



软件应用指南
ECB10-135A5M5M-I 单板机

目录

免责声明和版权公告	1
1. 概述	3
1.1. 软件资源	3
2. 使用前准备	3
2.1. 串口软件安装	3
2.2. 设置拨码开关	4
2.3. 安装软件开发工具	4
2.4. 安装交叉编译工具链	7
2.5. 安装相关库	8
3. 快速开始	9
4. 功能测试	10
4.1. 核心资源	10
4.2. 外设接口	17
4.3. 网络接口	30
5. 参考资料	37
6. 修订说明	37
7. 关于我们	38

免责声明和版权公告

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-TB13X 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 ECK10-135A5M1G-I 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

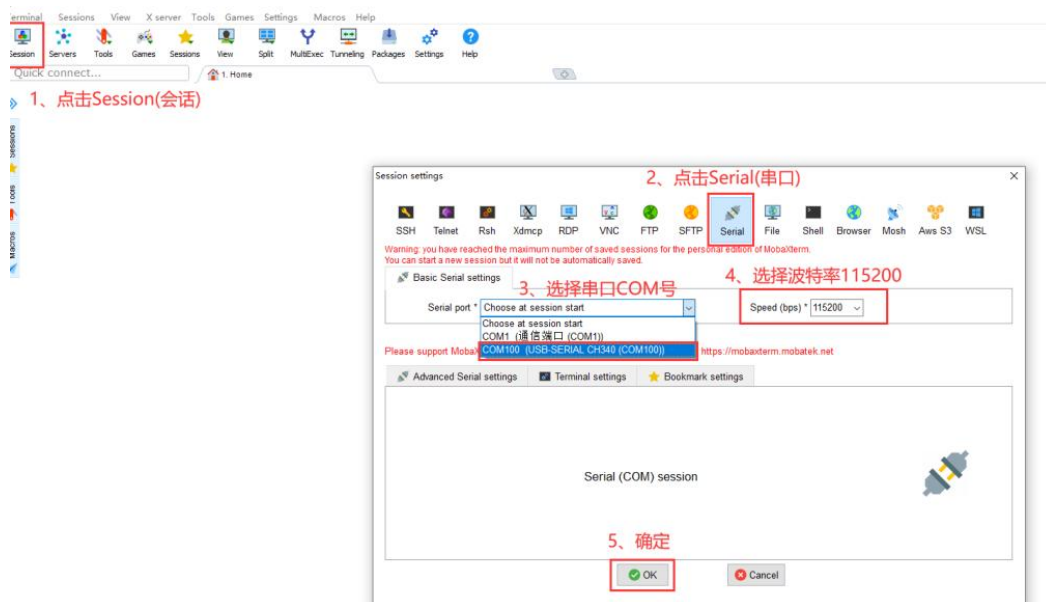
ECB10-TB13X 搭载基于 Linux 6.1.28 版本内核的操作系统，开发板出厂附带嵌入式 Linux 系统开发所需要的交叉编译工具链，TF-A 源代码，Optee-os 源码，U-boot 源代码，Linux 内核和各驱动模块的源代码，以及适用于 Windows 桌面环境和 Linux 桌面环境的各种开发调试工具，应用开发样例等。

2. 使用前准备

2.1. 串口软件安装

使用串口前主要需要安装 CH340 串口驱动和 MobaXterm 串口终端。具体安装方法可见核心板开发指南。

连接好串口之后，安装 MobaXterm 后进行配置，按如下方式打开串口。



2.2. 设置拨码开关

单板机支持三种启动模式，分别是 NandFlash，SD 卡和 USB（Download）启动模式。



单板机设置从 Nand 模式启动后，串口终端打印 TF-A、OP-TEE、U-Boot 和内核的运行信息，如下所示。

```
NOTICE: CPU: STM32MP135D Rev.Y
NOTICE: Model: EBYTE TECH STM32MP135 Discovery Board
INFO: PMIC version = 0x21
INFO: Reset reason (0x35):
INFO:   Power-on Reset (rst_por)
INFO: FCONF: Reading TB_FW firmware configuration file from: 0x2ffe0000
INFO: FCONF: Reading firmware configuration information for: stm32mp_io
INFO: Using FMC NAND
INFO:   Instance 1
INFO:   Boot used partition fsbl1
NOTICE: BL2: v2.8-stm32mp1-r1.0(debug):()
NOTICE: BL2: Built : 14:13:43, Jul 5 2024
INFO: BL2: Doing platform setup
INFO: RAM: DDR3-1066 bin F 1x4Gb 533MHz v1.53
INFO: Memory size = 0x20000000 (512 MB)
INFO: BL2: Loading image id 1
INFO: Loading image id=1 at address 0x30006000
INFO: Image id=1 loaded: 0x30006000 - 0x30006236
INFO: FCONF: Reading FW_CONFIG firmware configuration file from: 0x30006000
INFO: FCONF: Reading firmware configuration information for: mce_config
INFO: FCONF: Reading firmware configuration information for: dyn_cfg
INFO: FCONF: Reading firmware configuration information for: stm32mp1_firewall
INFO: BL2: Loading image id 4
INFO: Loading image id=4 at address 0xde000000
INFO: Image id=4 loaded: 0xde000000 - 0xde00001c
INFO: OPTEE ep=0xde000000
INFO: OPTEE header info:
INFO:   magic=0x4554504f
INFO:   version=0x2
INFO:   arch=0x0
INFO:   flags=0x0
INFO:   nb_images=0x1
```

2.3. 安装软件开发工具

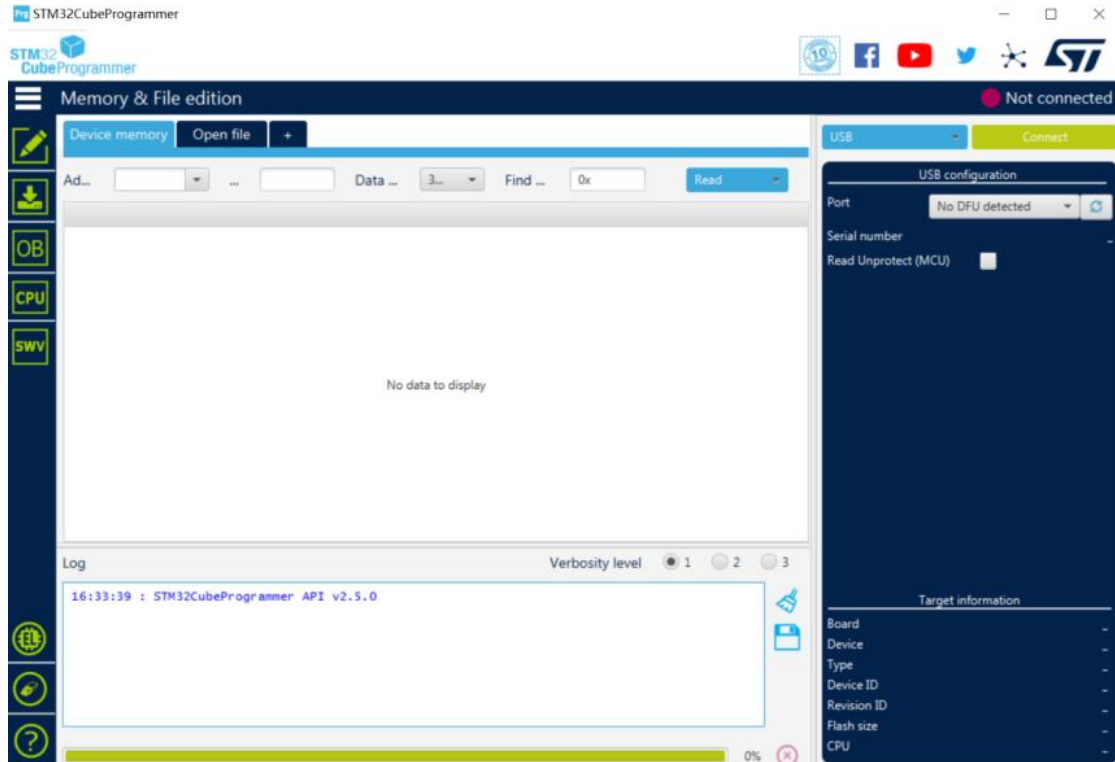
2.3.1. STM32CubeProg

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 03_Tools →

en.stm32cubeprog_v2-5-0.zip。

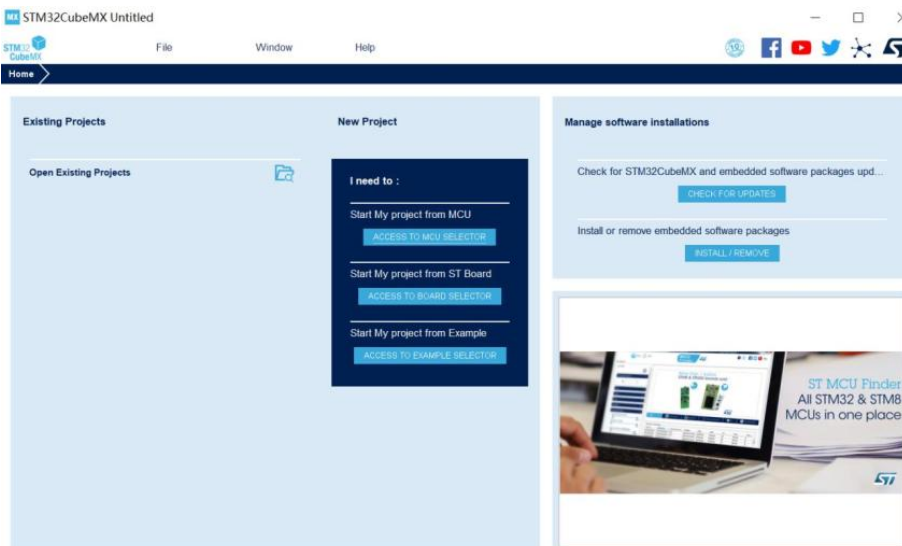
解压开发板中的 STM32CubeProgrammer 安装压缩包，然后双击解压出来的“SetupSTM32CubeProgrammer-2.5.0.exe”，安装过程很简单，根据提示进行安装即可，双击图标打开 STM32CubeProgrammer，如图所示



2.3.2. STM32CubeMX

STM32CubeMX 是 ST 公司推出的专门用于生成 STM32 的 HAL 代码的代码生成软件。它利用可视化界面来进行 STM32 时钟、定时器、DMA、串口、GPIO 等各种资源的配置。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, Optee, U-boot 以及 Kernel 的设备树和时钟，所以它可以用来在 Cortex A7 开发时生成设备树和时钟，下载地址如下：<https://www.st.com/zh/development-tools/stm32cubemx.html>。路径为 03_Tools → en.stm32cubemx_v6-0-1.zip。

双击 CubeMX 图标，打开以后如图所示：



2.3.3. JAVA 环境

在安装 STM32CbeMX 和 STM32CubeIDE 前我们要先安装 Java 的环境，Java 运行环境版本必须是 V1.7 及以上，否则会导致上述两个应用程序无法使用。
<https://www.java.com/zh-CN/download/manual.jsp> 查找下载最新的 64 位 Java 软件。路径为 03_Tools→jre-8u271-windows-x64.exe。



安装完 Java 运行环境之后，为了检测是否正常安装，我们可以打开 Windows 的 cmd 命令输入框，输入如下命令：

```
java -version //命令查询 Java 版本
```

如果安装成功的话就会打印出 Java 的版本号，如图所示：

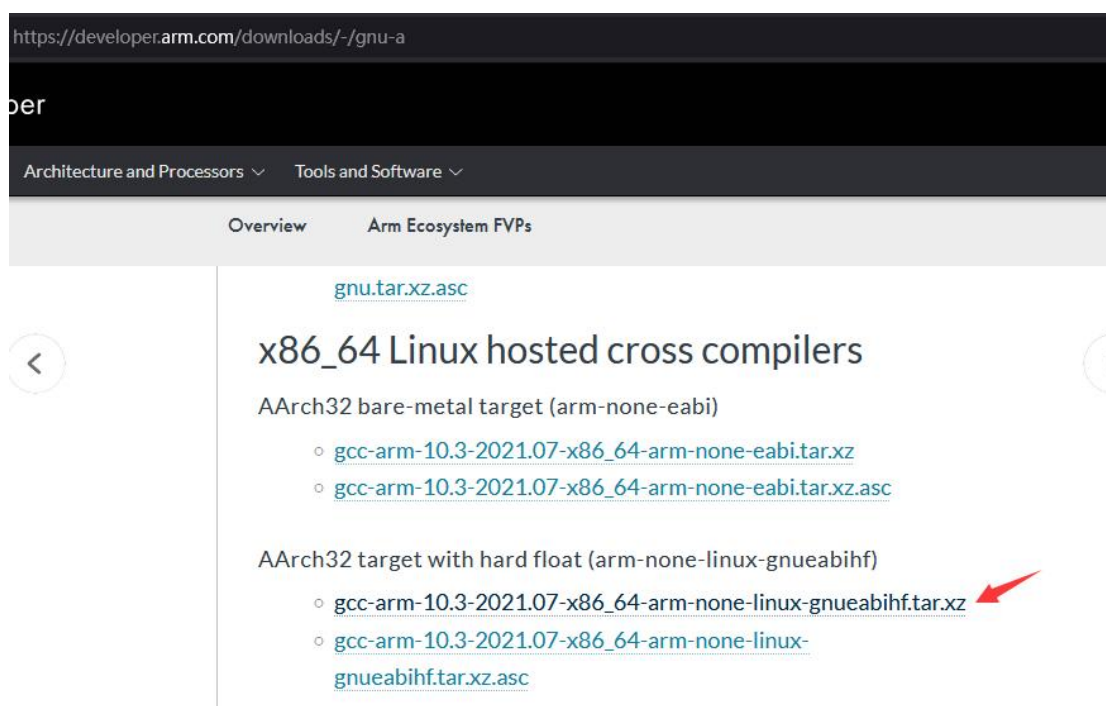

```
C:\Users\PC>java -version
java version "1.8.0_271"
Java(TM) SE Runtime Environment (build 1.8.0_271-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.271-b09, mixed mode)
```

2.4. 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。

使用 ARM 官方出品的交叉编译器, 编译器下载地址如下:

<https://developer.arm.com/tools-and-software/open-source-software/developertools/gnu-toolchain/gnu-a/downloads>, 打开后如图所示。



路径为 03_Tools→gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi.tar.xz。

在 Ubuntu 中创建目录: /usr/local/arm

```
sudo mkdir /usr/local/arm
```

将交叉编译器放到上面的目录中并解压

```
sudo cp gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi.tar.xz /usr/local/arm/ -f
```

```
sudo tar -vxf gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi.tar.xz
```

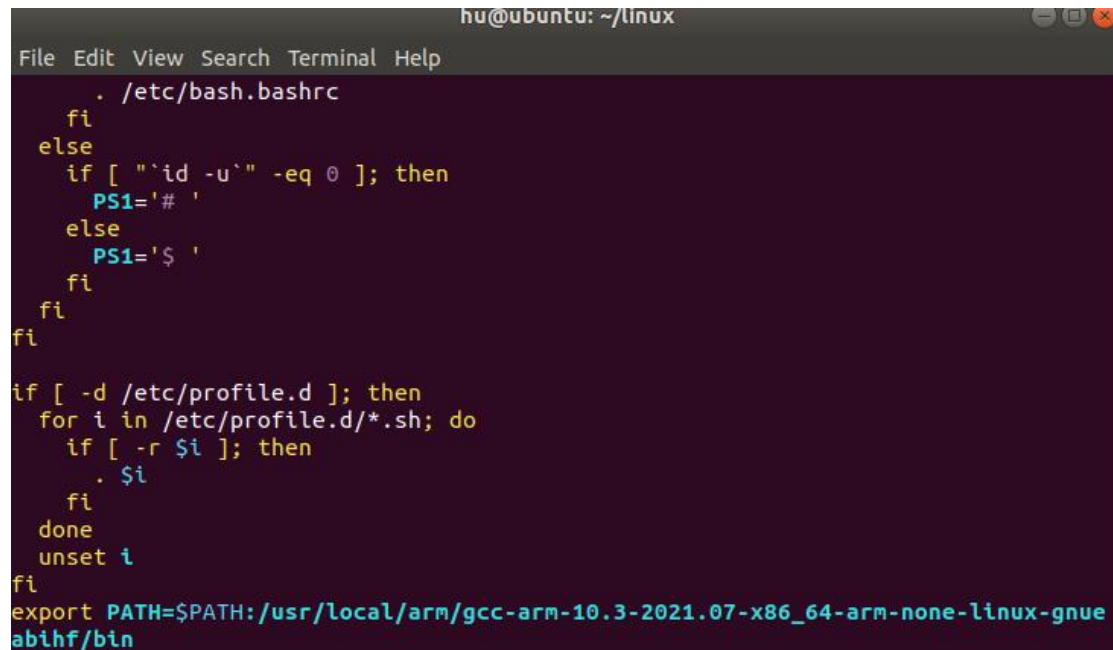
等待解压完成, 解压完成以后会生成一个名为“gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi”的文件夹, 这个文件夹里面就是我们的交叉编译工具链。

修改环境变量，使用打开/etc/profile 文件，命令如下：

```
sudo vi /etc/profile
```

打开/etc/profile 以后，在最后面输入如下所示内容：

```
Export PATH=$PATH:/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-  
gnueabi/bin
```

A terminal window titled 'hu@ubuntu: ~/linux' showing the editing of the /etc/profile file. The terminal displays the following code:

```
File Edit View Search Terminal Help  
. /etc/bash.bashrc  
fi  
else  
if [ "`id -u`" -eq 0 ]; then  
PS1='# '  
else  
PS1='$ '  
fi  
fi  
fi  
if [ -d /etc/profile.d ]; then  
for i in /etc/profile.d/*.sh; do  
if [ -r $i ]; then  
. $i  
fi  
done  
unset i  
fi  
export PATH=$PATH:/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi/bin
```

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。

2.5. 安装相关库

```
sudo apt-get update
```

```
sudo apt-get install lsb-core lib32stdc++6
```

安装验证

```
arm-none-linux-gnueabi-gcc -v
```

```

File Edit View Search Terminal Help
Using built-in specs.
COLLECT_GCC=arm-none-linux-gnueabi-hf-gcc
COLLECT_LTO_WRAPPER=/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabi-hf/bin/../libexec/gcc/arm-none-linux-gnueabi-hf/10.3.1/lto-wrapper
Target: arm-none-linux-gnueabi-hf
Configured with: /data/jenkins/workspace/GNU-toolchain/arm-10/src/gcc/configure
--target=arm-none-linux-gnueabi-hf --prefix= --with-sysroot=/arm-none-linux-gnueabi-hf/libc --with-build-sysroot=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabi-hf/install/arm-none-linux-gnueabi-hf/libc --with-bugurl=https://bugs.linaro.org/ --enable-gnu-indirect-function --enable-shared --disable-libssp --disable-libmudflap --enable-checking=release --enable-languages=c,c++,fortran --with-gmp=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabi-hf/host-tools --with-mpfr=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabi-hf/host-tools --with-mpc=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabi-hf/host-tools --with-isl=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabi-hf/host-tools --with-arch=armv7-a --with-fpu=neon --with-float=hard --with-mode=thumb --with-arch=armv7-a --with-pkgversion='GNU Toolchain for the A-profile Architecture 10.3-2021.07 (arm-10.29)'
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 10.3.1 20210621 (GNU Toolchain for the A-profile Architecture 10.3-2021.07 (arm-10.29))
    
```

“arm-none-linuxgnueabi-hf-gcc” 的含义如下：

- 1、 arm 表示这是编译 arm 架构代码的编译器。
- 2、 none 表示厂商，一般半导体厂商会修改通用的交叉编译器，此处就是半导体厂商的名字，ARM 自己做的交叉编译这里为 none，表示没有厂商。
- 3、 linux 表示运行在 linux 环境下。
- 4、 gnueabi-hf 表示嵌入式二进制接口，后面的 hf 是 hard float 的缩写，也就是硬件浮点运算，说明此交叉编译工具链支持硬件浮点运算。
- 5、 gcc 表示是 gcc 工具。

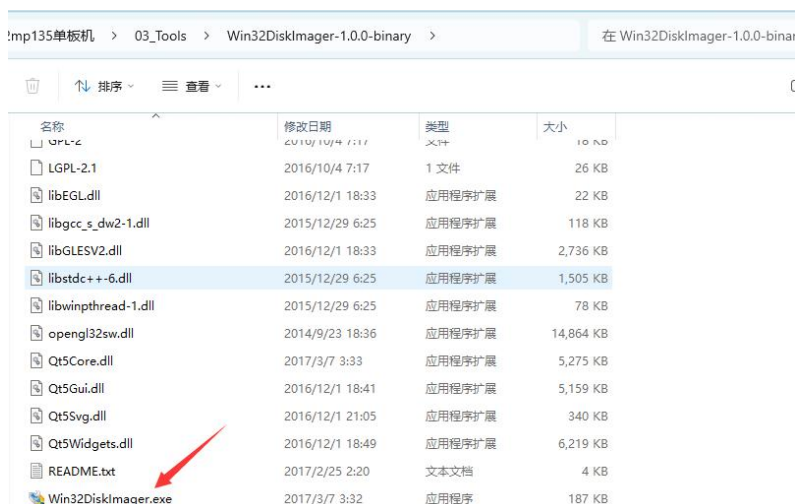
3. 快速开始

3.1. 镜像配置

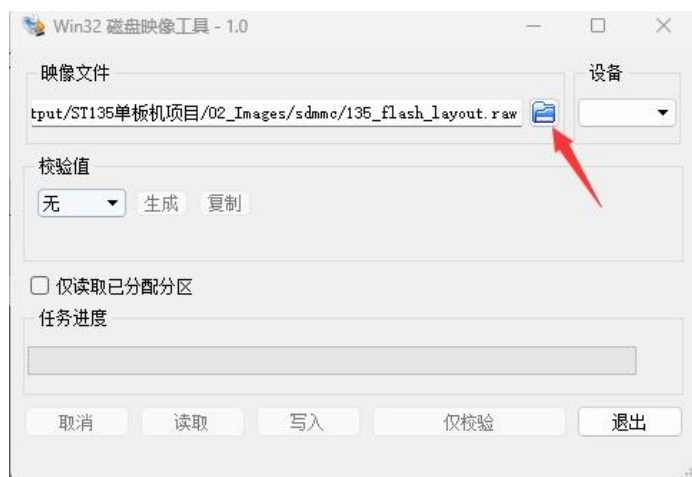
出厂开发板没有烧写镜像，请参考第二章内容配置拨码为从 SD 卡启动并制作相应 SD 启动卡，详细文档参考《Software_Development_Guide》。

下面简单描述如何制作 SD 卡：

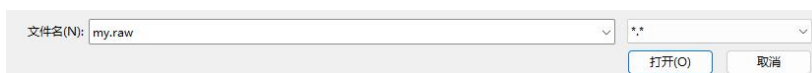
使用第三方的镜像烧录工具将位于 02_Images/sdmmc 中的 img 镜像文件 135_flash_layout.raw 写入 SD 卡，在 03_Tools/ 中已经包含了一个烧写工具：Win32DiskImager-1.0.0-binary。双击打开即可。



选择镜像文件 135_flash_layout.raw



文件类型要选择 *.* 才能找到烧录文件



使用读卡器找到设备之后，单击写入，然后等待烧写完成即可。

3.2. 上电启动

在板上找到 DEBUG 丝印，正确连接之后，使用 USB-TTL 串口连接上电脑，如果没有软件见第二章的内容。

单板机启动后，串口终端打印 U-Boot 和内核的运行信息。

4. 功能测试

4.1. 核心资源

在 Linux 系统中，提供了 proc 虚拟文件系统来查询各项核心资源的参数以及一些通用工具来评估资源的性能。下面将具体对 CPU, memory, eMMC, RTC 等核心资源的参数进行

读取与测试。

4.1.1. CPU

ECB10-135A5MA-I 核心芯片是 STM32MP135DAF7, 基于高性能单核 Arm®Cortex®-A7 32 位 RISC 核心, 工作频率为 650 MHz。Cortex-A7 处理器的 CPU 核心包括一个 32 kbyte L1 指令缓存, 一个 32 kbyte L1 数据缓存, 一个 128 kbyte 二级缓存。Cortex-A7 处理器是一款非常节能的应用处理器, 旨在为高端可穿戴设备以及其他低功耗嵌入式和消费应用提供丰富的性能。它提供了比 Cortex-A5 多 20% 的单线程性能, 并且提供了与 Cortex-A9 相似的性能。

4.1.1.1. 查看 CPU 信息

使用 `cat /proc/cpuinfo` 查看 CPU 信息。

```
root@ebyte-ubuntu:~# cat /proc/cpuinfo
processor      : 0
model name    : ARMv7 Processor rev 5 (v7l)
BogoMIPS     : 48.00
Features      : half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpv4 idiva idivt
              vfpd32 lpae evtstrm
CPU implementer : 0x41
CPU architecture: 7
CPU variant   : 0x0
CPU part      : 0xc07
CPU revision  : 5

Hardware      : STM32 (Device Tree Support)
Revision     : 0000
Serial       : 000A00103432511635373334
```

processor: 系统中逻辑处理核的编号, 对于多核处理器则可以是物理核、或者使用超线程技术虚拟的逻辑核

model name: CPU 属于的名字及其编号

BogoMIPS: 在系统内核启动时粗略测算的 CPU 每秒运行百万条指令数 (MillionInstructions Per Second)

4.1.1.2. 查看 CPU 使用率

```
top - 13:56:36 up 58 min,  0 users,  load average: 0.00, 0.00, 0.11
Tasks:  64 total,   1 running, 33 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  1.0 sy,  0.0 ni, 99.0 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem : 449616 total, 370472 free,   30388 used,   48756 buff/cache
KiB Swap:    0 total,    0 free,    0 used. 407640 avail Mem

  PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+
COMMAND
 255 root        20   0   4908   2284   1868 R   0.7   0.5    0:00.16 top
    1 root        20   0  28520   5696   4472 S   0.0   1.3    0:03.30 systemd
    2 root        20   0     0     0     0 S   0.0   0.0    0:00.01 kthreadd
    3 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 rcu_gp
    4 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 rcu_par_gp
    5 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 slub_flushwq
    6 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 netns
    8 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 kworker/0:0+
    9 root        20   0     0     0     0 I   0.0   0.0    0:03.06 kworker/u2:+
   10 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 mm_percpu_wq
   11 root        20   0     0     0     0 I   0.0   0.0    0:00.00 rcu_tasks_k+
   12 root        20   0     0     0     0 I   0.0   0.0    0:00.00 rcu_tasks_t+
   13 root        20   0     0     0     0 S   0.0   0.0    0:00.25 ksoftirqd/0
   14 root        20   0     0     0     0 I   0.0   0.0    0:00.34 rcu_preempt
   15 root        20   0     0     0     0 S   0.0   0.0    0:00.02 kdevtmpfs
   16 root         0 -20     0     0     0 I   0.0   0.0    0:00.00 inet_frag_wq
   20 root        20   0     0     0     0 I   0.0   0.0    0:00.77 kworker/0:4+
```

%usr: 表示用户空间程序的 cpu 使用率

%sys: 表示系统空间的 cpu 使用率

%idle: 空闲 cpu

%irq: cpu 处理硬中断的数量

%sirq: cpu 处理软中断的数量

4.1.1.3. 查看 CPU 温度信息

通过 `cat /sys/class/hwmon/hwmon0/temp1_input` 来查看 CPU 的内部温度。

```
root@ebyte-ubuntu:~# cat /sys/class/hwmon/hwmon0/temp1_input  
54988
```

上面的数值除以 1000 就是正确的温度值，单位为摄氏度。

4.1.2. 内存

4.1.2.1. 查看内存信息

通过 `cat /proc/meminfo` 可以查看内存信息。

```
root@ebyte-ubuntu:~# cat /proc/meminfo  
MemTotal:          449616 kB  
MemFree:           370472 kB  
MemAvailable:     407640 kB  
Buffers:            0 kB  
Cached:            39208 kB  
SwapCached:        0 kB  
Active:            13408 kB
```

MemTotal: 总内存量。这里显示的值为 449616 kB，表示系统总共有大约 449 MB 的物理内存可用。

MemFree: 空闲内存量。这里显示的值为 370472 kB，表示当前系统中有大约 370 MB 的内存是空闲的，未被使用。

MemAvailable: 可用内存量。这里显示的值为 407640 kB，表示当前可供系统使用的内存总量，包括已缓存的内存和可用的内存。

Buffers: 缓冲区使用量。这里显示的值为 0 kB，表示系统当前没有使用任何内存作为缓冲区。

Cached: 缓存的内存量。这里显示的值为 39208 kB，表示系统当前用于缓存的内存量。

SwapCached: 交换缓存的内存量。这里显示的值为 0 kB，表示当前没有被缓存到交换空间中的内存。

Active: 活跃的内存量。这里显示的值为 13408 kB，表示当前正在使用的内存量。

4.1.2.2. 内存压力测试

通过给定测试内存的大小和次数，可以对系统现有的内存进行压力上的测试。可使用系统工具 `memtester` 进行测试，如指定内存大小 10MB，测试次数为 10，测试命令为“`memtester 10M 10`”。

```
root@ebyte-ubuntu:~# memtester 10M 10
memtester version 4.3.0 (32-bit)
Copyright (C) 2001-2012 Charles Cazabon.
Licensed under the GNU General Public License version 2 (only).

pagesize is 4096
pagesizemask is 0xffff000
want 10MB (10485760 bytes)
got 10MB (10485760 bytes), trying mlock ...locked.
Loop 1/10:
  Stuck Address      : ok
  Random Value       : ok
  Compare XOR        : ok
  Compare SUB        : ok
  Compare MUL        : ok
  Compare DIV        : ok
  Compare OR         : ok
  Compare AND        : ok
  Sequential Increment: ok
  Solid Bits         : ok
  Block Sequential   : ok
  Checkerboard       : ok
  Bit Spread         : ok
  Bit Flip           : ok
  Walking Ones       : ok
  Walking Zeroes     : ok
  8-bit Writes       : ok
  16-bit Writes      : ok
*****
```


4.1.3. NandFlash 测试

4.1.3.1. 查看 Nand 容量及大小

```
root@ebyte-ubuntu:~# cat /proc/mtd
dev:   size   erasesize  name
mtd0: 00080000 00020000 "fsb11"
mtd1: 00080000 00020000 "fsb12"
mtd2: 00080000 00020000 "metadata1"
mtd3: 00080000 00020000 "metadata2"
mtd4: 00400000 00020000 "fip-a1"
mtd5: 00400000 00020000 "fip-a2"
mtd6: 00400000 00020000 "fip-b1"
mtd7: 00400000 00020000 "fip-b2"
mtd8: 0ee00000 00020000 "UBI"
```

4.1.4. RTC

RTC (Real-time clock) 本身是一个时钟, 用来记录真实时间, 当软件系统关机后保留系统时间并继续进行计时, 系统重新开启后在将时间同步进软件系统。

STM32MP135 芯片内部包含 RTC 时钟, 如果实际产品对 RTC 功耗要求不是很高, 对断电时间保持要求在一个月以内, 可以直接使用芯片内部 RTC, 否则就需要采用专用外部 RTC 芯片了。RTC 的测试通常采用 Linux 系统常用的 `hwclock` 和 `date` 命令配合进行, 下面测试将系统时间写入 RTC, 读取 RTC 时间并设置为系统时间并进行时间掉电保持的测试。

设置系统时间

将系统时间设置为 2024.07.12 16:35:15

```
root@ebyte-ubuntu:~# date 071216352024.15
Fri Jul 12 16:35:15 UTC 2024
```

将系统时间写入 RTC

```
root@ebyte-ubuntu:~# hwclock -w
```

```
root@ebyte-ubuntu:~# hwclock -r  
2000-01-01 07:07:15.965766+0000
```

掉电保持 RTC 时间

将开发板关机断开电源，一段时间后重新上电开机。查看 RTC 时间和系统时间：

```
root@ebyte-ubuntu:~# hwclock -r  
2024-07-12 16:52:42.648561+0000
```

4.1.5. WatchDog

使用测试例程测试看门狗，文件已经被编译好放在 `home` 目录下。运行看门狗应用,超时时间为 4s，每间隔 1s 喂一次狗：

```
root@ebyte-ubuntu:/home# ./watchdog 4 1 0  
Starting wdt_driver (timeout: 4, sleep: 1, test: ioctl)  
Trying to set timeout value=4 seconds  
The actual timeout was set to 4 seconds  
Now reading back -- The timeout is 4 seconds
```

如果将上面的 1s 改到大于 4s，则超过了要求的 4s 喂狗时间，开发板会重启。

4.1.6. 电源管理

本章节演示 Linux 电源管理的 Suspend 功能，让开发板睡眠，通过外部事件唤醒。Linux 内核一般提供了三种 Suspend: Freeze、Standby 和 STR(Suspend to RAM)，在用户空间向 `/sys/power/state` 文件分别写入 `freeze`、`standby` 和 `mem`，即可触发它们。目前只支持休眠到内存的方式，即 `mem` 方式。

查看当前支持的模式

```
root@ebyte-ubuntu:~# cat /sys/power/state  
mem
```

设置唤醒源

```
root@ebyte-ubuntu:/home#echoenabled >/sys/devices/platform/soc/40010000.serial/tty/ttyST  
M0/power/wakeup
```

休眠到内存

```
root@ebyte-ubuntu:~# echo "mem" > /sys/power/state
```

4.2. 外设接口

4.2.1. GPIO

GPIO 的测试是通过文件系统 sysfs 接口来实现的。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 PA8 对应的 pin 引脚编号为:PIN_NO('A',8)=(0x41-0x41)*0x10+8=(65-65)*16+8=8。

导出 GPIO

```
root@ebyte-ubuntu:~# cd /sys/class/gpio/  
root@ebyte-ubuntu:/sys/class/gpio# ls  
export      gpiochip112  gpiochip16  gpiochip48  gpiochip80  unexport  
gpiochip0   gpiochip128  gpiochip32  gpiochip64  gpiochip96  
root@ebyte-ubuntu:/sys/class/gpio# echo 8 > export
```

设置 GPIO 方向

输入

```
root@ebyte-ubuntu:/sys/class/gpio# echo "in" > PA8/direction
```

输出

```
root@ebyte-ubuntu:/sys/class/gpio# echo "out" > PA8/direction
```

设置 GPIO 值

```
root@ebyte-ubuntu:/sys/class/gpio# echo 1 > PA8/value
```

注销 GPIO

```
root@ebyte-ubuntu:/sys/class/gpio# echo 8 > unexport
root@ebyte-ubuntu:/sys/class/gpio# ls
export      gpiochip112  gpiochip16  gpiochip48  gpiochip80  unexport
gpiochip0   gpiochip128  gpiochip32  gpiochip64  gpiochip96
```

4.2.2. LED 灯

Linux 系统提供了一个独立的子系统以方便从用户空间操作 LED 设备, 该子系统以文件的形式为 LED 设备提供操作接口。这些接口位于 `/sys/class/leds` 目录下。在硬件资源列表中, 我们已经列出了开发板上所有的 LED。下面通过命令读写 `sysfs` 的方式对 LED 进行测试。下述命令均为通用命令, 也是操控 LED 的通用方法。

```
root@ebyte-ubuntu:/sys/class/leds# ls
green:heartbeat
```

点亮和熄灭 LED

```
root@ebyte-ubuntu:/sys/class/leds/green:heartbeat# echo 0 > brightness
root@ebyte-ubuntu:/sys/class/leds/green:heartbeat# echo 1 > brightness
```

查看 LED 触发模式

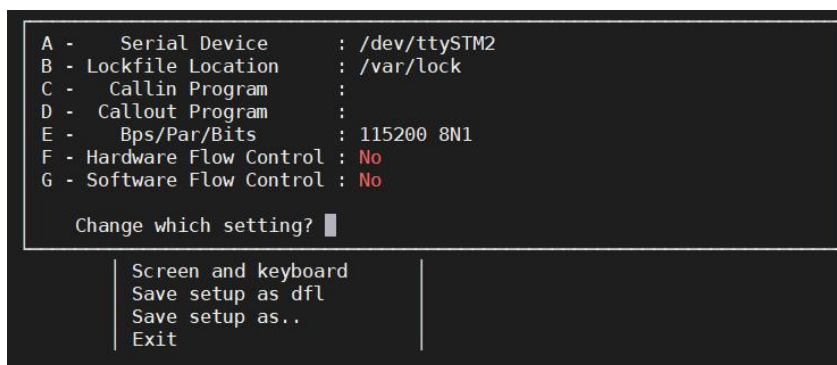
```
root@ebyte-ubuntu:/sys/class/leds/green:heartbeat# cat trigger
[none] rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrlock
kbd-ctrllllock kbd-ctrlrlock timer oneshot heartbeat backlight gpio cpu cpu0 default-on
transient flash torch mmc0 mmc1
```

4.2.3. 串口

本单板机引出了两路串口供使用，分别是 UART5 和 USART3,其中 UART5 是双线制，只有 RX 和 TX，USART3 是四线制，有 RX,TX,RTS 和 CTS。

4.2.3.1. UART5

此串口可以直接使用 usb-ttl 转接线连接 PC 进行测试。在单板机中被注册为 ttySTM2，使用 minicom 打开 ttySTM2 即可通讯。



```
A - Serial Device      : /dev/ttySTM2
B - Lockfile Location  : /var/lock
C - Callin Program    :
D - Callout Program   :
E - Bps/Par/Bits      : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No

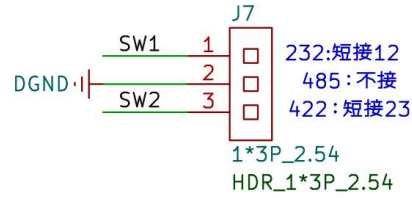
Change which setting? █

| Screen and keyboard |
| Save setup as dfl  |
| Save setup as..   |
| Exit               |
```

4.2.3.2. USART3

本串口支持三种协议，RS485,RS422 和 RS232，通过跳线帽来选择。在单板机中被注册为 ttySTM1。

	SW1	SW2
485	NO	NO
422	NO	NC
232	NC	NO



4.2.3.2.1. RS232

使用跳线短接 J7 的 12 脚，使用 RS232 转 USB 连接电脑，可以使用 minicom 和单板机进行通信。

支持开启硬件流控和关闭两种通信方式。

4.2.3.2.2. RS422

使用跳线短接 J7 的 23 脚，使用 RS422 转 USB 连接电脑，可以使用 minicom 和单板机进行通信。

4.2.3.2.3. RS485

可以使用 termios 库来操作 RTS 控制线来操作发送使能和接收使能，示例代码如下。

```
****

#include <termios.h>

int main() {

    int serial_fd = open("/dev/ttySTM1", O_RDWR);

    if (serial_fd == -1) {

        perror("Failed to open serial port");

        return 1;

    }

    int status;

    if (ioctl(serial_fd, TIOCMGET, &status) == -1) {

        perror("Failed to get modem status");

        close(serial_fd);

        return 1;

    }

    // 设置 RTS 信号

    status |= TIOCM_RTS; // 设置 RTS 为高电平（使能发送）high
// status &= ~TIOCM_RTS; // 设置 RTS 为低电平 low

    if (ioctl(serial_fd, TIOCMSET, &status) == -1) {

        perror("Failed to set modem status");

        close(serial_fd);

        return 1;

    }

    // 在这里可以进行其他操作或数据传输

    while(1);

    close(serial_fd);

    return 0;

}
```


使用 RS485 转 usb 连接 PC 之后, 继续使用 linux 系统调用 write 和 read 进行读写数据。

4.2.4. CAN

本节采用 Linux 系统常用的 cansend、candump 命令进行 SocketCAN 的通讯测试。

这里测试使用的 USB-CAN 模块对单板机的 CAN 接口进行测试。CANH, CANL 分别与 USB 转 CAN 模块的 CANH, CANL 相连。

初始化 CAN 网络接口

```
root@ebyte-ubuntu:/ # ifconfig can0 down  
root@ebyte-ubuntu:/ # ip link set can0 type can bitrate 1000000
```

上述命令设置 can0 速度为 1000Kbit/S, 两个 CAN 设备的速度要设置为一样的! 速度设置好以后打开 can0 网卡, 命令如下:

```
root@ebyte-ubuntu:/ # ifconfig can0 up
```

can0 打开以后就可以使用 can-utils 里面的小工具进行数据收发测试了。开发板输入如下命令进行接收和发送:

```
root@ebyte-ubuntu:~# candump can0 //接收数据  
root@ebyte-ubuntu:~# cansend can0 5A1#11.22.33.44.55.66.77.88
```

cansend 命令用于发送 can 数据, “5A1” 是帧 ID, “#” 号后面的 “11.22.33.44.55.66.77.88” 就是要发送的数据, 十六进制。CAN2.0 一次最多发送 8 个字节的数据, 8 个字节的数据之间用 “.” 隔开。如果 USB 转 CAN 设备工作正常的话接收端就会接收到上面发送过来的这 8 个字节的数据, 如图所示:

序号	时间标识	源通道	帧ID	CAN类型	方向	长度	数据
0	0.000000	0	0x5A1	CAN	Rx	8	11 22 33 44 55 66 77 88

4.2.5. USB

本节通过相关命令或热插拔、USB HUB 验证 USB Host 驱动的可行性, 实现读写 U 盘的功能、usb 枚举功能。

4.2.5.1.1. 插入 U 盘

插入 U 盘和使用 lsblk 查找存储设备。

```
root@ebyte-ubuntu:~# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                  8:0    1 58.6G  0 disk
|-sda1               8:1    1 58.6G  0 part
`-sda2               8:2    1    1M  0 part
mtdblock0           31:0    0  512K  0 disk
mtdblock1           31:1    0  512K  0 disk
mtdblock2           31:2    0  512K  0 disk
mtdblock3           31:3    0  512K  0 disk
mtdblock4           31:4    0    4M  0 disk
mtdblock5           31:5    0    4M  0 disk
mtdblock6           31:6    0    4M  0 disk
mtdblock7           31:7    0    4M  0 disk
mtdblock8           31:8    0 238M  0 disk
```

4.2.5.2. U 盘挂载读写

挂载 U 盘

```
root@ebyte-ubuntu:~# mount /dev/sda1 /mnt/sda/
```

读文件

提前在 u 盘中创建一些文件

```
root@ebyte-ubuntu:~# mount /dev/sda1 /mnt/sda/
root@ebyte-ubuntu:~# ls /mnt/sda/
MobaXterm_Installer_v12.3.zip      conf
'System Volume Information'       setup.exe
autorun.inf                        sources
boot                                support
```

写文件

```
root@ebyte-ubuntu:~# mount /dev/sda1 /mnt/sda/
root@ebyte-ubuntu:~# ls /mnt/sda/
MobaXterm_Installer_v12.3.zip      conf
'System Volume Information'      setup.exe
autorun.inf                        sources
boot                                support
```

4.2.6. OTG

硬件上并不支持 OTG 主从模式的自动切换，所以只能配置设备树 OTG 节点为主模式或者从模式。

```
/*otg*/
&usbotg_hs {
    phys = <&usbphyc_port1 0>;
    phy-names = "usb2-phy";
    // dr_mode = "peripheral"; //otg 从模式
    dr_mode = "host"; // otg 主模式
    vbus-supply = <&vbus_otg>;
    status = "okay";
};
```

配置为主模式之后，使用 usb 转 typec 母连接 usb 设备，将会自动挂载，比如鼠标，u 盘。



配置为从模式之后，使用 usb-typec 数据线连接 pc。然后加载 usb 大容量设备驱动 usb_f_mass_storage, libcomposite 之后，再将板上的存储设备挂载到 pc 即可。比如 u 盘，/dev/sda1, mmc 的某一个分区。

```
depmod  
挂载驱动, 并将 u 盘共享  
modprobe libcomposite  
modprobe usb_f_mass_storage  
modprobe g_mass_storage file=/dev/sda1 removable=1  
卸载驱动  
modprobe g_mass_storage -r  
modprobe usb_f_mass_storage -r  
modprobe libcomposite -r
```

4.2.7. SD 卡

ECB10-135A5M5M-I 使用 SDMMC1 连接 Micro SD。

4.2.7.1. 查看 TF 卡容量

通过 `fdisk -l` 命令可以查询到 TF 卡分区信息及容量:

```
lsblk  
  
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT  
|-mmcblk0p1 179:1    0   256K  0 part  
|-mmcblk0p2 179:2    0   256K  0 part  
|-mmcblk0p3 179:3    0   256K  0 part  
|-mmcblk0p4 179:4    0   256K  0 part  
|-mmcblk0p5 179:5    0     4M  0 part  
|-mmcblk0p6 179:6    0     4M  0 part  
|-mmcblk0p7 179:7    0   512K  0 part  
|-mmcblk0p8 179:8    0    64M  0 part  
|-mmcblk0p9 179:9    0  29.7G  0 part
```

4.2.7.2. TF 卡的性能测试

性能测试主要测试 eMMC 在 linux 系统下对文件的读写速度, 一般结合 `time` 与 `dd` 双命令进行测试。挂载需要测试的 TF 卡分区, 这里以最后一个分区 `/dev/mmcblk0p9` 为例, 挂

载目录为/mnt/rootfs。

写文件测试

```
time dd if=/dev/zero of=/mnt/rootfs/home/tempfile bs=1M count=100 conv=fdatasync  
  
100+0 records in  
  
100+0 records out  
  
104857600 bytes (105 MB, 100 MiB) copied, 5.95716 s, 17.6 MB/s  
  
real    0m6.203s  
user    0m0.000s  
sys     0m2.125s
```

读文件测试

读文件时忽略 cache 的影响。这是可以指定参数 iflag=direct, nonblock。

```
time dd if=/mnt/rootfs/home/tempfile of=/dev/null bs=1M count=100 iflag=direct,nonblock  
  
100+0 records in  
  
100+0 records out  
  
104857600 bytes (105 MB, 100 MiB) copied, 4.46744 s, 23.5 MB/s  
  
real    0m4.495s  
user    0m0.000s  
sys     0m0.059s
```

4.2.8. 显示

ECB10-135A5M5M-I 支持 HDMI 和 LCD 两种显示方案:

HDMI 显示: 由于 STM32MP135CPU 没有 HDMI 相关的控制器, 所以 HDMI 是通过显示转换芯片 SII9022A 将 RGB 转换为 HDMI 信号进行输出, 支持最大分辨率 1280 x 720@60fps。

LCD 显示: LCD 测试部分将演示对 Linux 的 drm 设备操作, 实现液晶输出显示 RGB 颜色和颜色合成测试。LCD 采用 RGB888 的显示模式。

在 uboot 阶段通过选择对应的设备树文件来切换显示方案。

4.2.8.1. HDMI 显示

如果用户使用过程需要 HDMI 显示功能, 启动开发板, 在 uboot 启动阶段在启动模式下选择 “3”, 然后按 “Enter”键即可。

```
1:      stm32mp135-ebyte-produce
2:      stm32mp135-discovery-lcd
3:      stm32mp135-discovery-hdmi
Enter choice: 3
3:      stm32mp135-discovery- hdmi
```

4.2.8.2. LCD 显示

如果用户使用过程需要 LCD 显示功能, 启动开发板, 在 uboot 启动 log 阶段在启动式下选择 “2”, 然后按 “Enter”键即可。

```
1:      stm32mp135-ebyte-produce
2:      stm32mp135-discovery-lcd
3:      stm32mp135-discovery-hdmi
Enter choice: 2
2:      stm32mp135-discovery- lcd
```

4.2.9. 触摸

evtest 命令测试

终端执行 “evtest” 进入测试界面。选择测试外设为触摸屏, 这里默认为输入中断 0, 测试界面选择 “0” 按下回车即可开始测试。

```
evtest
```

```
No device specified, trying to scan all of /dev/input/event*
```

```
Available devices:
```

```
/dev/input/event0:      Goodix Capacitive TouchScreen
```

```
/dev/input/event1:      wake_up
```

```
Select the device event number [0-1]: 0
```

```
Input driver version is 1.0.1
```

```
Input device ID: bus 0x18 vendor 0x416 product 0x38f version 0x1060
```

```
Input device name: "Goodix Capacitive TouchScreen"
```

```
Supported events:
```

```
Event type 0 (EV_SYN)
```

```
Event type 1 (EV_KEY)
```

```
Event code 59 (KEY_F1)
```

```
Event code 60 (KEY_F2)
```

```
Event code 61 (KEY_F3)
```

```
Event code 62 (KEY_F4)
```

```
Event code 63 (KEY_F5)
```

```
Event code 64 (KEY_F6)
```

```
Event code 125 (KEY_LEFTMETA)
```

```
Event code 330 (BTN_TOUCH)
```

```
Event type 3 (EV_ABS)
```

```
Event code 0 (ABS_X)
```

```
Value      0
```

```
Min        0
```

```
Max       1023
```

```
Event code 1 (ABS_Y)
```

```
Value      0
```

```
Min        0
```

```
Max       599
```

```
Event code 47 (ABS_MT_SLOT)
```

```
Value      0
```

```
Min        0
```


Min 0

Max 1023

Event code 54 (ABS_MT_POSITION_Y)

Value 0

Min 0

Max 599

Event code 57 (ABS_MT_TRACKING_ID)

Value 0

Min 0

Max 65535

Properties:

Property type 1 (INPUT_PROP_DIRECT)

Testing ... (interrupt to exit)

Event: time 946684926.419206, type 3 (EV_ABS), code 57 (ABS_MT_TRACKING_ID), value 0

Event: time 946684926.419206, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 248

Event: time 946684926.419206, type 3 (EV_ABS), code 54 (ABS_MT_POSITION_Y), value 313

Event: time 946684926.419206, type 3 (EV_ABS), code 48 (ABS_MT_TOUCH_MAJOR), value 50

Event: time 946684926.419206, type 3 (EV_ABS), code 50 (ABS_MT_WIDTH_MAJOR), value 50

Event: time 946684926.419206, type 1 (EV_KEY), code 330 (BTN_TOUCH), value 1

Event: time 946684926.419206, type 3 (EV_ABS), code 0 (ABS_X), value 248

Event: time 946684926.419206, type 3 (EV_ABS), code 1 (ABS_Y), value 313

Event: time 946684926.419206, ----- SYN_REPORT -----

Event: time 946684926.472114, type 3 (EV_ABS), code 54 (ABS_MT_POSITION_Y), value 314

Event: time 946684926.472114, type 3 (EV_ABS), code 1 (ABS_Y), value 314

由上面可知, 主要显示坐标值、键值, 具体信息如下:

EV_SYN: 同步事件

EV_KEY: 按键事件, 如 BTN_TOUCH 表示是触摸按键

EV_ABS: 绝对坐标, 如触摸屏上报的坐标

BTN_TOUCH: 触摸按键

ABS_MT_TRACKING_ID 表示采集信息开始, 后面一个 ABS_MT_TRACKING_ID 表示采集信息结束

单点触摸信息是以 ABS 承载并按一定顺序发送,如:

ABS_X: 是相对于屏幕绝对坐标 X

ABS_Y:是相对于屏幕绝对坐标 Y而多点触摸信息则是以 ABS_MT 承载并按一定顺序发送, 如:

ABS_MT_POSITION_X: 表示屏幕接触面的中心点 x 坐标位置.

ABS_MT_POSITION_Y: 表示屏幕接触面的中心点 Y 坐标位置

4.3. 网络接口

4.3.1. Ethernet

使用 net-tools 工具包中的 ifconfig 对网络进行手动配置, 首先通过通过 ifconfig 命令查看网络设备信息如下。

```
ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.0.250  netmask 255.255.255.0  broadcast 192.168.0.255
    ether 00:04:9f:04:d2:35  txqueuelen 1000  (Ethernet)
    RX packets 3524  bytes 3681632 (3.6 MB)
    RX errors 0  dropped 73  overruns 0  frame 0
    TX packets 1262  bytes 218100 (218.1 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
    device interrupt 44  base 0xc000
```

下面介绍给 eth0 手动配置 IP 地址 192.168.0.250 的方法, 命令如下:

```
ifconfig eth0 192.168.0.250 netmask 255.255.255.0 up
```

配置好以太网连接之后就可以使用 PING 对网络连接进行简单的测试, ping 同一网段的虚拟机测试。

```
ping 192.168.0.130  
PING 192.168.0.130 (192.168.0.130) 56(84) bytes of data.  
64 bytes from 192.168.0.130: icmp_seq=1 ttl=64 time=1.53 ms  
64 bytes from 192.168.0.130: icmp_seq=2 ttl=64 time=1.34 ms  
64 bytes from 192.168.0.130: icmp_seq=3 ttl=64 time=1.37 ms  
64 bytes from 192.168.0.130: icmp_seq=4 ttl=64 time=1.30 ms
```

4.3.2. WIFI

本节主要介绍 Linux 下 Wi-Fi 的配置和使用,通常 Wi-Fi 模块可以支持两种工作模式,分别是 STA 模式和 AP 模式,有些外设模块还支持 STA 和 AP 模式同时工作。STA 模式允许开发板连接外部 Wi-Fi 热点,AP 模式将开发板变成 Wi-Fi 热点,供其它外部设备连接。

开发板板载 AP6212 Wi-Fi 和 Bluetooth 二合一模块,当前不支持 STA 和 AP 同时工作,AP6212 Wi-Fi 模块对应的驱动为 brcmfmac,使用前加载驱动。

```
modprobe brcmfmac  
[ 2250.502879] cfg80211: Loading compiled-in X.509 certificates for regulatory database  
[ 2250.650222] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'  
[ 2250.728001] brcmfmac: brcmf_fw_alloc_request: using brcm/brcmfmac43430-sdio for  
chip BCM43430/1  
brcmfmac: brcmf_c_preinit_dcmds: Firmware: BCM43430/1 wl0: Sep 11 2018 09:22:09  
version 7.45.98.65 (r707797 CY) FWID 01-b54727f
```

查看加载的驱动

```
lsmod | grep brcm  
  
brcmfmac                225280  0  
cfg80211                647168  1 brcmfmac  
brcmutil                16384   1 brcmfmac
```

驱动加载的过程中会将位于/lib/firmware/brcm 的 Wi-Fi 固件加载到模块内部。WiFi 模块驱动加载成功之后生成 Wi-Fi 的网络节点 wlan0, 如下所示

```
ifconfig wlan0  
  
wlan0: flags=4098<BROADCAST,MULTICAST>  mtu 1500  
        ether 50:41:1c:b6:9c:32  txqueuelen 1000  (Ethernet)  
        RX packets 0  bytes 0 (0.0 B)  
        RX errors 0  dropped 0  overruns 0  frame 0  
        TX packets 0  bytes 0 (0.0 B)  
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4.3.2.1. STA 模式连接 WIFI 热点

下面尝试手动连接附近的 Wi-Fi 热点“TST-2.4G”, 这是一个采用 WPA2 加密方式的 Wi-Fi 热点, 密码为 12345678。

```
wpa_passphrase TST-2.4G TST12345678 >>> /etc/wpa_supplicant.conf
```

连接 wifi 热点

```
wpa_supplicant -Dnl80211 -B -c /etc/wpa_supplicant.conf -i wlan0
```

使用-B 参数可以在后台运行连接 WiFi, 如果去掉可以查看连接过程。

连接成功后需要分配 IP 地址, 使用 udphc 分配 IP 地址

```
udhcpc -i wlan0
udhcpc: started, v1.30.1
udhcpc: sending discover
udhcpc: sending select for 172.20.10.5
udhcpc: lease of 172.20.10.5 obtained, lease time 86400
ip: RTNETLINK answers: File exists
```

查看连接状态

```
iw wlan0 link
Connected to 4a:fe:e2:56:e0:36 (on wlan0)

    SSID: iPhone
    freq: 2437
    RX: 2652 bytes (17 packets)
    TX: 1806 bytes (13 packets)
    signal: -54 dBm
    tx bitrate: 24.0 MBit/s

    bss flags:          short-slot-time
    dtim period:       1
    beacon int:        100
```

扫描 WIFI 热点

```
iw dev wlan0 scan | grep SSID

SSID: iPhone

SSID: zhendu

SSID: TP-LINK_\xef\xbc\x88\xe7\xa1\xac\xe4\xbb\xb6\xe7\xbb\x84)

SSID: \x00\x00\x00\x00\x00\x00

SSID: TP-LINK_6C35

SSID: B05_2.4G_\xe6\x89\x93\xe5\x8d\xb0\xe6\x9c\xba

SSID: MERCURY_2.4G_D02D

SSID: KingOfSoftware

SSID: \xe5\xa4\xa7\xe5\x8e\x85\xe7\xbd\x91\xe7\xbb\x9cA-5G
```

4.3.2.2. AP 模式开启 WIFI 热点

Hostapd 是一个带加密功能的无线接入点程序，是 Linux 操作系统上构件无线接入点的一个比较方便的工具，支持 IEEE 802.11 协议和 IEEE 802.1X/WPA/WPA2/EAP/RADIUS 加密。板子作为 WiFi 热点，需要为每一个接入该热点的终端（例如手机）分配 IP，路由等网络参数。如创建 SSID 为 EBYTE，PASSWD 为 12345678 的无线 wifi 热点。下面先以手动方式进行配置：

当使用 AP 模式时，需要为激活 wlan0 并配置一个静态 IP 地址，这里配置一个默认的 IP 地址：192.168.10.1。

```
ifconfig wlan0 192.168.10.1 up
```

在前面我们有说明当前 Wi-Fi 模块不支持 STA 和 AP 模式同时工作，所以这里需要清除 STA 模式的配置，如下：

```
killall udhcpd  
killall wpa_supplicant
```

wlan0 工作在 AP 模式, 当其它外部设备连接这个 AP 热点的时候, 它需要通过 wlan0 为其它外部设备动态分配 IP 地址, 所以需要使用 wlan0 来运行 DHCP 服务程序 udhcpd。udhcpd 对应的配置文件为/etc/udhcpd.conf, 内容如下:

```
# File: /etc/udhcpd.conf  
  
# the start and end of the IP lease block  
start 192.168.10.10  
end 192.168.10.254  
  
# the interface that udhcpd will use  
interface wlan0  
opt    dns      8.8.8.8  
option subnet  255.255.255.0  
opt    router   192.168.10.1  
option domain  local  
option lease   864000
```

配置 AP 模式最关键的一步当然是启动 hostapd 服务。启动服务之前需要通过 /etc/hostapd.conf 配置 AP 模式的 ssid, password, 加密算法, 驱动类型, 工作模式等, 完整的参数配置说明参见: <http://w1.fi/cgi/hostap/plain/hostapd/hostapd.conf>, 下面是针对当前硬件的/etc/hostapd.conf 配置, 内容如下:


```
# File: /etc/hostapd.conf

interface=wlan0

driver=nl80211

# mode Wi-Fi (a = IEEE 802.11a, b = IEEE 802.11b, g = IEEE 802.11g)

hw_mode=g

ssid=EBYTE

channel=7

wmm_enabled=0

macaddr_acl=0

# Wi-Fi closed, need an authentication

auth_algs=1

ignore_broadcast_ssid=0

wpa=2

wpa_passphrase=12345678

wpa_key_mgmt=WPA-PSK

wpa_pairwise=TKIP

rsn_pairwise=CCMP
```

配置文件准备好之后，执行下面的命令启动 `hostapd` 服务：

```
hostapd -B /etc/hostapd.conf

Configuration file: /etc/hostapd.conf

wlan0: Could not connect to kernel driver

Using interface wlan0 with hwaddr 50:41:1c:b6:9c:32 and ssid "EBYTE"

wlan0: interface state UNINITIALIZED->ENABLED

wlan0: AP-ENABLED
```

如上面 log 显示，即可以正常使用热点服务了。如果以太网卡 `eth0` 已经连接 Internet，那么通过下面的配置进行 IP 转发，那连接到 EBYTE 的外部设备也可以连接 Internet 了。

```
echo "1" > /proc/sys/net/ipv4/ip_forward  
iptables -t nat -A POSTROUTING -s 192.168.10.1/24 -o eth0 -j MASQUERADE
```

可以用手机连接上述热点进行上网测试。

5. 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

6. 修订说明

修订说明表

版本	修改内容	修改时间	编制	校对	审批
V1.0	初稿	24-09-19	HSL	WYQ	WFX
V1.1	添加第三章快速开始部分	24-12-14	HSL	WYQ	WFX

7. 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
EBYTE Chengdu Ebyte Electronic Technology Co.,Ltd.