全志科技
Allwinner Technology

H3 项目

DragonBoard 使用说明书 V1.0

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2015-01-08		正式版本

目 录

1. 前言

1.1. 简介

Dragonboard 是基于 Linux BSP，集成 DirectFB 的图形化板卡测试系统。该系统旨在检测板卡能够在特定的环境下正常工作。

Dragonboard 测试系统的固件可以直接烧录到板卡上，同时也支持卡启动，减少对 pc 的依赖。

DragonBoard 测试系统的测试流程分为两个部分：自动测试和手动测试。自动测试包括内存、时钟、WIFI、等；手动测试包括 SD 卡、U 盘、HDMI、红外等。系统上电运行后会自动加载、运行用户启用的测试用例，并将结果显示到界面，用户可从中看到哪些测试项通过，哪些失败。

1.2. 目的

本文档主要向用户阐述在 H3 平台上 DragonBoard 测试系统的使用方法及其测试用例的编写方法。

1.3. 名称解释

1.4. 参考文档

2. Dragonboard 工具的使用

2.1. 使用入门

由于 DragonBoard 测试系统基于 Linux BSP，因此需要待测试平台的 Linux BSP，即 Linux 内核。

系统默认使用 Android 的 Linux 内核。DragonBoard 测试系统的源码位于 buildroot/target/dragonboard 目录下面，H8 SDK 默认包含了 dragonboard，编译内核时会自动编译这些源码，并生成所需的测试用例。

2.1.1. 配置测试项

DragonBoard 测试系统提供了一个灵活的配置脚本 test_config.fex，方便用户定制自己的测试项目，从而提高系统的效率。test_config.fex 能够完成以下基本工作：

1. 修改界面的语言和颜色
2. 启动或者禁用某个测试项目
3. 修改测试项目的参数
4. test_config.fex 位于 lichee\tools\pack\chips\sun8iw7p1\configs\xxxx\test_config.fex

测试项目的各项配置说明详见第 2.3 节配置文件

2.1.2. 编译 dragonboard

在 lichee 根目录下执行如下命令：

2.1.2.1. 配置环境变量

```
$ ./build.sh config
```

2.1.2.2. 选择芯片平台

```
Welcome to mkscript setup progress
All valid chips:
0. Sun8iw7p1
Choice: 0
```

2.1.2.3. 选择项目平台

```
All valid platforms:
0. android
1. dragonboard
2. Linux
Choice: 1
```

2.1.2.4. 选择配置项目

```
All available boards:
   1. dolphin-p1
Choice: 1
```

注意：完成以上配置后，系统会自动编译 dragonboard。

2.1.3. 打包固件

在完成编译后直接在 lichee 目录下执行：

```
./build.sh pack
```

2.1.4. 烧写固件

2.1.4.1. SD 卡启动

使用 PhoenixCard3.10 或更新的工具烧写制作卡启动固件，将制作好的启动卡，插入卡槽，上电，系统自动进入 dragonboard 测试系统。

2.1.4.2. 板卡启动

使用量产工具 PhoenixUSBpro 或升级工具 PhoenixSuit 将固件直接烧录到板卡上的 nand flash 或者是 emmc 上，烧录完成系统自动进入 dragonboard 测试系统。

2.1.5. 启动系统

如 2.1.4 描述

2.1.6. 测试流程

板卡上电之后系统自动启动，测试程序依照 test_config.fex 配置依次加载，并显示主界面。根据配置的不同，主界面的布局也将不同。当进入主界面之后，用户即看到各个测试项目的状态。这些测试项目被分为两组：自动测试项和手动测试项。自动测试项整个测试过程自动完成，无需用户干预，测试通

过测试项目描成蓝色，测试失败测试项目描成红色。手动测试项需要用户参与。其中带 wifi 测试与 mic 测试与 camera 测试的主界面如下图所示

2. 2. 配置文件

Dragonboard 板卡测试系统提供一个灵活的配置脚本 test_config.fex，位于 lichee\tools\pack\chips\sun8iw7p1\configs\xxxx\test_config.fex，方便用户定制自己的测试项目，从而提供系统效率。配置文件主要可以完成以下工作：

(1)、修改界面语言和颜色

(2)、启用或者禁用某个测试项目

(3)、修改测试项目的参数

2.2.1. 配置文件格式限制

该脚本使用 ini 文件格式，由段、键、值三者组成，通常一个段表示一个模块配置。目前要求该配置文件使用 UTF-8 编码。其他格式可能会导致未知错误。

2.2.2. 模块配置示例

测试模块配置示例：

```
[example]
display_name= "Example"
activated   = 1
program     = "example.sh"
category    = 0
run_type    = 1
```

- [example]

  [example]表示一个模块配置 example。

- display_name

当前测试模块显示到界面的名称，字符串类型，最多可容纳 64 个字节。如果为空，测试程序不会运行。如需显示双引号，使用下面语法：

```
display_name= string:"Example"
```

如果模块显示的名称需要不同的语言支持，修改 display_name 字段即可。

- **activated**

0：不测试该模块。

1：测试该模块。

如果用户的方案不需要测试当前模块，请将该项置 0；否则，请将该项置 1。

- **program**

模块的测试程序，字符串类型，最多可容纳 16 个字节。

- **category**

0：自动测试模块

1：手动测试模块

- **run_type**

0：等待当前模块的测试程序执行完毕再运行下一个模块的测试程序

1：不等待当前模块的测试程序执行完毕

一般为了提高整体的测试速度，对于耗时较长的测试程序建议填 0，反之填 1。注意，当 category = 1，即手动测试模块时，该项无效。因为手动测试模块会阻止其他测试模块加载，系统可能无法继续加载其他测试模块。

2.2.3. DRAM 配

内存测试配置。

```
[dram]
display_name= "内存"
activated   = 1
program     = "memtester.sh"
category    = 0
run_type    = 1
dram_size   = 2048
test_size   = 8
```

dram_size: 板卡上使用了多大的 dram，以 MB 为单位，在 dram 测试之初，程序会去检测板卡上 dram 的实际容量，如果小于配置容量，则说明贴片存在问题，测试不通过。

test_size：使用多大的容量测试 DRAM 的性能，单位是 MB，默认使用 8M。实际如果需要测出 DRAM 的性能，需要设定较大的容量，但是这样耗时较长，因此需要用户折中选择一个合适的值。8M 大概可以在 30 秒以内完成 DRAM 性能测试。

2.2.4. RTC 配置

时钟测试配置。

```
[rtc]
display_name= "时钟"
activated   = 1
```

```
program     = "rtctester.sh"
category    = 0
run_type    = 1
```

2.2.5. WIFI 配置

WIFI 测试配置。

```
[wifi]
display_name= "网络"
activated   = 0
program     = "wifitester.sh"
category    = 0
run_type    = 1
module_path = "/system/vendor/modules/8189eu.ko"
module_args = "fwpath=/system/vendor/modules/"
```

module_path：需要加载的模块全路径，DragonBoard 测试系统将模块文件放在/lib/modules/"kernel version"/目录下，并且创建了一个/system/vendor/modules，解决部分 wifi 驱动下载固件失败的问题。

module_args：模块的参数。

2.2.6. 蓝牙配置

```
[bluetooth]
display_name= "蓝牙"
activated   = 0
program     = "bttester.sh"
test_time   = 3
category    = 0
dst_bt      = "magicwu" ;需要连接的蓝牙设备名称
run_type    = 1
device_node = "/dev/ttyS2"
Module_path = "/system/vendor/modules/bcm20710a1.hcd"
```

2.2.7. 以太网配置

蓝牙测试配置

```
[ethernet]
display_name   = "以太网"
activated      = 0
program        = "ethtester.sh"
category       = 0
run_type       = 1
module_path    =
module_args    =
```

2.2.8. NAND 配置

闪存测试配置。

```
[nand]
display_name= "闪存"
activated   = 0
program     = "nandtester.sh"
category    = 0
run_type    = 0
test_size   = 128
```

test_size：测试 nand 读写的大小。

nand 方案请禁用 nand 测试。

2.2.9. EMMC 配置

emmc 测试配置

```
[emmc]
display_name= "emmc 闪存"
activated   = 0
program     = "emmctester.sh"
category    = 0
run_type    = 1
test_size   = 128
```

2.2.10. SDCARD 配置

SD 卡测试配置。

```
[mmc]
display_name= "SD 卡"
activated   = 1
program     = "mmctester.sh"
category    = 1
run_type    = 1
```

2.2.11. SATA 配置

```
[sata]
display_name= "sata"
activated   = 0
program     = "sata.sh"
category    = 0
run_type    = 1
```

2.2.12. USB_HOST1 配置

读出连接到 host1 的设备的 PID,VID 到 UI。

```
[Usb_HOST1]
display_name= "USB 主机 1"
activated   = 1
program     = "host1tester.sh"
category    = 1
run_type    = 1
```

2.2.13. USB_HOST2 配置

读出连接到 host2 的设备的 PID,VID 到 UI。

```
[Usb_HOST2]
display_name= "USB 主机 2"
activated   = 1
program     = "host2tester.sh"
category    = 1
run_type    = 1
```

2.2.14. USB_OTG 配置

读出连接到 otg 的设备的 PID,VID 到 UI。

```
[Usb_OTG]
display_name= "USB_OTG"
activated   = 1
program     = "otgtester.sh"
category    = 1
run_type    = 1
```

2.2.15. HDMI 配置

```
 [hdmi]
display_name= "HDMI"
activated   = 1
program     = "hdmitester"
category    = 1
run_type    = 1
sound_file  = "/dragonboard/data/test48000.pcm"
samplerate  = 48000
support_mode= 2
```

sound_file：向 HDMI 播放的音乐文件全路径。

samplerate：音乐的采样率。

support_mode：设置 HDMI 输出模块，默认优先输出 1080p，720p，如果两者都不支持，这个配置才会生效。支持以下输出模式。

```
; support_mode
;   0 : 480I
;   1 : 576I
;   2 : 480P
;   3 : 576P
;   4 : 720P 50HZ
;   5 : 720P 60HZ
;   6 : 1080I 50HZ
;   7 : 1080I 60HZ
;   8 : 1080P 24HZ
;   9 : 1080P 50HZ
;   10: 1080P 60HZ
```

2.2.16. TV 配置

TV 测试配置。

```
[tv]
display_name= "TV"
activated   = 1
program     = "tvtester"
category    = 1
run_type    = 1
sound_file  = "/dragonboard/data/test48000.pcm"
Samplerate  = 48000
```

2.2.17. SPDIF 配置

spdif 测试配置。

```
[spdif]
display_name= "SPDIF"
activated   = 0
program     = "spdiftester"
category    = 1
run_type    = 1
sound_file = "/dragonboard/data/test48000.pcm"
samplerate = 48000
module_count = 4
module1_path = "/system/vendor/modules/sunxi_spdif.ko"
module2_path = "/system/vendor/modules/sunxi_spdma.ko"
module3_path = "/system/vendor/modules/sndspdif.ko"
module4_path = "/system/vendor/modules/sunxi_sndspdif.ko"
```

2.2.18. IR 配置

红外测试配置。

```
[ir]
display_name= "红外"
activated   = 1
program     = "irtester"
category    = 1
run_type    = 1
module_path = "/system/vendor/modules/sunxi-ir-rx.ko"
```

2.2.19. 界面配置

```
;-------------------------------------------------------------------------------
; manual_menu_name
;   manual test case menu name, 32bytes.
; auto_menu_name
;   auto test case menu name, 32bytes.
; clear_button_name
;   clear screen button name, 8bytes.
; font_size
;   test case font display size, valid value: 20pixel(default), 24pixel.
; height_adjust
;   adjust height of manual test case and auto test case.
;
; Color Index
;   0: White
;   1: Yellow
;   2: Green
;   3: Cyan
;   4: Magenta
;   5: Red
;   6: Blue
;   7: Black
;   8: Beauty
; menu_bgcolor
;   The background color index of test case category menu.
; menu_fgcolor
;   The foreground color index of test case category menu.
; item_init_bgcolor
;   The background color index of test case item init status.
; item_init_fgcolor
;   The foreground color index of test case item init status.
; item_ok_bgcolor
;   The background color index of test case item OK status.
```

```
; item_ok_fgcolor
;   The foreground color index of test case item OK status.
; item_fail_bgcolor
;   The background color index of test case item fail status.
; item_fail_fgcolor
;   The foreground color index of test case item fail status.
;
; pass_str
;   The string display after test case display_name when test OK.
; fail_str
;   The string display after test case display_name when test Fail.
;------------------------------------------------------------------------------
[df_view]
tv_scale_factor     = 80  ;tv 显示界面缩放比例(80/100)
hdmi_scale_factor   = 95  ;hdmi 显示界面缩放比例(95/100)
manual_menu_name    = "手动测试项"
auto_menu_name      = "自动测试项"
clear_button_name   = "清屏"
font_size          = 20
menu_bgcolor        = 1
menu_fgcolor        = 7

item_init_bgcolor   = 7
item_init_fgcolor   = 0
item_ok_bgcolor     = 7
item_ok_fgcolor     = 6
item_fail_bgcolor   = 7
item_fail_fgcolor   = 5
tp_draw_color       = 0

;item_init_bgcolor   = 0
;item_init_fgcolor   = 7
;item_ok_bgcolor     = 0
;item_ok_fgcolor     = 6
;item_fail_bgcolor   = 0
;item_fail_fgcolor   = 5
;tp_draw_color       = 7

pass_str           = "通过"
fail_str           = "失败"
```

tv_scale_factor　: CVBS 输出缩放因子

现在市场上的很多电视都会对输入的视频裁边，导致视频内容不能完全的显示或者是电视四周有黑边。针对这种情况，dragonboard 会根据这个缩放因子对输出视频进行缩放。如: tv_scale-factor=80.

配置输出为 1280*720 分辨率输出时。实际的可显示的图像区域为:(1280*80%)*(720*80%)，也即是 1024*576。此缩放因子有效范围为 50~100，超出此范围的将自动认为不缩放，也就是 tv_scale-factor=100.
Hdmi_scale_factor : HDMI 显示初始缩放因子

同 tv_scale_factor 作用一样。

manual_menu_name：手动测试项的菜单显示内容。

auto_menu_name：自动测试项的菜单显示内容。

clear_button_name：清屏按键显示内容。

font_size：测试用例名称的字体大小。

menu_bgcolor：菜单的背景色，从 Color Index 中选择。

menu_fgcolor：菜单的前景色，从 Color Index 中选择。

item_init_bgcolor：测试项初始背景色，从 Color Index 中选择。

item_init_fgcolor：测试项初始前景色，从 Color Index 中选择。

item_ok_bgcolor：测试项通过背景色，从 Color Index 中选择。

item_ok_fgcolor：测试项通过前景色，从 Color Index 中选择。

item_fail_bgcolor：测试项失败背景色，从 Color Index 中选择。

item_fail_fgcolor：测试项失败前景色，从 Color Index 中选择。

tp_draw_color：触摸轨迹前景色，从Color Index中选择。为了能够清楚看到触摸轨迹，请选择和大部分区域的背景色相反的颜色。

pass_str：测试项通过提示语。

fail_str：测试项失败提示语。

Color Index

Color	Index
白色	0
黄色	1
绿色	2
青色	3
洋红	4
红色	5
蓝色	6
黑色	7

3. Dragonboard 测试用例的添加

3.1. 目录结构

DragonBoard 放在 buildroot/target 目录下，其目录结构大致如下：

```
|-- output/          # 输出目录
    |-- bin/         # 测试用例（程序）输出目录
|-- rootfs/          # 根目录
    |-- dragonboard/
        |-- bin/     # 测试用例（程序）rootfs 的输出目录
|-- src/                 # 源码目录
    |-- core/            # 系统核心模块
    |-- include/     # 系统公共头文件
    |-- lib/         # 系统公共库
    |-- testcases/   # 测试用例源码目录
        |-- example/ # 示例
            |-- example.c
            |-- Makefile
        |-- Makefile
    |-- view/            # UI
    |-- Makefile     # 顶层 Makefile
    |-- rule.mk      # 编译变量，include by Makefile
|-- sysroot/             # 交叉编译环境依赖目录
|-- build.sh             # 生成 rootfs.ext4
|-- README.txt
```

3.2. C 测试用例

3.2.1. 创建模块测试用例的目录

```
$ cd buildroot/target/dragonboard
$ mkdir src/testcases/example
$ cd src/testcases/example
```

3.2.2. 编写测试用例源码 example.c

参照 3.4.2 节。

3.2.3. 编写 Makefile

参照 3.4.2 节。

3.2.4. 让系统编译你的模块测试用例

```
在 testcases/Makefile 中添加
all:
    make -C example
```

3.3. Shell 测试用例

Shell 测试用例就是用 shell 脚本完成模块的测试任务。launcher 进程会向脚本传递 4 个参数，分别是：$0 是脚本的名称，$1 是 DragonBoard 系统版本号，目前没有使用，$2 是配置脚本的共享内存 id，$3 是当前测试用例的 id，返回结果的时候会使用。下面是一个简单的示例（memtester）：

```
#!/bin/sh

memtester 128K 1 > /dev/null

echo "$3 $?" >> /tmp/cmd_pipe
```

为了编译的时候能够将脚本拷贝到 output/bin 目录下，我们还需要一个 Makefile，如：

```
# define sources root directory before everything
SRC_ROOT := ../..

# include rule.mk
include $(SRC_ROOT)/rule.mk

.PHONY: all
all:
    cp memtester.sh $(BINDIR)/
```

然后在 testcases/Makefile 添加：

```
all:
    make -C memtester
```

更加详细的说明参照 4.3 节。

3.4. 测试用例示范

3.4.1. 与 core 交互

DragonBoard 系统的 core 进程负责接收测试用例返回的结果，目前这是与 core 唯一的交互方式，通过命名管道（/tmp/cmd_pipe）实现。core 进程会循环地一行一行地读取该管道，因此要求测试用例返回结果时必须按行的方式写入。

吐槽一下，建议测试用例的程序名称以 tester 或者 tester.sh 结尾。

3.4.2. C 测试用例

以下是一个简单的 C 测试用例：

```
/*
 * \file       example.c
 * \brief      just an example.
 *
 * \version    1.0.0
 * \date       2012 年 05 月 31 日
 * \author     James Deng <csjamesdeng@allwinnertech.com>
 *
 * Copyright (c) 2012 Allwinner Technology. All Rights Reserved.
 *
 */

/* include system header */
#include <string.h>

/* include "dragonboard_inc.h" */
#include "dragonboard_inc.h"

/* C entry.
 *
 * \param argc the number of arguments.
 * \param argv the arguments.
 *
 * DO NOT CHANGES THE NAME OF PARAMETERS, otherwise your program will get
 * a compile error if you are using INIT_CMD_PIPE macro.
 */
int main(int argc, char *argv[])
{
    char binary[16];

    /* init cmd pipe after local variable declaration */
    INIT_CMD_PIPE();

    strcpy(binary, argv[0]);
    db_msg("%s: here is an example.\n", binary);

    /* send OK to core if test OK, otherwise send FAIL
     * by using SEND_CMD_ PIPE_FAIL().
     */
    SEND_CMD_PIPE_OK();

    return 0;
```

```
}
```

几个需要注意的地方：

- 必须包含 dragonboard_inc.h，里面定义了与 core 交互的宏，建议用户直接使用；
- main 函数的参数个数和名字固定，不建议更改；
- main 函数的参数 argc = 4，表示 args 有 4 个值，args[0]是测试程序的名称，args[1]是 DragonBoard 系统版本号，目前没有使用，args[2]是配置脚本的共享内存 id，args[3]是当前测试用例的 id，返回结果的时候会使用；
- 建议使用 SEND_CMD_PIPE_OK()和 SEND_CMD_ PIPE_FAIL()返回结果。

编译链接：

```
# define sources root directory before everything
SRC_ROOT := ../..

# change compiler and linker option before you include rule.mk
#
# link to libscript.a when you need to fetch configuration
# from test_script
#
#CFLAGS := $(CFLAGS) -g
LDFLAGS := -lscript

# include rule.mk
include $(SRC_ROOT)/rule.mk

# define objects collection variable
example_objs = example.o

# add your target(s) to all
.PHONY: all
all: example

# define you target, the target will be output to dragonboard/output/bin
# directory
example: $(example_objs)
    $(LINK_MSG)
    $(LINKX)

# change 'example_objs' to your objects collection variable
$(example_objs): %.o: %.c
    $(COMPILE_MSG)
    $(COMPILEX)
```

上面是一个 Makefile 的示例，用户可以根据里面的注释修改。如果想让您的测试用例随 DragonBoard 系统一同编译，那么需要在 testcases/Makefile 中添加以下语句：

```
all:
    make -C example
```

将 example 换成你的模块测试用例所在目录。

3.4.3. Shell 测试用例

3.4.3.1. shell 脚本的参数

```
$0：脚本的名称；
$1：DragonBoard 系统版本号，目前没有使用；
$2：配置脚本的共享内存 id，目前没有找到在 shell 脚本使用共享内存的方法；
$3：是当前测试用例的 id，返回结果的时候会使用；
```

3.4.3.2. 返回结果

测试用例通过向命名管道（/tmp/cmd_pipe）写入一行字符的方式通知当前模块的测试结果，可以通过 echo 重定向实现：

```
echo "$3 $?" >> /tmp/cmd_pipe
```

$?是脚本中上一个命令的返回值，注意一般返回 0 表示 OK，非 0 表示 Fail。请使用“>>”，而不是“>”。“>>”表示把内容追加到/tmp/cmd_pipe 末尾，这样就不会影响其他模块的结果；“>”表示把内容写入/tmp/cmd_pipe，可能清除其他模块的结果。

3.4.3.3. 测试过程的调试信息

不建议将测试过程的调试信息输出到串口，可以重定向到/dev/null，例如：

```
memtester 128K 1 > /dev/null
```

4. FAQs

4.1. 如何添加新的方案？

在 lichee\tools\pack\chips\sun8iw7p1\configs\xxx\目录下添加新的方案目录，从标案 dolphin-p1 上拷贝 test_config.fex 到新的方案目录。针对新的方案，修改 test_config.fex 文件。一般需要修改启用或者禁用（activated）某些测试项目，修改模块文件的全路径，或者修改测试项目在界面上显示的语言（display_name）。下面列举几个经常修改的模块：

- **WIFI 模块**

  主要修改模块文件的全路径和参数。

- **IR 模块**

  主要修改模块文件的全路径。

4.2. 如何关闭内核的打印？

修改 lichee\tools\pack\chips\sun8iw7p1\configs\default\env.cfg 文件，找到下面这行：

```
loglevel=8
```

改成：

```
loglevel=4
```

5. Declaration