软件开发指南

ECB10-PGD

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. 概述

本文主要介绍基于亿佰特核心板定制一个完整的嵌入式 Linux 系统的完整流程，其中包括开发环境的准备，代码的获取，以及如何进行 Bootloader, Kernel 的移植，定制适合自身应用需求的 Rootfs 等。我们首先介绍如何基于我们提供的源代码构建适用于 ECB10 开发板的系统镜像，如何将构建好的镜像烧录到开发板。针对那些基于 ECK10 核心板进行项目开发的用户，我们重点介绍了将这一套系统移植到用户的硬件平台上的方法和一些要点，并通过一些实际的移植案例和 Rootfs 定制的案例，使用户能够迅速定制适合自己硬件的系统镜像。

1.1. 软件资源

ECB10-TB13X 搭载基于 Linux 5.15 版本内核的操作系统，开发板出厂附带嵌入式 Linux 系统开发所需要的交叉编译工具链，TF-A 源代码，Optee-os 源码，U-boot 源代码，Linux 内核和各驱动模块的源代码，以及适用于 Windows 桌面环境和 Linux 桌面环境的各种开发调试工具，应用开发样例等。

2. 如何烧录系统镜像

使用 stm32cubeprogrammer 可以烧录到 ECB10 板上上的 emmc 存储器。或者可以制作 SD 卡启动器来从 SD 卡启动 ECB10。

2.1. 使用 STM32CubeProg 烧写

2.1.1. 烧写工具和材料

- CH340 串口线和串口软件
- USB Type_C 数据线
- 电源适配器 12V
- STM32CubeProgrammer 软件
- 软件镜像包

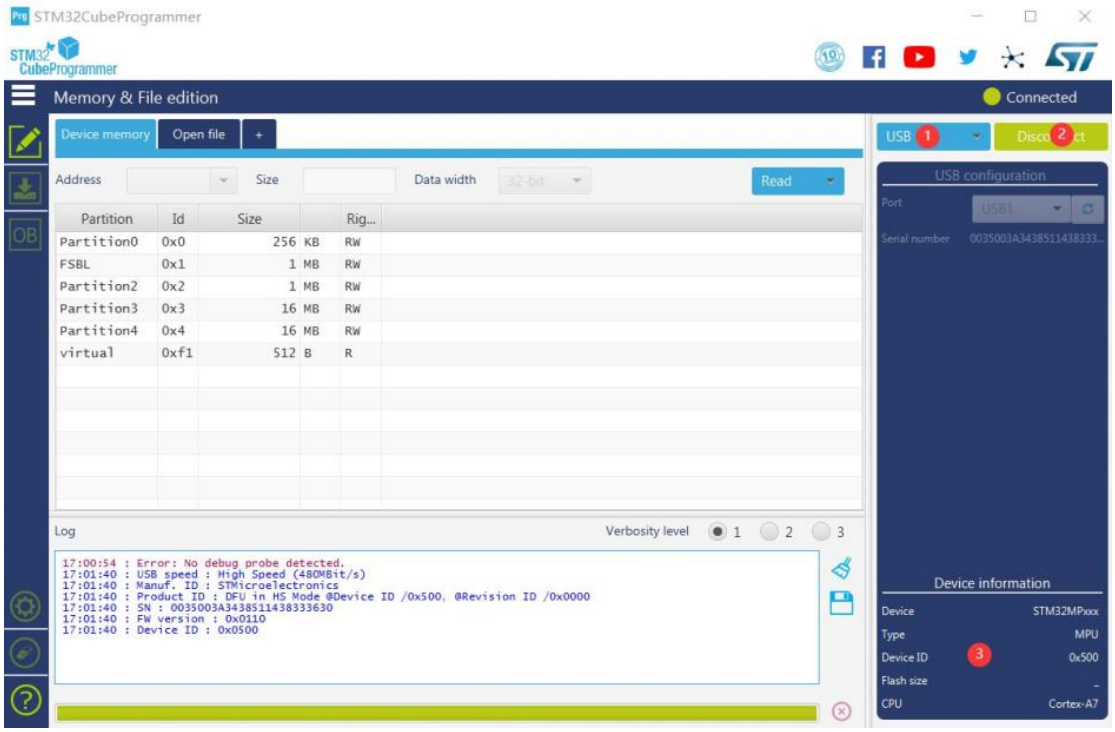
串口软件 mobaexterm 和 STM32CubeProgrammer 软件在 3.5 章节下讲解了如何安装。

2.1.2. 设置硬件和 programer

在有了上述软件之后，我们连接 CH340 串口线的 RX,TX,GND 到板上的 TX,RX,GND，再连接上 typec 接口，这里注意虚拟机不要占用了 usb 端口，最后我们接入电源线。

将拨码开关拨到 Download 模式（B2/B1/B0 : 0 0 0）。

打开 Windows 平台已经安装好的 STM32CubeProgrammer 软件，如果没有安装软件参考 3.5 小节。选择 USB 烧录方式。点击 connect 按钮连接。检查能否正常显示开发板的信息，如下图。 如果左下角的日志显示窗口提示了设备的一些信息，那么说明已经正常连接 ECB10 了。



2.1.3. 选择配置文件

ECB10-PGD 有四种配置的核心板，分别是 131-D2，131-D2E8，135-D2135-D5E8。

131-D2： cpu：131 内存：256M emmc：无（只能从 sd 卡启动）

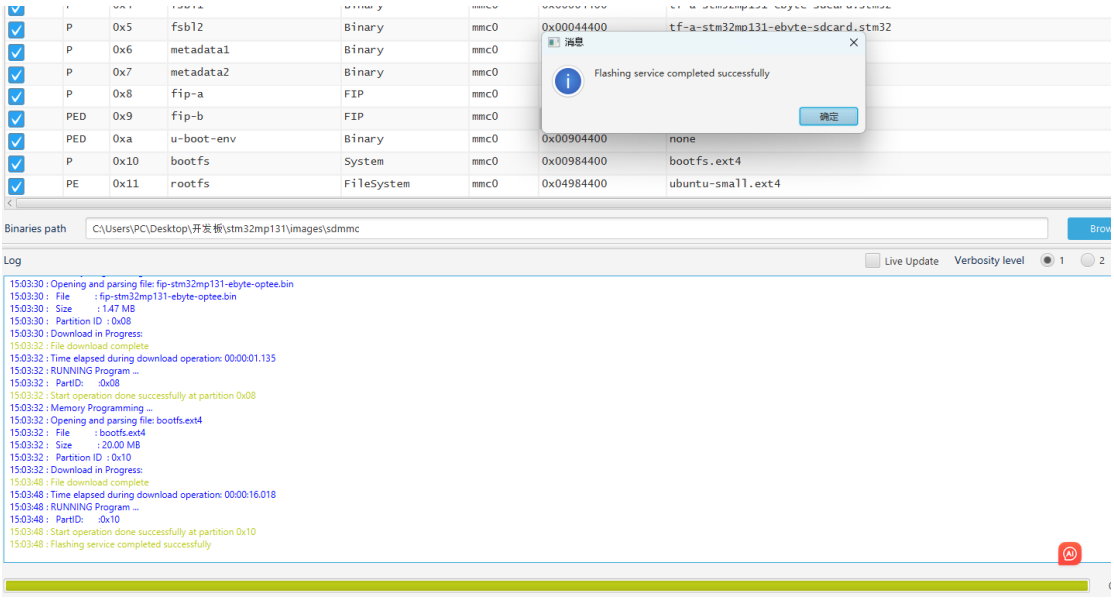
131-D2E8： cpu：131 内存：256M emmc：8G（从 sd 卡或 emmc 启动）

135-D2： cpu：135 内存：256M emmc：无（只能从 sd 卡启动）

135-D5E8： cpu：135 内存：512M emmc：8G（从 sd 卡或 emmc 启动）

我们以 135-D5E8 为例，CubeProgram 烧写工具只能下载到板上的 flash，所以请使用带 emmc 的 135-D5E8 来烧录。

选择打开 02_Images/FlashLayout_emmc_ebyte-stm32mp135-D5E8-optee.tsv ， 点击 download 进行烧录。



烧录完成之后。

将拨码开关拨到 emmc 模式（B2/B1/B0 : 0 1 0）。按下复位键就可以启动镜像了。可以使用串口软件来观察启动流程。

2.2.制作 SD 卡启动器

2.2.1. 准备材料

SD 卡

镜像制作工具（04_Tools/ Win32DiskImager-1.0.0-binary/Win32DiskImager.exe）

2.2.2. 制作 SD 卡启动

格式化 SD 卡

打开 Win32DiskImager-1.0.0-binary

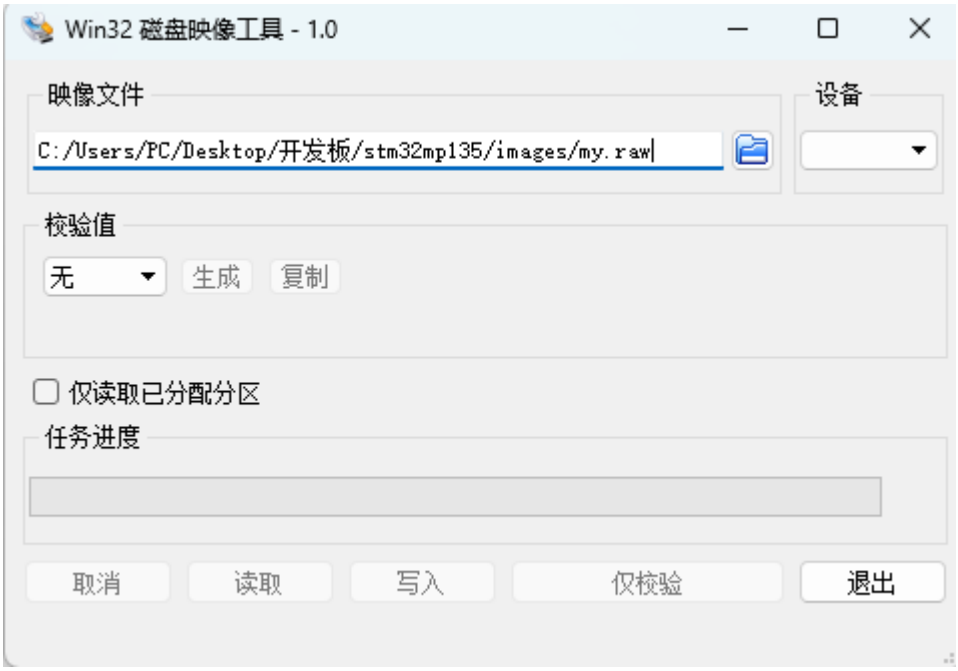
选择镜像文件 FlashLayout_sdcard_ebyte-stm32mp135-D5E8-optee.raw。

这里仅提供了 135-D5E8 型号的 SD 卡烧录镜像，如果需要其他型号的镜像。拷贝 Sourecs 下面的 stm32mp13x-PGD.tar.gz 到虚拟机，解压之后进入 FIP_artifacts 文件夹，使用制作镜像的脚本和对应的 tsv 文件制作即可。制作 stm32mp131-D2 的 SD 卡镜像示例如下。

```
# cd FIP_artifacts/script/
# ./create_sdcard_from_flashlayout_13x.sh ../FlashLayout_sdcard_ebyte-stm32mp131-D2-optee.tsv
```

就会得到 FlashLayout_sdcard_ebyte-stm32mp131-D2-optee.raw。然后按照后续的方法烧

录即可。



文件类型要选择*.*才能找到烧录文件

点击写入进行烧录，等待写入完成，大约 1 分钟完成，此速度取决于 SD 的读写速度。

烧录完成之后。插入 SD 卡。

将拨码开关拨到 SD Card 模式（B2/B1/B0：1 0 1）。按下复位键就可以启动镜像了。可以使用串口软件来观察启动流程。

2.3.修改启动配置

因为不同核心板支持的硬件功能有差异，而且内存容量也不同，所以需要按照核心板的型号来选择加载的设备树文件。如果需要修改默认配置，需要改动 bootfs.ext4 里面的配置文件。可以在烧录之前修改 bootfs.ext4（推荐，更安全）或者启动开发板之后进行修改。

```
Hit any key to stop autoboot:  0
Boot over mmc0!
switch to partitions #0, OK
mmc0 is current device
Scanning mmc 0:8...
Found /mmc0_extlinux/extlinux.conf
Retrieving file: /mmc0_extlinux/extlinux.conf
983 bytes read in 11 ms (86.9 KiB/s)
Retrieving file: /splash_landscape.bmp
7802 bytes read in 12 ms (634.8 KiB/s)
Select the boot mode
1:      ebyte-stm32mp135-D5E8
2:      ebyte-stm32mp135-D2
3:      ebyte-stm32mp131-D2E8
4:      ebyte-stm32mp131-D2
Enter choice: 1:        ebyte-stm32mp135-D5E8
```

mmc0 对应 SD 卡，mmc1 对应 emmc。

如果在启动开发板之后进行修改，我们需要直接访问挂载好的/media/mmcblk0p8/。然后进行修改。

```
#cd /media/mmcblk0p8/
#ls
boot.scr.uimg                  ebyte-stm32mp135-D5E8.dtb   splash_landscape.bmp
ebyte-stm32mp131-D2.dtb        lost+found                  splash_portrait.bmp
ebyte-stm32mp131-D2E8.dtb      mmc0_extlinux               st-image-resize-initrd
ebyte-stm32mp135-D2.dtb        mmc1_extlinux               uImage
```

这里的 mmc0_extlinux 和 mmc1_extlinux 分别代表 SD 卡和 EMMC。我们是从 SD 卡启动的，所以这里修改 mmc0_extlinux 下的配置文件 extlinux.conf。

```
# vi mmc0_extlinux/extlinux.conf
```

将其中的 DEFALT 改为下面的任意一个 LABEL 即可。

如果需要在烧录之前就提前修改好，则按照上述的方法挂载提供的 Images 里面的 bootfs.ext4，然后修改对应位置。最后使用 cubeprogrammer 烧录到 emmc，或者制作 raw 文件烧录到 sd 卡即可。

3. 开发环境准备

本节将介绍如何搭建适用于 STM32MP1XX 系列处理器平台的开发环境，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。

主机硬件：i7-12700 和 VMware 虚拟机，可供参考。

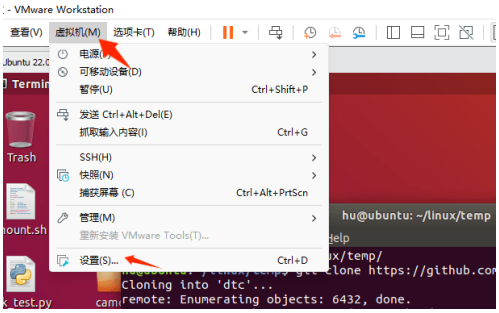
虚拟机操作系统：ubuntu18.04 64bit，在亿佰特提供的镜像中，路径为： 04_Tools→ubuntu-18.04.5-desktop-amd64.iso。

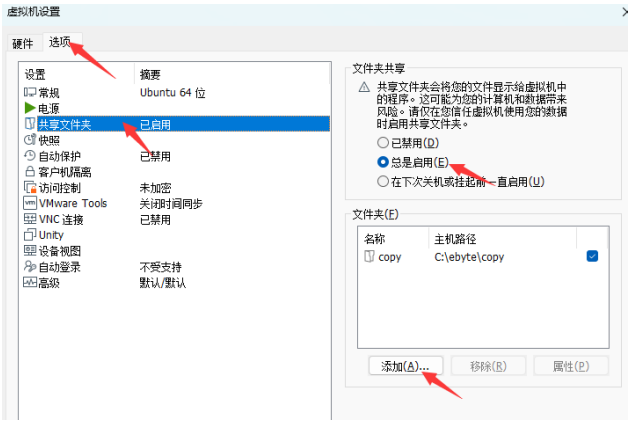
用户可以根据一些网络教程使用 vmware 和此镜像来搭建 ubuntu 虚拟机，本文不做相关内容介绍。安装好虚拟机之后，就可以使用下面的配置内容来对 ubuntu 进行配置。

后面讲解安装一些传输文件的工具，如果对下面的工具没有需要的话，可以直接跳到 2.6 章节配置 ubuntu 来编译 SDK。如果需要安装一些文件传输工具则参考下面的章节。Ubuntu 虚拟机和 windows 传输文件可以使用 vmware 共享目录的方式，参考 2.2 小节。Ubuntu 虚拟机和开发板传输文件可以使用 nfs 共享目录，参考 2.3 小节。使用串口工具远程登录 ubuntu 需要开启 ssh 服务，参考 2.4 小节。后面的内容仅对 ubuntu-1804 进行验证。

3.1.Vmware 共享目录

通过设置 vmware 的共享目录，可以实现虚拟机和 windows 访问同一文件夹，从而达到方便传输文件的目的。





然后在这里添加一个 windows 的路径即可。这里我选择的是 C:ebyte/copy，名称默认就是 copy。接下来我们在 ubuntu 里验证能否访问到这个文件夹。

```
# cd /mnt/hgfs/copy/
# ls
```

这样就可以在 windows 和 ubuntu 虚拟机方便的传输文件了。在这个文件夹里最好不要做编译的操作，拷到虚拟机的其他位置再编译。如果虚拟机重启之后找不到这个文件夹了，那么需要手动重新挂载一下。

```
# sudo mount -t fuse.vmhgfs-fuse .host:/ /mnt/hgfs -o allow_other -o nonempty
```

3.2.NFS 服务

如果需要用从网络启动开发板，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-get install nfs-kernel-server rpcbind
```

等待安装完成，安装完成以后在用户根目录下创建一个名为“linux”的文件夹，在“linux”文件夹里面新建一个名为“nfs”的文件夹。

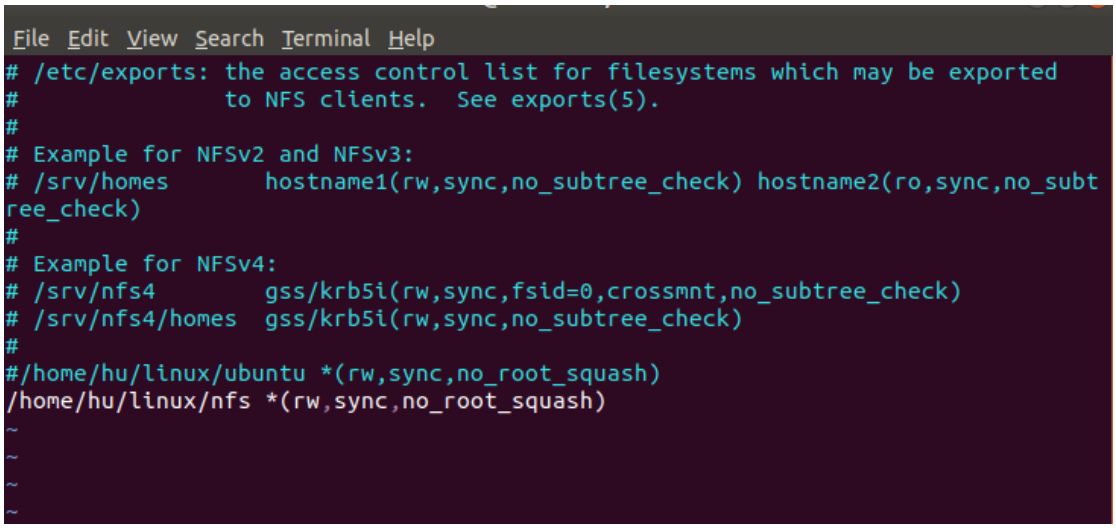
```
# sudo mkdir -p /home/hu/linux/nfs
```

创建的 nfs 文件夹供 nfs 服务器使用，以后我们可以在开发板上通过网络文件系统来访问 nfs 文件夹，要先配置 nfs，使用如下命令打开 nfs 配置文件/etc/exports：

```
# sudo vi /etc/exports
```

打开/etc/exports，在最后面添加如下所示内容：

```
/home/hu/linux/nfs *(rw,sync,no_root_squash)
```



重启 NFS 服务，使用命令如下：

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

3.3.SSH 服务

开启 Ubuntu 的 SSH 服务以后我们就可以在 Windwos 下使用终端软件登陆到 Ubuntu，比如使用 MobaXterm， Ubuntu 下使用如下命令安装 SSH 服务：

```
# sudo apt-get install openssh-server
```

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。用户即可使用 SSH 远程登录虚拟机了。需要保证主机可以 ping 通开发板。

3.4.TFTP 服务

开启 ubuntu 的 TFTP 服务，我们可以从网络来启动开发板，本质上就是开发板通过网络来访问虚拟机上的文件路径，获取到内核文件之后，就可以启动。

```
# sudo apt-get install vsftpd
```

等待软件自动安装，安装完成以后使用如下 VI 命令打开/etc/vsftpd.conf，命令如下：

```
# sudo vi /etc/vsftpd.conf
```

打开 vstfpd.conf 文件之后找到如下两行：

```
local_enable=YES
write_enable=YES
```

确保上面两行前面没有“#”，有的话就取消掉，完成以后如图所示：

```
27 # Uncomment this to allow local users to log in.
28 local_enable=YES
29 #
30 # Uncomment this to enable any form of FTP write command.
31 write_enable=YES
```

修改完 vsftpd.conf 以后保存退出，使用如下命令重启 FTP 服务：

```
# sudo /etc/init.d/vsftpd restart
```

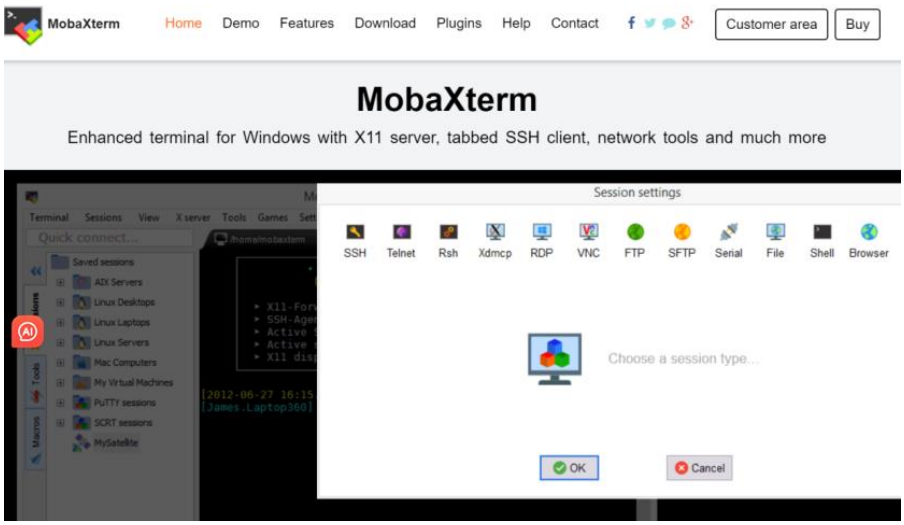
3.5.windows 安装软件开发工具

3.5.1. MobaXterm

此软件用作开发板的调试串口的通讯，用户可以使用调试串口和开发板交互 linux 命令。

3.5.1.1. 软件下载安装

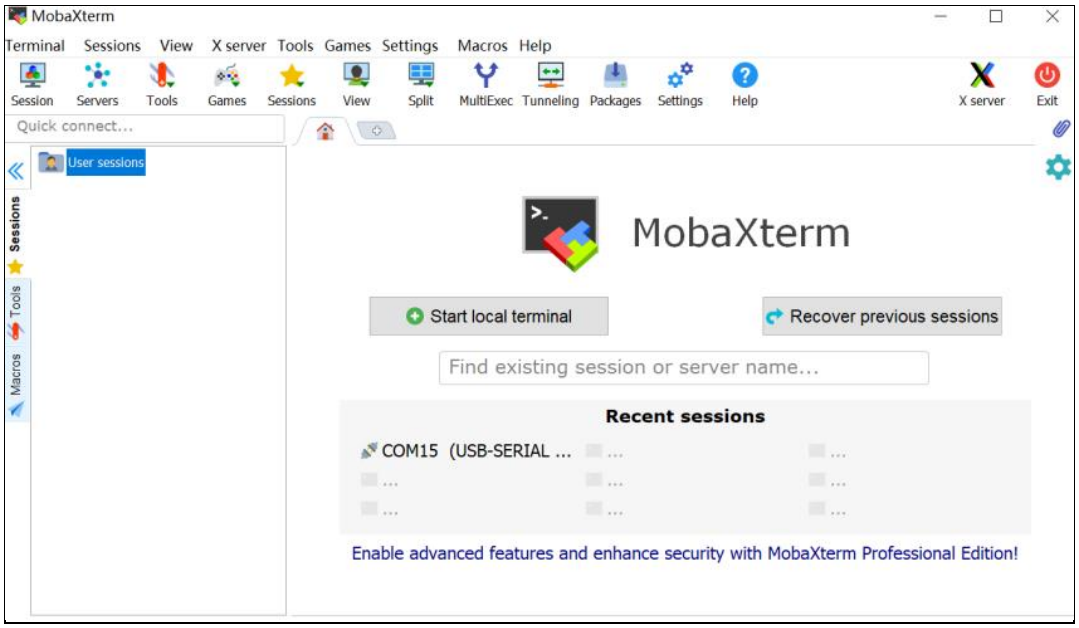
MobaXterm 是一款终端软件，功能强大而且免费，推荐使用此软件作为终端调试软件，MobaXterm 软件在其官网下载即可，地址为 https://mobaxterm.mobatek.net/，如图所示：



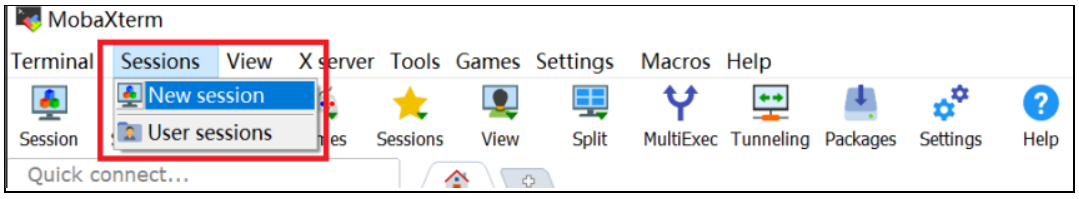
安装包已经放到了开发板中，路径为： 04_Tools ->MobaXterm_Portable.zip。打开此压缩包即可。

3.5.1.2. 软件使用

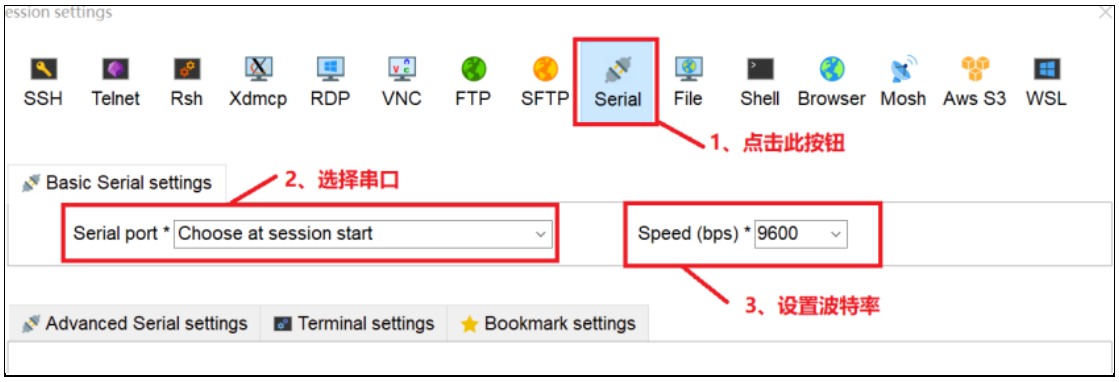
双击 MobXterm 图标，打开此软件，软件界面如图所示：



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：



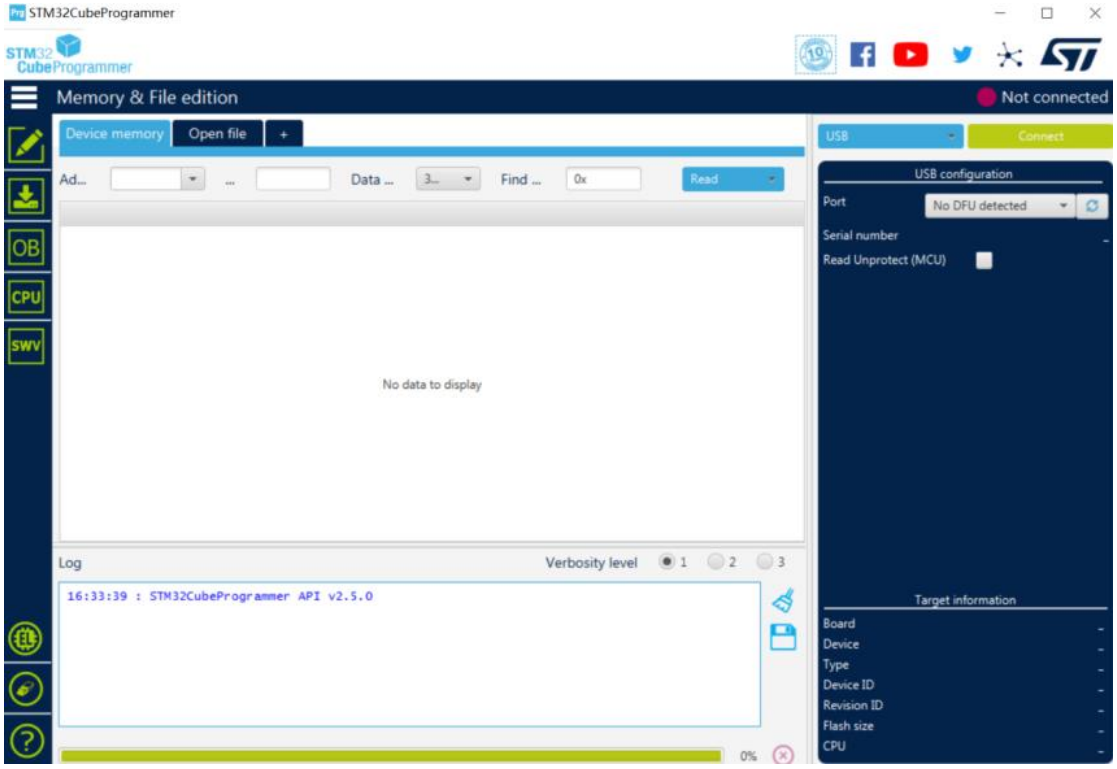
打开串口设置窗口以后先选择要设置的串口号，因此要先用串口线将开发板连接到电脑上上，然后设置波特率为 115200(根据自己实际需要设置)。

3.5.2. STM32CubeProg

ST 推出的高集成度的编程工具 STM32CubeProgrammer，它可以在烧录的过程中对未分区的的存储设备进行分区，一旦分区就可以使用已经编译好的二进制文件对某个分区单独进行更新。用户可以根据需要选择使用合适的版本，下载地址如下：https://www.st.com/zh/development-tools/stm32cubeprog.html 。路径为 04_Tools →

en.stm32cubeprog_v2-5-0.zip。此软件主要用来使用 USB 方式烧录我们的开发板。

解压开发板中的 STM32CubeProgrammer 安装压缩包，然后双击解压出来的“SetupSTM32CubeProgrammer-2.5.0.exe”，安装过程很简单，根据提示进行安装即可，双击图标打开 STM32CubeProgrammer，如图所示



3.6.Ubuntu 配置开发环境

3.6.1. 安装交叉编译工具链

用户可以直接使用这个交叉编译工具链来单独编译 Bootloader，Kernel 或者编译自己的应用程序，具体过程在后面的章节中将会详细介绍。这里先介绍交叉编译工具链的安装步骤。

提供的交叉编译器路径为 04_Tools→SDK-x86_64-stm32mp1-openstlinux.tar.gz。

将交叉编译器放到虚拟机中并解压

tar -xvzf SDK-x86_64-stm32mp1-openstlinux.tar.gz

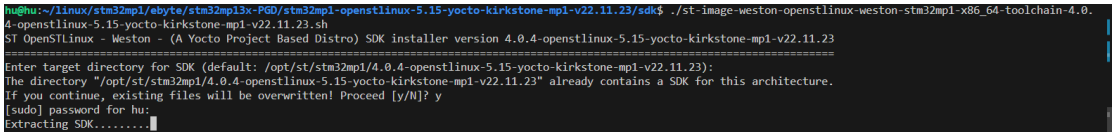
等待解压完成，解压完成以后会生成一个 stm32mp1-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23 文件夹

进入目录安装工具链

cd stm32mp1-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/sdk/

./st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23.sh

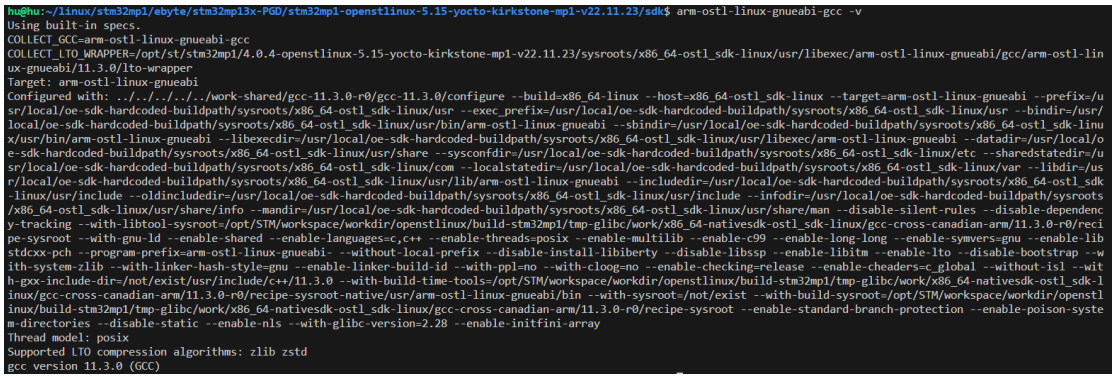
按照提示按下 enter 并输入 y，最后输入 root 密码就可以安装了



修改好以后就保存退出，重启 Ubuntu 系统，交叉编译工具链(编译器)就安装成功了。

接下来我们验证下是否安装成功。出现下面的版本信息说明安装成功。

```
#source /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# arm-ostl-linux-gnueabi-gcc -v
```



我们在编译时需要依赖许多库，这里集中进行安装。

```
# sudo apt-get update
# sudo apt-get install lsb-core lib32stdc++6 u-boot-tools libyaml-dev bison flex sed wget curl git-core coreutils unzip texi2html texinfo docbook-utils gawk cvs subversion python-pysqlite2 diffstat help2man make gcc build-essential g++ chrpath libxml2-utils xmlto docbook bsdmainutils iputils-ping cpio python-wand python-pycryptopp python-crypto
# sudo apt-get install libsdl1.2-dev xterm corkscrew nfs-common nfs-kernel-server mercurial u-boot-tools libarchive-zip-perl
# sudo apt-get install ncurses-dev bc linux-headers-generic gcc-multilib libncurses5-dev libncursesw5-dev lrzsz dos2unix lib32ncurses5 repo libssl-dev
```

ubuntu-1804 默认的 dtc 版本过低，我们手动安装 device-tree-compiler。从 github 上获取

最新的源码，或者使用我们已经下载的源码。

04_Tools→dtc.tar.gz。

拷贝到虚拟机之后，解压，编译，添加到系统目录即可。

```
# sudo apt-get install pkg-config
# tar -xvzf dtc.tar.gz
# cd dtc
# make
# sudo cp dtc /usr/bin
```

最后我们验证下，能看到正确的 dtc 版本即可。

4. 如何构建镜像

前面的章节已经比较完整的讲述了将镜像烧录到开发板上的完整流程。由于 ECB10 核心板的很多管脚都具有多种功能复用的特性，所以实际项目中基于 ECK10 核心板设计的底板和 ECB10 总会有一些差异。这些差异可能是去掉显示，增加更多 GPIO，或者需要增加更多串口，还有可能通过 SPI，I2C，USB 等扩展一些外设等等，本章从一个系统开发人员的角度来讲述开发和定制自己系统的具体过程。用户如果没有特殊需求的话，只需要对 kernel 部分进行修改，其他的镜像可使用亿佰特编译好的镜像。

4.1.TF-A

在使用 Trusted Boot Chain（关于 Boot Chain 的详细信息，请查看 https://wiki.st.com/stm32mpu/wiki/Boot_chain_overview）方式启动时，TF-A 用作 FSBL（First Stage Boot Loader），主要实现 DDR, Clock 等核心资源的初始化，基于 ECB10-TB13X 核心板设计的硬件一般不需要修改 TFA，如果特殊情况下，需要修改某个时钟，或者需要对某个管脚进行早期的控制时，可以参照下面的方法进行移植。

4.1.1. 获取源码

获取源码后进行解压，源码路径为 02_Source/tf-a.tar.gz，解压后如图所示。

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/tf-a$ ls
build  deploy  ebyte-st-tfa  Makefile.sdk  README.HOW_TO.txt
```

4.1.2. Fiptool 安装

本节介绍如何使用 fiptool 工具来将 OP-TEE 和 U-Boot 等固件文件打包在一起，封装成单个镜像文件 fip-stm32mp135d-atk-optee.bin， 后续烧写到开发板。

FIP，英文全称 Firmware Image Package，直译为固件镜像包。 FIP 是 TF-A 使用的一种打包格式，用于将引导加载程序镜像文件（bootloader） 以及其他有效载荷文件打包为单个二进制文件。 现在 ST 发布的生态系统开发包中推荐使用 FIP 来启动 STM32MP1 平台。只有 TF-A（带有 STM32 头部结构信息)镜像被 ROM 代码加载，而其他二进制文件比如 OP-TEE、 U-Boot 及其各自的设备树文件则封装为 FIP 镜像包形式。 TF-A 运行时能够加载并鉴权 FIP 镜像。

故 ECB10-TB13X 核心板的出厂系统启动、运行过程，可通俗地概述为：

ROM 代码->TF-A->FIP 镜像（含 OP-TEE 和 U-Boot） ->Linux 内核->文件系统。

在 TF-A 源码中，提供了一个实现 FIP 功能的工具 fiptool。下面我们进入 TF-A 的 fiptool 源码目录进行编译， 来生成 fiptool 工具，编译指令如下：

```
# cd ebyte-st-tfa/tools/fiptool / //进入到自己的 TF-A 源码目录
# cd tools/fiptool //进入 fiptool 源码
# make //编译 fiptool 源码
```

将上面编译生成的 fiptool 工具复制到 ubuntu 的/usr/bin/目录下

```
# sudo cp fiptool /usr/bin/
```

```
hu@ubuntu:/usr/bin$ ls fiptool
fiptool
```

执行如下指令，查看 fiptool 工具帮助信息；

```
# fiptool help
```

```
hu@ubuntu:/usr/bin$ fiptool help
usage: fiptool [--verbose] <command> [<args>]
Global options supported:
  --verbose      Enable verbose output for all commands.

Commands supported:
  info           List images contained in FIP.
  create         Create a new FIP with the given images.
  update         Update an existing FIP with the given images.
  unpack         Unpack images from FIP.
  remove         Remove images from FIP.
  version        Show fiptool version.
  help           Show help for given command.
```

4.1.3. stm32wrapper4dbg 工具安装

编译 TF-A 或者 Uboot 的时候需要用到 stm32wrapper4dbg 这个工具，否则编译会报错。 ST 提供了这个工具的源码，我们需要在 Ubuntu 下编译并安装这个源码，源码的下载地址为： https://github.com/STMicroelectronics/stm32wrapper4dbg，这个我们已经下载下来并放到了开发板中，路径为：04_Tools→stm32wrapper4dbg-master.zip。 将源码压缩包拷贝到 Ubuntu 下，然后进行解压，命令如下：

```
# unzip stm32wrapper4dbg-master.zip
# cd stm32wrapper4dbg-master
# make
```

编译完成以后就会得到一个名为“stm32wrapper4dbg”的工具，拷贝到 Ubuntu 的/usr/bin 目录下

```
# sudo cp stm32wrapper4dbg /usr/bin
```

在终端验证安装是否成功

```
# stm32wrapper4dbg -s
```

```
hu@ubuntu:~/linux/temp/dtc$ stm32wrapper4dbg -s
stm32wrapper4dbg: option requires an argument -- 's'
Usage: stm32wrapper4dbg -s srcfile -d destfile [-b] [-f]
  Add a debug wrapper to a stm32 image.
  If "-b" is not specified, the wrapper would be placed
  after the last byte of the image.

Options:
  -s srcfile   input image in stm32 file format
  -d destfile  output image in stm32 file format
  -b           place the wrapper before the image
  -f           force re-adding the wrapper
```

4.1.4. 编译

Makefile.sdk 文件主要定义了一些编译属性，比如要使用的交叉编译器、编译的一些选项等等， Makefile.sdk 最终会调用 TF-A 内部的 Makefile 来编译 TF-A 源码。

```
23
24  # Init default config settings
25  TF_A_DEVICETREE_optee ?= ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
26  TF_A_EXTRA_OPTFLAGS_optee ?= AARCH32_SP=optee
27  TF_A_BINARY_optee ?= tf-a
28  TF_A_MAKE_TARGET_optee ?= dtbs
29
30  # Init default config settings
31  TF_A_DEVICETREE_emmc ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
32  TF_A_EXTRA_OPTFLAGS_emmc ?= STM32MP_EMMC=1 PSA_FWU_SUPPORT=1
33  TF_A_BINARY_emmc ?= tf-a
34  TF_A_MAKE_TARGET_emmc ?= all
35
36  # Init default config settings
37  TF_A_DEVICETREE_sdcard ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
38  TF_A_EXTRA_OPTFLAGS_sdcard ?= STM32MP_SDMMC=1 PSA_FWU_SUPPORT=1
39  TF_A_BINARY_sdcard ?= tf-a
40  TF_A_MAKE_TARGET_sdcard ?= all
41
42  # Init default config settings
43  TF_A_DEVICETREE_uart ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
44  TF_A_EXTRA_OPTFLAGS_uart ?= STM32MP_UART_PROGRAMMER=1
45  TF_A_BINARY_uart ?= tf-a
46  TF_A_MAKE_TARGET_uart ?= all
47
48  # Init default config settings
49  TF_A_DEVICETREE_usb ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
50  TF_A_EXTRA_OPTFLAGS_usb ?= STM32MP_USB_PROGRAMMER=1
```

打开源码根目录后，执行编译脚本，如果没有可执行权限，需要先添加权限。然后再进行编译。编译脚本如下图所示。

```
# tar -xvzf tf-a.tar.gz
# cd tf-a/ebyte-st-tfa/
# chmod +x ./tools/stm32image/stm32image
# chmod +x ./tools/fwu_gen_metadata/fwumd_tool.py
#  source  /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

```
#!/bin/bash
rm ../build ../deploy -rf      You, 上个月 • init
make -f ../Makefile.sdk all -j 8
cp ../deploy/tf-a-ebyte-stm32mp13* ../../FIP_artifacts/arm-trusted-firmware/
cp ../deploy/metadata.bin ../../FIP_artifacts/arm-trusted-firmware/
cp ../deploy/fwconfig/ebyte-stm32mp13* ../../FIP_artifacts/arm-trusted-firmware/fwconfig/
cp ../deploy/debug/* ../../FIP_artifacts/arm-trusted-firmware/debug/
```

编译完成后会在上一层目录，也就是 tf-a 目录下生成两个目录“build” 和“deploy”，如下图所示。会在上层目录的 deploy 目录下生成编译文件。

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/tf-a/deploy$ ls -l
total 708
drwxrwxr-x 2 hu hu  4096  9月  8 13:29 debug
drwxrwxr-x 2 hu hu  4096  9月  8 13:29 fwconfig
-rw-rw-r-- 1 hu hu    96  9月  8 13:29 metadata.bin
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-emmc.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-sdcard.stm32
-rw-rw-r-- 1 hu hu 84723  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-uart.stm32
-rw-rw-r-- 1 hu hu 88819  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-usb.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-emmc.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-sdcard.stm32
-rw-rw-r-- 1 hu hu 84723  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-uart.stm32
-rw-rw-r-- 1 hu hu 88819  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-usb.stm32
```

4.2.Optee-os

OP-TEE 是实现 Arm TrustZone 技术的开源可信执行环境 (TEE)，与安全领域相关 。

源码目录在 02_Source/optee.tar.gz，解压之后如下图所示。

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/optee$ ls
build  deploy  ebyte-st-optee  Makefile.sdk  README.HOW_TO.txt
```

由上图可见， 和 TF-A 源码类似， 解压后出现一个 Makefile.sdk 文件和 OP-TEE 源码目录。其中 Makefile.sdk 文件主要定义了一些编译属性，比如要使用的交叉编译器、编译的一些选项等等， Makefile.sdk 最终会调用 OP-TEE 内部的 Makefile 来编译 OP-TEE 源码。

```
# Set default path
SRC_PATH ?= $(PWD)
BLD_PATH ?= $(SRC_PATH)/../build
DEPLOYDIR ?= $(SRC_PATH)/../deploy

# Set default optee-os config
CFG_EMBED_DTB_SOURCE_FILE ?=   ebyte-stm32mp135-D5E8
#CFG_EMBED_DTB_SOURCE_FILE ?=   ebyte-stm32mp135-D2

# Remove default variables
LDFLAGS =
CFLAGS =
CPPFLAGS =
# Define default make options
```

编译出厂源码时，用户不必修改任何内容。 适配于开发板硬件资源的 OPTEE 源码设备树路径为 core/arch/arm/dts/ ， 设备树文件为 ebyte-stm32mp135-D5E8 和 ebyte-stm32mp135-D2。

用户根据自己手上实际的核心板型号来配置 CFG_EMBED_DTB_SOURCE_FILE。

获取源码并解压之后执行编译脚本会生成镜像文件

```
# cd ebyte-st-optee
# source /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

和 TF-A 编译结果类似， OP-TEE 源码编译完成后会在上一层目录，也就是 optee 目录下生成两个目录“build” 和“deploy”。

build 目录包含编译过程生成的所有文件，比如.a、 .o 等格式文件，这些文件大部分对用户来说也是不需要接触到的。

而 deploy 目录下的文件则很重要，它是由 Makefile.sdk 脚本对 build 目录里面生成的二进制文件进行二次处理， 复制到 deploy 目录下。它包含 3 个文件 tee-header_v2-ebyte-stm32mp135-D5E8.bin， tee-pageable_v2-ebyte-stm32mp135-D5E8.bin 和 tee-pager_v2-ebyte-stm32mp135-D5E8.bin。这 3 个文件，就是编译出厂 OP-TEE 源码的最终产物。

如果选择了 ebyte-stm32mp135-D2 配置则生成 tee-header_v2-ebyte-stm32mp135-D2.bin，tee-pageable_v2-ebyte-stm32mp135-D2.bin 和 tee-pager_v2-ebyte-stm32mp135-D2.bin。

出厂 OP-TEE 源码已编译完成，下面介绍 U-Boot 源码编译。

4.3. 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/。

4.3.1. 获取源码并编译

在 ubuntu 的目录下创建一个名为“uboot” 的目录，然后把亿佰特出厂 U-Boot 源码包 02_Source/uboot.tar.gz 拷贝到 uboot 这个目录下。然后进行解压，解压之后如图。

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/u-boot$ ls
0001-release-v1.7.5.1_U-boot_2022_maxio_driver.patch  build  deploy  ebyte-st-uboot  Makefile.sdk  README.HOW_TO.txt
```

由上图可见，和前面源码类似， 解压后出现一个 Makefile.sdk 文件和 U-Boot 源码 ebyte-st-uboot 目录。

这里 Makefile.sdk 文件主要定义了一些编译属性，比如 U-Boot 默认源码配置、 指定编译设备树名称、编译的一些选项等等，如下图所示。Makefile.sdk 最终会调用 U-Boot 内部的 Makefile 来编译 UBoot 源码。

```
# Set default path
SRC_PATH ?= $(PWD)
BLD_PATH ?= $(SRC_PATH)/../build
DEPLOYDIR ?= $(SRC_PATH)/../deploy

# Default U-Boot overall settings to null
UBOOT_CONFIG ?=
UBOOT_DEFCONFIG ?=
UBOOT_BINARY ?=
UBOOT_DEVICETREE ?=

# Init default config settings
#UBOOT_CONFIGS += trusted
#UBOOT_DEFCONFIG_trusted += stm32mp13_defconfig
#UBOOT_BINARY_stm32mp13_defconfig ?= u-boot.dtb
#UBOOT_DEVICETREE_stm32mp13_defconfig ?=
# Init default config settings
UBOOT_CONFIGS += trusted
#UBOOT_DEFCONFIG_trusted += ebyte_stm32mp135_D2_defconfig
#UBOOT_BINARY_ebyte_stm32mp135_D2_defconfig ?= u-boot.dtb
#UBOOT_DEVICETREE_ebyte_stm32mp135_D2_defconfig ?= ebyte-stm32mp135-D2

UBOOT_DEFCONFIG_trusted += ebyte_stm32mp135_D5E8_defconfig
UBOOT_BINARY_ebyte_stm32mp135_D5E8_defconfig ?= u-boot.dtb
UBOOT_DEVICETREE_ebyte_stm32mp135_D5E8_defconfig ?= ebyte-stm32mp135-D5E8
```

编译出厂源码时，用户不必修改任何内容。

135-D2 则选择 ebyte_stm32mp135_D2_defconfig 对应的三行配置

135-D5E8 则选择 ebyte_stm32mp135_D5E8_defconfig 对应的三行配置

```
# cd ebyte-st-uboot
# source /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

和前面源码类似， U-Boot 源码编译完成后会在上一层目录，也就是 uboot 目录下生

成两个目录“build” 和“deploy”。

build 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。而 deploy 目录下的文件则很重要，它是由 Makefile.sdk 脚本对 build 目录里面的二进制文件进行二次处理，复制到 deploy 目录下。它包含 2 个文件 u-boot-ebyte-stm32mp135-D5E8.dtb 和 u-boot-nodtb-stm32mp13.bin。这 2 个文件，就是编译出厂 U-Boot 源码的最终产物，也就是 ECB10-135D5E8 核心板的 U-Boot 固件。如果是选择了 D2 的配置，则生成 u-boot-ebyte-stm32mp135-D2.dtb 和 u-boot-nodtb-stm32mp13.bin。

4.3.2. 设置 Uboot 启动内核方式

Uboot 中有许多的环境变量，这里主要将影响 Kernel 启动方式的一些环境变量。用户如何没有特殊需求一般不用修改本章节的命令，可以跳过。

4.3.2.1. bootcmd

bootcmd 保存着 uboot 默认命令， uboot 倒计时结束以后就会执行 bootcmd 中的命令。这些命令一般都是用来启动 Linux 内核的，比如读取 EMMC 或者 NAND Flash 中的 Linux 内核镜像文件和设备树文件到 DRAM 中，然后启动 Linux 内核。可以在 uboot 启动以后进入命令行设置 bootcmd 环境变量的值。

bootcmd 的默认值就是 CONFIG_BOOTCOMMAND，bootargs 的默认值就是 CONFIG_BOOTARGS。我们可以直接在 stm32mp1.h 文件中通过设置宏 CONFIG_BOOTCOMMAND 来设置 bootcmd 的默认值。

4.3.2.2. bootargs

bootargs 保存着 uboot 传递给 Linux 内核的参数，比如指定 Linux 内核所使用的 console、指定根文件系统所在的分区等，如下面 bootargs 环境变量值：

console=ttySTM0,115200 root=/dev/mmcblk2p3 rootwait rw

console 用来设置 linux 终端(或者叫控制台)，也就是通过什么设备来和 Linux 进行交互，是串口还是 LCD 屏幕？如果是串口的话应该是串口几等等。一般设置串口作为 Linux 终端，这样我们就可以在电脑上通过 MobaXterm 来和 linux 交互了。这里设置 console 为 ttySTM0，因为 linux 启动以后 STM32MP1 的串口 4 在 linux 下的设备文件就是 /dev/ttySTM0，在 Linux 下，一切皆文件。ttySTM0 后面有个“,115200”，这是设置串口的波特率， console=ttySTM0,115200 综合起来就是设置 ttySTM0（也就是串口 4）作为 Linux 的终端，并且串口波特率设置为 115200。使用调试串口终端，开发板启动到 uboot

启动之后通过在 uboot 中设置 bootargs 和 bootcmd 来控制启动方式。设置完成后需要使用命令保存环境变量。后续就可以自动从设置的环境变量来启动内核和文件系统了。

root 用来设置根文件系统的位置， root=/dev/mmcblk2p3 用于指明根文件系统存放在 mmcblk2 设备的分区 3 中。STM32MP1 核心板启动 linux 以后会存在/dev/mmcblk1、/dev/mmcblk2 、/dev/mmcblk1p1 、/dev/mmcblk1p2 、/dev/mmcblk2p1 、/dev/mmcblk2p2 和/dev/mmcblk2p3 这样的文件， 其中 /dev/mmcblkx(x=0~n) 表示 mmc 设备，而 /dev/mmcblkxpy(x=0~n,y=1~n)表示 mmc 设备 x 的分区 y。在 STM32MP1 开发板中 /dev/mmcblk2 表示 EMMC，而/dev/mmcblk2p3 表示 EMMC 的分区 3。

root 后面有“rootwait rw”， rootwait 表示等待 mmc 设备初始化完成以后再挂载，否则的话 mmc 设备还没初始化完成就挂载根文件系统会出错的。 rw 表示根文件系统是可以读写的，不加 rw 的话可能无法在根文件系统中进行写操作，只能进行读操作。

rootfstype 此选项一般配合 root 一起使用， rootfstype 用于指定根文件系统类型，如果根文件系统为 ext 格式的话此选项无所谓。如果根文件系统是 yaffs、 jffs 或 ubifs 的话就需要设置此选项，指定根文件系统的类型。

在设置完 bootcmd 和 bootargs 环境变量之后都需要保存。

saveenv

```
STM32MP> saveenv
Saving Environment to MMC... Writing to MMC(0)... OK
```

下面分别介绍三种不同的启动方式。

4.3.2.3. 从 SD 卡启动

```
# setenv bootargs 'console=ttySTM0,115200 root=/dev/mmcblk0p9 rootwait rw'
# setenv bootcmd 'ext4load mmc 0:8 c2000000 uImage;ext4load mmc 0:8 c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000'
```

4.3.2.4. 从 nandflash 启动

```
# setenv bootcmd 'ubifsmount ubi:boot;ubifsload c2000000 uImage;ubifsload c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000;ubifsmount ubi:rootfs'
# setenv bootargs 'ubi.mtd=UBI rootfstype=ubifs root=ubi:rootfs rootwait rw console=ttySTM0,115200'
```

4.3.2.5. 从网络启动

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。需要配置 IP 地址，MAC 地址，网关，掩码和虚拟机的 IP 地址。

```
# setenv ipaddr 192.168.0.250
# setenv ethaddr 00:04:9f:04:d2:35
# setenv gatewayip 192.168.0.1
# setenv netmask 255.255.255.0
# setenv serverip 192.168.0.130
```

设置从网络启动

```
# setenv bootcmd 'tftp c2000000 uImage;tftp c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000'
# setenv bootargs 'console=ttySTM0,115200 root=/dev/nfs nfsroot=192.168.0.130:/home/hu/linux/ubuntu/ubuntu-small,proto=tcp rw ip=192.168.0.250:192.168.0.130:192.168.0.1:255.255.255.0::eth0:off
```

4.3.2.6. 代码设置自动启动

亿佰特提供的 SDK 已经设置了自动启动。下面的内容供用户参考。

uboot 源码下 include/configs/stm32mp13_st_common.h：用于 STM32MP13x 的 STMicroelectronics 板。

我们可以在这里更改 uboot 启动内核的方式，从而实现烧录固件之后自动启动内核，而不需要使用上面章节的方式启动 uboot 之后，手动设置启动方式。

```
#define ST_STM32MP13_BOOTCMD "bootcmd_stm32mp=" \
 "echo \"Boot over ${boot_device}${boot_instance}!\";" \
 "if test ${boot_device} = serial || test ${boot_device} = usb;" \
 "then stm32prog ${boot_device} ${boot_instance}; " \
 "else " \
  "run env_check;" \
  "if test ${boot_device} = mmc;" \
  "then env set bootargs \"console=ttySTM0,115200 root=/dev/mmcblk0p9 rootwait rw\";" \
  "ext4load mmc 0:8 c2000000 uImage;"\
  "ext4load mmc 0:8 c4000000 stm32mp135-ebyte.dtb;"\
  "bootm c2000000 - c4000000;"\
  "fi;" \
  "if test ${boot_device} = nand ||" \
     " test ${boot_device} = spi-nand ;" \
  "then env set bootargs \"ubi.mtd=UBI rootfstype=ubifs root=ubi:rootfs rootwait rw
console=ttySTM0,115200\";" \
  "ubifsmount ubi:boot;"\
  "ubifsload c2000000 uImage;"\
  "ubifsload c4000000 stm32mp135-ebyte.dtb;"\
  "bootm c2000000 - c4000000;"\
  "fi;" \
  "if test ${boot_device} = nor;" \
  "then env set boot_targets mmc0; fi;" \
  "run distro_bootcmd;" \
 "fi;\0"
```

如上述代码所示，如果是从 sd 卡启动（mmc），就从 sd 卡的分区来加载设备树，内核和根文件系统。如果是从 nand 启动，就从 nand 来进行加载。

至此出厂 U-Boot 源码已编译完成，下面介绍使用 fiptool 工具封装 OP-TEE 和 U-Boot 固件。

4.4.Kernel

4.4.1. 获取源码并编译

在 ubuntu 的 目录下创建一个名为“linux” 的目录，然后把出厂 linux 源码包 02_Source/linux.tar.gz 拷贝到 linux 这个目录下。

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/linux$ ls
0001-release-v1.7.5.1_linux_5.15.x_maxio_driver.patch  CONTRIBUTING.md  drivers  Kbuild    MAINTAINERS  samples  tools
arch                                                   COPYING          fs       Kconfig   Makefile     scripts  usr
block                                                  CREDITS          include  kernel    mm           security virt
build.sh                                               crypto           init     lib       net          SECURITY.md
certs                                                  Documentation    ipc      LICENSES  README       sound
```

编译出厂源码时，用户不必修改任何内容。 适配于 ECK10-135A5M1G-I 开发板硬件资源的 uboot 源码设备树路径为 arch/arm/boot/dts/，设备树文件为 stm32mp135-ebyte.dts、stm32mp13-pinctrl-ebyte.dtsi。方便大家的编译，提供了编译脚本，第一次编译的时候可以直接运行脚本编译，脚本如下图所示：

```
#!/bin/bash
rm ../build -rf      You, 1秒钟前 • Uncommitted changes
make ARCH=arm O="$PWD/../build" ebyte_stm32mp135_D5E8_defconfig
# make ARCH=arm O="$PWD/../build" menuconfig
make ARCH=arm uImage vmlinux dtbs LOADADDR=0xC2000040 O="$PWD/../build" -j 8
make ARCH=arm modules O="$PWD/../build" -j 8
make ARCH=arm INSTALL_MOD_PATH="$PWD/../build/install_artifact" modules_install O="$PWD/../build" -j 8
mkdir -p $PWD/../build/install_artifact/boot/
cp $PWD/../build/arch/arm/boot/uImage $PWD/../build/install_artifact/boot/
cp $PWD/../build/arch/arm/boot/dts/ebyte-stm32mp13* $PWD/../build/install_artifact/boot/
sudo cp ../build/install_artifact/boot/uImage /mnt/bootfs
sudo cp ../build/install_artifact/boot/ebyte-stm32mp13* /mnt/bootfs
```

我们使用提供的 bootfs.ext4 来挂载到/mnt/bootfs，编译脚本会自动将编译好的设备树和内核镜像拷贝到/mnt/bootfs。

```
# cd FIP_artifacts/
# mkdir /mnt/bootfs
# sudo mount bootfs.ext4 /mnt/bootfs
# cd ../linux/
#  source  /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

然后执行./build.sh 编译脚本，第一次编译可能会比较慢。编译之后会在/mnt/bootfs 文件夹中生成 uImage 镜像，ebyte-stm32mp135-D5E8.dtb，ebyte-stm32mp135-D2.dtb 设备树文件。

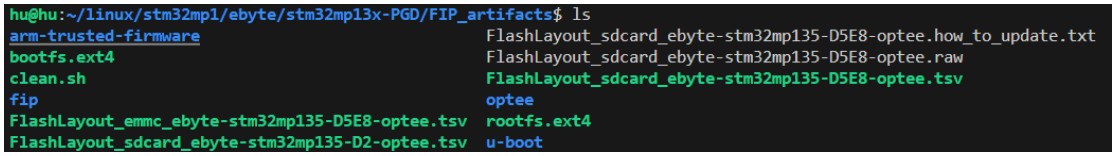
```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/FIP_artifacts$ ls /mnt/bootfs
boot.scr.uimg           ebyte-stm32mp135-D5E8.dtb  mmc0_extlinux  splash_landscape.bmp  st-image-resize-initrd
ebyte-stm32mp135-D2.dtb lost+found                 mmc1_extlinux  splash_portrait.bmp   uImage
```

最后取消挂载即可

```
# sudo umount /mnt/bootfs
```

4.5.小结

通过上述步骤，我们得到了所有源码部分的镜像文件，如下图所示。



5. 如何适配不同的硬件平台

5.1. 在设备树中添加硬件资源

用户需要对 CPU 的芯片手册，以及 ECB10-TB13X 核心板的产品手册，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。

用户可以在 BSP 源码里创建自己的设备树，一般情况下不需要修改 Bootloader 部分中的 TF-a、optee 和 u-boot。用户只需要根据实际的硬件资源对 Linux 内核设备树进行适当的调整即可。在此将 ECB10-TB13X 的 BSP 各个部分中的设备树列表罗列出来，方便用户开发。

项目	设备树	说明
TF-A	ebyte-stm32mp13x-base.dts	资源设备树
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D2-fw-config.dts ebyte-stm32mp135-D5E8.dts ebyte-stm32mp135-D5E8-fw-config.dts	核心板资源描述
	stm32mp13-ddr3-1x2Gb-1066-binF.dtsi stm32mp13-ddr3-1x4Gb-1066-binF.dtsi	单片 256MB DDR3 描述 单片 512MB DDR3 描述
	ebyte-stm32mp13-pinctrl.dtsi	管脚复用描述
Optee	ebyte-stm32mp13x-base.dts	资源设备树
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D5E8.dts	核心板资源描述

	ebyte-stm32mp13-pinctrl.dtsi	管脚复用描述
U-Boot	ebyte-stm32mp13x-base.dts	资源设备树
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D2-u-boot.dtsi ebyte-stm32mp135-D5E8.dts ebyte-stm32mp135-D5E8-u-boot.dtsi	核心板资源描述
	ebyte-stm32mp13-pinctrl.dtsi	管脚复用描述
	ebyte_stm32mp135_D2_defconfig	Uboot 配置文件
	ebyte_stm32mp135_D5E8_defconfig	
Kernel	ebyte_stm32mp135_D5E8_defconfig	内核配置文件
	ebyte-stm32mp13x-base.dts	资源设备树
	ebyte-stm32mp135-D2.dts	ECK10-135D2-C 设备树
	ebyte-stm32mp135-D5E8.dts	ECK10-135D5E8-C 设备树
	ebyte-stm32mp13-pinctrl.dtsi	管脚复用

用户一般只需要在内核的设备树中添加不同的硬件资源，再根据具体的硬件资源适配驱动即可。

5.2. 设备树的添加

Linux 内核设备树是一种数据结构，它通过特有的语法格式描述片上片外的设备信息。由 BootLoader 传递给 kernel，kernel 进行解析后形成和驱动程序关联的 dev 结构供驱动代码使用。

在内核源码下 arch/arm/boot/dts 下可以看到大量的平台设备树。如适合 ECK10-135A5M1G-I 的设备树，可在当前路径下增加自定义设备树，如：ebyte-stm32mp135a-xxx.dts。如下图所示。

```
hu@ubuntu:~/linux/ebyte/stm32mp131/linux-6.1.28/arch/arm/boot/dts$ ls stm32mp13*
stm32mp131.dtsi       stm32mp133.dtsi       stm32mp135-ebyte.dts          stm32mp135f-dk.dtb            stm32mp13-pinctrl.dtsi  stm32mp13xc.dtsi
stm32mp131-ebyte.dtb  stm32mp135.dtsi       stm32mp135f-dk-a7-examples.dtb  stm32mp135f-dk.dts          stm32mp13xa.dtsi        stm32mp13xd.dtsi
stm32mp131-ebyte.dts  stm32mp135-ebyte.dtb  stm32mp135f-dk-a7-examples.dts  stm32mp13-ebyte-pinctrl.dtsi  stm32mp13x-base.dts     stm32mp13xf.dtsi
```

我们将 ECB10-TB13X 核心板相关的资源编写进 stm32mp135-base.dts。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）：

```
/dts-v1/;

#include "stm32mp13x-base.dts"

/ {
    model = "EBYTE TECH STM32MP131 Discovery Board";
    compatible = "st,stm32mp131-ebyte", "st,stm32mp135";

    memory@c0000000 {
        device_type = "memory";
        reg = <0xc0000000 0x10000000>;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;

        optee_framebuffer@cd000000 {
            reg = <0xcd000000 0x1000000>;
            no-map;
        };
        optee@ce000000 {
            reg = <0xce000000 0x2000000>;
            no-map;
        };
    };

```

用户增加了新的设备树源文件之后，还需要在同目录下的 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。

在 arch/arm/boot/dts/Makefile 中

dtb-$(CONFIG_ARCH_STM32) += \的配置项下添加 stm32mp131-ebyte.dtb \

```
    stm32mp157f-ev1.dtb \
    stm32mp157f-ev1-a7-examples.dtb \
    stm32mp157f-ev1-m4-examples.dtb \
    stm32mp131-ebyte.dtb \      You, 4天前 • aud
    stm32mp135-ebyte.dtb
```

增加完成后即可增加设备驱动的板载描述，通过 4.5.1 节的方法编译生成设备树 dtb 文件 stm32mp131-ebyte.dtb。

5.3. 如何根据硬件配置 CPU 功能管脚

实现一个功能引脚的控制是一个较为复杂的系统开发过程之一，其中包含了引脚的配置，驱动的开发，应用的实现等等步骤，本节不具体分析每个部分的开发过程，而是以实例来讲解功能管脚的控制实现。

5.3.1. GPIO 管脚的配置方法

GPIO: General-purpose input/output，通用的输入输出口，在嵌入式设备中是一个十分重要的资源，可以通过它们输出高低电平或者通过它们读入引脚的状态-是高电平或是低电平。STM32MP1 封装大量的外设控制器，这些外设控制器与外部设备交户一般是通过控制 GPIO 来实现，而将 GPIO 被外设控制器使用我们称为复用（Alternate Function），给它们赋予了更多复杂的功能，如用户可以通过 GPIO 口和外部硬件进行数据交互(如 UART)，控制硬件工作(如 LED、蜂鸣器等),读取硬件的工作状态信号（如中断信号）等。所以 GPIO 口的使用非常广泛。

STM32MPU 的 GPIO 管脚配置方法一般使用 STM32CubeMX 配置或使用 Datasheet 查表的方式来配置。

5.3.1.1. STM32CubeMX 配置方法

目前 ST 已经将 STM32MPU 系列 CPU 添加进 STM32CubeMX，我们也可以用此工具来配置 TF-A，U-boot 以及 Kernel 的 GPIO 功能设备树和外设时钟。本节不重点讲解其使用方法，您可以通过官方网站获取详细的开发指导说明。

WIKI: https://wiki.st.com/stm32mpu/wiki/STM32CubeMX

ST 官方:https://www.st.com/zh/development-tools/stm32cubemx.html

5.3.1.2. 查询手册方式

GPIO 的配置可通过亿佰特整理的 Datasheet 找到描述文件（01_Documents\Datasheet\STM32MP135A&D 数据手册.pdf），示例如下：

```
i2c1_pins_a: i2c1-0 {
    pins {      You, 上个月 • add ebyte board support
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
        bias-disable;
        drive-open-drain;
        slew-rate = <0>;
    };
};
```

通过查询数据手册可以得到 I2C_SCL 在 PB8 管脚的复用关系为 AF4，I2C_SDA 在 PD3 引脚的复用关系为 AF5。

PB7	-	TIM17_CH1N	TIM4_CH2	-	-	I2S4_CK	I2C4_SDA	-
PB8	-	TIM16_CH1	TIM4_CH3	-	I2C1_SCL	I2C3_SCL	DFSDM1_DATIN1	-
PB9	TRACED3	-	TIM4_CH4	-	-	-	I2C4_SDA	-

PD2	TRACED4	-	TIM3_ETR	-	I2C1_SMBA	SPI3_NSS/ I2S3_WS	SAI2_D1	USART3_RX
PD3	-	-	TIM2_CH1/ TIM2_ETR	USART2_CTS/ USART2_NSS	DFSDM1_ CKOUT	I2C1_SDA	SAI1_D3	-
PD4	-	-	-	USART2_RTS/	-	SPI3_MISO/	DFSDM1_	-

5.3.2. 设备树中引用 GPIO

配置功能管脚为 I2C 功能实例，i2c 主要使用两个 GPIO 分别用作 I2C_SCL 和 I2C_SDA，在 pinctrl 子系统中定义好 i2c 的引脚位置和配置之后，在板级设备树中添加对其的引用即可，如下图所示。这里的 i2c 使用了两种引脚配置，分别是正常使用的默认状态和休眠时的引脚状态，分别为 default 和 sleep，系统根据状态进行自动切换。

```
&i2c1 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&i2c1_pins_a>;
    pinctrl-1 = <&i2c1_sleep_pins_a>;
    i2c-scl-rising-time-ns = <100>;
    i2c-scl-falling-time-ns = <7>;
    status = "okay";
    /delete-property/dmas;
    /delete-property/dma-names;
```

在 pinctrl 系统对 i2c 引脚的定义如下图所示。

```
i2c1_pins_a: i2c1-0 {        You, 上个月 • add ebyte board support
    pins {
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
        bias-disable;
        drive-open-drain;
        slew-rate = <0>;
    };
};

i2c1_sleep_pins_a: i2c1-sleep-0 {
    pins {
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
    };
};
```

5.3.3. 如何使用自己配置的管脚

5.3.3.1. U-boot 中使用 GPIO 管脚

uboot 可以直接在终端使用命令来控制 GPIO 的设置。如设置 GPIOB6，可使用下列命令。

```
STM32MP> gpio set GPIOB6
gpio: pin GPIOB6 (gpio 22) value is 1
STM32MP> gpio clear GPIOB6
gpio: pin GPIOB6 (gpio 22) value is 0
```

5.3.3.2. 用户空间使用 GPIO 管脚

Linux 操作系统的体系架构分为用户态和内核态（或者用户空间和内核）。用户态即上层应用程序的活动空间，应用程序的执行必须依托于内核提供的资源，包括 CPU 资源、存储资源、I/O 资源等。为了使上层应用能够访问到这些资源，内核必须为上层应用提供访问的接口：即系统调用。

Shell 是一个特殊的应用程序，俗称命令行，本质上是一个命令解释器，它下通系统调用，上通各种应用。使用 Shell 脚本，通常短短的几行 Shell 脚本就可以实现一个非常大的功能，原因就是这些 Shell 语句通常都对系统调用做了一层封装。为了方便用户和系统交互。

本节将讲述在用户态如何使用 GPIO 管脚的控制三种基本方式。

5.3.3.2.1. Shell 命令

GPIO 的测试是通过文件系统 sysfs 接口来实现的，下面内容以 PB6 为例说明 GPIO 的使用过程。

pin 脚的编号定义为 #define PIN_NO(port, line) (((port) - 'A') * 0x10 + (line))，其中 port 为 gpio 端口，line 为该 gpio 对应引脚，((port)-'A')代表 ASCII 码相减。如 PB6 对应的 pin 引脚编号为:PIN_NO('B',6)=(0x42-0x41)*0x10+6=(66-65)*16+6=22。

1） 导出 GPIO

echo 94 > /sys/class/gpio/export

导出成功后会在/sys/class/gpio/目录下生成 PB6 这个目录。

2） 设置/查看 GPIO 方向

设置输入

echo "in" > /sys/class/gpio/PB6/direction

设置输出

echo "out" > /sys/class/gpio/PB6/direction

查看 GPIO 方向

cat /sys/class/gpio/PB6/direction

out

3） 设置/查看 GPIO 的值

设置输出低

echo "0" > /sys/class/gpio/PB6/value

设置输出高

echo "1" > /sys/class/gpio/PB6/value

查看 GPIO 的值

cat /sys/class/gpio/PB6/value

可以看到 PB6 输出高电平，可以用万用表测量扩展 IO 的 PB6 引脚，可以看到电压为 3.3V 左右。

另外用户如果需要在应用程序控制 GPIO，可以参考一下 wiki：

https://wiki.st.com/stm32mpu/wiki/How_to_control_a_GPIO_in_userspace

5.3.3.2.2. 库函数

从 Linux 4.8 版本开始，Linux 引入了新的 gpio 操作方式，GPIO 字符设备。不再推荐使用以前 SYSFS 方式在"/sys/class/gpio"目录下来操作 GPIO，而是基于"文件描述符"的字符设备，每个 GPIO 组在"/dev"下有一个对应的 gpiochip 文件，例如"/dev/gpiochip0 对应 GPIOA, /dev/gpiochip1 对应 GPIOB"等等。

Libgpiod 库函数实现基于 gpiochip 的方式，提供了一些工具和更简易的 C API 接口。Libgpiod (Library General Purpose Input/Output device) 提供了完整的 API 给开发者，同时还提供了一些用户空间下的应用来操作 GPIO。

Libgpiod 常用基本接口描述：

gpiodetect - 列出系统中出现的所有 gpiochip，它们的名称，标签和 GPIO 行数。

gpioinfo - 列出指定的 gpiochips 的所有行、它们的名称、使用者、方向、活动状态和附加标志。

gpioget -读取指定的 GPIO 行值。

gpioset - 设置指定的 GPIO 行值，潜在地保持这些行导出并等待超时、用户输入或信号。

gpiofind - 查找给定行名称的 gpiochip 名称和行偏移量。

gpiomon - 等待 GPIO 行上的事件，指定要观察哪些事件，退出前要处理多少事件，或者是否应该将事件报告到控制台。

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。

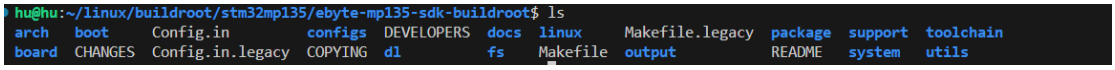
6.1. Buildroot 制作最小系统

6.1.1. 获取源码并编译

亿佰特提供了一个 buildroot 文件系统的例子，可以直接烧录。如果需要对其进行更改，将 buildroot 源码 ebyte-mp135-sdk-buildroot 拷贝到 ubuntu 虚拟机中，拷贝完成以后对其进行解压。

```
# tar -vxjf buildroot-2020.02.6.tar.bz2
```

解压后如下图所示

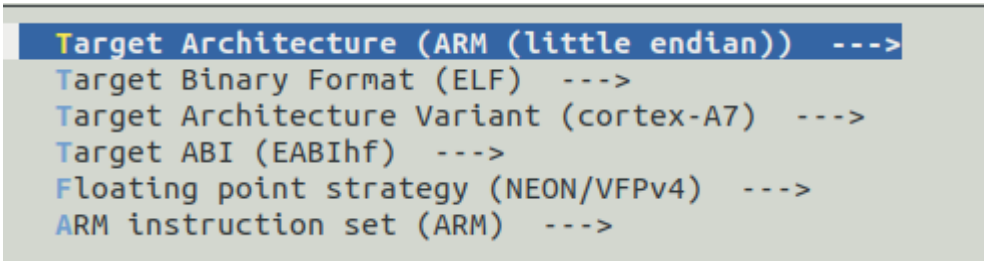


6.1.2. 配置 buildroot

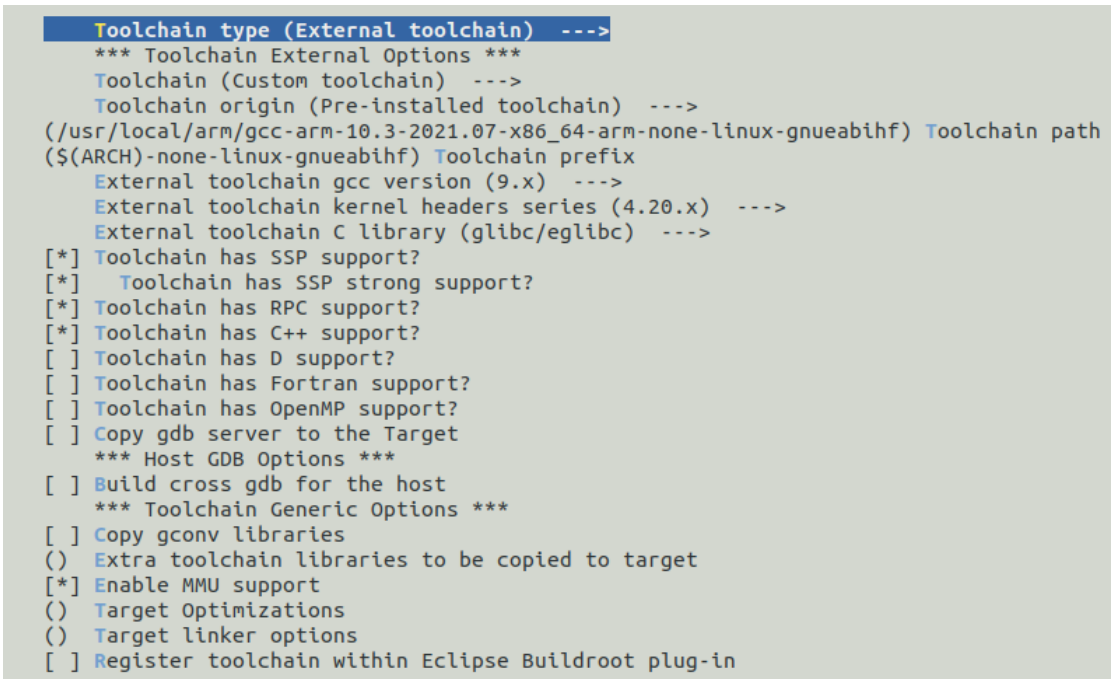
我们使用图形界面配置 buildroot，在根目录下使用如下命令：

```
# make menuconfig
```

配置 target options，配置结果如下所示。



配置 toolchain，配置结果如下所示



配置 system configuration

-> System hostname = stm32mp1 //平台名字，自行设置

-> System banner = Welcome to STM32MP135 //欢迎语

-> Init system = BusyBox //使用 busybox

-> /dev management = Dynamic using devtmpfs + mdev //使用 mdev

-> [*] Enable root login with password (NEW) //使能登录密码

-> Root password = 123456 //登录密码为 123456

配置 Filesystem images

-> [*] ext2/3/4 root filesystem //如果是 EMMC 或 SD 卡的话就用 ext3/ext4

-> ext2/3/4 variant = ext4 //选择 ext4 格式

-> exact size =1G //ext4 格式根文件系统 1GB(根据实际情况修改)

-> [*] ubi image containing an ubifs root filesystem //如果使用 NAND 的话就用 ubifs

配置内核支持 UBIFS

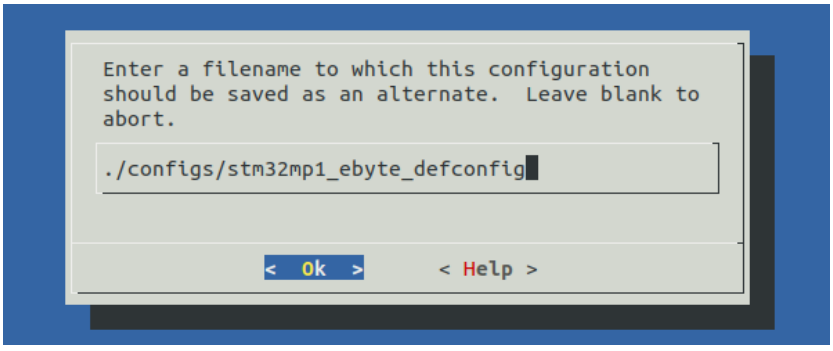
Device Drivers --->Memory Technology Device (MTD) support --->UBI - Unsorted block images --->Enable UBI

配置 mtd 支持 UBI 接口

File systems --->Miscellaneous filesystems --->UBIFS file system support

保存配置项

和 uboot、 kernel 一样，通过图形化界面配置好 buildroot 以后最好保存一下配置项，防止清除工程以后将配置项给删除掉。 buildroot 的默认配置项都保存在 configs 目录下，配置完成以后选择<Save>，然后输入要设置的配置项名字，如图所示：



编译 buildroot

直接使用 make -j8 进行编译即可。

6.2. 使用 ubuntu-base 制作系统

亿佰特提供了一个 ubuntu 文件系统的例子，可以直接烧录。

如果需要自定义，则参考下面的内容。ubuntu-base 是 Ubuntu 官方构建的 ubuntu 最小文件系统，包含 debain 软件包管理器，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。

6.2.1. 获取源码

官网地址：

http://cdimage.ubuntu.com/ubuntu-base/releases/

或者获取 04_Tools/ubuntu-base-18.04.5-base-armhf.tar.gz 并解压

```
hu@ubuntu:~/linux/ubuntu/ubuntu-1804$ tree -L 1
.
├── bin
├── boot
├── dev
├── etc
├── home
├── lib
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

6.2.2. 准备 chroot 环境

安装模拟器

```
# sudo apt-get install qemu-user-static
# sudo cp /usr/bin/qemu-arm-static ./ubuntu-1804/usr/bin/
增加 DNS 配置，后期可直接使用网络更新包
# echo "nameserver 8.8.8.8" > ./ubuntu-1804/etc/resolv.conf
```

制作挂载脚本

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() {
        echo "MOUNTING"
        sudo mount -t proc /proc ${2}/proc
        sudo mount -t sysfs /sys ${2}/sys
        sudo mount -o bind /dev ${2}/dev
        sudo chroot ${2}
}
function umnt(){
        echo "UNMOUNTING"
        sudo umount ${2}/proc
        sudo umount ${2}/sys
        sudo umount ${2}/dev
}
if [ "$1" == "-m" ] && [ -n "$2" ] ;
then
        mnt $1 $2
elif [ "$1" == "-u" ] && [ -n "$2" ];
then
        umnt $1 $2
else
        echo ""
        echo "Either 1'st, 2'nd or both parameters were missing"
        echo ""
        echo "1'st parameter can be one of these: -m(mount) OR -u(umount)"
        echo "2'nd parameter is the full path of rootfs directory(with trailing '/')"
        echo ""
        echo "For example: ch-mount -m /media/sdcard/"
        echo ""
        echo 1st parameter : ${1}
        echo 2nd parameter : ${2}
fi
```

6.2.3. 安装包文件

挂载系统

```
# ./ch-mount.sh -m ubuntu-1804
```

```
hu@ubuntu:~/linux/ubuntu$ ./ch-mount.sh -m ubuntu-1804
MOUNTING
[sudo] password for hu:
root@ubuntu:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ubuntu:/#
```

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
桌面系统的安装
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

设置登录串口

#systemctl enable getty@ttySTM0.service

6.2.4. 卸载系统

以上步骤操作完成后即可卸载系统。直接在系统中输入 exit 退出系统，并使用命令来卸载。

```
root@ubuntu:/# exit
exit
hu@ubuntu:~/linux/ubuntu$ ./ch-mount.sh -u ubuntu-1804/
UNMOUNTING
```

7. 参考资料

- Linux kernel 开源社区：

  https://www.kernel.org/

- STM32MPU 开发社区：

  https://wiki.st.com/stm32mpu/wiki/Development_zone

- STM32MP131 数据手册
- STM32MP135 数据手册

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	25-9-8	HSL	WYQ	WFX

9. 关于我们

销售热线：4000-330-990
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.