第1章　前言

我还没有录完驱动入门、应用程序入门，在录完这 2 部分入门知识之后，我才开始讲解项目开发。

但是有很多同学急需一个能上手的项目，有些是用来刷经验，有些是在工作中就要用到。所以我先写一下文档，这个文档里，不讲代码，只讲操作；看得懂的人就看，看不懂的人就等视频。

本文档讲解一个实际的项目：电子产品量产测试与烧写工具。这是一套软件，用在我们的实际生产中，有如下特点：

① 简单易用：

把这套软件烧写在 SD 卡上，插到 IMX6ULL 板子里并启动，它就会自动开始测试各个模块、烧写 EMMC 系统。

工人只要按照说明接入几个模块，就可以完成整个测试过程。

测试结果一目了然：等 LCD 上所有模块的图标都变绿时，就表示测试通过。

② 软件可配置、易扩展：

通过配置文件添加测试项，可以添加不限个数的测试项。

每个测试项有自己的测试程序，测试通过后把结果发送给 GUI 即可。各个测试程序互不影响。

③ 纯 C 语言编程

下图是这个工具的界面，它可以一边测试一边烧写：

上图中的 led、speaker 按钮，可以点击：

① 当你看到 LED 闪烁时，就点击 led 按钮，它变成绿色表示测试通过；

② 当你从耳机里听到声音时，就点击 speaker 按钮，它变成绿色表示测试通过。

其他按钮无法点击，接上对应模块后会自动测试，测试通过时图标就会变绿。

上图中的蓝色按钮表示烧写 EMMC 的进度，烧写成功后它也会变绿。

LCD 上所有图标都变绿时，就表示测试、烧写全部完成；某项保持红色的话，就表示对应模块测试失败。

第2章　文件获取与制作

1.1 下载文件

使用 GIT 下载代码，如下所示：

```
$ git clone https://e.coding.net/weidongshan/01_all_series_quickstart.git
```

执行上述命令后，可以得到一个“01_all_series_quickstart”目录，本文的文件位于如下目录中：

1.2 文件说明

上图中，

app 目录下是源代码；

image 下是映像文件的下载方法，由于 GIT 对文件大小有限制，而映像文件太大了，所以另存在网盘中；

scripts_and_executable 目录下是配置文件、脚本、编译好的可执行程序；

tools 目录下是一些运行于 Ubuntu 的工具。

1.2.1 app目录

这个目录下有如下内容：

① test_gui：

GUI 界面程序，它用于显示测试、烧写界面。

② detect_dev：

这只是一个简单的发送网络信息的程序，比如要向 GUI 程序发送信息时，可以执行以下命令，它表示 AP3216C 模块测试通过了：

```
detect_dev 127.0.0.1 "ap3216c ok"
```

③ dd.c：

这是修改过的 dd 命令，可以打印 dd 执行的进度，我们使用 dd 命令烧写 EMMC，从它的输出获得烧写进度，然后把进度发送给 GUI。

④ serial_test.c：

它用来测试串口，向串口发送“uname”命令，如果能读回“linux”字符，就表示串口正常。

⑤ 其他库文件：

freetype-2.4.10.tar.bz2：矢量字符库

tslib-1.21.tar.bz2：触摸屏库

1.2.2 scripts_and_executable目录

这个目录有如下内容：

① etc_test_gui 子目录：

里面的内容要放到板子的/etc/test_gui 目录，里面有 2 个文件：配置文件 gui.conf、字体文件 simsun.ttc。

test_gui 程序根据配置文件 gui.conf 来生成界面，配置文件示例如下：

```
# name   can_be_pressed command
led         1
speaker     1
record      0
key1        0
key2        0
ap3216c     0
icm20608    0
RS485toCAN  0
CANtoRS485  0
4G          0
usb         0
otg_device  0
otg_host    0
serial      0
wifi        0
net0        0
net1        0
burn        0
ALL         0      test_sleep_key.sh
```

第 1 列是测试项的名字，这会在 LCD 上显示出来。

第 2 列表示该测试项能否被点击：1 表示能点击，0 表示不能点击。对于 led 这样的测试项必须通过人眼观察，如果它能闪烁就用手点击图标把它变为绿色。对于 wifi 这样的模块，测试程序会自动改变图标颜色，不允许手工点击图标。

第 3 项是对应的是命令，这项是可以省略的。如果提供了“命令”，当某个测试项的状态发生变化时，test_gui 会调用测试项对应的命令。比如如果有这项：

```
led  1  led.sh
```

当我们点击 led 图标让它变到绿色时，test_gui 会调用“led.sh ok”；当我们再次点击 led 图标让它变为红色时，test_gui 会调用“led.sh err”。

在配置文件里最后一个测试项是“ALL”，当屏幕上除了“ALL”图标之外所有的模块都测试通过后，就会调用“test_sleep_key.sh processing”，表示正在处理最后一项，在这里我们是让系统进入休眠，然后测试唤醒按钮是否有效。如果唤醒按钮正常，那么 test_sleep_key.sh 会通知 test_gui 把“ALL”按钮也变成绿色，这表示全部测试通过。

② 6ull_test.sh：

这是所有模块的测试脚本，把它放到板子的/usr/bin 目录。它里面有很多 shell 函数，比如 test_ap3216c，它是用来测试 AP3216C 模块的，当通过 i2ctransfer 命令能写、读时就表示这个模块正常：

```
test_ap3216c() {
    i2ctransfer -f -y 0 w2@0x1e 0 0x3
    while :
    do
        i2ctransfer -f -y 0 w1@0x1e 0 r1 | grep -q "0x03"
        if [ $? -eq 0 ]
        then
            detect_dev 127.0.0.1 "ap3216c ok"
            echo "ap3216c ok" > $DEBUG_DEV
        else
            echo "ap3216c err" > $DEBUG_DEV
        fi
        sleep 1
    done
}
```

检测到 AP3216C 模块后，就可以通过以下命令通过 test_gui，test_gui 会把对应图标变为绿色：

```
detect_dev 127.0.0.1 "ap3216c ok"
```

③ detect_dev：

这是一个发送网络数据的可执行程序，它向 8765 端口发送数据；test_gui 会监听 8765 端口，根据这些数据来修改图标颜色。

用法示例如下，需要指定 IP，一般是“127.0.0.1”：

```
detect_dev 127.0.0.1 “ap3216c ok”
```

④ serial_test：

这是测试串口的可执行程序，把开发板的调试串口接到开发板的 USB Host，这样 serial_test 就可以向调试串口发送“uname”命令，如果能读到“linux”字符就表示开发板的调试串口正常。

⑤ test_gui：

显示测试界面的可执行程序。

⑥ test_sleep_key.sh：

一个脚本，用来测试系统的唤醒按钮。

1.2.3 tools目录

genimage 工具的源码及配置文件。

1.2.4 image目录

根据 image 目录里的说明文件，去网盘中可以下载到以下文件：

① sdcard.img：

可以使用 Win32DiskImager 把它直接烧到 TF 卡上，用 TF 卡启动 IMX6ULL 就可以测试系统、烧写 EMMC。

② sdcard_rootfs.tar.bz2：

根文件系统，里面的 root 目录下含有 emmc.img，这会烧录到 EMMC 上。

③ u-boot-dtb.imx：

这是 u-boot，制作 sdcard.img 时 genimage 的配置里指明要用到它。

1.3 编译程序

注意：你可以直接使用我们提供的 sdcard_rootfs.tar.bz2，在它的基础上修改根文件系统。这样就不需要自己去编译程序了。

编译之前都要先设置工具链，如果你使用的是我们的 IMX6ULL 配套开发环境，执行以下命令：

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
export PATH=$PATH:/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin
```

请注意上述命令中 PATH 后面的文字之间没有空格、没有换行。

可以执行下面命令验证是否设置成功：

```
$ arm-linux-gnueabihf-gcc -v    // 这条命令有输出的话就表示成功
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
COLLECT_LTO_WRAPPER=/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/../libexec/gcc/arm-linux-gnueabihf/6.2.1/lto-wrapper
Target: arm-linux-gnueabihf
```

有些 Ubuntu 可能需要先安装一些工具：

```
$ sudo apt-get  update
$ sudo apt-get install automake libtool
```

1.3.1 编译test_gui

test_gui 程序要用到 tslib、freetype，所以要先编译这 2 个依赖。

① 编译 tslib：

```
$ tar xjf tslib-1.21.tar.bz2
```

```
$ cd tslib-1.21
$ ./autogen.sh

$ mkdir tmp
$ ./configure --host=arm-linux-gnueabihf --prefix=$(pwd)/tmp
$ make
$ make install
```

编译出来的头文件应该放入：

```
/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include
$ cd tmp/include
$ cp *  /home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include
```

编译出来的库文件应该放入：

```
/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib
$ cd tmp/lib
$ cp * -rfd /home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib
```

② 编译 freetype：

```
$ tar xjf freetype-2.4.10.tar.bz2
$ cd freetype-2.4.10/
$ ./configure --host=arm-linux-gnueabihf
$ make
$ make DESTDIR=$PWD/tmp install
```

编译出来的头文件应该放入：

```
/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include
$ cd tmp/usr/local/include/
$ cp * -rf  /home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include
$ cd /home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include
$ mv freetype2/freetype .
```

编译出来的库文件应该放入：

```
/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib
```

```
$ cd tmp/usr/local/lib/
$ cp * -rfd /home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib
```

③ 编译 test_gui：

```
$ cd test_gui/
$ make
```

1.3.2 编译detect_dev

```
$ cd detect_dev/
$ arm-linux-gnueabihf-gcc -o detect_dev detect_dev.c
```

1.3.3 编译serial_test

```
$ arm-linux-gnueabihf-gcc -o serial_test  serial_test.c
```

1.3.4 编译dd

dd.c 来自 coreutils，你下载 coreutils 的源码后替换 dd.c 就可以去编译了。
为了省事，我不再去编译，你直接使用 scripts_and_executable\coreutils 就可以。

第3章　制作映像文件

1.1 我们要做什么？

我们要制作一个 sdcard.img，它里面含有：emmc.img、test_gui、6ull_test.sh 及必需的库文件。

把 sdcard.img 烧录到 SD 卡后，使用 SD 卡启动 IMX6ULL 开发板，它就会执行测试、烧写：

① 运行 test_gui、6ull_test.sh：

6ull_test.sh 会自动测试众多模块，把测试结果发送给 test_gui；test_gui 在 LCD 上显示结果。

② 烧写系统：

6ull_test.sh 还会把 emmc.img 烧写到 EMMC Flash，同时发送烧写进度给 test_gui；test_gui 在 LCD 上显示进度。

如果你要体验这套系统，直接烧写我们提供的 sdcard.img 就可以。

1.2 准备工具 genimage

如果你使用的是我们提供的 buildroot 系统，执行以下命令，就可以生成 emmc.img、sdcard.img，你可以在 Buidlroot 系统里修改里面的内容：

```
book@100ask:~/100ask_imx6ull-sdk$ cd Buildroot_2019.02
book@100ask:~/100ask_imx6ull-sdk/Buildroot_2019.02$ make clean
book@100ask:~/100ask_imx6ull-sdk/Buildroot_2019.02$ make 100ask_imx6ull_defconfig
book@100ask:~/100ask_imx6ull-sdk/Buildroot_2019.02$ make all
```

但是很多人需要定制自己的产品，他们有自己的 rootfs.tar.bz2，所以需要手工生成 emmc.img。

我们需要一个工具：genimage。

```
$ cd tools
$ tar xJf genimage-10.tar.xz
$ ./configure
会提示：
configure: error: Package requirements (libconfuse) were not met:
No package 'libconfuse' found
$ sudo apt-get install libconfuse-dev
$ ./configure
$ make
$ sudo make install
```

1.3 准备配置文件

在 genimage 的配置文件里，指定了怎么分区、每一个分区使用哪种格式的文件系统、每一个分区放什么内容。

我们有 2 个配置文件：emmc_genimage.cfg、sdcard_genimage.cfg。

emmc_genimage.cfg 的内容如下。

它的意思是制作出来的 emmc.img 中含有 3 个分区：u-boot、arduino、rootfs。

其中 u-boot 分区不在分区表里，即你只可以看到后面 2 个分区。u-boot 分区从偏移地址 1024 开始，内容是 u-boot-dtb.img 文件。

arduino 分区我们用不到，之所以放在这里只是为了兼容我们的 arduino 教程。

rootfs 分区有 2048M，里面的内容来自 emmc_rootfs.ext4。

```
image emmc.img {
  hdimage {
  }

  partition u-boot {
    in-partition-table = "no"
    image = "u-boot-dtb.imx"
    offset = 1024
  }
  partition arduino {
    partition-type = 0xC
    size = 50M
    offset = 10M
  }
  partition rootfs {
    partition-type = 0x83
    image = "emmc_rootfs.ext4"
    size = 2048M
  }
}
```

sdcard_genimage.cfg 的内容如下。

它的意思是制作出来的 sdcard.img 中含有 3 个分区：u-boot、arduino、rootfs。

其中 u-boot 分区不在分区表里，即你只可以看到后面 2 个分区。u-boot 分区从偏移地址 1024 开始，内容是 u-boot-dtb.img 文件。

arduino 分区我们用不到，之所以放在这里只是为了兼容我们的 arduino 教程。

rootfs 分区有 4096M，里面的内容来自 sdcard_rootfs.ext4。

```
image sdcard.img {
  hdimage {
  }
```

```
  partition u-boot {
    in-partition-table = "no"
    image = "u-boot-dtb.imx"
    offset = 1024
  }
  partition arduino {
    partition-type = 0xC
    size = 10M
    }
  partition rootfs {
    partition-type = 0x83
    image = "sdcard_rootfs.ext4"
    size = 4096M
  }
}
```

1.4 制作 emmc.img

假设你已经有了一个 emmc_rootfs.tar.gz(我们没有提供这个文件)，把它解压到某个目录里，你可以在里面添加内容：

```
$ mkdir tmp_emmc_rootfs
$ sudo tar xzf emmc_rootfs.tar.gz -C tmp_emmc_rootfs/
```

使用 tmp_emmc_rootfs 制作 ext4 映像文件，下面的命令制作 650M 的 ext4 映像文件：

```
$ dd if=/dev/zero of=emmc_rootfs.ext4 bs=1M count=650
$ mkfs.ext4  -F -E nodiscard -O ^metadata_csum,^64bit  emmc_rootfs.ext4
$ mkdir  tmp
$ sudo mount -t ext4 emmc_rootfs.ext4 tmp
$ sudo cp -rfd tmp_emmc_rootfs/* tmp/
$ sudo umount tmp
```

当前目录下要有这 3 个文件：

① emmc_genimage.cfg：这个配置指明要使用下面 2 个文件来制作 emmc.img

② emmc_rootfs.ext4

③ u-boot-dtb.imx

就可以使用 genimage 生成 emmc.img：

```
mkdir root   // 没什么用，genimage 要用到它
genimage  --inputpath ./  --outputpath ./ --config emmc_genimage.cfg
```

1.5 制作 sdcard.img

假设你已经有了一个 sdcard_rootfs.tar.bz2，把它解压到某个目录里，你可以在里面添加内容：

```
$ mkdir tmp_sdcard_rootfs
$ sudo tar xjf sdcard_rootfs.tar.bz2 -C tmp_sdcard_rootfs/
```

如果你想修改，请留意以下目录中的文件(以下的根目录，指的是 tmp_sdcard_rootfs 的根目录)：

```
/etc/init.d/S04test_gui     // /etc/init.d 下其他使用 LCD 的程序要删掉
/etc/test_gui/gui.conf
/etc/test_gui/simsun.ttc
/usr/bin/test_gui
/usr/bin/6ull_test.sh
/usr/bin/detect_dev
/usr/bin/serial_test
/usr/bin/coreutils     // 里面含有我们修改的 dd 命令
/lib/libts*  // 来自上面编译的 tslib-1.21/tmp/lib
/lib/ts       // 来自上面编译的 tslib-1.21/tmp/lib/ts/
/etc/ts.conf  // tslib-1.21/tmp/etc/ts.conf
              //  注意 ts.conf 的“# module_raw input”改为“module_raw input”，要顶格写
# ls lib/libfreetype.so*   // 来自上面编译的 freetype-2.4.10/tmp/usr/local/lib
lib/libfreetype.so        lib/libfreetype.so.6      lib/libfreetype.so.6.9.0

/usr/bin/coreutils  // 来自 scripts_and_executable\coreutils
/bin/dd             // 这是一个链接文件，指向../usr/bin/coreutils

/root/emmc.img        // 要烧写到 EMMC 去的文件
/root/u-boot-dtb.imx  // 要烧写到 EMMC 去的文件
```

使用 tmp_sdcard_rootfs 制作 ext4 映像文件，下面的命令制作 650M 的 ext4 映像文件：

```
$ dd if=/dev/zero of=sdcard_rootfs.ext4 bs=1M count=650
$ mkfs.ext4  -F -E nodiscard -O ^metadata_csum,^64bit  sdcard_rootfs.ext4
$ mkdir  tmp
$ sudo mount -t ext4 sdcard_rootfs.ext4 tmp
$ sudo cp -rfd tmp_sdcard_rootfs/* tmp/
$ sudo umount tmp
```

当前目录下要有这 3 个文件：

① sdcard_genimage.cfg：这个配置要使用下面 2 个文件来制作 sdcard.img

② sdcard_rootfs.ext4

③ u-boot-dtb.imx

就可以使用以下命令生成 sdcard.img：

```
mkdir root   // 没什么用，genimage 要用到它
genimage  --inputpath ./  --outputpath ./ --config sdcard_genimage.cfg
```

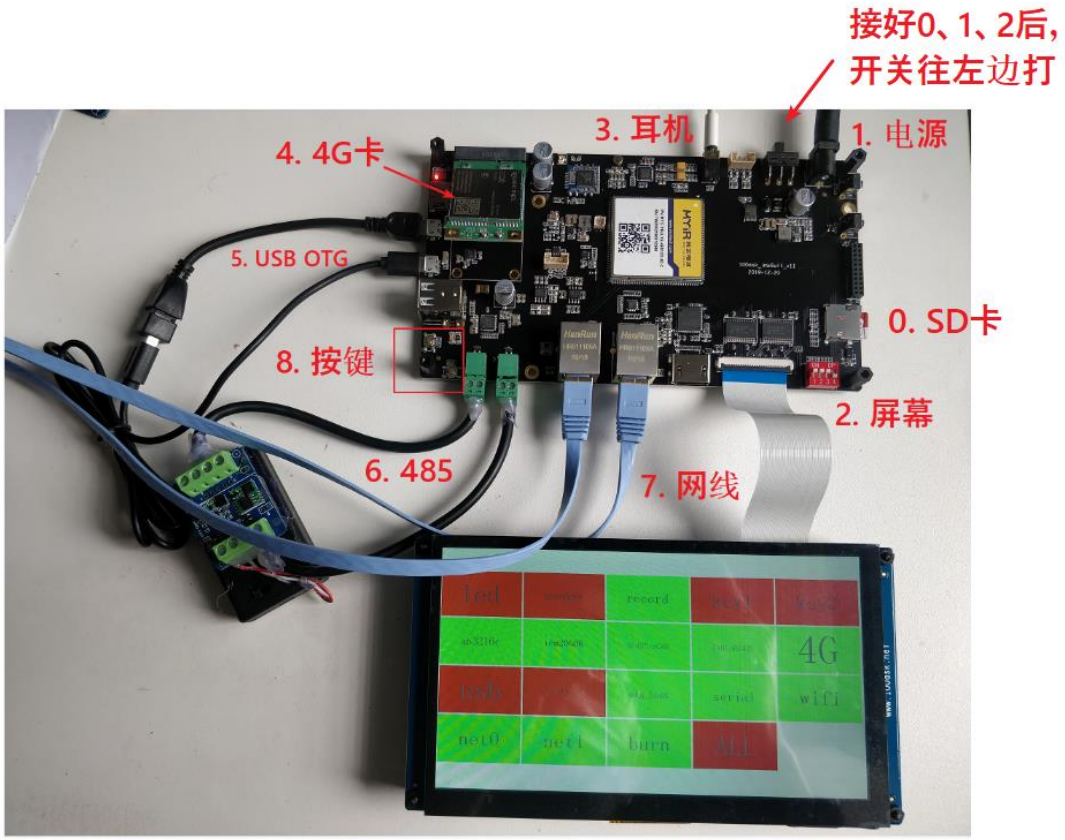
第4章　使用系统

这是一线工人就可以完成的操作，你首先要给他们提供一张烧录好了 sdard.img 的 TF 卡。

1.1 设置启动开关为 SD 卡启动

1、2、3 往上拨：

1.2 接线

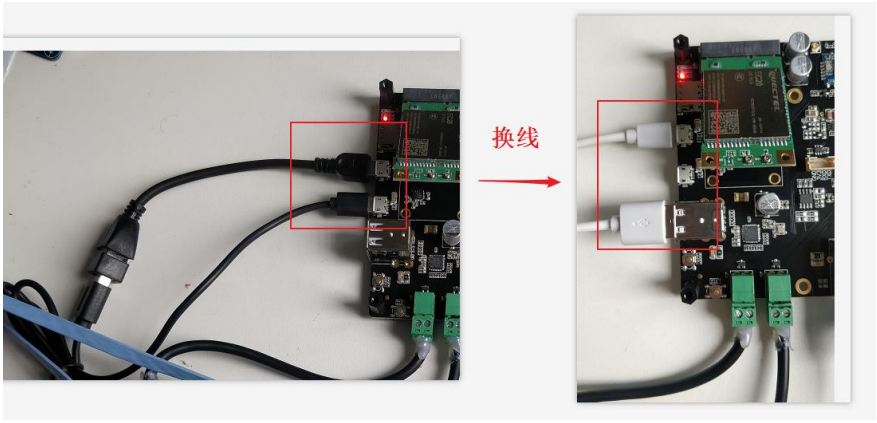


1.3 上电

接好 0、1、2 后，就可以上电：把电源开关打向左边。
其他的模块可以在开电后再插。
注意：一上电时就看红灯亮不亮，不亮的话就是板子有问题。

1.4 测试

1. 绿色 LED 是否闪，闪的话点屏幕上的“LED”按钮让它变亮
2. 听耳机，两个耳机都要有声音，都有声音的话点“speaker”按钮让它变亮
3. 点击右下角 2 个按钮，屏幕上的“key1”、“key2”会变亮
4. 等“otg_host”、“serial”这 2 个按钮变亮后，换 USB 线：观察“usb”、”otg_device”按钮变绿



5. 全部测试完后，屏幕会自动变黑，这时按右上角按钮，屏幕会变亮：

6. 屏幕上全部变亮后，测试通过，所有线取下。

1.5 恢复启动开关

只有 3 往上拨：

1.6 测 HDMI

接好 HDMI 屏，上电，屏幕有图像就通过。

第5章　阅读源码

test_gui 程序是整个项目的核心，等我录完驱动入门、应用入门后，就开始讲解它。
现在只贴下面这张结构图：