



MYD-YF13X_Linux 软件评估指南

文件状态： [] 草稿 [√] 正式发布	文件标识：	MYIR-MYD-YF13X-SW-EG-ZH-L5.15.67
	当前版本：	V1.0[文档]
	作 者：	Nico
	创建日期：	2023-05-06
	最近更新：	2023-05-10

版本历史

版本	作者	参与者	日期	备注
V1.0	Nico	licy	20230506	初始版本, u-boot2021.1 Linux Kernel 5.15.67, Yocto 4.1.



目 录

版本历史.....	- 2 -
目 录.....	- 3 -
1. 概述.....	- 6 -
1.1. 硬件资源.....	- 6 -
1.2. 软件资源.....	- 6 -
1.3. 文档资源.....	- 6 -
1.4. 环境准备.....	- 7 -
2. 核心资源.....	- 8 -
2.1. CPU.....	- 8 -
2.2. Memory.....	- 10 -
2.3. eMMC/Nand.....	- 12 -
2.3.1. eMMC 测试.....	- 13 -
2.3.2. Nand 测试.....	- 15 -
2.4. RTC.....	- 17 -
2.5. Watchdog.....	- 18 -
2.6. Power Management.....	- 20 -
3. 外设接口.....	- 22 -
3.1. GPIO.....	- 22 -
3.2. LED 灯.....	- 23 -
3.3. Key(按键).....	- 24 -
3.4. RS232.....	- 25 -
3.5. RS485.....	- 27 -
3.6. CAN.....	- 29 -
3.7. USB.....	- 31 -
3.8. Micro SD 卡.....	- 32 -
3.9. ADC.....	- 35 -
3.10. Display.....	- 36 -



3.11. Touch Panel	37 -
4. 网络接口	41 -
4.1. Ethernet	41 -
4.2. 4G 模块	44 -
5. 网络应用	47 -
5.1. PING	47 -
5.2. SSH	48 -
5.3. SCP	50 -
5.4. TFTP	51 -
5.5. DHCP	52 -
5.6. Iptables	54 -
5.7. Ethtool	56 -
5.8. iperf3	58 -
6. 图形系统	63 -
6.1. Wayland	64 -
6.2. QT	64 -
7. 多媒体应用	67 -
7.1. Camera	67 -
7.2. Audio	68 -
7.3. Video	68 -
8. 系统工具	69 -
8.1. 压缩解压工具	69 -
8.2. 文件系统工具	71 -
8.3. 磁盘管理工具	75 -
8.4. 进程管理工具	77 -
9. 开发支持	84 -
9.1. 开发语言	84 -
9.2. 数据库	88 -
9.3. Qt 应用程序本地化	88 -



10. 参考资料.....	- 93 -
附录一 联系我们.....	- 94 -
附录二 售后服务与技术支持.....	- 95 -



1. 概述

Linux 软件评估指南用于介绍在米尔的开发板上运行开源 Linux 系统下的核心资源与外设资源的测试步骤与评估方法。本文可作为前期评估指南使用，也可以作为通用系统开发的测试指导书使用。

1.1. 硬件资源

米尔电子的 MYD-YF13X 系列板卡，是基于 ST 公司的高性能嵌入式 ARM 处理器 ST M32MP135DAF7 模块开发的一套开发平台，此开发平台由核心板 MYC-YF13X 和底板 MYB-YF13X 两部分组成。有关硬件部分的详细配置参数请查看《MYD-YF13X 产品手册》。同时用户在评估测试过程中会用到一些配件，参见下面的列表。

表 1-1. 选配模块

配件	接口方式	说明及链接
摄像头	USB 接口	MY-CAM011B(200 万像素) https://www.myir-tech.com/product/my_cam011b.htm
液晶屏	RGB 接口	MY-TFT070CV2 (7 寸带电容触摸屏) http://www.myir-tech.com/product/my-tft070cv2.htm
RGB 转 HDMI 接口模块	RGB、HDMI 接口	MY-RGB2HDMI (RGB LCD 显示输出信号转换为 HDMI 输出的模块) https://www.myir-tech.com/product/MY-RGB2HDMI.htm

1.2. 软件资源

MYD-YF13X 系列开发板的 BSP 是基于 ST 官方开源社区版 Linux BSP 移植与修改而来，系统镜像采用 Yocto 项目进行构建。Bootloader, Kernel 以及文件系统各部分软件资源全部以源码的形式开放，具体内容请查看《MYD-YF13X SDK 发布说明》。

开发板在出厂时已经烧录了 myir-image-full 镜像，您只需要上电即可使用。

1.3. 文档资源

根据用户使用开发板的各个不同阶段，SDK 中包含了各阶段的文档，发布说明，评估指南，开发指南，应用笔记，常用问答等不同类别的文档和手册。具体的文档列表参见《MYD-YF13X SDK 发布说明》表 2-4 中的说明。



1.4. 环境准备

在开始评估开发板软件之前，您需要对开发板做一些必要的准备和配置一些基础环境，包括正确硬件接线，配置调试串口，设置启动等步骤。详细的步骤可以参照《MYD-Y F13X 快速入门指南》。

接下来的部分重点介绍如何对系统的硬件资源和接口以及软件功能进行评估和测试。主要借助一些 Linux 下常用的工具和命令，以及自己开发的应用进行测试。软件评估指南分为多个部分来描述，包括：核心资源，外设资源，网络应用，多媒体应用，开发支持应用，系统工具等几大类。后面的章节会针对各个部分做全方位的讲解，并详细描述各部分资源的具体评估方法和步骤。



2. 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。

2.1. CPU

MYD-YF13X 核心芯片是 STM32MP135DAF7。STM32MP135A 是基于高性能单核 Arm®Cortex®-A7 32 位 RISC 核心，工作频率为 1 GHz。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。

1). 查看 CPU 信息命令

读取系统中的 CPU 的提供商和参数信息，则可以通过 /proc/cpuinfo 文件得到。

```
root@myd-yf13x:~# 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         : 000C00103232511438303631
```

- 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 (Million Instructions Per Second)

2). 查看 CPU 使用率

```
root@myd-yf13x:~# top
MMem: 77408K used, 374352K free, 8832K shrd, 5388K buff, 29848K cached
CPU:  0% usr  9% sys  0% nic 90% idle  0% io  0% irq  0% sirq
Load average: 0.10 0.20 0.09 1/93 665
```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
665	656	root	R	2320	1%	9%	top
635	1	root	S	23180	5%	0%	/usr/sbin/iiod
572	1	root	S	16548	4%	0%	/lib/systemd/systemd-udev
598	1	root	S	14284	3%	0%	/lib/systemd/systemd-logind
651	649	root	S	9136	2%	0%	(sd-pam)
1	0	root	S	8008	2%	0%	{systemd} /sbin/init
649	1	root	S	7380	2%	0%	/lib/systemd/systemd --user
626	1	systemd-	S	6992	2%	0%	/lib/systemd/systemd-networkd
633	1	systemd-	S	6204	1%	0%	/lib/systemd/systemd-resolved
645	644	root	S	6024	1%	0%	systemd-userwork
646	644	root	S	6024	1%	0%	systemd-userwork
647	644	root	S	6024	1%	0%	systemd-userwork
644	1	root	S	5928	1%	0%	/lib/systemd/systemd-userdbd

- %usr: 表示用户空间程序的 cpu 使用率 (没有通过 nice 调度)
- %sys: 表示系统空间的 cpu 使用率, 主要是内核程序
- %nic: 表示用户空间且通过 nice 调度过的程序的 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

3). 获取 CPU 温度信息

CPU 内置温度传感器作为 CPU 温度采集, 可以很方便的获取 CPU 内部温度。

```
root@myd-yf13x:~# cat /sys/class/hwmon/hwmon0/temp1_input
39425
```



上面显示数字为千分之一度，除以 1000 就是当前温度值。

4). CPU 压力测试

CPU 的压力的测试方式有很多，可通过 bc 命令来计算圆周率方法来测试 CPU 在运算过程中的稳定性。

```
root@myd-yf13x:~# echo "scale=5000; 4*a(1)" | bc -l -q &
root@myd-yf13x:~# 3.141592653589793238462643383279502884197169399375
1058209749445923078164062862089986280348253421170679821480865132823
0664709384460955058223172535940812848111745028410270193852110555964
462294895493038196
.....
[1]+  Done                  echo "scale=5000; 4*a(1)" | bc -l -q
```

上述命令将在后台计算的 PI，并精确到小数点后 5000 位。计算过程需要一段时间。此时，我们可以通过 top 命令检查 CPU 利用率的变化，如下所示：

```
root@myd-yf13x:~# top
MMem: 77352K used, 374408K free, 8840K shrd, 5488K buff, 30632K cached
CPU: 99% usr  0% sys  0% nic  0% idle  0% io  0% irq  0% sirq
Load average: 0.68 0.23 0.13 2/89 691
  PID  PPID  USER    STAT  VSZ  %VSZ  %CPU  COMMAND
  687   656  root     R      2188  0% 100%  bc -l -q
  691   656  root     R      2408  1%  0%  top
    6     2  root    IW         0  0%  0%  [kworker/0:0-eve]
```

约 3 分钟后，PI 结果被计算出来。在此期间 CPU 使用率达到 100%，没有发生异常，说明 CPU 压力测试通过。还可以继续增加精确值，可进一步提高测试压力。

2.2. Memory

STM32MP135DAF7 提供了一个外部 SDRAM 接口，支持外部存储器高达 8Gbit 容量 (1GB)，16 位 LPDDR2/LPDDR3 或 DDR3/DDR3L，工作频率 533 MHz。

1). 查看内存信息

读取系统中的内存的参数信息，则可以通过 /proc/meminfo 文件得到。

```
root@myd-yf13x:~# cat /proc/meminfo
MemTotal:          451760 kB
```



```
MemFree:      374408 kB
MemAvailable: 399876 kB
Buffers:      5488 kB
Cached:       30632 kB
SwapCached:   0 kB
Active:       12784 kB
Inactive:     31628 kB
...
```

- MemTotal : 所有可用的 RAM 大小，物理内存减去预留位和内核使用
- MemFree : LowFree + HighFree
- Buffers : 用来给块设备做缓存的大小
- Cached : 文件的缓冲区大小
- SwapCached : 已经被交换出来的内存。与 I/O 相关
- Active : 经常（最近）被使用的内存
- Inactive : 最近不常使用的内存

2). 获取内存使用率

可使用 free 命令来读取内存的使用情况，-m 参数代表单位为 MByte。

```
root@myd-yf13x:~# free -m
              total        used         free       shared  buff/cache   available
Mem:          451760        31844        374408           8840         45508
399876
Swap:           0           0           0
```

- total : 内存总量
- used : 被使用的内存量
- free : 可使用的内存量

3). 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 300MB，测试次数为 10，测试命令为“memtester 300M 10”。

下列以使用 300MB 内存空间，单次测试为例：

```
root@myd-yf13x:~# memtester 300M 1
```



```
memtester version 4.5.1 (32-bit)
Copyright (C) 2001-2020 Charles Cazabon.
Licensed under the GNU General Public License version 2 (only).
```

```
pagesize is 4096
pagesizemask is 0xffff000
want 300MB (314572800 bytes)
got 300MB (314572800 bytes), trying mlock ...locked.
```

```
Loop 1/1:
```

```
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     : testing 49
```

```
settiok          70
```

```
Done.
```

2.3. eMMC/Nand

eMMC 是一个数据存储设备，包括一个 MultiMediaCard (MMC)接口，一个 NAND Flash 组件。它的成本、体积小、Flash 技术独立性和高数据吞吐量使其成为嵌入式产品的



理想选择。STM32MP135 支持 8 位的 SDMMC 接口 (eMMC v5.1) , 最大支持 128G B。本节将讲解在 Linux 系统下查看与操作 eMMC/Nand 的步骤与方法。

2.3.1. eMMC 测试

1). 查看 eMMC 容量

通过 fdisk -l 命令可以查询到 eMMC 分区信息及容量。

```
root@myd-yf13x:~# fdisk -l
Disk /dev/mmcblk1: 3.59 GiB, 3850371072 bytes, 7520256 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: ADFF86F4-9408-4A45-90D8-198535757B7FF
```

Device	Start	End	Sectors	Size	Type
/dev/mmcblk1p1	1024	2047	1024	512K	Linux reserved
/dev/mmcblk1p2	2048	3071	1024	512K	Linux reserved
/dev/mmcblk1p3	3072	11263	8192	4M	unknown
/dev/mmcblk1p4	11264	19455	8192	4M	unknown
/dev/mmcblk1p5	19456	20479	1024	512K	Linux reserved
/dev/mmcblk1p6	20480	151551	131072	64M	Linux filesystem
/dev/mmcblk1p7	151552	184319	32768	16M	Linux filesystem
/dev/mmcblk1p8	184320	6475775	6291456	3G	Linux filesystem
/dev/mmcblk1p9	6475776	7519231	1043456	509.5M	Linux filesystem

- /dev/mmcblk1p3 : 用于 fip 存放分区
- /dev/mmcblk1p5 : 用于存放 uboot env
- /dev/mmcblk1p6 : 用于 kernel 和设备树资源存放分区
- /dev/mmcblk1p7 : 用于存放第三方库文件
- /dev/mmcblk1p8 : 用于存放根文件系统
- /dev/mmcblk1p9 : 可用于用户使用分区

2). 查看 eMMC 分区信息

通过 df 命令可以查询到 eMMC 分区信息, 使用情况, 挂载目录等信息。



```
root@myir-yf13x:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        187M   0  187M   0% /dev
/dev/mmcblk1p8  2.9G  806M  1.9G  30% /
tmpfs           221M  160K  221M   1% /dev/shm
tmpfs           89M   8.8M   80M  10% /run
tmpfs           4.0M   0   4.0M   0% /sys/fs/cgroup
tmpfs           221M   20K  221M   1% /tmp
/dev/mmcblk1p6   55M   12M   39M  24% /boot
/dev/mmcblk1p7   14M   24K   13M   1% /vendor
/dev/mmcblk1p9  472M   36M  406M   9% /usr/local
tmpfs           221M  100K  221M   1% /var/volatile
tmpfs           45M    0   45M   0% /run/user/0
```

- tmpfs: 内存虚拟文件系统，挂载到不同的目录下。
- /dev/mmcblk1p6: 存放 kernel 资源分区
- /dev/mmcblk1p8: 根文件系统，挂载到根目录下。
- /dev/mmcblk1p9: 单独分区，可用于用户使用，挂载在/usr/local

3). eMMC 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。

● 写文件测试

```
root@myd-yf13x:~# time dd if=/dev/zero of=tempfile bs=1M count=100 conv
=fdatasync
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 8.46865 s, 12.4 MB/s
real    0m 8.48s
user    0m 0.00s
sys     0m 1.55s
```

使用 dd 命令写文件时，需要加 conv=fdatasync 参数，表示当 dd 写 N 次结束之后，会 flush cache 同步到磁盘。因为对磁盘的写一般是先写到缓存还没有写到磁盘就返回了。这里测试出写磁盘速度为 12.4MB/s。



● 读文件测试

在嵌入式系统中，经常需要测试系统文件读写性能，读文件时忽略 cache 的影响。这时可以指定参数 iflag=direct, nonblock。

```
root@myd-yf13x:~# dd if=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, 3.28757 s, 31.9 MB/s
```

可知，从磁盘直接读取读速度为 31.9MB/s。

2.3.2. Nand 测试

Nand-flash 内存是 flash 内存的一种，其内部采用非线性宏单元模式，为固态大容量

内存的实现提供了廉价有效的解决方案。下面在 Linux 文件系统中去查看 Nand 分区容量大小以及信息。

1). 查看 Nand 容量及大小

```
root@myd-yf13x:~# cat /proc/mtd
dev:   size  erasesize  name
mtd0: 00080000 00020000 "fsbl1"
mtd1: 00080000 00020000 "fsbl2"
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"
```

2). 查看 Nand 分区信息

```
root@myd-yf13x:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
ubi0_3          146M   77M   69M  53% /
devtmpfs        62M    0    62M   0% /dev
```




```
tmpfs          94M      0   94M    0% /dev/shm
tmpfs          38M    8.6M   29M   23% /run
tmpfs          4.0M      0   4.0M    0% /sys/fs/cgroup
tmpfs          94M      0   94M    0% /tmp
/dev/ubi0_2    59M    12M   48M   20% /boot
tmpfs          94M    12K   94M    1% /var/volatile
tmpfs          19M      0   19M    0% /run/user/0
```

3). Nand 性能测试

● 写文件测试

```
root@myd-yf13x:~# time dd if=/dev/zero of=tempfile bs=1M count=100 conv
=fdatasync
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 2.84869 s, 36.8 MB/s
real    0m 2.86s
user    0m 0.00s
sys     0m 1.01s
```

使用 dd 命令写文件时，需要加 conv=fdatasync 参数，表示当 dd 写 N 次结束之后，会 flush cache 同步到磁盘。因为对磁盘的写一般是先写到缓存还没有写到磁盘就返回了。这里测试出写磁盘速度为 36.8MB/s。

● 读文件测试

在嵌入式系统中，经常需要测试系统文件读写性能，读文件时忽略 cache 的影响。先执行下下面的命令清除 cache。

```
root@myd-yf13x:~# sync; echo 3 > /proc/sys/vm/drop_caches
[ 7744.684308] sh (511): drop_caches: 3
```

然后继续测试读文件速度

```
root@myd-yf13x:~# dd if=tempfile of=/dev/null bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 5.65183 s, 18.6 MB/s
```



可知，从磁盘直接读取速度为 18.6MB/s。

2.4. 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。

- 查看系统 RTC 设备

```
root@myd-yf13x:~# ls /dev/rtc* -al
lrwxrwxrwx 1 root root      4 Apr 29 10:18 /dev/rtc -> rtc0
crw----- 1 root root 253, 0 Apr 29 10:18 /dev/rtc0
```

rtc 属于 linux 设备，在/dev 下有其设备节点 rtc0 可供用户操作。

- 设置系统时间

在将系统时间设置为 Mon May 1 00:00:00 UTC 2023

```
root@myd-yf13x:~# date 050100002023.00
[ 97.013942] systemd-journald[526]: Time jumped backwards, rotating.
Mon May 1 00:00:00 UTC 2023
```

- 将系统时间写入 RTC

将上一步 date 命令设置的系统时间写入到 RTC 设备：

```
root@myd-yf13x:~# hwclock -w
```

- 读取 RTC 时间并设置为系统时间

```
root@myd-yf13x:~# hwclock -r
2023-05-01 00:01:02.263476+00:00
```

- 掉电保持 RTC 时间

将开发板关机断开电源，经过 2 分钟左右，重新上电开机。查看 RTC 时间和系统时间：

```
root@myd-yf13x:~# hwclock -r
```



2023-05-01 00:03:55.021306+00:00

重新开机之后查看的 RTC 时间和系统时间比之前设置的时候增加了大约 2 分钟，说明 RTC 工作正常。如果需要详细测试 RTC 的精度，可以将断电时间延长如 24 小时，测试 RTC 时间与标准时间的差异。

- 将系统时间与 RTC 时间同步

```
root@myd-yf13x:~# hwclock -s
root@myd-yf13x:~# date
Mon May 1 00:04:50 UTC 2023
```

如果将 `hwclock -s` 命令添加到启动脚本中就可以保证每次启动都可以将系统时间和 RTC 时间保持同步了。

2.5. Watchdog

Linux 内核包含 Watchdog 子系统，硬件设计过程中一般可以利用芯片内部的看门狗定时器或者使用外部看门狗芯片来实现 Watchdog 的功能，用于监测系统的运行。当系统出现异常情况无法喂狗时系统将进行自动复位。MYD-YF13X 芯片内部有 1 个看门狗，本章节将讲解 linux 下看门狗的测试方法。

1). 关闭看门狗测试

向看门狗节点写入“V”，尝试关闭看门狗：

```
root@myd-yf13x:~# echo V > /dev/watchdog0
[ 71.038670] watchdog: watchdog0: nowayout prevents watchdog being stopped!
[ 71.044297] watchdog: watchdog0: watchdog did not stop!
```

有上面信息可知，由于 NOWAYOUT 属性，系统并不能关闭看门狗。

2). 用户空间测试看门狗

模拟内核崩溃，测试看门狗复位功能，默认 32s 重启系统：

```
root@myd-yf13x:~# echo c > /proc/sysrq-trigger
[ 381.425435] sysrq: Trigger a crash
[ 381.427400] Kernel panic - not syncing: sysrq triggered crash
[ 381.433145] CPU: 0 PID: 656 Comm: sh Not tainted 5.15.67 #1
[ 381.438794] Hardware name: STM32 (Device Tree Support)
```



```
[ 381.443843] [<c010e8ec>] (unwind_backtrace) from [<c010c158>] (show_stack+0x10/0x14)
[ 381.451623] [<c010c158>] (show_stack) from [<c0c32f30>] (panic+0xfc/0x2f0)
[ 381.458490] [<c0c32f30>] (panic) from [<c06bfc1c>] (sysrq_reset_seq_param_set+0x0/0x84)
[ 381.466462] [<c06bfc1c>] (sysrq_reset_seq_param_set) from [<00000002>] (0x2)
[ 381.473532] ---[ end Kernel panic - not syncing: sysrq triggered crash ]---
. . .
[ 0.000000] Kernel command line: root=PARTUUID=491f6117-415d-4f53-88c9-6e0de54deac6 rootwait rw console=ttySTM0,115200
[ 0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes, linear)
[ 0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes, linear)
[ 0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[ 0.000000] Memory: 315700K/524288K available (12288K kernel code, 1255K rwd data, 3444K rodata, 1024K init, 186K bss, 77516K reserved, 131072K cm a-reserved, 0K highmem)
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.000000] trace event string verifier disabled

root@myd-yf13x:~#
```

3). 应用程序测试看门狗

执行下面命令禁用看门狗，然后重启开发板：

```
root@myd-yf13x:~# echo V > /dev/watchdog0
```

- 设置看门狗超时时间

通过 `ioctl` 实现超时时间，具体命令为：`WDIOC_SETTIMEOUT`，需要一个参数，即超时时间 `timeout`。使用示例：



```
ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
```

以上为设置看门狗当前超时时间的参考代码。其中 fd 为看门狗设备的文件句柄。

● 看门狗应用程序测试

编译生产执行文件 watchdog 并拷贝到开发板，如下命令执行：

```
root@myd-yf13x:~# ./watchdog
Usage: wdt_driver_test <timeout> <sleep> <test>
    timeout: value in seconds to cause wdt timeout/reset
    sleep: value in seconds to service the wdt
    test: 0 - Service wdt with ioctl(), 1 - with write()
```

运行看门狗应用,超时时间为 4s，每间隔 1s 喂一次狗：

```
root@myd-yf13x:~# ./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。如需了解 ST 看门狗外设请参考以下链接：

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Watchdog_overview。

(watchdog 例程在 04_Sources/Example/目录下)

2.6. Power Management

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。MYD-YF13X 只支持休眠到内存的方式，即“mem”方式。

1). 查看当前开发板支持的模式

```
root@myd-yf13x:~# cat /sys/power/state
mem
```

2). 设置唤醒源

测试休眠唤醒需要先设置唤醒源，设置 Debug 串口(UART4)为唤醒源：

- 20 -



```
root@myd-yf13x:~# echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
```

3). 休眠到内存

在用户空间向/sys/power/state 写入字符串即进入相应的电源管理模式，休眠到内存的方式如下：

```
root@myd-yf13x:~# echo "mem" > /sys/power/state
[ 465.180354] PM: suspend entry (deep)
[ 465.197085] Filesystems sync: 0.014 seconds
[ 465.208816] Freezing user space processes ... (elapsed 0.001 seconds) done.
[ 465.216194] OOM killer disabled.
[ 465.219426] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[ 465.226782] printk: Suspending console(s) (use no_console_suspend to debug)
```

4). 通过 Debug 串口终端唤醒

输入休眠命令后开发板休眠，将运行状态数据存到内存，并关闭外设，进入等待模式，唤醒较慢，心跳灯停止闪烁。此时因为设置的是 debug 唤醒，键盘随意输入字符即能成功唤醒系统，如下：

```
[ 465.876499] stm32-dwmac 5800a000.eth1 eth0: configuring for phy/rgmii-id link mode
[ 466.885622] stm32-dwmac 5800a000.eth1: Failed to reset the dma
[ 466.885649] stm32-dwmac 5800a000.eth1 eth0: stmmac_hw_setup: DMA engine initialization failed
[ 466.886230] stm32-dwmac 5800e000.eth2 eth1: configuring for phy/rgmii-id link mode
[ 467.889448] stm32-dwmac 5800e000.eth2: Failed to reset the dma
[ 467.938198] Restarting tasks ... done.
[ 467.982982] PM: suspend exit.
[ 368.676566] PM: suspend exit
```

此时终端重新进入命令行，心跳灯开始闪烁。



3. 外设接口

3.1. GPIO

GPIO 的测试是通过文件系统 sysfs 接口来实现的，下面内容以 PF14 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 PF14 对应的 pin 引脚编号为：`PIN_NO('F',14)=(0x46-0x41)*0x10+14=(70-65)*16+14=94`。

1). 导出 GPIO

```
root@myd-yf13x:~# echo 94 > /sys/class/gpio/export
```

导出成功后会在 `/sys/class/gpio/` 目录下生成 PF14 这个目录。

2). 设置/查看 GPIO 方向

- 设置输入

```
root@myd-yf13x:~# echo "in" > /sys/class/gpio/PF14/direction
```

- 设置输出

```
root@myd-yf13x:~# echo "out" > /sys/class/gpio/PF14/direction
```

- 查看 gpio 方向

```
root@myd-yf13x:~# cat /sys/class/gpio/PF14/direction
out
```

返回 in 表示输入，返回 out 表示输出。

3). 设置/查看 GPIO 的值

- 设置输出低

```
root@myd-yf13x:~# echo "0" > /sys/class/gpio/PF14/value
```

- 设置输出高

```
root@myd-yf13x:~# echo "1" > /sys/class/gpio/PF14/value
```

- 查看 gpio 的值

```
root@myd-yf13x:~# cat /sys/class/gpio/PF14/value
1
```



可以看到 PF14 输出高电平，可以用万用表测量 J7 扩展 IO 的 PF14 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

3.2. 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。

1). 操作 LED 的目录为 `/sys/class/leds`

```
root@myd-yf13x:~# ls /sys/class/leds/  
blue:heartbeat
```

2). 测试 LED

- 读取蓝灯 LED 状态

其中 0 表示 LED 关闭，1 表示 LED 开启：

```
root@myd-yf13x:~# cat /sys/class/leds/blue:heartbeat/brightness  
1
```

可知当前蓝灯 LED 为开启状态。

- 熄灭 LED

```
root@myd-yf13x:~# echo 0 > /sys/class/leds/blue:heartbeat/brightness
```

- 点亮 LED

```
root@myd-yf13x:~# echo 1 > /sys/class/leds/blue:heartbeat/brightness
```

- 开启 LED 触发模式

开启 “timer” 出发模式后，LED 默认以 1Hz 周期闪烁，占空比为 50%：

```
root@myd-yf13x:~# echo "timer" > /sys/class/leds/blue:heartbeat/trigger
```

改变 LED 灯闪烁时的亮灭占空比

通过调整 `led` 下的 `delay_on` 和 `delay_off` 属性，可以调整 LED 闪烁周期和亮灭占空比，例如：




```
root@myd-yf13x:~# echo "100" > /sys/class/leds/blue:heartbeat/delay_on
root@myd-yf13x:~# echo "500" > /sys/class/leds/blue:heartbeat/delay_off
```

3.3. Key(按键)

Linux 的/dev/input/eventx 设备可以用来方便地调试鼠标、键盘、触摸板等输入设备。本节主要是测试 key。通过 hexdump 命令以及 dmesg 命令来查看按键是否有反应。MYD-YF13X 有两个按键，S1 是系统复位按键；S2 是用户按键（User KEY），已经在设备树配置。

1). 设备树配置信息

打开配套的设备树文件/arch/arm/boot/dts/myb-stm32mp135f.dts，可以看到对应的 gpio-keys 节点：

```
Path:/
    gpio-keys {
        compatible = "gpio-keys";
        #size-cells = <0>;
        button-0 {
            label = "usr_button";
            linux,code = <KEY_ENTER>;
            interrupt-parent = <&gpioi>;
            interrupts = <1 IRQ_TYPE_EDGE_RISING>;
        };
    };
```

● 按键测试

查看对应的输入 event 信息

```
root@myd-yf13x:~# cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="gpio-keys"
P: Phys=gpio-keys/input0
S: Sysfs=/devices/platform/gpio-keys/input/input0
U: Uniq=
H: Handlers=kbd event0
B: PROP=0
```



B: EV=3

B: KEY=10000000

由上可以知 gpio-keys 的对应设备事件为 event0。

- **dump 按键信息**

执行下面命令，操作按键 S2，串口终端会打印出如下信息：

```
root@myd-yf13x:~# hexdump /dev/input/event0
0000000 2729 644f d4c1 000b 0001 001c 0001 0000
0000010 2729 644f d4c1 000b 0000 0000 0000 0000
0000020 2729 644f e879 000b 0001 001c 0000 0000
0000030 2729 644f e879 000b 0000 0000 0000 0000
0000040 272a 644f 6723 0005 0001 001c 0001 0000
0000050 272a 644f 6723 0005 0000 0000 0000 0000
```

每按一次 S2 当前终端会打印出当前事件码值，即按键正常。

3.4. RS232

本节将使用 Linux API 配置开发板 RS232 的收发功能。Linux 的串口设备文件一般命名为/dev/ttySTMn(n=0,1,2,3.....)。n 表示串口在 Linux 系统中的设备编号，“ttySTM”是内核已经定义好的串口设备名字。本节是以 MYD-YF13X 底板上 J19 接口（即 uart5）为例进行测试，uart5 的编号设备节点为 ttySTM2。其测试配置如下表：

表 3-1.RS232 接口配置

	MYD-YF13X	windows 10
硬件接口	RS232	USB-RS232 模块
设备节点	ttySTM2	com12
测试软件	uart_test	sscom

这里将 J19 的 232_RX 和 232_TX 分别与 USB-RS232 转换器的 RXD 和 TXD 连接。

(uart_test 测试例程在 04_Sources/Example/目录下)。

1). 测试开发板 RS232 接收数据

- **windows 下 sscom 发送数据**

```
[22:46:53.902]发→◇12345678
[22:46:54.117]发→◇12345678
[22:46:54.415]发→◇12345678
```



```
[22:46:54.482]发→◇12345678
```

```
[22:46:54.661]发→◇12345678
```

```
[22:46:54.850]发→◇12345678
```

```
[22:46:55.058]发→◇12345678
```

● 开发板接收数据

在开发板上执行下面命令，接收数据。命令执行完成后中断会进入一个阻塞状态，等待接收电脑串口发送过来的数据：

```
root@myd-yf13x:~# ./uart_test -d /dev/ttySTM2 -b 115200
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
/dev/ttySTM2 RECV[8]: 12345678
```

windows 下发送的数据与开发板接收数据一致，即开发板 RS232 接收数据正常。

2). 测试开发板 RS232 发送数据

● 开发板发送数据

```
root@myd-yf13x:~# ./uart_test -d /dev/ttySTM2 -b 115200 -m 1
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
/dev/ttySTM2 SEND: 1234567890
```

上述命令执行完成后，会自动发送对应数据。

● windows 下 sscom 接收数据

```
[11:49:30.366]收←◆1234567890
```



```
[11:49:31.366]收←◆1234567890
[11:49:32.366]收←◆1234567890
[11:49:33.366]收←◆1234567890
[11:49:34.367]收←◆1234567890
[11:49:35.367]收←◆1234567890
[11:49:36.367]收←◆1234567890
[11:49:37.368]收←◆1234567890
```

windows 下 sscom 接收的数据与开发板发送数据一致，即开发板 RS232 发送数据正常。

3.5. RS485

本例程演示如何使用 Linux API 测试开发板 RS485 发送和接收数据功能，RS485 设备别名 serial3 = &uart5，设备节点为 ttySTM1。以 J19 接口的 RS485 为例进行测试，硬件接口配置如下表：

表 3-2.RS485 接口配置

	MYD-YF13X	windows 10
接口	RS485	USB-RS485
设备节点	ttySTM1	com12
测试软件	rs485_read、rs485_write	sscom

将 J19 的 485A 和 485B 分别与 USB-RS485 转换器的 485A 和 485B 连接。

(rs485_read、rs485_write 测试例程在 04_Sources/Example/目录下)。

1). 开发板下 RS485 接收数据

- windows 下 sscom 发送数据

```
[20:01:05.903]Send→◇12345678
[20:01:06.734]Send→◇12345678
[20:01:12.231]Send→◇12345678
[20:01:12.752]Send→◇12345678
[20:01:13.171]Send→◇12345678
[20:01:13.426]Send→◇12345678
[20:01:13.699]Send→◇12345678
[20:01:13.940]Send→◇12345678
```



- 设置接收模式

```
root@myd-yf13x:~# echo 60 > /sys/class/gpio/export
root@myd-yf13x:~# echo out > /sys/class/gpio/PD12/direction
root@myd-yf13x:~# echo 0 > /sys/class/gpio/PD12/value
```

- 开发板上接收数据

```
root@myd-yf13x:~# ./rs485_read -d /dev/ttySTM1 -b 115200
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
RECV[-163754450]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
```

windows 下发送数据与开发板接收数据一致，即开发板 RS485 接收数据正常。

2). 开发板下 RS485 发送数据

- 设置发送模式

```
root@myd-yf13x:~# echo 60 > /sys/class/gpio/export
root@myd-yf13x:~# echo out > /sys/class/gpio/PD12/direction
root@myd-yf13x:~# echo 1 > /sys/class/gpio/PD12/value
```

- 开发板发送数据

```
root@myd-yf13x:~# ./rs485_write -d /dev/ttySTM1 -b 115200
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
SEND[08]: 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38
```

上述命令执行完成后应用程序会自动发送数据。



- windows 下 sscom 接收数据

```
[20:27:27.972]收←◆31 32 33 34 35 36 37 38
[20:27:28.972]收←◆31 32 33 34 35 36 37 38
[20:27:29.971]收←◆31 32 33 34 35 36 37 38
[20:27:30.972]收←◆31 32 33 34 35 36 37 38
[20:27:31.971]收←◆31 32 33 34 35 36 37 38
[20:27:32.972]收←◆31 32 33 34 35 36 37 38
[20:27:33.972]收←◆31 32 33 34 35 36 37 38
[20:27:33.972]收←◆31 32 33 34 35 36 37 38
```

windows 下接收到的数据与开发板发送数据一致，即开发板 RS485 发送数据正常。

3.6. CAN

本节采用 Linux 系统常用的 cansend、candump 命令进行 SocketCAN 的通讯测试。这里测试使用的两块开发板对接测试。

J19 座子的 CANH、CANL 引脚和同类型的板子 CANH、CANL 相连。

1). 初始化 CAN 网络接口

- 设置 CAN 波特率

使用 canfd 需要先设置波特率，并开启 CAN 网络接口。参考下面命令分别将两块开发板的仲裁波特率设置为 5KHz，数据波特率为 4M，并开启 canfd 功能：(两个开发板都需要做以下操作)

```
root@myd-yf13x:~# ifconfig can0 down
root@myir-yf13x:~# ip link set can0 up type can bitrate 500000 sample-point
0.75 dbitrate 4000000 dsample-point 0.8 fd on
[ 125.963519] m_can_platform 4400f000.can can0: bitrate error 3.8%
[ 125.984407] IPv6: ADDRCONF(NETDEV_CHANGE): can0: link becomes ready
root@myir-yf13x:~# ifconfig can0 up
```

至此，即开启 canfd 功能。

2). 收发数据

- 发送数据

设置其中一块板子为发送，并使用 cansend 发送 64 字节固定格式的字符串来测试：

```
root@myd-yf13x:~# cansend can0 100#11.22.33.44
```

- 29 -



```
root@myd-yf13x:~# cansend can0 100#11.22.33.44
root@myd-yf13x:~# cansend can0 100#11.22.33.44
root@myd-yf13x:~# cansend can0 100#11.22.33.44
root@myd-yf13x:~# cansend can0 100#11.22.33.44
root@myd-yf13x:~# cansend can0 100#11.22.33.44
```

● 接收数据

设置另外一块板子为接收，可以使用 `candump` 来查看 CAN 的接收数据：

```
root@myd-yf13x:~# candump can0 -L
(1712819690.336718) can0 100#11223344
(1712819693.324284) can0 100#11223344
(1712819693.735206) can0 100#11223344
(1712819694.134187) can0 100#11223344
(1712819694.526169) can0 100#11223344
(1712819695.004645) can0 100#11223344
```

3). 统计 can0 信息

CAN 数据收发之后显示 CAN 设备的详情和收发统计信息，其中“clock”的值代表 can 的时钟，“drop”的值代表丢包，“overrun”的值代表溢出，“error”代表总线错误。

```
root@myd-yf13x:~# ip -details -statistics link show can0
2: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UP mode
  DEFAULT group default qlen 10
    link/can  promiscuity 0 minmtu 0 maxmtu 0
    can <FD>  state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
      bitrate 500000 sample-point 0.750
      tq 20 prop-seg 37 phase-seg1 37 phase-seg2 25 sjw 1 brp 1
      m_can: tseg1 2..256 tseg2 2..128 sjw 1..128 brp 1..512 brp_inc 1
      dbitrte 3846153 dsample-point 0.769
      dtq 20 dprop-seg 4 dphase-seg1 5 dphase-seg2 3 dsjw 1 dbrp 1
      m_can: dtseg1 1..32 dtseg2 1..16 dsjw 1..16 dbrp 1..32 dbrp_inc 1
      clock 50000000
      re-started bus-errors arbit-lost error-warn error-pass bus-off
```



```

0          0          0          0          0          0          nu
mtxqueues 1 gso_max_size 65536 gso_max_segs 65535 parentbus platform pa
rentdev 4400f000.can
RX:  bytes packets errors dropped missed mcast
      0          0          0          0          0          0
TX:  bytes packets errors dropped carrier collsns
      24          6          0          0          0          0

```

3.7. 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。

1). 查看插入 usb 的打印信息

- 查看 USB 设备信息

将 U 盘连接到开发板 USB Host 接口(J11)，内核提示信息如下：

```

root@myd-yf13x:~#
[ 1431.459516] usb 1-1.1: new high-speed USB device number 3 using ehci-pl
atform
[ 1431.719887] usb-storage 1-1.1:1.0: USB Mass Storage device detected
[ 1431.743089] scsi host0: usb-storage 1-1.1:1.0
[ 1432.810835] scsi 0:0:0:0: Direct-Access    TU100    128GB thinkplus  0000
PQ: 0 ANSI: 4
[ 1432.837064] sd 0:0:0:0: [sda] 245760001 512-byte logical blocks: (126 GB/1
17 GiB)
[ 1432.845080] sd 0:0:0:0: Attached scsi generic sg0 type 0
[ 1432.863871] sd 0:0:0:0: [sda] Write Protect is off
[ 1432.890792] sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, do
esn't support DPO or FUA
[ 1432.933455] sda: sda1
[ 1432.941359] sd 0:0:0:0: [sda] Attached SCSI removable disk

```

从上述信息可以得出需要挂载的设备为 sda1。

2). U 盘挂载读写



- 挂载 U 盘

```
root@myd-yf13x:/# mount /dev/sda1 /mnt/
```

- 读文件

需要提前在 U 盘上建立一个 test.txt 文件。

```
root@myd-yf13x:/# ls /mnt
test.txt
root@myd-yf13x:/# cat /mnt/test.txt
hello world!
```

- 写文件

```
root@myd-yf13x:/# touch test.txt
root@myd-yf13x:/# echo "hello world !!!" > test.txt
root@myd-yf13x:/# cp test.txt /mnt
root@myd-yf13x:/# cat /mnt/test.txt
hello world !!!
```

写完文件后需要执行下 sync 命令，确保数据完全写入到 U 盘里面之后，才可以卸载 U 盘设备。

3). 卸载 U 盘

- 卸载操作

```
root@myd-yf13x:/# umount /mnt
root@myd-yf13x:~# [ 1666.532207] usb 1-1.1: USB disconnect, device number 5
```

3.8. Micro SD 卡

Micro SD Card，原名 Trans-flash Card(TF 卡)，Micro SD 卡是一种极细小的快闪存储器卡。Micro SD 卡相比标准 SD 卡，外形上更加小巧，是 SD 卡类型中尺寸最小的一种 SD 卡。尽管 Micro SD 卡的外形大小及接口形状与原来的 SD 卡有所不同，但接口规范保持不变，确保了兼容性。若将 Micro SD 插入特定的转接卡中，可当作标准 SD 卡来使用，SD 卡已成为目前消费数码设备中应用最广泛的一种存储卡，具有大容量、高性能、安全等多种特点的多功能存储卡。Micro SD 卡背面一般有 9 个引脚，包含 4 根数据线，支持 1bit/4bit 两种数据传输宽度。



STM32MP135 支持 2 路 8bit SDMMC 接口，MYD-YF135 开发板上使用 SDMMC1 连接 Micro SD。

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：

```
root@myir:~# fdisk -l
略
Disk /dev/mmcbk0: 29.72 GiB, 31914983424 bytes, 62333952 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 432D1A65-71C3-41A5-BA22-86A16067C169

Device            Start      End  Sectors  Size Type
/dev/mmcbk0p1       34        545     512    256K Linux reserved
/dev/mmcbk0p2      546       1057     512    256K Linux reserved
/dev/mmcbk0p3     1058       1569     512    256K Linux reserved
/dev/mmcbk0p4     1570       2081     512    256K Linux reserved
/dev/mmcbk0p5     2082      10273    8192     4M unknown
/dev/mmcbk0p6    10274      18465    8192     4M unknown
/dev/mmcbk0p7     18466      19489    1024    512K Linux reserved
/dev/mmcbk0p8     19490     150561  131072    64M Linux filesystem
/dev/mmcbk0p9    150562     183329    32768    16M Linux filesystem
/dev/mmcbk0p10   183330   11197985 11014656    5.3G Linux filesystem
/dev/mmcbk0p11  11197986  13385694  2187709     1G Linux filesystem
```

- /dev/mmcbk0p1：用于存放 TF-A
- /dev/mmcbk0p2：用于存放 TF-A
- /dev/mmcbk0p5：用于存放 FIP
- /dev/mmcbk0p7：用于存放 uboot env
- /dev/mmcbk0p8：用于存放 kernel 和设备树资源
- /dev/mmcbk0p9：用于存放第三方库文件



- /dev/mmcblk0p10：用于存放根文件系统
- /dev/mmcblk0p11：用户空间分区

● 查看 TF 卡分区信息

通过 df 命令可以查询到 TF 卡分区信息，使用情况，挂载目录等信息：

```
root@myd-yf13x:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        187M   0  187M   0% /dev
/dev/mmcblk0p10  4.9G  2.2G  2.5G  47% /
tmpfs           221M  172K  221M   1% /dev/shm
tmpfs           89M   8.8M   80M  10% /run
tmpfs           4.0M    0  4.0M   0% /sys/fs/cgroup
tmpfs           221M   20K  221M   1% /tmp
/dev/mmcblk0p8   55M   12M   39M  24% /boot
/dev/mmcblk0p9   14M   24K   13M   1% /vendor
tmpfs           221M  104K  221M   1% /var/volatile
/dev/mmcblk0p11 995M   36M  902M   4% /usr/local
tmpfs           45M    0   45M   0% /run/user/1000
tmpfs           45M    0   45M   0% /run/user/0
```

- tmpfs: 内存虚拟文件系统，挂载到不同的目录下
- devtmpfs: 用于系统创建 dev
- /dev/mmcblk0p8: 用于 kernel 资源、启动脚本设置的存放分区，挂载在/boot 目录
- /dev/mmcblk0p10: 根文件系统
- /dev/mmcblk0p11: 单独分区，可用于用户使用，挂载在/usr/local

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/mmcblk0p11 为例，挂载目录为/usr/local。

● 写文件测试

```
root@myd-yf13x:/mnt# time dd if=/dev/zero of=/usr/local/tempfile bs=1M count=100 conv=fdatasync
```



```
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 12.8163 s, 8.2 MB/s
real    0m 12.82s
user    0m 0.00s
sys     0m 1.59s
```

使用 dd 命令写文件时，需要加 conv=fdatasync 参数，表示当 dd 写 N 次结束之后，会 flush cache 同步到磁盘。因为对磁盘的写一般是先写到缓存还没有写到磁盘就返回了。这里测试出写磁盘速度为 8.2MB/s。

● 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。

```
root@myd-yf13x:/mnt# time dd if=/usr/local/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, 4.44917 s, 23.6 MB/s
real    0m 4.45s
user    0m 0.00s
sys     0m 0.04s
```

可知，直接从 SD 卡读数据速度为 23.6MB/s。

3.9. ADC

MYD-YF13X 提供了两路 ADC，都提供一个默认的电值，ADC 的测试是通过文件系统 sysfs 接口来实现的，下面以 ADC 通道 2 和通道 8 为例进行说明。

1). 读取 ADC 的 sysfs 接口

使用命令查看 ADC 读取接口：

```
root@myd-yf13x:/mnt# grep -H "" /sys/bus/iio/devices/*/name | grep adc
/sys/bus/iio/devices/iio:device0/name:48003000.adc:adc@0
```

2). ADC 读取操作

读取 ADC 值如下：



● 读取通道 2 值

```
root@myd-yf13x:~# cat /sys/bus/iio/devices/iio:device0/in_voltage2_raw
3151
```

上面得到的值除以 1000 就得到测量的电压值。

● 读取通道 8 值

```
root@myd-yf13x:~# cat /sys/bus/iio/devices/iio:device0/in_voltage8_raw
2043
```

上面得到的值除以 1000 就得到测量的电压值。

3.10. Display

MYD-YF13X 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 SII9022ACNU 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，支持米尔 7 寸电容屏。

显示方案均可在 uboot 阶段通过选择对应的设备树文件来切换，MYD-YF13X 的设备树信息列表如下：

表 3-3.设备树信息列表

设备树	描述
myb-stm32mp135x-512m	包含最基本设备树，并增加 LCD 显示描述，出厂镜像默认为此设备树。
myb-stm32mp135x-512m-hdmi	包含最基本设备树，并增加 HDMI 显示描述。

本节将介绍 uboot 显示方案切换的方法以及显示的测试方式。

1). HDMI 显示

如果用户使用过程需要 HDMI 显示功能，需要使用到表 1-1 中的 RGB 转 HDMI 模块，连接好后，启动开发板，在 uboot 启动 log 阶段在启动模式下选择 “2”，然后按“Enter”键即可。

```
Select the boot mode
1:      OpenSTLinux
```



```
2:      myb-stm32mp135x-512m-hdmi
3:      myb-stm32mp135x-512m
Enter choice: 2
```

2). LCD 显示

MYD-YF13X 适配米尔的 LCD 屏型号为 MY-TFT070CV2。这款屏幕是 7 寸电容触摸屏，是彩色有源矩阵薄膜晶体管（TFT）液晶显示器（LCD）。屏幕先通过 50pin 柔性连接线接到底板的 J15 LCD 接口。

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 阶段在启动模式下选择“3”，然后按“Enter”键即可

```
Select the boot mode
1:      OpenSTLinux
2:      myb-stm32mp135x-512m-hdmi
3:      myb-stm32mp135x-512m
Enter choice: 3
```

3.11. Touch Panel

触摸有电容触摸和电阻触摸，MYD-YF13X 系列开发板硬件目前不支持电阻触摸，但是支持电容触摸，米尔科技提供 7 寸触摸的液晶屏配件，见表 1-1。可根据实际需求自行购买配件。电容屏在使用中较为灵敏，很少出现问题。另外，电容屏不需要较准。因为根据电容屏的原理，电容屏在使用中是可以准确的识别出手指与屏幕接触的位置，具有很高的灵敏性。我们在使用中如果出现点击软件选不中的现象，一般只有一种情况：屏幕出现了问题。下面是通过 evtest 命令测试电容屏触摸功能的简单测试。

1). 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 1，测试界面选择“1”按下回车即可开始测试：

```
root@myd-yf13x:~# evtest
No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      gpio-keys
/dev/input/event1:      generic ft5x06 (79)
Select the device event number [0-1]: 1
```



Input driver version is 1.0.1

Input device ID: bus 0x18 vendor 0x0 product 0x0 version 0x0

Input device name: "generic ft5x06 (79)"

Supported events:

Event type 0 (EV_SYN)

Event type 1 (EV_KEY)

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

Max 4

Event code 53 (ABS_MT_POSITION_X)

Value 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)

可知，接上触摸屏后会识别为一个输入接口，此触摸屏的 event number 为 1。

点击触摸屏，终端会打印对应的信息：

```
Event: time 1651169395.095608, type 3 (EV_ABS), code 57 (ABS_MT_TRACKING_ID), value 0
Event: time 1651169395.095608, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 445
Event: time 1651169395.095608, type 3 (EV_ABS), code 54 (ABS_MT_POSITION_Y), value 154
Event: time 1651169395.095608, type 1 (EV_KEY), code 330 (BTN_TOUCH), value 1
Event: time 1651169395.095608, type 3 (EV_ABS), code 0 (ABS_X), value 445
Event: time 1651169395.095608, type 3 (EV_ABS), code 1 (ABS_Y), value 154
Event: time 1651169395.095608, ----- SYN_REPORT -----
Event: time 1651169395.111154, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 447
Event: time 1651169395.111154, type 3 (EV_ABS), code 54 (ABS_MT_POSITION_Y), value 150
Event: time 1651169395.111154, type 3 (EV_ABS), code 0 (ABS_X), value 447
Event: time 1651169395.111154, type 3 (EV_ABS), code 1 (ABS_Y), value 150
Event: time 1651169395.111154, ----- SYN_REPORT -----
Event: time 1651169395.123595, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 445
Event: time 1651169395.123595, type 3 (EV_ABS), code 54 (ABS_MT_POSITION_Y), value 151
Event: time 1651169395.123595, type 3 (EV_ABS), code 0 (ABS_X), value 445
Event: time 1651169395.123595, type 3 (EV_ABS), code 1 (ABS_Y), value 151
Event: time 1651169395.123595, ----- SYN_REPORT -----
Event: time 1651169395.137485, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 446
Event: time 1651169395.137485, type 3 (EV_ABS), code 0 (ABS_X), value 446
```




```
Event: time 1651169395.137485, ----- SYN_REPORT -----
Event: time 1651169395.156492, type 3 (EV_ABS), code 53 (ABS_MT_POSITION_X), value 445
Event: time 1651169395.156492, type 3 (EV_ABS), code 0 (ABS_X), value 445
Event: time 1651169395.156492, ----- SYN_REPORT -----
Event: time 1651169395.172975, type 3 (EV_ABS), code 57 (ABS_MT_TRACKING_ID), value -1
Event: time 1651169395.172975, type 1 (EV_KEY), code 330 (BTN_TOUCH), value 0
Event: time 1651169395.172975, ----- SYN_REPORT -----)
```

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. 网络接口

MYD-YF13X 开发板包含两个千兆以太网接口，下面对网络外设接口的配置进行介绍。

4.1. Ethernet

Linux 下网络配置的工具很多，常见的有 net-tools, iproute2, systemd-networkd, network manager 以及 connman 等，这些都可以在系统定制的时候根据实际需要进行选择，这里介绍几种常用的以太网手动临时配置和自动永久配置方式。

1). 手动临时配置以太网 IP 地址

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：

```
root@myd-yf13x:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 42:0F:18:CC:11:F9
          inet addr:192.168.40.119  Bcast:192.168.40.255  Mask:255.255.255.0
          inet6 addr: fe80::400f:18ff:fecc:11f9/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:289 errors:0 dropped:32 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:39356 (38.4 KiB)  TX bytes:11393 (11.1 KiB)
          Interrupt:44 Base address:0x8000
```

eth0 为实际的以太网设备，默认采用的是随机硬件 MAC 地址，如果用户需要自己写入 MAC 地址，请参照 ST 的官方 Wiki 说明：

https://wiki.st.com/stm32mpu/wiki/How_to_update_OTP_with_U-Boot#MAC_addresses_example

下面介绍给 eth0 手动配置 IP 地址 192.168.0.100 的方法，命令如下：

```
root@myd-yf13x:~# ifconfig eth0 192.168.0.100 netmask 255.255.255.0 up
```

上面的命令手动配置 eth0 的 IP 地址为 192.168.0.100，子网掩码为 255.255.255.0，以及默认配置的广播地址 192.168.40.255，并通过 up 参数进行激活，如下所示：

```
root@myd-yf13x:~# ifconfig eth0
```



```
eth0      Link encap:Ethernet  HWaddr 42:0F:18:CC:11:F9
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::400f:18ff:fecc:11f9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:840 errors:0 dropped:75 overruns:0 frame:0
          TX packets:118 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:113293 (110.6 KiB)  TX bytes:19267 (18.8 KiB)
          Interrupt:44 Base address:0x8000
```

使用 iproute2 工具包中的 ip 命令对网络进行手动配置

ifconfig 命令手动设置 IP 地址的方法也可以使用 ip addr 和 ip link 进行替代，更多的信息请查看 <https://wiki.linuxfoundation.org/networking/iproute2> 中的说明。

```
root@myd-yf13x:~# ip addr flush dev eth0
root@myd-yf13x:~# ip addr add 192.168.0.101/24 brd + dev eth0
root@myd-yf13x:~# ip link set eth0 up
```

如果之前已经配置过 IP 地址，再使用 ip addr add 配置的 IP 地址将会成为 Secondary 地址，所以这里先使用 ip addr flush 清除之前的地址之后再行配置然后激活。完成配置之后，通过 ip addr show 命令查看 eth0 信息如下：

```
root@myd-yf13x:~# ip addr show eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
P group default qlen 1000
    link/ether 42:0f:18:cc:11:f9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.101/24 brd 192.168.0.255 scope global eth0
        valid_lft forever preferred_lft forever
```

2). 自动永久配置以太网 IP 地址

通过 ifconfig 命令和 ip 命令配置的 IP 地址断电之后就会丢失，如果需要使 IP 地址永久生效，就需要修改网络管理工具相应的配置文件。

使用 systemd-networkd 管理工具配置动态获取 IP 地址

MYD-YF13X 默认出厂镜像采用 systemd-networkd 进行网卡的配置，具体可以查看 /lib/systemd/network/ 目录下的配置文件 80-wired.network。



```
root@myd-yf13x:~# cat /lib/systemd/network/80-wired.network
[Match]
Type=ether
Name=!veth*
KernelCommandLine=!nfsroot
KernelCommandLine=!ip

[Network]
DHCP=yes

[DHCP]
UseMTU=yes
RouteMetric=10
ClientIdentifier=mac
```

以上配置文件将会对 en*和 eth*匹配的网卡进行配置，如果内核启动参数中不包含 nfs root 参数时，则通过 DHCP 自动为匹配到的网卡配置 IP 地址，网关，DNS 等信息。详细的配置参数参见以下链接：

<https://www.freedesktop.org/software/systemd/man/systemd.network.html>

使用 systemd-networkd 管理工具自动永久配置静态 IP 地址

如果需要对 eth0 配置永久 IP 地址，则可以再编写一个 50-static.network 文件放置到 etc/systemd/network/目录下，50-static.network 内容如下：

```
[Match]
Name=eth0
[Network]
Address=192.168.30.100/24
Gateway=192.168.30.1
```

配置完成之后，重新启动 systemd-networkd.service，会发现 eth0 网卡的地址已经配置为设定的 192.168.30.100 了，如下所示：

```
root@myd-yf13x:~# systemctl restart systemd-networkd.service
root@myd-yf13x:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 42:0F:18:CC:11:F9
```



```
inet addr:192.168.30.100 Bcast:192.168.30.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1425 errors:0 dropped:149 overruns:0 frame:0
TX packets:194 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:182644 (178.3 KiB) TX bytes:32745 (31.9 KiB)
Interrupt:44 Base address:0x8000
```

4.2. 4G 模块

Linux 设备也可以外接 4G 模块来拨号上网，MYD-YF13X 开发板使用的是 M5700 4G 模块，下面简单的进行 4G 模块的测试。

1). 查看 VID 和 PID

```
root@myd-yf13x:~# lsusb
Bus 001 Device 003: ID 1782:4d11 Spreadtrum Communications Inc. M5700
Bus 001 Device 002: ID 1a40:0101 Terminus Technology Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

1782:4d11 : M5700 的 VID 和 PID 的信息。

2). 查看 kernel 识别模块

如果 kernel 增加了此模块的 VID 和 PID 配置，那么会生成/dev/ttyUSB*的节点：

```
root@myd-yf13x:~# ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 Apr 28 18:30 /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 1 Apr 28 18:30 /dev/ttyUSB1
crw-rw---- 1 root dialout 188, 2 Apr 28 18:30 /dev/ttyUSB2
crw-rw---- 1 root dialout 188, 3 Apr 28 18:30 /dev/ttyUSB3
crw-rw---- 1 root dialout 188, 4 Apr 28 18:30 /dev/ttyUSB4
crw-rw---- 1 root dialout 188, 5 Apr 28 18:30 /dev/ttyUSB5
crw-rw---- 1 root dialout 188, 6 Apr 28 18:30 /dev/ttyUSB6
crw-rw---- 1 root dialout 188, 7 Apr 28 18:30 /dev/ttyUSB7
```

3). 使用 AT 指令进行初步测试



使用 AT 指令可以方便的来查询信号强度，是否插入 SIM 卡，SIM 卡当前是否搜索到运营商，也可以用 AT 来打电话测试下当前卡功能。这里进行 AT 通讯还需要知道哪个设备是通讯口，这里需要查询模块文件，M5700 采用 ttyUSB5 进行 AT 通讯。这里采用 microcom 举例，也可以用 minicom。如：microcom /dev/ttyUSB5 进入模式 ctrl+x 退出。

```
root@myd-yf13x:~# microcom /dev/ttyUSB5
at
OK
```

● 查询信号质量

```
at+csq
+CSQ: 12,99
OK
```

12,99: 其中 12 是信号质量，这个数字应在 0 到 31 之间(为 99 时表示无信号)，数值越大表明信号质量越好

● 查询卡是否识别

```
at+cops?
+COPS: 0,1,"UNICOM",7
OK
```

CPIN:READY : READY 代表就绪。

● 查看运营商

```
at+cops?
+COPS: 0,1,"UNICOM",7
OK
```

UNICOM,7: UNICOM 代表联通，7 代表采用 2G，3G，4G，还是 5G 根据模块手册查看。

4). 使用 ecm 脚本拨号

```
root@myd-yf13x:~# ecm
AT+SYSNV=1,"usbmode",5
```



```
OK
AT+CGDCONT=1,"IP","CMNET"

OK
AT+CGACT=1,1

+CGACT: 1, 1, 10.250.181.217

OK
IP 10.250.181.217 available.
udhcpc: started, v1.35.0
udhcpc: broadcasting discover
udhcpc: broadcasting select for 10.250.181.217, server 192.168.1.1
udhcpc: lease of 10.250.181.217 obtained from 192.168.1.1, lease time 30840
RTNETLINK answers: File exists
/etc/udhcpc.d/50default: Adding DNS 218.104.111.122
```

5). Ping 外网测试

```
root@myd-yf13x:~# ping www.baidu.com -I eth2
PING www.a.shifen.com (112.80.248.75) from 172.22.188.79 eth2: 56(84) bytes
of data.
64 bytes from 112.80.248.75 (112.80.248.75): icmp_seq=1 ttl=55 time=42.2 ms
64 bytes from 112.80.248.75 (112.80.248.75): icmp_seq=2 ttl=55 time=30.7 ms
64 bytes from 112.80.248.75 (112.80.248.75): icmp_seq=3 ttl=55 time=25.7 ms
64 bytes from 112.80.248.75 (112.80.248.75): icmp_seq=4 ttl=55 time=37.8 ms
^C
--- www.a.shifen.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 25.709/34.114/42.211/6.350 ms
```



5. 网络应用

开发板出厂烧录的 myir-image-full 镜像默认包含了一些常见的网络应用程序，方便用户进行开发或调试。

5.1. PING

PING 主要用来测试网络的连通性，也可以测试网络延迟以及丢包率。下面使用 PING 对网络连接进行简单的测试。

1). 接线与信息输出

通过 CAT6 网线将开发板连接到交换机或路由器，控制台会显示内核输出的连接信息，如下：（通过测试 CAT6 类网线速度稳定，使用其他网线速度可能不稳定）

```
root@myd-yf13x:~#  
[ 41.932837] stm32-dwmac 5800e000.eth2 eth1: Link is Up - 1Gbps/Full - flow control off  
[ 41.939390] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
```

2). 测试外网网址

将网线接入 j18 接口，会自动获取 ip 地址，然后使用以下命令进行测试：

```
root@myd-yf13x:~# ping www.baidu.com -I eth0  
PING www.baidu.com (112.80.248.76): 56 data bytes  
64 bytes from 112.80.248.76: seq=0 ttl=55 time=12.672 ms  
64 bytes from 112.80.248.76: seq=1 ttl=55 time=12.784 ms  
64 bytes from 112.80.248.76: seq=2 ttl=55 time=13.136 ms  
64 bytes from 112.80.248.76: seq=3 ttl=55 time=12.750 ms  
^C  
--- www.baidu.com ping statistics ---  
4 packets transmitted, 4 packets received, 0% packet loss  
round-trip min/avg/max = 12.672/12.835/13.136 ms
```

注：ping 公网需要确保 DNS 正常工作。



上面结果显示 www.baidu.com 经过域名解析之后的 IP 地址为 112.80.248.76, icmp_seq 代表 icmp 包的编号, 如果编号连续说明没有丢包; time 代表响应的延迟时间, 当然这个时间越短越好。

5.2. SSH

SSH 为 Secure Shell 的缩写, 由 IETF 的网络小组 (Network Working Group) 所制定; SSH 为建立在应用层基础上的安全协议, 是较可靠, 专为远程登录会话和其他网络服务提供安全性的协议。通常 Linux 平台下使用 dropbear 或 OpenSSH 来实现 SSH 的服务端和客户端。下面在以太网连接上分别测试 SSH 客户端和服务端的使用。当前出厂默认包含 openssh v2020.81 版本提供的客户端和服务程序。

首先配置好开发板以太网接口到 SSH 服务器的连接, 配置后的以太网卡地址如下:

```
root@myd-yf13x:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 7A:F8:25:74:24:DF
          inet addr:192.168.40.220  Bcast:192.168.40.255  Mask:255.255.255.0
          inet6 addr: fe80::78f8:25ff:fe74:24df/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:428 errors:0 dropped:67 overruns:0 frame:0
          TX packets:99 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:65361 (63.8 KiB)  TX bytes:13172 (12.8 KiB)
          Interrupt:45 Base address:0x8000
```

SSH 服务器的 IP 地址为 192.168.1.13, 用 ping 命令可测试开发板和 SSH 服务器之间的连接是否正常。

● SSH 客户端测试

开发板作为客户端连接 SSH 服务器, 在开发板上使用 ssh 命令登陆服务器, 命令和结果如下:

```
root@myd-yf13x:~# ssh sur@192.168.40.242

Host '192.168.40.242' is not in the trusted hosts file.
(ssh-ed25519 fingerprint sha1!! 02:e2:70:b5:17:64:42:f3:26:9e:74:f3:9f:2f:03:fc:e4:06:d6:24)
Do you want to continue connecting? (y/n) y
```




```
sur@192.168.40.242's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-148-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

37 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Sun May  7 15:07:23 2023 from 192.168.40.122
sur@myir:~$
```

其中 sur 为服务器上的用户名。

登录成功之后，自动进入 SSH 服务器上的 console 控制台，用户就可以在客户端对远程服务器执行 sur 用户权限内的控制。如果需要退出，直接在当前控制台执行"exit"命令即可。

● SSH 服务端测试

开发板作为 SSH 服务端，其它外部设备远程连接到此台开发板。

开发板端默认启动了 SSH 服务，因此我们也可以在其它具有 SSH 客户端的外部设备（开发板或者 PC）上使用 ssh 命令登陆到当前的开发板上，命令和结果如下：

```
sur@myir:~$ ssh root@192.168.40.220
The authenticity of host '192.168.40.220 (192.168.40.220)' can't be established.
RSA key fingerprint is SHA256:OUHiUofsoGKS7A8ELAHBCfAqy6j2V6GqkfHKEYU
9fAE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.40.220' (RSA) to the list of known hosts.
root@myd-yf13x:~#
```



上面的示例中，我们从远程以 root 账户登录到了此开发板上，并进入 console 控制台，可以对开发板执行 root 用户权限内的控制。如果需要退出，直接在控制台执行"exit"命令即可。

OpenSSH 是使用 SSH 协议远程登录的主要连接工具。它加密所有流量以消除窃听、连接劫持和其他攻击。此外，OpenSSH 还提供一系列大型安全隧道功能、多种身份验证方法和复杂灵活的配置选项。用户可以根据自身需要修改位于电脑主机/etc/ssh/目录下的配置文件 ssh_config 和 sshd_config。

例如，如果希望 SSH 服务端允许 root 账户不用密码远程登录，则可以修改 SSH 服务端的/etc/ssh/sshd_config，添加下面两行配置。

```
PermitRootLogin yes
PermitEmptyPasswords yes
```

上面的配置有比较大的安全风险，一般用于调试阶段远程部署。实际产品中考虑到安全性，一般都是关掉的。

5.3. SCP

SCP 是 Secure Copy 的缩写，它是 linux 系统下基于 SSH 协议的安全的远程文件拷贝命令，在系统调试阶段非常实用。

前面已经介绍过使用 SSH 协议以及 SSH 客户端和服务端进行远程登录的示例，这里再介绍通过 SCP 命令进行文件远程拷贝的示例：

1). 从远程拷贝文件到本地

```
PC $ scp test root@192.168.40.220:/home/root
The authenticity of host '192.168.40.101 (192.168.40.220)' can't be established.
ECDSA key fingerprint is SHA256:C6fXFGctxoRu2eBuCPe04fEcSRzI82WJ1Rbbqji
kp4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.40.101' (ECDSA) to the list of known ho
sts.
test                               100% 17MB 2.9MB/s 14:28 ETA
```

进入开发板 home 目录可以看到此文件，如下：

```
root@myd-yf13x:~# ls
```



```
myb-stm32mp135x-256m.dtb myb-stm32mp135x-512m.dtb rs485_read rs485_write tempfile test ulmage watchdog_test
```

2). 从本地拷贝文件到远程

```
root@myd-yf13x:~# scp test sur@192.168.40.242:~/
The authenticity of host '192.168.40.106 (192.168.40.242)' can't be established.
ECDSA key fingerprint is SHA256:KCTJPoHzafew1fRJmTx2hV4BNymgaZ1WDg2o
vdtqtCw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.40.106' (ECDSA) to the list of known ho
sts.
sur@192.168.40.242's password:
test                               100% 17MB 2.2MB/s 14:28 ETA
```

拷贝的过程中需要按照提示输入，验证成功之后文件从开发板上拷贝到服务器上指定账户的\$HOME 目录。

通过添加“-r”参数，还可以进行目录的拷贝，具体操作请参照 scp 命令的帮助。

5.4. TFTP

TFTP 使用客户端和服务端软件在不同设备之间进行连接和传输文件，TFTP 使用的是 UDP 协议，不具备登录功能，它非常简洁，特别适合在设 备和服务端传输和备份固件，配置文件等信息。例如常见的 u-boot 中就支持 TFTP 协 议，可以通过网络加载服务器端的 Linux 系统并实现网络启动的功能。其命令语法如下：

开发主机为 Windows10 系统的 PC 机，打开 cmd 命令窗口，用 FTP 登陆到开发板，用户名为 ftp，密码任意。

```
root@myd-yf13x:~# tftp --help
BusyBox v1.35.0 () multi-call binary.

Usage: tftp [OPTIONS] HOST [PORT]
```

详细参数说明如下：

- -g : 获取文件
- -p : 上传文件



- -l : 本地文件
- -r : 远程文件
- HOST: 远程主机 IP 地址

服务端可以选择 Linux 平台下的 tftp-hpa,也可以选择 windows 平台下的 tftpd 32/64(http://tftpd32.jounin.net/tftpd32_download.html)。下面以 ubuntu 平台为例 说明 tftp 服务端的配置。

● 安装 TFTP 服务端

```
PC $ sudo apt-get install tftp-hpa tftpd-hpa
```

● 配置 TFTP 服务

创建 TFTP 服务器工作目录,并打开 TFTP 服务配置文件,如下:

```
PC $ mkdir -p /home/sur/tftpboot
PC $ chmod -R 777 /home/sur/tftpboot
PC $ sudo vi /etc/default/tftpd-hpa
```

修改或添加以下字段:

```
TFTP_DIRECTORY="/home/sur/tftpboot"
TFTP_OPTIONS="-l -c -s"
```

● 重启 TFTP 服务

```
PC $ sudo service tftpd-hpa restart
```

配置好 tftp 服务端之后, 将一个测试文件 zImage 放置到上面配置的<WORKDIR>/tftpboot/目录, 就可以在开发板上使用 tftp 客户端进行文件的下载和上传了

```
root@myd-yf13x:~# tftp -g -r zImage -l zImage 192.168.40.242
```

上面的命令会把 tftp 服务端/tftpboot 目录下的 zImage 下载到开发板当前目录下。

```
root@myd-yf13x:~# tftp -p -l config -r config_01 192.168.40.242
```

上面的命令会把开发板上当前目录下的 config 文件上传到 tftp 服务端之前配置的<WORKDIR>/tftpboot 目录下, 并重新命名为 config_01。

5.5. DHCP

DHCP (动态主机配置协议) 是一个局域网的网络协议。指的是由服务器控制一段 IP 地址范围, 客户机登录服务器时就可以自动获得服务器分配的 IP 地址和子网掩码。



DHCP 也包含服务器端和客户端两种角色，这里介绍一下使用 dhclient 命令和 udhcpd 命令手动获取 IP 地址的方法，方便用户在调试网络时使用。

用 CAT6 网线连接开发板和路由器，使用命令手动为 eth0 网卡分配 IP 地址，观察 dhcp 获取 ip 的过程。

由上面测试 log 可以观察到 DHCP 通讯获取地址的四个过程：DHCP Discover、DHCP Offer、DHCP Request、DHCP ACK。用 Ctrl+c 结束 DHCP 测试。

1). 使用 udhcpd 命令配置 IP 地址

```
root@myd-yf13x:~# udhcpd -i eth0
udhcpd: started, v1.35.0
udhcpd: broadcasting discover
udhcpd: broadcasting select for 192.168.40.220, server 192.168.40.1
udhcpd: lease of 192.168.40.220 obtained from 192.168.40.1, lease time 7200
/etc/udhcpd.d/50default: Adding DNS 192.168.40.1
```

最终可以为 eth0 配置好 IP 地址，以及网关，子网掩码，DNS 等信息如下：

```
root@myd-yf13x:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 7A:F8:25:74:24:DF
          inet addr:192.168.40.220  Bcast:192.168.40.255  Mask:255.255.255.0
          inet6 addr: fe80::78f8:25ff:fe74:24df/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:73960 errors:0 dropped:779 overruns:0 frame:0
          TX packets:57773 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:50012427 (47.6 MiB)  TX bytes:2842661 (2.7 MiB)
          Interrupt:45 Base address:0x8000

root@myd-yf13x:~# cat /etc/resolv.conf
# This is /run/systemd/resolve/resolv.conf managed by man:systemd-resolved
(8).
# Do not edit.
#
# This file might be symlinked as /etc/resolv.conf. If you're looking at
```



```
# /etc/resolv.conf and seeing this text, you have followed the symlink.
#
# This is a dynamic resolv.conf file for connecting local clients directly to
# all known uplink DNS servers. This file lists all configured search domains.
#
# Third party programs should typically not access this file directly, but only
# through the symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a
# different way, replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes
of
# operation for /etc/resolv.conf.
nameserver 192.168.40.1
search.
```

5.6. Iptables

iptables 是一个用于 IPv4 包过滤和 NAT 的管理工具。它用于设置、维护和检查 Linux 内核中的 IP 包过滤规则表。可以定义几个不同的表。每个表包含许多内置链，也可以包含用户定义的链。每个链是一个规则列表，它可以匹配一组数据包。每个规则指定如何处理匹配的数据包。

使用 Linux 系统的开发板通常使用 iptables 工具来配置防火墙。iptables 就根据包过滤规则所定义的方法来处理各种数据包，如放行 (accept)、拒绝 (reject) 和丢弃 (drop) 等。下面使用 iptables 来测试拦截 icmp 包，禁止网络上的其它外部设备对其进行 ping 探测。具体命令使用参见：<https://linux.die.net/man/8/iptables>

1). 配置开发板 iptables

在开发板上使用 iptables 配置丢弃输入的 icmp 包，不回应其他主机的 ping 探测，命令如下：

```
root@myd-yf13x:~# iptables -A INPUT -p icmp --icmp-type 8 -j DROP
root@myd-yf13x:~# iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
```



```
-A INPUT -p icmp -m icmp --icmp-type 8 -j DROP
```

2). ping 测试

在开发主机上 ping 开发板，并指定 deadline 为 10，结果如下：

```
PC$ ping 192.168.40.220 -w 10
PING 192.168.0.60 (192.168.0.60) 56(84) bytes of data.

--- 192.168.0.60 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9064ms
```

以上结果表明，设置防火墙后开发主机无法 ping 通开发板。

3). 删掉对应的防火墙规则

```
root@myd-yf13x:~# iptables -F
root@myd-yf13x:~# iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
```

4). 再次测试 ping 开发板

```
PC$ ping 192.168.40.220 -w 5
PING 192.168.0.60 (192.168.0.60) 56(84) bytes of data.
64 bytes from 192.168.0.60: icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from 192.168.0.60: icmp_seq=2 ttl=64 time=0.219 ms
64 bytes from 192.168.0.60: icmp_seq=3 ttl=64 time=0.222 ms
64 bytes from 192.168.0.60: icmp_seq=4 ttl=64 time=0.226 ms
64 bytes from 192.168.0.60: icmp_seq=5 ttl=64 time=0.238 ms
64 bytes from 192.168.0.60: icmp_seq=6 ttl=64 time=0.236 ms

--- 192.168.0.60 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4996ms
rtt min/avg/max/mdev = 0.219/0.232/0.254/0.019 ms
```



清除 iptables 规则之后，再次从开发主机 ping 开发板，就可以 ping 通了。上述示例只是一个简单的演示，实际上 iptables 配合各种规则可以实现非常强大的功能，这里就不详细介绍了。

5.7. Ethtool

ethtool 是一个查看和修改以太网设备参数的工具，在网络调试阶段具有一定的作用，下面使用该命令查看一下以太网卡的信息，并尝试修改其参数。

首先，我们通过 ethtool -h 查看该命令的帮助信息：

```
root@myd-yf13x:~# ethtool --help
ethtool version 5.16
Usage:
    ethtool [ FLAGS ] DEVNAME      Display standard information about
device
    ethtool [ FLAGS ] -s|--change DEVNAME  Change generic options
    [ speed %d ]
    [ lanes %d ]
    [ duplex half|full ]
    [ port tp|au|bnc|mii|fibre|da ]
    [ mdix auto|on|off ]
    [ autoneg on|off ]
    [ advertise %x[%x] | mode on|off ... [--] ]
    [ phyad %d ]
    [ xcvr internal|external ]
    [ wol %d[%d] | p|u|m|b|a|g|s|f|d... ]
    [ sopass %x:%x:%x:%x:%x:%x ]
    [ msglvl %d[%d] | type on|off ... [--] ]
    [ master-slave preferred-master|preferred-slave|forced-master|fo
rced-slave ]
    ethtool [ FLAGS ] -a|--show-pause DEVNAME  Show pause optio
ns
    ethtool [ FLAGS ] -A|--pause DEVNAME  Set pause options
```

查看开发板以太网卡的基本信息：




```
root@myd-yf13x:~# ethtool eth0
Settings for eth0:

    Supported ports: [ TP        MII ]
    Supported link modes:   10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
    Supported pause frame use: Symmetric Receive-only
    Supports auto-negotiation: Yes
    Supported FEC modes: Not reported
    Advertised link modes:  10baseT/Half 10baseT/Full
                           100baseT/Half 100baseT/Full
                           1000baseT/Full
    Advertised pause frame use: Symmetric Receive-only
    Advertised auto-negotiation: Yes
    Advertised FEC modes: Not reported
    Speed: 1000Mb/s
    Duplex: Full
    Auto-negotiation: on
    Port: Twisted Pair
    PHYAD: 0
    Transceiver: internal
    MDI-X: Unknown
    Supports Wake-on: ug
    Wake-on: d
    Current message level: 0x0000003f (63)
                           drv probe link timer ifdown ifup

    Link detected: yes
```

通过 ethtool 命令可以查看到当前以太网卡支持的连接模式为十兆，百兆和千兆半双工与全双工六种模式，当前连接状态为协商的千兆，全双工模式，使用 MII 接口等等。

可以使用 ethtool 工具对以太网的参数进行设置，这些在进行以太网调试和诊断的时候有一定的作用，例如我们强制将以太网设置为百兆全双工，并且关闭自协商，命令如下：



```
root@myd-yf13x:~# ethtool -s eth0 speed 100 duplex full autoneg off
[ 3012.546094] stm32-dwmac 5800a000.eth1 eth0: Link is Down
[ 3015.692806] stm32-dwmac 5800a000.eth1 eth0: Link is Up - 100Mbps/Full
- flow control off
```

关于 ethtool 的更多说明请参考: <http://man7.org/linux/man-pages/man8/ethtool.8.html>

5.8. iperf3

iperf3 是在 IP 网络上主动测量最大可实现带宽的工具。它支持调节测试时间、缓冲区大小和协议(IPV4 和 IPV6 下的 TCP、UDP、SCTP)等各种参数。iperf3 按角色可以分为服务端模式或客户端模式, 我们可以用它来测试和查看 TCP 模式下的网络带宽, TCP 窗口值, 重传的概率等, 也可以测试指定 UDP 带宽下丢包率, 延迟和抖动情况。

在开发主机上打开 Windows PowerShell, 带千兆网卡的服务器作为 iperf3 的服务端, 被测试的开发板作为客户端分别测试开发板网卡 TCP 和 UDP 的性能。

将服务器和开发板通过 CAT6 网线直连, 并配置好各自的 IP 地址。例如我们设置服务器 ip 为 192.168.40.120, 设开发板 IP 为 192.168.40.110, 并使用 ping 命令测试确保它们之间是连通的。

注意: 尽量不要连接路由器或交换机, 以免测试结果受到中间设备传输转发的影响。

1). 测试 TCP 性能

服务端 (192.168.40.120)

服务器上 iperf3 使用-s 参数表示工作在服务端模式。

```
PC $ $ iperf3 -s -i 2
```

```
-----
Server listening on 5201
-----
```

客户端 (192.168.40.110)

开发板上运行的 iperf3 程序工作在客户端, TCP 模式, 其中参数说明如下:

- -c 192.168.40.120 : 工作在客户端, 连接服务端 192.168.40.120
- -i 2 : 测试结果报告时间间隔为 2 秒
- -t 10 : 总测试时长为 10 秒



```
root@myd-yf13x:~# iperf3 -c 192.168.40.120 -i 2 -t 10
Connecting to host 192.168.40.120, port 5201
[ 5] local 192.168.40.110 port 46604 connected to 192.168.40.120 port 5201
[ ID] Interval            Transfer        Bitrate          Retr  Cwnd
[ 5]  0.00-2.01      sec    199 MBytes     831 Mbits/sec     0    214 KBytes
[ 5]  2.01-4.00      sec    197 MBytes     828 Mbits/sec     0    214 KBytes
[ 5]  4.00-6.01      sec    201 MBytes     842 Mbits/sec     0    214 KBytes
[ 5]  6.01-8.01      sec    199 MBytes     836 Mbits/sec     0    214 KBytes
[ 5]  8.01-10.01     sec    200 MBytes     836 Mbits/sec     0    214 KBytes

-----
[ ID] Interval            Transfer        Bitrate          Retr
[ 5]  0.00-10.01     sec    996 MBytes     835 Mbits/sec     0
[ 5]  0.00-10.01     sec    996 MBytes     835 Mbits/sec
r

iperf Done.
```

客户端经过 10 秒之后测试结束并显示上面的测试结果，表明 TCP 带宽为 830Mbps 左右，没有重传，测试时 TCP 窗口值为 560KBytes.

同时服务端也显示测试结果如下，然后继续监听 5201 端口等待客户端连接：

```
PC $ iperf3 -s -i 2
-----
Accepted connection from 192.168.40.110, port 46594
[ 5] local 192.168.40.120 port 5201 connected to 192.168.40.110 port 46604
[ ID] Interval            Transfer        Bandwidth
[ 5]  0.00-2.00      sec    193 MBytes     811 Mbits/sec
[ 5]  2.00-4.00      sec    197 MBytes     826 Mbits/sec
[ 5]  4.00-6.00      sec    201 MBytes     842 Mbits/sec
```



```
[ 5] 6.00-8.00 sec 200 MBytes 837 Mbits/sec
[ 5] 8.00-10.00 sec 200 MBytes 837 Mbits/sec
[ 5] 10.00-10.06 sec 5.70 MBytes 857 Mbits/sec
-----
[ ID] Interval          Transfer      Bandwidth
[ 5] 0.00-10.06 sec 0.00 Bytes 0.00 bits/sec      sender
[ 5] 0.00-10.06 sec 996 MBytes 831 Mbits/sec      receiver
-----
Server listening on 5201
```

2). 测试 UDP 性能

服务端 (192.168.40.120)

服务器上继续运行 iperf3 使用-s 参数表示工作在服务端模式。

```
PC $ $ iperf3 -s -i 2
-----
Server listening on 5201
-----
```

客户端 (192.168.40.110)

开发板上 iperf3 工作在客户端，UDP 模式，其中参数说明如下：

- -u：工作在 UDP 模式
- -c 192.168.40.120：工作在客户端，连接服务端 192.168.40.120
- -i 2：测试结果报告时间间隔为 2 秒
- -t 10：总测试时长为 10 秒
- -b 100M：设定 UDP 传输带宽为 100Mbps.

```
root@myd-yf13x:~# iperf3 -u -c 192.168.40.120 -i 2 -t 10 -b 100M
Connecting to host 192.168.40.120, port 5201
[ 5] local 192.168.40.110 port 43355 connected to 192.168.40.120 port 5201
[ ID] Interval          Transfer      Bitrate        Total Datagrams
[ 5] 0.00-2.00 sec 23.8 MBytes 100 Mbits/sec 17262
[ 5] 2.00-4.00 sec 23.8 MBytes 100 Mbits/sec 17263
[ 5] 4.00-6.00 sec 23.8 MBytes 100 Mbits/sec 17266
[ 5] 6.00-8.00 sec 23.8 MBytes 100 Mbits/sec 17265
```



```
[ 5] 8.00-10.00 sec 23.8 MBytes 100 Mbits/sec 17265
-----
[ ID] Interval          Transfer      Bitrate          Jitter    Lost/Total Datagrams
ms
[ 5] 0.00-10.00 sec 119 MBytes 100 Mbits/sec 0.000 ms 0/86321
(0%) sender
[ 5] 0.00-10.00 sec 119 MBytes 100 Mbits/sec 0.040 ms 12/86321
(0.014%) receiver

iperf Done.
```

客户端经过 10 秒之后测试结束并显示上面的测试结果，表明 UDP 在指定带宽为 100 Mbps 时没有丢包。

同时服务端也显示测试结果如下，然后继续监听 5201 端口等待客户端连接：

```
PC $ iperf3 -s -i 2
-----
Server listening on 5201
-----
Accepted connection from 192.168.40.110, port 39812
[ 5] local 192.168.40.120 port 5201 connected to 192.168.40.110 port 43355
[ ID] Interval          Transfer      Bandwidth          Jitter    Lost/Total Datagrams
[ 5] 0.00-2.00 sec 23.3 MBytes 97.6 Mbits/sec 0.057 ms 12/16856 (0.071%)
[ 5] 2.00-4.00 sec 23.8 MBytes 100 Mbits/sec 0.050 ms 0/17265 (0%)
[ 5] 4.00-6.00 sec 23.8 MBytes 100 Mbits/sec 0.036 ms 0/17266 (0%)
[ 5] 6.00-8.00 sec 23.8 MBytes 100 Mbits/sec 0.045 ms 0/17264 (0%)
[ 5] 8.00-10.00 sec 23.8 MBytes 100 Mbits/sec 0.060 ms 0/17267 (0%)
[ 5] 10.00-10.05 sec 570 KBytes 99.4 Mbits/sec 0.040 ms 0/403 (0%)
-----
[ ID] Interval          Transfer      Bandwidth          Jitter    Lost/Total Datagrams
```



```
[ 5] 0.00-10.05 sec 0.00 Bytes 0.00 bits/sec 0.040 ms 12/86321 (0.014%)
```

客户端修改-b 参数，继续增大指定的 UDP 带宽，当开始出现丢包时测到的 UDP 带宽即为实际的 UDP 带宽。如果指定带宽达到以太网卡最高的 1Gbps，仍然没有丢包，那么 iperf3 测试返回的 bandwidth 即为实际的 UDP 带宽。例如下面的例子指定带宽为 1000Mbps，仍然没有丢包，但实际带宽其实为 300Mbps 左右。

```
root@myd-yf13x:~# iperf3 -u -c 192.168.40.120 -i 2 -t 10 -b 1000M
Connecting to host 192.168.40.120, port 5201
[ 5] local 192.168.40.110 port 41567 connected to 192.168.40.120 port 5201
```

[ID]	Interval		Transfer	Bitrate	Total Datagrams
[5]	0.00-2.00	sec	72.9 MBytes	306 Mbites/sec	52808
[5]	2.00-4.00	sec	72.3 MBytes	303 Mbites/sec	52376
[5]	4.00-6.00	sec	72.3 MBytes	303 Mbites/sec	52325
[5]	6.00-8.00	sec	72.2 MBytes	303 Mbites/sec	52299
[5]	8.00-10.00	sec	71.8 MBytes	301 Mbites/sec	51969

```
-----
```

[ID]	Interval		Transfer	Bitrate	Jitter	Lost/Total Datagrams
[5]	0.00-10.00	sec	361 MBytes	303 Mbites/sec	0.000 ms	0/261777

(0%) sender

[5]	0.00-10.00	sec	361 MBytes	303 Mbites/sec	0.033 ms	0/261777

(0%) receiver

```
iperf Done.
```

iperf3 在测试的过程中还有很多参数可以配置，用户可以根据实际应用需要进行有针对性的调整测试。比如可以增大-t 参数的值进行长时间压力测试，或者指定-P 参数进行多个连接并发的压力测试等。关于 iperf3 测试的更多信息请参考：<https://iperf.fr/iperf-doc.php#3doc>



6. 图形系统

Linux 图形系统是 Linux 系统中比较复杂的一个子系统，一直处于不停的变革当中。它位于底层显示相关的设备驱动和上层用户界面应用之间，屏蔽了形形色色的底层硬件差异，同时又向上层用户界面提供统一的 API。

Linux/Unix 环境下最早的图形系统是 Xorg 图形系统，随着软硬件的发展，特别是嵌入式系统的发展，Xorg 显得过于笨重，随之又出现了一些更简洁，易于开发和维护的图形系统，比如 Wayland (<https://wayland.freedesktop.org/>)。下面是一个典型的嵌入式系统下图形系统的结构图。

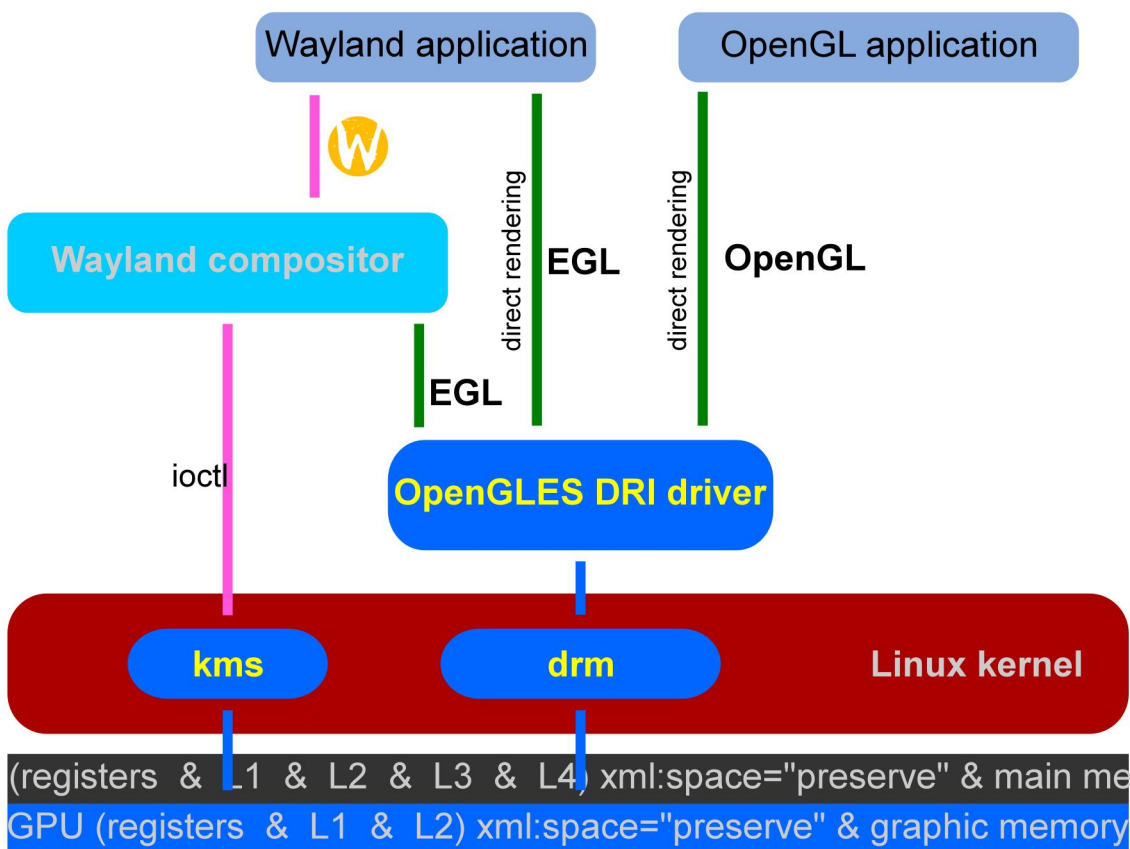


图 6-1. Embedded Linux Graphics Stack Overview

(by Shmuel Csaba Otto Traian; GFDL 1.2+ and CC-BY-SA 3.0+; created 2013-11-04)

上图可以看出，Wayland 并非进行图形界面开发的必需组件，例如我们出厂镜像中包含的 MEasy HMI V2.x 就可以通过指定平台插件直接在 linuxfb 上运行 Qt5。以下章节将针对图形系统中部分关键内容进行测试和说明。



6.1. Wayland

本节主要介绍 wayland/weston。Wayland/Weston 框架专用于显示器的管理，包括其内容的组成，其相关输入外设事件的支持(触摸屏、鼠标、键盘...)和其设置(背景壁纸、分辨率、多屏...):

- Wayland 是一个协议，它指定了显示服务器与应用层客户端之间的通信。Wayland 的目的是作为一个更简单的 X 窗口系统的替代品，更容易开发和维护。
- Weston 是一个 Wayland 复合程序的参考实现，它是一个使用 Wayland 协议的显示服务器。Weston 是一个最小化和快速的组合程序，适合于许多嵌入式和移动用例。

1). 启动 weston 显示服务器

```
root@myd-yf13x:~# systemctl restart weston-launch
```

6.2. QT

QT 是一种跨平台 C++图形用户界面应用程序开发框架。它既可以开发 GUI 程序，也可用于开发非 GUI 程序，比如控制台工具和服务器。Qt 是面向对象的框架，使用特殊的代码生成扩展以及一些宏，Qt 很容易扩展，并且允许真正地组件编程。

开发板会在出厂的时候烧写带有 Qt 运行时库的系统，并且提供了一个丰富的 HMI 演示系统，具体内容可以查看《MYD-YF13X_QT 及 MEasy HMI2.0 软件开发指南》。

1). 获取 qt 的信息

首先查看当前系统支持的 QT 版本，如下：

```
root@myd-yf13x:~# qmake -v
QMake version 3.1
Using Qt version 5.15.3 in /usr/lib
```

2). QT 运行环境介绍

在运行 Qt 应用程序时，可以根据不同的软硬件要求，对 Qt 的运行环境，如平台插件，显示参数，输入设备以及光标指针等进行适当的配置。

● 平台插件配置

在嵌入式 Linux 系统上，可以使用多个平台插件：EGLFS，LinuxFB，DirectFB 或 Wayland。但是，这些插件的可用性取决于实际硬件平台的特性以及 Qt 的配置方式。EGLFS 是许多主板上的默认插件，实际测试中 EGLFS 不合适，所以采用 Linuxfb。使用 QT_QP



A_PLATFORM 环境变量来请求另一个插件。另外，对于快速测试，请使用-platform 具有相同语法的命令行参数。例如在用户控制台下，可以使用如下命令进行配置：

```
root@myir-yf13x:~# export QT_QPA_PLATFORM=linuxfb
root@myir-yf13x:~# /home/mxapp2
```

或者使用“-platform linuxfb”指令平台插件，如下所示：

```
root@myir-yf13x:~# /home/mxapp2 -platform linuxfb
```

注意：从 Qt 5.0 开始，Qt 不再具有自己的窗口系统（QWS）实现，因此不再支持 Qt 早期版本上使用的 -qws 参数。

● 输入外设配置

当嵌入式 Linux 设备上没有窗口系统（例如 XWindow 或 Weston）时，鼠标，按键或者触目设备通过直接读取 evdev 或者使用其它中间库获取输入设备信息，如 libinput 或 tslib。eglfs 和 linuxfb 平台插件包含这两种输入方式。关于 Qt5 输入设备配置如下：

linuxfb 平台插件默认使用的是 EvdevTouch 输入处理程序，这种方式常用于处理电容触摸，电容触摸驱动上报的事件坐标与实际屏幕区域坐标完全对应的话，就不需要做额外的处理，如果出现反向，则可以通过环境变量进行一些调整，EvdevTouch 输入处理程序支持以下一些额外参数：

表 6-1. QT linuxfb 插件触摸处理程序相关的环境变量参数

参数	描述
/dev/input/...	指定输入设备的名称。如果未指定，Qt 将通过 libudev 或遍历可用节点来寻找合适的设备。
rotate	在某些触摸屏上，必须通过将坐标设置 rotate 为 90、180 或 270 来旋转坐标。
invertx/inverty	指定用于在输入事件中反转 X 或 Y 坐标的参数。

例如，如果 QT_QPA_EVDEV_TOUCHSCREEN_PARAMETERS 在启动应用程序之前将以下值传递给平台插件，则明确指定触摸设备为/dev/input/event1，其坐标翻转 180 度。当实际屏幕和触摸屏的方向不匹配时，这很有用。

```
export QT_QPA_EVDEV_TOUCHSCREEN_PARAMETERS=/dev/input/event1:rotate=180
```



如果要启用 tslib 支持，需要将 QT_QPA_EGLFS_TSLIB (for eglfs) 或 QT_QPA_FB_TSLIB (for linuxfb) 环境变量设置为 1。关于 tslib 的具体使用方法参考 <https://github.com/libts/tslib/blob/master/README.md>

注意：tslib 输入处理程序常用于电阻触摸，会生成鼠标事件并仅支持单点触摸，初始使用还需要进行屏幕校准。

3). 启动 Qt 程序

通常使用触摸的设备都不需要显示鼠标指针，用户可以在执行应用前设置环境变量将鼠标指针隐藏。

使用的是 linuxfb 平台插件，设置如下：

```
export QT_QPA_FB_HIDE_CURSOR=1
```

MYD-YF13X 开发板出厂的 myir-image-full 镜像带有 MEasy HMI 演示程序，使用 linuxfb 平台插件和 EvdevTouch 触摸处理程序，对应启动命令程序如下：

```
root@myir-yf13x:~# /home/mxapp2 -platform linuxfb &
root@myir-yf13x:~# qt.qpa.input: xkbcommon not available, not performing key mapping
qml: index=0
qml: currentIndex=0
qml: index=0
stop !
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
```

mxapp2 默认已经启动，如果需要运行自己的 Qt 应用程序，需要先终止 mxapp2 再启动其他应用。

可以通过 kill 方式对进程直接退出

```
root@myir-yf13x:~# killall mxapp2
```



7. 多媒体应用

7.1. Camera

本节采用系统自带 gstreamer, v4l2-utils, media-ctl 命令对米尔 DVP ov2659 摄像头 (型号 MY-CAM011B) 进行评估测试。主要测试摄像头的预览、抓帧 (拍照)。

1). 摄像头窗口预览、拍照测试

- 查询开发板 ip (此 IP 需要能访问外网)

```
root@myir-yf13x:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr AE:7D:A5:7E:C4:A2
          inet addr:192.168.40.146  Bcast:192.168.40.255  Mask:255.255.255.0
          inet6 addr: fe80::ac7d:a5ff:fe7e:c4a2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1669 errors:0 dropped:102 overruns:0 frame:0
          TX packets:440 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:243972 (238.2 KiB)  TX bytes:58552 (57.1 KiB)
          Interrupt:44 Base address:0x8000
```

- 执行预览命令

```
root@myir-yf13x:~# uvc_stream -d /dev/video0 -y -P 123456 -r 640x480
Using V4L2 device: /dev/video0
Format: YUYV
JPEG quality: 40
Resolution: 640 x 480 @ 5 fps
TCP port: 8080 user: uvc_user pass: *****
```

- 打开预览网页

打开网页输入:

192.168.40.146:8080/stream.mjpeg

输入的 ip 地址需要与前面的 ip 地址一致。进入该网页时会提示输入用户名和密码, 这里随意填即可。



- 执行拍照命令

```
root@myir-yf13x:~# v4l2grab -d /dev/video0 -W 640 -H 480 -o 1.jpeg
Unable to set frame interval.
```

此时会在当前目录生成 1.jpeg 文件，将此文件移至 windows 电脑后，用看图软件打
卡即可。

7.2. Audio

这节是测试播放音频。有两种方式，一种是 HMI2.0 应用直接播放，一种是通过 aplay
工具手动播放音频。

1). HMI2.X 音乐播放

HMI 多媒体音乐播放请参考 HMI 手册 《MYD-YF13X_QT 及 MEasy HMI2.0 软件开
发指南》。

2). 调节音量

```
root@myir-yf13x:~# amixer cset name='PCM Playback Volume' 180
numid=1,iface=MIXER,name='PCM Playback Volume'
; type=INTEGER,access=rw---R--,values=2,min=0,max=192,step=0
: values=192,192
amixer: Control default element TLV read error: No such device or address
```

从上面信息可以看出音量最小值为：0，最大值为：192，这里调节音量为 180

3). 手动播放音乐

播放 wav 格式的音乐

```
root@myir-yf13x:~# aplay 01+Singalongsong.wav
Playing WAVE '01+Singalongsong.wav' : Signed 16 bit Little Endian, Rate 441
00 Hz, Stereo
```

7.3. Video

1). HMI2.0 视频播放

HMI 多媒体视频播放请参考 HMI 手册 《MYD-YF13X_QT 及 MEasy HMI2.0 软件开
发指南》。



8. 系统工具

默认映像中包含了一些常用的系统工具，便于用户在系统调试或实际部署的产品中查看和管理系统的各种资源，也可以在 SHELL 脚本或其他应用程序中调用。这些工具可能不完全满足用户的系统定制需求，此时系统开发人员需要根据实际情况做出适当的调整。

8.1. 压缩解压工具

本节主要测试系统的解压缩工具。压缩是可以把多个文件压缩成一个压缩包可以把多个文件压缩成一个压缩包，方便进行文件的传输。而解压可以把经过压缩的压缩文件还原成原始大小方便使用。本节将在文件系统以 tar、gzip、gunzip 等工具为例进行说明。

1). tar 工具

现在我们在 Linux 中使用的 tar 工具，它不仅可以对文件打包，还可以对其进行压缩，查看，添加以及解压等一系列操作。这里是将打包操作。

- 语法格式

输入以下命令查看 tar 语法格式：

```
root@myir-yf13x:~# tar --help
Usage: tar [OPTION...] [FILE]...
GNU 'tar' saves many files together into a single tape or disk archive, and can
restore individual files from the archive.
```

Examples:

```
tar -cf archive.tar foo bar # Create archive.tar from files foo and bar.
tar -tvf archive.tar        # List all files in archive.tar verbosely.
tar -xf archive.tar         # Extract all files from archive.tar.
```

详细参数说明如下：

- -c : 建立一个压缩文件的参数指令(create 的意思);
- -x : 解开一个压缩文件的参数指令!
- -t : 查看 tarfile 里面的文件! 特别注意, 在参数的下达中, c/x/t 仅能存在一个! 不可同一时候存在! 由于不可能同一时候压缩与解压缩。
- -z : 是否同一时候具有 gzip 的属性? 亦即是否须要用 gzip 压缩?



- -j : 是否同一时候具有 bzip2 的属性? 亦即是否须要用 bzip2 压缩?
- -v : 压缩的过程中显示文件! 这个经常使用, 但不建议用在背景运行过程!
- -f : 使用档名, 请留意, 在 f 之后要马上接档名, 不要再加参数! 比如使用『tar -zcvfP tfile sfile』就是错误的写法, 要写成『tar -zcvPf tfile sfile』才对。
- -p : 使用原文件的原来属性 (属性不会根据使用者而变)
- -P : 能够使用绝对路径来压缩!
- -N : 比后面接的日期(yyyy/mm/dd)还要新的才会被打包进新建的文件里!
- --exclude FILE: 在压缩的过程中, 不要将 FILE 打包!

● 使用 tar 压缩

新建 test.txt 文件, 并输入以下命令将文件打包成.gz 格式:

```
root@myir-yf13x:~# tar -czf test.tar.gz test.txt
root@myir-yf13x:~# ls
OpenAMP_TTY_echo.elf  rs485_write  test.tar.gz  uart_test
rs485_read            test.c       test.txt     wifi.conf
```

所以上面加 z 参数, 则以 .tar.gz 或 .tgz 来代表 gzip 压缩过的 tar file 。

● 使用 tar 解压

把打包成 tar.gz 格式文件解压:

```
root@myir-yf13x:~# tar -xvf test.tar.gz
test.txt
root@myir:~# ls
OpenAMP_TTY_echo.elf  rs485_write  test.tar.gz  uart_test
rs485_read            test.c       test.txt     wifi.conf
```

2). gzip 压缩工具

gzip 是在 Linux 系统中经常使用的一个对文件进行压缩和解压缩的命令, 既方便又好用。

● 语法格式

在开发板终端输入以下命令查看 gzip 语法:

```
root@myir-yf13x:~# gzip --help
BusyBox v1.31.1 () multi-call binary.
Usage: gzip [-cfkdt] [FILE]...
```



- 用 **gzip** 把文件压缩

```
root@myir-yf13x:~# gzip test.txt
root@myir-yf13x:~# ls
OpenAMP_TTY_echo.elf  rs485_write  test.tar.gz  uart_test
rs485_read            test.c       test.txt.gz  wifi.conf
```

- 用 **gunzip** 工具解压

用 gunzip 把文件解压，示例如下：

```
root@myir-yf13x:~# gunzip test.txt.gz
root@myir-yf13x:~# ls
test.tar.gz  test.c  test.txt
```

8.2. 文件系统工具

主要测试系统的文件系统工具，本节将介绍几种常见的文件系统管理工具。系统自带的文件系统工具 mount、mkfs、fsck、dumpe2fs。

1). mount 挂载工具

mount 是 Linux 下的一个命令，它可以将分区挂接到 Linux 的一个文件夹下，从而将分区和该目录联系起来，因此我们只要访问这个文件夹，就相当于访问该分区了，其应用语法格式如下：

```
root@myir-yf13x:~# mount -h

Usage:
mount [-lhV]
mount -a [options]
mount [options] [--source] <source> | [--target] <directory>
mount [options] <source> <directory>
mount <operation> <mountpoint> [<target>]

Mount a filesystem.
```

挂载 sd 卡第 1 个分区示例如下：

```
root@myir-yf13x:~# mount /dev/mmcblk0p1 /mnt/
```

2). mkfs 格式工具



硬盘分区后，下一步的工作是 Linux 文件系统的建立。类似于 Windows 下的格式化硬盘。在硬盘分区上建立文件系统会冲掉分区上的数据，而且不可恢复，因此在建立文件系统之前要确认分区上的数据不再使用。建立文件系统的命令是 mkfs,应用语法格式如下：

```
root@myir-yf13x:/mnt# mkfs -h
Usage:
mkfs [options] [-t <type>] [fs-options] <device> [<size>]

Make a Linux filesystem.

Options:
-t, --type=<type>  filesystem type; when unspecified, ext2 is used
fs-options         parameters for the real filesystem builder
<device>           path to the device to be used
<size>             number of blocks to be used on the device
-V, --verbose      explain what is being done;
                   specifying -V more than once will cause a dry-run
-h, --help         display this help
-V, --version      display version

For more details see mkfs(8).
```

格式化 sd 卡第 1 个分区示例如下：

```
root@myir-yf13x:~# mkfs -t ext3 -V -c /dev/mmcblk0p1
mkfs from util-linux 2.37.4
mkfs.ext3 -c /dev/mmcblk0p1
mke2fs 1.46.5 (30-Dec-2021)
/dev/mmcblk0p1 contains a vfat file system labelled 'Volumn'
Proceed anyway? (y,N) y
[ 108.937869] mmc_erase: erase error -110, status 0x0
Discarding device blocks: done
Creating filesystem with 20480 1k blocks and 5112 inodes
Filesystem UUID: fd6238ad-340e-447b-b40d-2115472289ee
Superblock backups stored on blocks:
```



8193

Checking for bad blocks (read-only test): done

Allocating group tables: done

Writing inode tables: done

Creating journal (1024 blocks): done

Writing superblocks and filesystem accounting information: done

3). fsck 文件修复工具

fsck 命令主要用于检查文件系统的正确性，当文件系统发生错误时，可用 fsck 指令尝试加以修复。并对 Linux 磁盘进行修复。例如：

```
root@myir-yf13x:~# fsck -a /dev/mmcbk0p1
fsck from util-linux 2.37.4
/dev/mmcbk0p1: clean, 11/5112 files, 2490/20480 blocks
```

4). dumpe2fs

打印特定设备上现存的文件系统的超级块(super block)和块群(blocks group)的信息。开发板输入以下命令查看应用语法：

```
root@myir-yf13x:~# dumpe2fs -h
dumpe2fs 1.46.5 (30-Dec-2021)
Usage: dumpe2fs [-bfghimxV] [-o superblock=<num>] [-o blocksize=<num>]
device
```

查看文件系统格式化后的详细属性，例如，输入命令查看某个磁盘的详细信息：

```
root@myir-yf13x:~# dumpe2fs /dev/mmcbk0p1
dumpe2fs 1.46.5 (30-Dec-2021)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          fd6238ad-340e-447b-b40d-2115472289ee
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype sp
arse_super large_file
```



```
Filesystem flags:      unsigned_directory_hash
Default mount options: user_xattr acl
Filesystem state:     clean
Errors behavior:      Continue
Filesystem OS type:   Linux
Inode count:          5112
Block count:          20480
Reserved block count: 1024
Overhead clusters:    2471
Free blocks:          17990
Free inodes:           5101
First block:          1
```

.....

Group 0: (Blocks 1-8192)

Primary superblock at 1, Group descriptors at 2-2

Reserved GDT blocks at 3-81

Block bitmap at 82 (+81)

Inode bitmap at 83 (+82)

Inode table at 84-509 (+83)

6640 free blocks, 1693 free inodes, 2 directories

Free blocks: 1553-8192

Free inodes: 12-1704

查看某磁盘的 inode 数量，inode 也会消耗硬盘空间，所以硬盘格式化的时候，操作系统自动将硬盘分成两个区域。一个是数据区，存放文件数据；另一个是 inode 区 (inode table)，存放 inode 所包含的信息。

```
root@myir-yf13x:~# dumpe2fs /dev/mmcblk0p1 | grep -i "inode size"
dumpe2fs 1.46.5 (30-Dec-2021)
Inode size:          256
```

查看某磁盘的 block 数量，操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个"块" (block)。这种由多个扇区组成的"块"，是文件存取的最小单位。



```
root@myir-yf13x:~# dumpe2fs /dev/mmcblk0p1 | grep -i "block size"
dumpe2fs 1.46.5 (30-Dec-2021)
Block size:                1024
```

8.3. 磁盘管理工具

主要测试系统的磁盘管理工具，本节将介绍几种常见的磁盘管理工具。系统自带的磁盘管理工具 fdisk、dd、mkfs、du、df 等。通过这些命令可以监控平时的磁盘使用情况。

1). fdisk 磁盘分区工具

fdisk 磁盘分区工具在 DOS、Windows 和 Linux 中都有相应的应用程序。在 Linux 系统中，fdisk 是基于菜单的命令。用 fdisk 对硬盘进行分区，可以在 fdisk 命令后面直接加上要分区的硬盘作为参数，其应用语法格式如下：

```
root@myir-yf13x:~# fdisk -h
Usage:
fdisk [options] <disk>      change partition table
fdisk [options] -l [<disk>] list partition table(s)
```

● 对 eMMC 进行分区：

```
root@myir-yf13x:~# fdisk /dev/mmcblk1p9
Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
The old ext4 signature will be removed by a write command.
Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xd2dcd6ef.
Command (m for help):
```

2). dd 拷贝命令

dd 命令用于将指定的输入文件拷贝到指定的输出文件上。并且在复制过程中可以进行格式转换。dd 命令与 cp 命令的区别在于：dd 命令可以在没有创建文件系统的软盘上进行，拷贝到软盘的数据实际上是镜像文件。类似于 DOS 中的 diskcopy 命令的作用。dd 命令的格式为：dd [<if=输入文件名/设备名>] [<of=输出文件名/设备名>] [bs=块字节大小] [count = 块数]



创建一个大小为 100M 的文件示例如下：

```
root@myir-yf13x:~# time dd if=/dev/zero of=temp bs=1M count=100 conv=
fsync
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 9.09095 s, 11.5 MB/s
real    0m 9.09s
user    0m 0.00s
sys     0m 1.16s
```

● du 磁盘用量统计工具

du 命令用于显示磁盘空间的使用情况。该命令逐级显示指定目录的每一级子目录占用文件系统数据块的情况。du 一般使用语法如下：

```
root@myir-yf13x:~# du --help
Usage: du [OPTION]... [FILE]...
    or: du [OPTION]... --files0-from=F
Summarize disk usage of the set of FILEs, recursively for directories.
```

部分参数说明：

- -a:显示所有目录或文件的大小
- -h:以 K,M,G 为单位，提高信息可读性
- -k:以 KB 为单位输出
- -m:以 MB 为单位输出

统计 dd 命令生成的文件大小：

```
root@myir-yf13x:~# du temp
102400    temp
root@myir-yf13x:~# du -h temp
100M     temp
```

3). df 磁盘统计工具

用于显示目前在 Linux 系统上的文件系统的磁盘使用情况统计，一般用法如下：

```
root@myir-yf13x:~# df --help
Usage: df [OPTION]... [FILE]...
```



Show information about the file system on which each FILE resides, or all file systems by default.

部分参数说明:

- -h: 可以根据所使用大小使用适当的单位显示
- -i: 查看分区下 inode 的数量和 inode 的使用情况
- -T: 打印出文件系统类型

查看分区下 inode 的数量和 inode 的使用情况, 使用如下命令:

```
root@myir-yf13x:~# df -i
Filesystem      Inodes IUsed   IFree IUse% Mounted on
devtmpfs         47667   486   47181    2% /dev
/dev/mmcblk1p8  723840 24114  699726    4% /
tmpfs            56483     3   56480    1% /dev/shm
tmpfs           819200   714  818486    1% /run
tmpfs           1024     13    1011    2% /sys/fs/cgroup
tmpfs          1048576   22 1048554    1% /tmp
/dev/mmcblk1p6   16384     22   16362    1% /boot
/dev/mmcblk1p7   4096     22    4074    1% /vendor
tmpfs           56483    27   56456    1% /var/volatile
tmpfs           11296    14   11282    1% /run/user/0
```

inode 是我们在格式化的时候系统给我们划分好的, inode 与磁盘分区大小有关。当我们的 inode 使用已经达到百分百时, 即使我们的磁盘空间还是有剩余, 也是写不了数据到磁盘的。

8.4. 进程管理工具

进程也是操作系统中的一个重要概念, 它是一个程序的一次执行过程, 程序是进程的一种静态描述, 系统中运行的每一个程序都是在它的进程中运行的。Linux 系统中所有的进程是相互联系的, 除了初始化进程外, 所有进程都有一个父进程。新的进程不是被创建, 而是被复制, 或是从以前的进程复制而来。Linux 中所有的进程都是由一个进程号为 1 的 init 进程衍生而来的。Linux 系统包括 3 种不同类型的进程, 每种进程都有自己的特点和属性:

- 交互进程: 由一个 Shell 启动的进程, 既可以在前台运行, 又可以在后台运行。



- 批处理进程：这种进程和终端没有联系，是一个进程序列。这种进程被提交到等待队列顺序执行的进程。
- 监控进程(守护进程)：守护进程总是活跃的，一般是在后台运行，守护进程一般是由系统在开始时通过脚本自动激活启动或 root 启动
- 对于 linux 系统来说，进程的管理是重要的一环，对于进程的管理通常是通过进程管理工具实现的，Linux 系统中比较常用的进程管理命令有以下几种： ps top vmstat kill 。

1). ps 显示当前进程工具

显示当前系统进程的运行情况，一般语法如下：

```
root@myir-yf13x:~# ps --help

Usage:
ps [options]

Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.

For more details see ps(1)
```

部分参数组合说明：

- -u：以用户为中心组织进程状态信息显示；
- -a：与终端无关的进程；
- -x：与终端有关的进程；（线程，就是轻量级进程；）

通常上面命令组合使用：aux。

- --e：显示所有进程；相当于 ax；
- -f：显示完整格式程序信息；

通常以上命令组合使用：ef

- -H：以进程层级显示进程数的
- -F：显示更多的程序信息

通常组合使用命令：eHF



显示所有进程的信息情况示例如下：

```
root@myir-yf13x:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
MMAND
root         1  0.2  1.3   8132   5876 ?        Ss   21:39   0:04 /sbin/init
root         2  0.0  0.0      0      0 ?        S    21:39   0:00 [kthreadd]
root         3  0.0  0.0      0      0 ?        I<   21:39   0:00 [rcu_gp]
root         4  0.0  0.0      0      0 ?        I<   21:39   0:00 [rcu_par_gp]
root         5  0.0  0.0      0      0 ?        I<   21:39   0:00 [netns]
root         8  0.0  0.0      0      0 ?        I    21:39   0:01 [kworker/u2:0
-events_unbound]
root         9  0.0  0.0      0      0 ?        I<   21:39   0:00 [mm_percpu_
wq]
root        10  0.0  0.0      0      0 ?        S    21:39   0:00 [rcu_tasks_kt
hre]
```

对其上面一些任务栏进行简单解释说明：

- VSZ: vittual memory size 虚拟内存集
- RSS: resident size, 常驻内存集
- STAT: 进程状态有以下几个
 - R: runing
 - S: interruptable sleeping 可中断睡眠
 - D: uninterruptable sleeping 不可中断睡眠
 - T: stopped
 - Z: zombie 僵尸进程
 - +:前台进程
 - N:低优先级进程
 - l:多线程进程

2). top 显示 linux 进程



top 命令将相当多的系统整体性能信息放在一个屏幕上。显示内容还能以交互的方式进行改变。动态的持续监控进程的运行状态，top 语法一般如下：

```
root@myir-yf13x:~# top -help
procps-ng 3.3.17-dirty
Usage:
top -hv | -bcEeHiOSs1 -d secs -n max -u|U user -p pid(s) -o field -w [cols]
```

动态查看系统进程示例如下：

```
root@myir-yf13x:~# top
top - 22:05:13 up 25 min, 1 user, load average: 0.96, 0.68, 0.57
Tasks: 99 total, 1 running, 98 sleeping, 0 stopped, 0 zombie
%Cpu(s): 15.0 us, 10.0 sy, 0.0 ni, 75.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 441.3 total, 155.6 free, 63.9 used, 221.7 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 357.5 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+
COMMAND
1588 root        20   0   4280   1972   1476 R   11.1   0.4   0:00.06 top

1022 root        39  19 190816 21668   4880 S    5.6   4.8   0:17.57 netda
ta
```

3). vmstat 虚拟内存的统计工具

这个命令可查看内存空间的使用状态，能获取整个系统的性能粗略信息，vmstat 运行于两种模式：采样模式和平均模式。如果不指定参数，则 vmstat 统计运行于平均模式下，vmstat 显示从系统启动以来所有统计数据的均值。常用语法以及参数如下：

```
root@myir-yf13x:~# vmstat -h

Usage:
```



vmstat [options] [delay [count]]

Options:

-a, --active active/inactive memory
 -f, --forks number of forks since boot
 -m, --slabs slabinfo
 -n, --one-header do not redisplay header
 -s, --stats event counter statistics
 -d, --disk disk statistics
 -D, --disk-sum summarize disk statistics
 -p, --partition <dev> partition specific statistics
 -S, --unit <char> define display unit
 -w, --wide wide output
 -t, --timestamp show timestamp

 -h, --help display this help and exit
 -V, --version output version information and exit

For more details see vmstat(8).

vmstat 运行于平均模式，显示从系统启动以来所有统计数据的均值：

```
root@myir-yf13x:~# vmstat
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sy id wa
st
 0  0     0 159352 12520 214560    0    0   80   76  256  408  3  4
92  1  0
```

对其上面一些任务栏进行简单解释说明

- r: 当前可运行的进程数。这些进程没有等待 I/O,而是已经准备号运行。理想状态,可运行进程数与可用 cpu 数量相等
- b: 等待 I/O 完成的被阻塞进程数
- forks: 创建新进程的次数
- in: 系统发生中断的次数



- cs: 系统发生上下文切换的次数
- us: 用户进程消耗的总 CPU 时间百分比
- sy: 系统代码消耗的总 CPU 时间的百分比, 其中包括消耗在 system、irq 和 softirq 状态的时间
- wa: 等待 I/O 消耗的总 CPU 的百分比
- id: 系统空闲消耗的总 CPU 时间的百分比

统计系统各种数据详细信息如下:

```
root@myir-yf13x:~# vmstat -s
451864 K total memory
65432 K used memory
30732 K active memory
219400 K inactive memory
159352 K free memory
12520 K buffer memory
214560 K swap cache
0 K total swap
0 K used swap
0 K free swap
1864 non-nice user cpu ticks
2825 nice user cpu ticks
7588 system cpu ticks
163145 idle cpu ticks
1317 IO-wait cpu ticks
0 IRQ cpu ticks
122 softirq cpu ticks
0 stolen cpu ticks
139309 pages paged in
132883 pages paged out
0 pages swapped in
0 pages swapped out
450684 interrupts
718821 CPU context switches
1651181992 boot time
```



1599 forks

对其上面一些任务栏进行简单解释说明：

- total memory: 系统总内存
- used memory: 已使用内存
- CPU ticks: 显示的是自系统启动的 CPU 时间, 这里的“tick”是一个时间单位
- forks: 大体上表示的是从系统启动开始已经创建的新进程的数量

vmstat 提供了关于 linux 系统性能的众多信息。在调查系统问题时, 它是核心工具之一。

4). kill 进程终止工具

发送指定的信号到相应进程。不指定型号将发送 SIGTERM (15) 终止指定进程。如果任无法终止该程序可用“-KILL”参数, 其发送的信号为 SIGKILL(9), 将强制结束进程, 使用 ps 命令或者 jobs 命令可以查看进程号。root 用户将影响用户的进程, 非 root 用户只能影响自己的进程。kill 命令一般语法如下:

```
kill [ -s signal | -p ] [ -a ] pid ...
kill -l [ signal ]
```

部分参数组合说明:

- -s: 指定发送的信号
- -p: 模拟发送信号
- -l: 指定信号的名称列表
- pid: 要中止进程的 ID 号
- Signal: 表示信号

首先使用 ps -ef 与管道命令确定要杀死进程的 PID。

```
root@myir-yf13x:~# ps -ef | grep mxapp2
root      1601   1050   0 22:10 ttySTM0  00:00:00 grep mxapp2
```

然后输入以下命令终止进程:

```
root@myir-yf13x:~# kill 1050
```

killall 命令终止同一进程组内的所有进程, 允许指定要终止的进程的名称而非 PID 进程号:

```
root@myir-yf13x:~# killall mxapp2
```



9. 开发支持

本章主要介绍针对当前 SDK 进行二次开发的一些基本信息，当前 SDK 提供两种配置的参考镜像，一种是 myir-image-core，主要针对无 GUI 的应用；另外一种是我 myir-image-full，在 myir-image-core 的基础上增加一些需要 GUI 的应用，如 QT 运行时库以及参考的 HMI 界面。关于这两种镜像的信息请参考《MYD-YF13X SDK 发布说明》。

9.1. 开发语言

● SHELL

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。常见的 Linux 的 Shell 种类众多，常见的有：

- Bourne Shell (/usr/bin/sh 或/bin/sh)
- Bourne Again Shell (/bin/bash)
- C Shell (/usr/bin/csh)
- K Shell (/usr/bin/ksh)
- Shell for Root (/sbin/sh)

这里重点介绍，也是当前 SDK 默认支持的 bash。下面示例为 myir-image-full 镜像中 MEasy HMI 2.0 开机自启动的脚本 start.sh，内容如下：

```
#!/bin/sh -e

echo "start myir HMI 2.0..."

export QT_QPA_EGLFS_ALWAYS_SET_MODE="1"
export QT_QPA_EGLFS_KMS_ATOMIC='1'
export QT_QPA_EGLFS_KMS_CONFIG='/usr/share/qt5/cursor.json'
#export QT_QPA_PLATFORM='eglfs'
psplash-drm-quit

strA=$(cat /sys/firmware/devicetree/base/compatible)
strB="ya157c"

result=$(echo $strA | grep "${strB}")
```



```
if [[ "$result" != "" ]]
then
export QT_QPA_PLATFORM='eglfs'
/home/mxapp2 -platform eglfs &
else
/home/mxapp2 -platform linuxfb &
fi

exit 0
```

这个脚本首先设置 MEasy HMI V2.0 的环境变量，然后根据不同的内核设备树信息，实用不同的参数调用/home/mxapp2 可执行程序启动 MEasy HMI V2.0，参数最后的"&"代表程序将进入后台运行。

● C/C++

C/C++是 Linux 平台下进行底层应用开发最为常用的编程语言，也是仅次于汇编的最为高效的语言。使用 C/C++进行开发通常采用的是交叉开发的方式，即在开发主机端进行开发，编译生成目标机器上运行的二进制执行文件，然后部署到目标机器上运行。

采用这种方式，首先需要安装基于 Yocto 构建的 SDK，安装步骤请参《MYD-YF13X Linux 软件开发指南》，安装完成后需要配置一下 SDK 环境，如下：

```
PC $:source /opt/st/myir-yf13x/4.0.4-snapshot/environment-setup-cortexa7t2hf-
neon-vfpv4-ostl-linux-gnueabi
PC $:arm-ostl-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4
-mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/myir-yf13x/4.0.4-snapsho
t/sysroots/cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

本节通过编写一个简单的 Hello World 实例来演示应用程序的开发，以下为在开发主机端用 C 编写的演示程序 hello-CC.c:

```
//file: hello-CC.c
#include<stdio.h>
int main(int argc,char *argv[])
{
    printf("hello world!\n");
    return 0;
```



```
}
```

C++编写的演示程序 hello-CXX.cpp

```
//file: hello-CXX.cpp
#include <iostream>
using namespace std;
int main(int argc,char *argv[])
{
    cout << "hello world!";
    return 0;
}
```

接着编译应用程序，这里 c 语言用\$CC 编译，c++用\$CXX 编译，因为编译的时候需要对应的头文件和链接，\$CC/\$CXX 包含有对应的系统库和配置信息，如果直接用 arm-stl-linux-gnueabi-gcc/g++ 来编译会出现找不到头文件的情况，这个时候可以加入参数-v 来查看详细的链接过程。

```
$CC -v hello-CC.c -o hello-CC
or
$CXX -v hello-CXX.cpp -o hello-CXX
.....
```

然后通过 scp 命令把生成的执行文件拷贝到目标机器上执行，结果如下：

```
root@myir-yf13x:~# ./hello-CC
hello world!
or
root@myir-yf13x:~# ./hello-CXX
hello world!
```

更复杂的示例和开发方式请参考《MYD-YF13X Linux 软件开发指南》应用移植部分的说明。

● Python

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言。Python 由 Guido van Rossum 于 1989 年底发明，第一个公开发行业版发行于 1991 年。像 Perl 语言一样，Python 源代码同样遵循 GPL(GNU General Public License) 协议。本节主要测试 python 的使用，从 python 命令行和脚本两个方面来说明。



1). 查看系统支持的 python 版本

```
root@myir-yf13x:~# python3.10 -V
Python 3.10.4
```

2). python 命令行测试

启动 python，并在 python 提示符中输入以下文本信息，然后按 Enter 键查看运行效果：

```
root@myir-yf13x:~# python3
Python 3.10.4 (main, Mar 23 2022, 20:25:24) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello, Python!")
```

在 Python 3.10 版本中,以上实例输出结果如下：

```
Hello, Python!
>>>
```

退出 Python 命令行，执行 exit()即退出 Python：

```
>>> exit()
root@myir-yf13x:~#
```

3). 编写脚本测试 Python

编写一个简单的 Python 脚本程序，所有 Python 文件将以 .py 为扩展名。

```
root@myir-yf13x:~# vi test.py
root@myir-yf13x:~# cat test.py
#!/usr/bin/python3
print ("Hello, Python!")
```

执行脚本文件，Python3 解释器在/usr/bin 目录中，使用以下命令执行脚本。

```
root@myir-yf13x:~# chmod +x test.py
root@myir-yf13x:~# ./test.py
Hello, Python!
```

通过脚本参数调用 Python3 解释器开始执行脚本，直到脚本执行完毕。当脚本执行完成后，解释器不再有效。当前系统不支持 pip 命令与 pip 安装 python 包，如果客户需要 pip 工具需要自行移植。



9.2. 数据库

数据库(Database)是按照数据结构来组织、存储和管理数据的仓库。数据库有很多种类型,常用的数据库有 Access、Oracle、Mysql、SQL Server、SQLite 等。

- **System SQLite**

SQLite 是一个嵌入式 SQL 数据库引擎。与大多数其他 SQL 数据库不同,SQLite 没有单独的服务器进程。SQLite 直接读写普通磁盘文件。包含多个表、索引、触发器和视图的完整 SQL 数据库包含在单个磁盘文件中。这是一款轻型的数据库,是遵守 ACID 的关联式数据库管理系统,它的设计目标是嵌入式的,而且目前已经在很多嵌入式产品中使用了它,它占用资源非常的低,在嵌入式系统中,可能只需要几百 K 的内存就够了。这款数据库的运行处理速度比 Mysql、PostgreSQL 这两款都要快,MYD-YF135 使用的是 SQLite 3,下面进行简单测试。

启动 sqlite3,并创建一个新的数据库 <testDB.db>,在终端界面输入以下命令就可以进入操作界面。

```
root@myir-yf13x:~# sqlite3 testDB.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite>
```

上面的命令将在当前目录下创建一个文件 testDB.db。该文件将被 SQLite 引擎用作数据库。注意到 sqlite3 命令在成功创建数据库文件之后,将提供一个 sqlite> 提示符。

数据库被创建,您就可以使用 SQLite 的 .databases 命令来检查它是否在数据库列表中,如下所示:

```
sqlite> .databases
main: /home/root/testDB.db
sqlite>
```

使用 .quit 命令退出 sqlite 提示符,如下所示:

```
sqlite> .quit
root@myir-yf13x:~#
```

如果想要详细了解 SQLite 相关信息,请参考官网 <https://www.sqlite.org/docs.html>

9.3. Qt 应用程序本地化



本节讨论本地化相关的设置和测试。本地化，指的是一个程序或软件在支持国际化的基础上，给定程序特定区域的语言信息使其在信息的输入输出等处理上适应特定区域人群的使用。这里允许程序所使用的一些语言环境变量在程序执行时动态配置。本章主要针对 QT 应用的本地化，以米尔演示镜像 MEasy HMI 2.0 为例进行说明。

1). 多语言

本节主要以米尔演示镜像 MEasy HMI 2.0 为例说明多语言在 QT 项目中的实际应用。

阅读下面内容前请参考《MYD-YF13X_QT 及 MEasy HMI2.0 软件开发指南》第 3.1 章 环境搭建，搭建起 MEasy HMI 的编译环境。

● 打开 mxapp2 工程

MEasy HMI 2.0 对应的工程源码为 mxapp2.tar.gz,拷贝至上述文档所构建的环境里面去，并使用 QT Creator 打开这个工程。

● 生成 ts 文件

通过终端进入 mxapp2 工程所在源码目录，并执行下面命令生成翻译文件。

```
xmr@xmr-VirtualBox:~/download/mxapp2$ lupdate mxapp2.pro
Info: creating stash file /home/qinlh/download/MXAPP/.qmake.stash
Updating 'languages/language_zh.ts'...
Found 202 source text(s) (0 new and 202 already existing)
Updating 'languages/language_en.ts'...
Found 202 source text(s) (0 new and 202 already existing)
```

工程在发布之前已完成翻译的工作，此处不会重新生成 language_zh.ts 和 language_en.ts 文件，中文显示使用的是 language_zh.ts 生成的 qm 文件，英文显示使用的是 language_en.ts 生成的 qm 文件。

● 翻译文本

打开 languages/language_en.ts 文件，针对翻译的源字符串为<source>节点里面的内容，翻译的目标字符串为<translation>里面的内容，用户可根据需求进行修改。

```
<message>
  <location filename=" ../Album.qml" line="50"/>
  <source>返回</source>
  <translation type="unfinished">Return</translation>
</message>
```



- 生成 qm 文件

Ts 文件修改完成后需要手动生成翻译模版文件供应用使用，进入 languages 目录，使用如下命令即可生成翻译模版文件。

```
xmr@xmr-VirtualBox:~/download/mxapp2/languages$ lrelease language_en.ts -qm language_en.qm
Updating 'language_en.qm'...
    Generated 183 translation(s) (2 finished and 181 unfinished)
Ignored 21 untranslated source text(s)
qinlh@qinlh-VirtualBox:~/download/mxapp2/languages$ lrelease language_zh.ts -qm language_zh.qm
Updating 'language_en.qm'...
    Generated 183 translation(s) (2 finished and 181 unfinished)
Ignored 21 untranslated source text(s)
```

- 应用翻译文件

翻译文件的使用需要通过应用程序代码调用，具体调用方法参考源代码“translator.cpp”文件中的 LoadLanguage 成员函数 “。

1). 字体

本节主要以米尔演示镜像 MEasy HMI 2.0 为例说明字体在 QT 项目中的实际应用。阅读下面内容前请参考《MYD-YF13X_QT 及 MEasy HMI2.0 软件开发指南》第 3.1 章环境搭建，搭建起 MEasy HMI 的编译环境。

- 安装字体文件

字体文件可以直接放置到开发板文件系统/usr/lib/fonts/目录。

```
root@myir-yf13x:/usr/lib/fonts# ls
msyh.ttc
```

或者直接添加的 QT 工程里面。

```
xmr@xmr-VirtualBox:~/download/mxapp2/fonts$ tree
├── DIGITAL
│   ├── DIGITAL.TXT
│   └── DS-DIGIB.TTF
└── fontawesome-webfont.ttf
```



- 使用字体文件

字体文件的使用是需要通过应用的代码来调用的，具体调用方法参考 main.cpp 中 iconFontInit() 函数。

```
void iconFontInit()
{
    int fontId_digital = QFontDatabase::addApplicationFont(":/fonts/DIGITAL/DS-DIGIB.TTF");
    int fontId_fws = QFontDatabase::addApplicationFont(":/fonts/fontawesome-webfont.ttf");
    QString fontName_fws = QFontDatabase::applicationFontFamilies(fontId_fws).at(0);
    QFont iconFont_fws;
    iconFont_fws.setFamily(fontName_fws);
    QApplication::setFont(iconFont_fws);
    iconFont_fws.setPixelSize(20);
}
```

2). 软键盘

本章节主要以米尔演示镜像 MEasy HMI 2.0 为例说明软键盘在 QT 项目中的实际应用。

阅读下面内容前请参考《MYD-YF13X_QT 及 MEasy HMI2.0 软件开发指南》第 3.1 章 环境搭建，搭建起 MEasy HMI 的编译环境。

QT 从 5.7 版本开始在官方源码里面加入了 qt virtual keyboard 组件，myir 在发布的 MEasy HMI 2.0 配套的镜像中已经使用了 QT 5.15 版本，并配置了 virtual keyboard 这个组件。

- 软键盘嵌入到 qml 代码

Qt 提供的软键盘只能在 qml 代码里面进行调用，调用前需要定义软键盘弹出的位置及软键盘的大小，如下面代码所示。

```
InputPanel {
id: inputPanel
x: adaptive_width/8
y: adaptive_height/1.06
```



```
z:99
anchors.left: parent.left
anchors.right: parent.right

states: State {
name: "visible"
when: inputPanel.active
PropertyChanges {
target: inputPanel
    y: adaptive_height/1.06 - inputPanel.height
}
}
```

● 触发软键盘

软键盘是通过 QML 的 TextField、TextEdit 组件触发的，用户只需要在 UI 组件中加入此组件即可在 UI 上调出软键盘并使用。如果是 qt 系统，可以使用 QT 自带的 QT virtual keyboard 输入。

```
TextField{
id: netmask_input
InputMethodHints: Qt.ImhFormattedNumbersOnly
onAccepted: digitsField.focus = true
font.family: "Microsoft YaHei"
color: "white"
}
```

● 使用软键盘

软键盘的使用方法请参考《MYD-YF13X_QT 及 MEasy HMI2.0 软件开发指南》第 2.6 章系统设置 UI 界面。用户点击可编辑的界面组件就会弹出软键盘。



10. 参考资料

- STM32MPU 开发社区

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

- Yoto 项目 BSP 开发指南

<https://docs.yoctoproject.org/4.0.9/bsp-guide/index.html>

- Yocto 项目 Linux 内核开发手册

<https://docs.yoctoproject.org/4.0.9/kernel-dev/index.html>

- Linux 内核看门狗介绍

<https://www.kernel.org/doc/html/latest/watchdog/index.html>

- 嵌入式 Linux 下的 Qt

<https://doc.qt.io/qt-5/embedded-linux.html>

- Wayland 架构

<https://wayland.freedesktop.org/architecture.html>

- Systemd 网络配置

<https://www.freedesktop.org/software/systemd/man/systemd.network.html>



附录一 联系我们

深圳总部

地址：深圳市龙岗区坂田街道发达路云里智能园 2 栋 6 楼 04 室

负责区域：广东、广西、海南、重庆、云南、贵州、四川、西藏、香港、澳门

传真：0755-25532724 电话：0755-25622735

武汉研发中心

地址：武汉东湖新技术开发区关南园一路 20 号当代科技园 4 号楼 1601 号

电话：027-59621648

华东地区

地址：上海市浦东新区金吉路 778 号浦发江程广场 1 号楼 805 室

负责区域：上海、福建、浙江、江苏、安徽、山东

传真：021-62087085 电话：021-62087019

华北地区

地址：北京市大兴区荣华中路 8 号院力宝广场 10 号楼 901 室

负责区域：辽宁、吉林、黑龙江、北京、天津、河北、山西、内蒙古、

湖北、湖南、江西、河南、陕西、甘肃、宁夏、青海、新疆

传真：010-64125474 电话：010-84675491

销售联系方式

网址：www.myir-tech.com

邮箱：sales.cn@myirtech.com

技术支持联系方式

电话：0755-22316235（深圳）027-59621647/027-59621648(武汉)

邮箱：support.cn@myirtech.com

如果您通过邮件获取帮助时，请使用以下格式书写邮件标题：

[公司名称/个人--开发板型号] 问题概述

这样可以使我们更快速跟进您的问题，以便相应开发组可以处理您的问题。



附录二 售后服务与技术支持

凡是通过米尔电子直接购买或经米尔电子授权的正规代理商处购买的米尔电子全系列产品，均可享受以下权益：

- 1、6 个月免费保修服务周期
- 2、终身免费技术支持服务
- 3、终身维修服务
- 4、免费享有所购买产品配套的软件升级服务
- 5、免费享有所购买产品配套的软件源代码，以及米尔电子开发的部分软件源代码
- 6、可直接从米尔电子购买主要芯片样品，简单、方便、快速；免去从代理商处购买时，漫长的等待周期
- 7、自购买之日起，即成为米尔电子永久客户，享有再次购买米尔电子任何一款软硬件产品的优惠政策
- 8、OEM/ODM 服务

如有以下情况之一，则不享有免费保修服务：

- 1、超过免费保修服务周期
- 2、无产品序列号或无产品有效购买单据
- 3、进液、受潮、发霉或腐蚀
- 4、受撞击、挤压、摔落、刮伤等非产品本身质量问题引起的故障和损坏
- 5、擅自改造硬件、错误上电、错误操作造成的故障和损坏
- 6、由不可抗拒自然因素引起的故障和损坏

产品返修：

用户在使用过程中由于产品故障、损坏或其他异常现象，在寄回维修之前，请先致电米尔电子客服部，与工程师进行沟通以确认问题，避免故障判断错误造成不必要的运费损失及周期的耽误。

维修周期：

收到返修产品后，我们将即日安排工程师进行检测，我们将在最短的时间内维修或更换并寄回。一般的故障维修周期为 3 个工作日（自我司收到物品之日起，不计运输过程时间），由于特殊故障导致无法短期内维修的产品，我们会与用户另行沟通并确认维修周期。

维修费用：



在免费保修期内的产品，由于产品质量问题引起的故障，不收任何维修费用；不属于免费保修范围内的故障或损坏，在检测确认问题后，我们将与客户沟通并确认维修费用，我们仅收取元器件材料费，不收取维修服务费；超过保修期限的产品，根据实际损坏的程度来确定收取的元器件材料费和维修服务费。

运输费用：

产品正常保修时，用户寄回的运费由用户承担，维修后寄回给用户的费用由我司承担。非正常保修产品来回运费均由用户承担。

