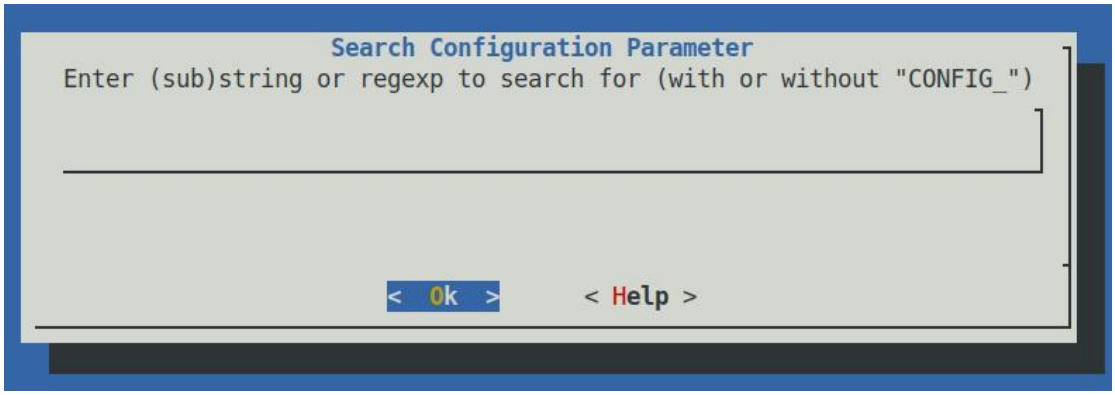
If you know the function name but don't know which menu it is in, you can press the "/" button in the following interface to enter the search function:



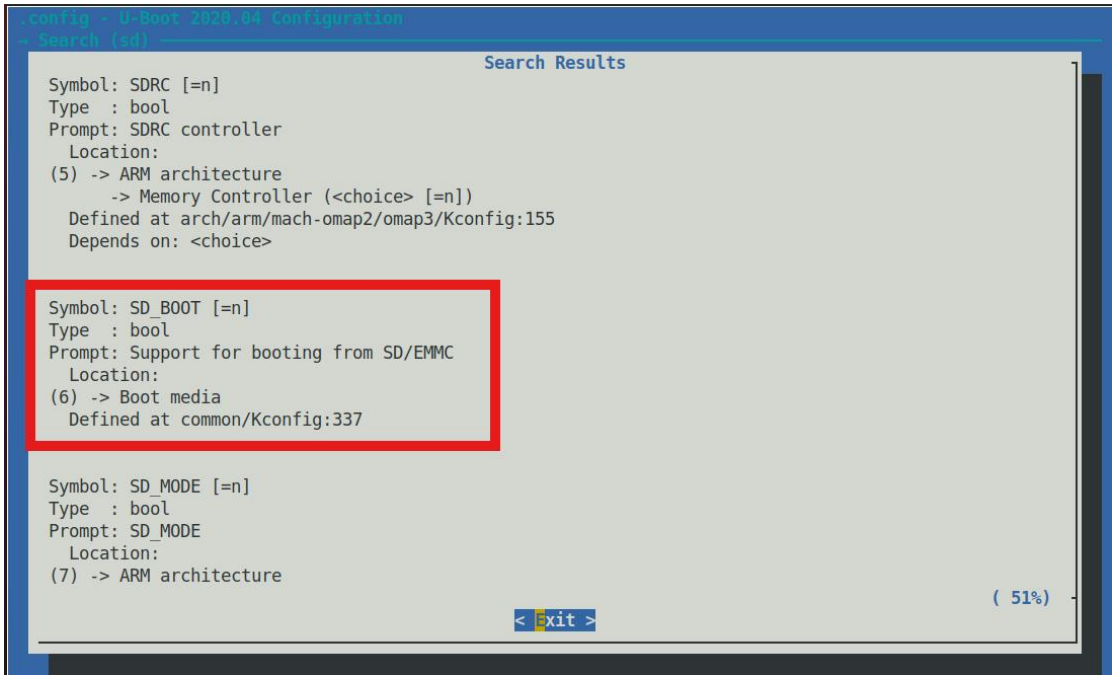
After entering the search, enter the function item you want to find and press Enter to

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

confirm:



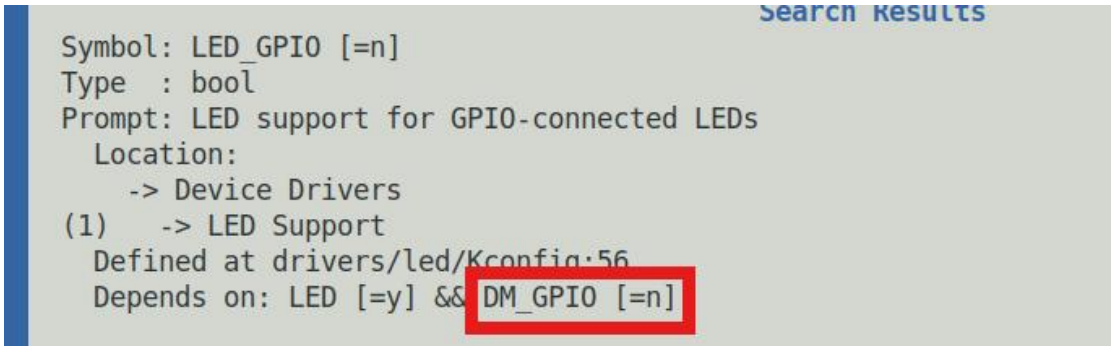
Search for SD functions like here:



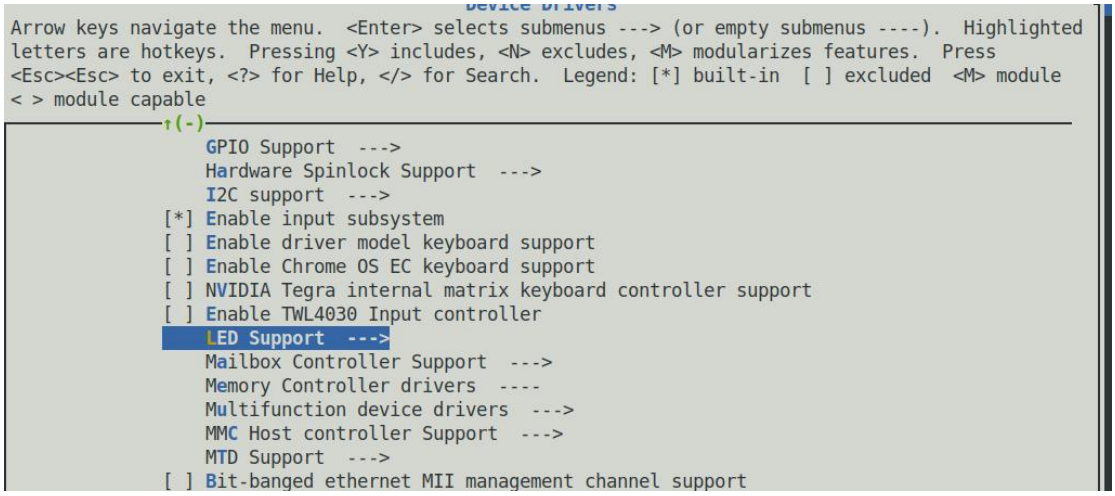
After moving up and down, you will find a large number of options. There will be a number in front of each option. When we find the option we need, tap the number 6 in front to enter the menu sub-item where the option is located.

However, sometimes you may find that the option is not available when you should theoretically enter the location where the option is located. In this case, you need to return to the search interface to determine whether the dependencies of the option are met, such as:

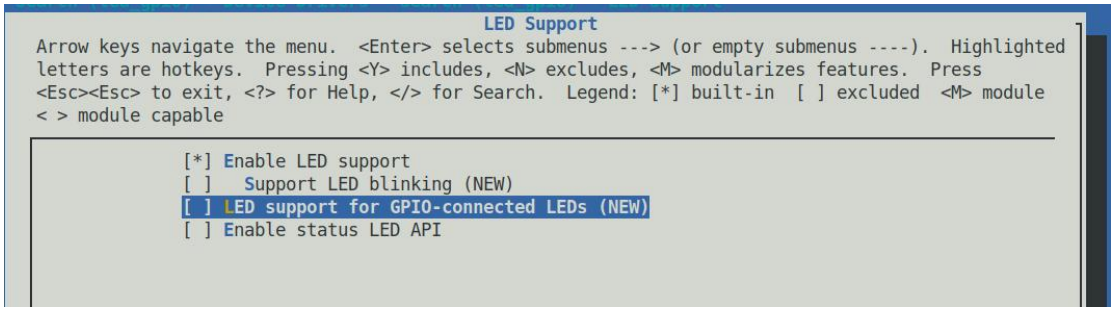
Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



At this time, press 1 to enter the interface as follows:



After enabling the missing DM_GPIO function above, press button 1 again to enter the following interface:



5.1.4.2. Save Configuration

After completing all module function selections, execute the following command to save the configuration:

```
make O=out savedefconfig
```

Then the defconfig file will be generated in the out directory. This file can be used as the default configuration file for the target development board.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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

In order to facilitate users to modify and adapt to their own baseboards, the kernel example modifications and additions provided by Ebyte are listed here:

Linux Modify File Table

document	illustrate
arch/arm/boot/dts/Makefile	Add device tree for compiling device tree
arch/arm/boot/dts/ebyte-imx6ull-base.dtsi	Basic configuration of ECK20-6Y2XA
arch/arm/boot/dts/ebyte-imx6ull-gpmi-weim.dts	eMMC board device tree for ECK20-6Y2XA
arch/arm/boot/dts/ebyte-imx6ull-emmc.dts	NAND board device tree for ECK20-6Y2XA
arch/arm/boot/dts/ebyte-imx6ull-btb-base.dtsi	Basic configuration of ECK20-6Y2XC
arch/arm/boot/dts/ebyte-imx6ull-btb-gpmi-weim.dts	eMMC board device tree for ECK20-6Y2XC
arch/arm/boot/dts/ebyte-imx6ull-btb-emmc.dts	NAND board device tree for ECK20-6Y2XC
arch/arm/configs/ebyte_imx6ull_defconfig	By default, kernel feature profiles
drivers/video/fbdev/core/fbcon.c	Disable framebuffer terminal cursor blinking
drivers/video/fbdev/core/fbmem.c	Configure the cursor to be centered by default
drivers/video/logo/logo_linux_clut224.ppm	Logo for display

5.2.1. Device Tree

For the arch/arm/boot/dts/ebyte-imx6ull-base.dtsi file, other extended interfaces and devices can reference them as follows (for reference only):

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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";
};

```

After the user adds a new device tree source file, it is also necessary to add the device tree compilation information in the arch/arm/dts/Makefile in the same directory, so that the corresponding device tree binary file can be generated when compiling the kernel.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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) += \
```

After the addition is completed, you can add the onboard description of the device driver, and compile and generate the device tree dtb files ebyte-imx6ull-emmc.dtb and ebyte-imx6ull-gpmi-weim.dtb through the method in Chapter 4.

5.2.2. Logo

For users who need to use a custom logo, just replace `drivers/video/logo/logo_linux_clut224.ppm` with your own logo file.

For deb systems, you can install the netpbm package and then use the following command to create a logo file that can be used during the kernel boot phase:

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

5.2.3. defconfig configuration

Some device drivers are enabled in the examples provided by Ebyte, such as the PHY

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

function and the audio wm8960 driver. Users should configure and enable the functions actually used by the baseboard.

During the debugging phase, it would be troublesome to modify the arch/arm/configs/*_defconfig files every time, and the configuration may have dependency-related problems. It is recommended to use the manual configuration function in menuconfig. For this part, please refer to the description in Section 5.1.4 of this article.

6. Making a file system

This is the last step of Linux system transplantation. After the root file system is built, it means that we have a complete and runnable minimum system. In the future, we will write and test Linux drivers on this minimum system, transplant some third-party components, and gradually improve this minimum system. Finally, we will get a relatively complete operating system with complete functions and complete drivers.

6.1. BuildRoot builds the root file system

Buildroot is a tool set for building embedded Linux systems. It is an open source project designed to help developers create minimal, customized Linux systems. Buildroot provides a complete set of automated build processes that can build the Linux kernel, Bootloader (such as U-Boot), root file system, and other dependent libraries and applications from source code.

6.1.1. Get the source code

At present, the buildroot source code adapted for our company's ECB20 series single-board computers has been attached in the download materials. Its path is located in 04_Sources/buildroot.tar.xz. At the same time, in order to facilitate users to quickly build and avoid network files during the build process, the source code corresponding to most packages required to build the final image is also attached. Its path is located in 04_Sources/buildroot_dl.tar.xz. Please unzip it to the dl subdirectory of the buildroot root directory.

```
# Create a working directory and unzip the buildroot source code
mkdir work && cd work
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
tar -xaf <your_path>/buildroot.tar.xz

# Enter the buildroot directory and unzip the dl package (optional)

cd buildroot

tar -xaf <your_path>/buildroot_dl.tar.xz
```

After decompressing the buildroot source code, the following contents are available:

```
$ ~/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. Compile source code

We provide a `build.sh` script to facilitate users to build and adapt to our company Image for the ECB20 single board computer.

```
$ ~/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
```

The `config` parameter is used to configure buildroot. Users can customize the configuration `configs/ebyte_imx6ulx_qt_defconfig` instead of the initial configuration provided by our company, but please at least make sure that your configuration refers to the official `configs/freescale_imx6ullevk_defconfig` of NXP.

Note: For users using BTB connector, configure it as `configs/ebyte_imx6ulx_qt_btb_defconfig`

After completing the initial configuration, execute the following command to build buildroot:

```
# Not using build.sh

make

# Use build.sh to modify the dl directory name to .downloads

mv dl .downloads
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
./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'" (stderr
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
```

As shown above, the compilation is complete. You can see the compiled file system in the output/images directory, and it has been packaged into an image file suitable for the ECB20 single board computer:

```
 $ ~/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 build root file system

Since the chip manufacturer has customized the standard Yocto, the following is the basic process of using the version provided by the chip manufacturer. For more details on the use process, refer to the NXP official document "i.MX_Yocto_Project_User's_Guide.pdf", which is located in `01_Documents/Datasheet/NXP/imx-yocto-LF5.10.9_1.0.0`.

6.2.1. Get the source code

Download the repo tool:

```
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
```

Configure the repo tool to update the source using a mirror (optional in China):

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

Enable relevant environment variables:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
source ~/.profile
```

After entering the project directory, use NXPOfficial REPOGet the source code and check the directory as follows:

```
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. Install compilation dependencies

Compiling Yocto requires more dependencies than those required to compile the kernel and u-boot.

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

6.2.3. Initial configuration

Use the provided imx-setup-release.sh script to create a workspace and then build the image in this space. After executing the script, you will be asked to read and agree to the copyright statement before entering the built directory. At the same time, the script will create and enter the build directory by default. If a specific directory name is required, you can use the -b parameter, such as "-b out_dir".

The optional parameters of MACHINE and DISTRO can be viewed in the `README-IMXBSP` file in the directory:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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>
```

Execute the following command to compile the example provided by NXP and use the framebuffer as the display function:

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

Read the User Agreement and agree to:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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. Start Compiling

6.2.4.1. Compile target selection

Before starting to compile, you need to first understand which projects can be compiled and generated under the current configuration of Yocto. Use the following command to view the optional compilation targets:

```
bitbake -s
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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
```

Among so many projects above, the main goal we need to focus on is this part:

```
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
```

The compilation target `fsl-image-mfgtool-initramfs` is the initramfs image we need to

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

use when burning the system firmware. For the rest, please refer to Chapter 5.2 in the official NXP document "i.MX_Yocto_Project_User's_Guide.pdf":

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. Start Compiling

After selecting the target to be compiled, the compilation officially begins.

```
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)
```

If you interrupt the compilation and exit the terminal, you can execute the following command to re-establish the configuration:

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

After the file system is built, there will be a manifest file in the output directory, which contains the installed software packages in the corresponding file system. The first Yocto build will take a long time, depending on the number of CPU cores and hardware read and

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

write speed of the computer. Yocto recommends using eight cores and SSD hard drives to speed up the build. After the first build is completed, a cache will be generated, and the time for subsequent modified builds will be much shorter. After the build is completed, the final file will be generated in the "tmp/deploy/images/imx6ull14x14evk/" directory.

6.2.5. Speed up compilation

As mentioned above, after the first compilation is completed, a compilation cache will be generated, and its path is: `<build_dir>/sstate-cache`. We can also specify different storage locations, so that when compiling different DISTROs, we can share a part to provide acceleration.

To achieve this, just modify the `<build_dir>/conf/local.conf` file and add a line:

```
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 [+]
```

This will place the compilation cache in the sstate-cache directory under the root

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

directory of the Yocto project, which is the directory `~/work/yocto/sstate-cache` according to the configuration in this article.

6.3. Use ubuntu-base to create a system

ubuntu-base is the minimum Ubuntu file system officially built by Canonical. It includes the apt manager. The basic package size is usually only tens of megabytes. It is supported by the entire ubuntu software source. Ubuntu software is generally stable. Linux software can be installed on demand based on ubuntu-base, and it is deeply customizable. It is often used for embedded rootfs construction.

6.3.1. Get the source code

Official website address:http://cdimage.ubuntu.com/ubuntu-base/releases/

Get ubuntu-base-20.04.5-base-armhf.tar.gz and unzip it.

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

6.3.2. Preparing the chroot environment

Install qemu emulator

```
sudo apt-get install qemu-user-static
sudo cp /usr/bin/qemu-arm-static ~/work/ubuntu/usr/bin/
```

Add DNS configuration, you can directly use the network update package later, you can customize the DNS server and replace 8.8.8.8

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

Copy the following script to the ch-mount.sh file and change the permission to executable.

```
#!/bin/bash

function mnt_chroot() {
    echo "Mounting chroot environment..."
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    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."
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
        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. Installing the system

Mounting system

```
./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:/#
```

After the mount is successful, you can configure the Ubuntu file system and install some necessary software.

Basic package installation:

```
# apt update

# apt install sudo

# apt install language-pack-en-base
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
# 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
```

Add log, user debugging ubuntu system debugging

```
# touch /var/log/rsyslog
# chown syslog:adm /var/log/rsyslog
# chmod 666 /var/log/rsyslog
# systemctl unmask rsyslog
# systemctl enable rsyslog
```

Installing network and language pack support

```
# 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 Desktop System Installation

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

Set root password

```
passwd root
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Create a user named: ebyte

```
adduser ebyte

passwd ebyte
```

Add user ebyte to the sudo group

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

6.3.4. Uninstall System

After completing the above steps, you can exit the system. Simply enter exit in the system to exit the system and use the script to uninstall.

```
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. References

- Linux kernel open source community: https://www.kernel.org/
- NXP Development Community: https://community.nxp.com
- IMX6ULL date-sheet
- imx-yocto-lf5.10.9_1.0.0

8. Revision Notes

Revision Notes Table

Version	Modifications	Modification time	prepared by	Proofreading	Approval
V1.0	First Draft	24-06-21	WQY	LJQ	WFX
V1.1	Added buildroot section Added description of ECB20-6Y28C series	25-03-05	WQY	LJQ	WFX

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

9. About us

Technical support: service@cdebyte.com

Documents and RF Setting download link: www.cdebyte.com

Thank you for using Ebyte products! Please contact us with any questions or suggestions: info@cdebyte.com

---

Web: www.cdebyte.com

Address: B5 Mould Park, 199# Xiqu Ave, High-tech District, Sichuan, China

Chengdu Ebyte Electronic Technology Co.,Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000