



龙芯 2K300 软件用户手册 (试用版)

2023年2月

手册版本：

龙芯中科技股份有限公司

V0.1

版权声明

本文档版权归龙芯中科技术股份有限公司所有，并保留一切权利。未经书面许可，任何公司和个人不得将此文档中的任何部分公开、转载或以其他方式散发给第三方。否则，必将追究其法律责任。

免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

龙芯中科技术股份有限公司

Loongson Technology Corporation Limited

地址：北京市海淀区中关村环保科技示范园龙芯产业园 2 号楼

Building No.2, Loongson Industrial Park,

Zhongguancun Environmental Protection Park, Haidian District, Beijing

电话(Tel)：010-

62546668

传真(Fax)：010-

62600826

阅读指南

《龙芯2K300LA用户手册》主要介绍龙芯 2K300LA相关软件及源码的使用、编译。帮助用户了解和快速搭建应用环境。所涉及包括：PMON源码编译、Linux内核源码编译、文件系统制作、交叉工具链、EJTAG软件使用。

目录

版权声明	2
免责声明	2
龙芯中科技术股份有限公司	2
阅读指南	3
目录	4
1. 快速开始	6
1.1 PMON编译环境搭建	6
1.2 编译PMON :	7
1.3 LINUX内核编译环境搭建	8
1.3 文件系统制作	9
1.4 PMON烧录	10
1.5 启动内核	13
2. 接口配置与使用	14
2.1 内存	15
2.2 GMAC	15
2.3 UART	18
2.4 SPI	19
2.5 OTG	20
2.6 IIC	21
2.7 CAN	22
2.8 GPIO	23
2.9 PWM	24
2.10 EMMC	24
2.11 SDIO	25
2.12 LIO	25

2.13 I2S	26
----------------	----

1. 快速开始

本文是龙芯2K300的用户手册，和该手册一起的包括：PMON源码、Linux内核源码、文件系统、交叉工具链、EJTAG软件、龙芯2K300LA处理器用户手册。本文主要说明上述软件及源码的使用、编译。

1.1 PMON编译环境搭建

1) 环境说明

本文档涉及的龙芯软件开发环境均在linux环境下进行，且均为交叉编译环境说明，x86环境下首先推荐使用ubuntu 20.04，其他ubuntu版本需要注意python版本是否为python 2.7。

2) pmon源码解压

将源码压缩包拷贝到工作目录，使用下述命令对源码进行解压。

（注：如果使用的工作环境是虚拟机，请不要直接在共享文件夹下进行解压）

```
1| tar -zxvf pmon-loongson.tar.gz
```

3) 交叉编译工具安装

将交叉工具链拷贝到常用目录，并解压：

```
tar -zxvf loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.3-1.tar.xz
```

注：本手册以工具链位于/opt/为例

工具链名称或因迭代出现名称差异，此处工具链名称仅为举例，具体工具链请以所获取到的为准。

4) 安装编译依赖：

```
1| sudo apt install aptitude xutils-dev bison flex acpica-tools
```

5) pmoncfg 工具安装

该操作仅在源码包解压后的第一次编译前需要执行，如果之前已经安装过，可忽略该步骤。

pmoncfg的编译和安装操作如下所示：

```
1| cd PMON源码/tools/pmoncfg
2| make pmoncfg
3| sudo cp pmoncfg /usr/bin
```

1.2 编译PMON：

```
1| cd zloader.soc/
2| make ls2k300
3| make cfg all tgt=rom CROSS_COMPILE=/opt/loongson-gnu-
  toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.3-
  1/bin/loongarch64-linux-gnu- DEBUG=-g
4| make dtb CROSS_COMPILE=/opt/loongson-gnu-toolchain-8.3-
  x86_64-loongarch64-linux-gnu-rc1.3-1/bin/loongarch64-linux-gnu-
```

编译选项解释：

make cfg 对pmon进行配置；

make ls2k300 选择2k300芯片配置

all为Makefile里的编译项；

tgt=rom，指定tgt为rom，则会生成gzrom.bin文件；

CROSS_COMPILE=loongarch64-linux-gnu-，指定编译工具前缀名；

DEBUG=-g，设置编译的时候携带调试信息。

make dtb，编译设备树

注：如更改了配置文件 Targets/soc/conf/ls2k300，则在编译前要执行make

cfg，使得更改生效，如果普通编译没有更改配置，则每次无需都执行make cfg命令

令。

编译脚本如下，在PMON源码目录下执行：

```
#!/bin/sh
cd zloader.soc/
make ls2k300
make cfg all tgt=rom CROSS_COMPILE=/opt/ loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.2/bin/loongarch64-linux-gnu-
DEBUG=g
make dtb CROSS_COMPILE=/opt/loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.2/bin/loongarch64-linux-gnu-
cp gzrom-dtb.bin ./
cp gzrom.bin ./
cp loongson.dtb ./
```

在PMON源码目录生成bin文件和dtb文件，其中gzrom-dtb.bin带dtb，gzrom.bin不带dtb，loongson.dtb为单独的dtb文件。

1.3 LINUX内核编译环境搭建

安装编译依赖：

```
1| sudo apt install libncurses5-dev libssl-dev
```

指定交叉工具链：

```
1| export PATH=/opt/loongson-gnu-toolchain-8.3-x86_64-
loongarch64-linux-gnu-rc1.2/bin:$PATH
```

采用2K300的配置文件

```
1| cp arch/loongarch/configs/ls2K300_defconfig .config
2| make menuconfig ARCH=loongarch
```

编译内核：

```
1| make vmlinuz ARCH=loongarch CROSS_COMPILE=loongarch64-
linux-gnu- -j 4
```

编译完成后，会在当前目录下看到生成的vmlinuz文件，与压缩后的内核文件vmlinuz。

编译脚本mymake如下：

```
#!/bin/sh
export LC_ALL=C LANG=C
export PATH=/opt/loongarch64-linux-gnu-2021-12-10-vector/bin:$PATH
make vmlinuz ARCH=loongarch CROSS_COMPILE=loongarch64-linux-gnu- -j 4 "$@"
```

编译时执行：


```
1| ./mymake menuconfig
2| ./mymake vmlinuz
```

注：

a、一般情况下，随本文档一起的文件里有制作好的文件系统（initrd.cpio），可以将制作好的文件系统直接编译进内核，执行make menuconfig ARCH=loongarch进行配置，配置路径如下：

```
General setup ---->
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
(填写路径及文件名称) Initramfs source file(s)
```

b、内核下DTB文件的配置（推荐使用pmon的dtb文件）

```
Kernel type --->
[*] Enable builtin dtb in kernel
(loongson_2K300) Built in DTB
```

内核下的dtb文件位于：

arch/loongarch/boot/dts/loongson/loongson_2K300.dts

1.3 文件系统制作

编译前安装依赖参照PMON编译章节所述进行安装。

1.3.1 buildroot文件系统制作

解压buildroot-la.tar.gz文件

```
1| tar -zxvf buildroot-la.tar.gz
```

设置编译配置文件

```
1| cp config-la.my .config
```

或者

```
1| cp configs/loongisa_defconfig .config
```

指定交叉编译工具链

```
1| export PATH=/opt/loongarch_toolchain/bin:$PATH
```

打开图形化配置界面，修改工具链路径名，并保存退出

```
1| make menuconfig ARCH=loongarch64
```

```
Toolchain type (External toolchain) --->
*** Toolchain External Options ***
Toolchain (Custom toolchain) --->
Toolchain origin (Pre-installed toolchain) --->
(/opt/loongarch-toolchain) Toolchain path
($(ARCH)-linux-gnu) Toolchain prefix
External toolchain gcc version (8.x) --->
External toolchain kernel headers series (4.19.x) --->
External toolchain C library (glibc/eglibc) --->
```

联网编译文件系统

```
1| make ARCH=loongarch64 CROSS_COMPILE=loongarch64-linux-
gnu- -j4 "$@"
```

编译完成后，在buildroot源码的output/images/目录下会生成文件系统镜像文件。

1.3.2 Yocto文件系统制作

解压yocto-la.tar.gz

```
1| tar zxvf yocto-la.tar.gz
```

解压交叉工具链

```
1| sudo tar zxvf loongarch-toolchain.tar.gz
```

进入yocto目录，执行下面的命令

```
1| cd yocto
2|
3| . oe-init-build-env
```

编译文件系统，执行下面的命令

```
1| bitbake core-image-minimal
```

1.4 PMON烧录

pmon支持多种烧写方式，具体如下：

1.4.1 调试器更新

linux下烧写器程序为ejtag-debug.tar.gz,直接解压即可使用。

将需要烧录的gzrom-dtb.bin 放入ejtag目录。

进入调试器目录输入如下命令

```
1| sudo ./la_dbg_tool_usb -t
2| source configs/config.ls2K300
3| set      #先set后上电，set返回寄存器的值，其中pc需要时
0x1c000000
4| program_cachelock ./gzrom-dtb.bin
```

如遇烧录问题请先查看FAQ相关问题解决方案。

windows烧写

windows下直接右键解压，可参考解压后文件doc/ejtag1.pdf安装驱动。其中给出的下载网址或已失效，应用软件可通过技术支持获取。

电脑端插上EJTAG的usb端口且设备管理中出现了一个新的未知设备后，就可以双击运行EJTAG中的zadig-*.exe来进行安装，只需在列表中选择对应的未知设备并选择对应驱动后点击安装等待其完成即可。

(mingw版选winusb，cygwin版选libusb，版本会体现在压缩包的名字里，一般为mingw版)

将需要烧录的gzrom-dtb.bin 放入ejtag目录。

直接双击la_dbg_tool_usb.exe启动后使用如下命令烧录。

```
1| source configs/config.ls2k
2| set      #先set后上电，set返回寄存器的值，其中pc需要时
0x1c000000
3| program_cachelock ./gzrom-dtb.bin
```

执行过program_cachelock后打印回到cpu0- 则烧录结束，此时可以重启板卡。

在双击exe打开终端后有可能出现无法输入的情况，此时可稍作等待或连续使用ctrl+c至其回应break.

当前windows应用在烧录时，可能会在如下位置停止十秒左右后开始烧录。

```
s5:0x51cfde734243622a s6:0x20c17125698f3347 s7:0xf1592
pc:0x1c000004

csr0-crmd:0x8          csr1-prmd:0x0
csr4-excfg:0x10000     csr5-exst:0x0
csr8-badi:0x0          csrc-ebase:0x1c000000
cpu0 -program_cachelock ./gzrom-dtb.bin
#stop
#setconfig callbin.stacksize 0x1000
#expr 0x8000000000000000+0x1fe00000+0x2000+0x80
#expr 0x1c000000+0x00f2
#m8 0x800000001fe02080 0x1c0000f2
#expr 0x8000000000000000+0x1fe00000+0x2000+0x90
#expr 0x1c000000+0x00f2
#expr 0x1c000000+0x00f2
#m8 0x800000001fe02090 0x1c0000f2 0x1c0000f2
#call cache_lock
##cache 0x13 {$cabase+$spi_membase} 0x40000
#expr 0x8000000000000000+0x1fe00000+0x0200
#expr 0x8000000000000000+0x1c000000
#m8 0x800000001fe00200 0x800000001c000000
#
#expr 0x8000000000000000+0x1fe00000+0x0240
#m8 0x800000001fe00240 0xfffffffffc0000
#
#expr 0x9000000000000000+0x1c000000
#memset4 0x900000001c000000 0 0x40000
```

若在烧录过程中鼠标点击了终端则终端窗口会来到如图状态

```
选择C:\Users\HP\Desktop\ejtag-debug\la_dbg_tool_usb.exe
s5:0x51cfde734243622a s6:0x20c17125698f3347 s7:
pc:0x1c000004
```

且此时烧录会停止，请使用按下任意键使其回到如下状态后，烧录会继续进行。

```
C:\Users\HP\Desktop\ejtag-debug\la_dbg_tool_usb.exe
#put ./gzrom-dtb.bin 0x900000001c010000 0x10000 0
pack: 0 time: 1 download size: 0x10000 downlo
```

1.4.2 U盘更新

将gzrom-dtb.bin放入U盘中，进入PMON，在pmon shell里输入如下命令

```
1| fload (usb0,0)/gzrom-dtb.bin
```

1.4.3 网络更新

需要有tftp服务器，将gzrom-dtb.bin放入tftp指定的目录中，进入PMON，在pmon shell里输入如下命令（假设用2K上的GMAC0口更新）

```
1| ifaddr syn0 ip
2| fload tftp://server-ip/gztom-dtb.bin
```

ip为要设置的该板卡的IP地址，server-ip为连接该板卡的终端机的IP地址

1.4.4 只更新DTB

如果有只更新DTB的需求，pmon下也可以使用load_dtb命令来更新，同样支持U盘

更新和tftp更新：

```
1| load_dtb (usb0,0)/loongson.dtb
2| load_dtb tftp://server-ip/loongson.dtb
```

1.5 启动内核

pmon下支持内核启动的外设接口有emmc、sdio、nvme、usb、sata和网络，分别对应的设备名称为tfcard0、emmc0、nvme0、usb0、wd0、syn0(cpu自带的gmac接口)，网络加载的命令形式比较特殊，后面单独介绍，支持读访问的文件系统包括ext、fat、iso9660等常用文件系统，下面以wd0为例进行介绍，如需其他外设启动，请自行更换成相应的设备名。

1.5.1 手动启动

进入pmon命令行，依次输入如下命令：

```
1| load (wd0,0)/vmlinuz          #加载内核
2| initrd (wd0,0)/initrd.cpio    #加载文件系统
3| g console=ttyS0,115200        #启动linux
```

此处为将文件系统加载入内存中以Ramdisk方式启动的参考命令，如果硬盘已经写入文件系统，可通过传参root=/dev/sda1或root=UUID=来使其以挂载对应硬盘中的文件系统的方式启动或通过添加传参root、nfsroot、ip来使用NFS挂载文件系统的方式启动。

如果在编译内核时，已经将initrd.cpio文件系统编译进内核，则再不需要进行initrd的命令操作。

1.5.2 自动启动

1.5.2.1 使用boot.cfg

PMON启动最后会去常用存储设备的boot目录找boot.cfg文件，并按照boot.cfg的参数启动，例：

```
timeout 3
default 0
showmenu 1

title 'boot test'
    kernel (wd0,0)/boot/vmlinuz
    initrd (wd0,0)/initrd.cpio
    args console=tty console=ttyS0,115200 root=/dev/sda1
```

其中kernel为内核二进制所在路径，args为内核传参，initrd为加载ramdisk文件

系统，如果想让对应的条目不生效，可以在对应条目前加入‘#’号键，如当

initrd.cpio已经编译进内核（参见内核编译章节），则不需要initrd的条目：

```
# initrd (wd0,0)/initrd.cpio
```

若U盘与硬盘同时存在boot.cfg系统优先读取使用U盘中的，若仅需使用ramdisk启动则可参考如下boot.cfg：

```
timeout 3
default 0
showmenu 1

title 'boot test'
    kernel (wd0,0)/boot/vmlinuz
    initrd (wd0,0)/boot/rootfs.cpio.gz
    args console=ttyS0,115200
```

1.5.2.2 使用环境变量

如果没有boot.cfg, PMON会执行环境变量autocmd提供的命令，设置autocmd如下：

```
set autocmd "load (wd0,0)/vmlinuz;initrd (wd0,0)/rootfs.cpio.gz;g c
onsole=ttyS0,115200 rdinit=/sbin/init"
```

pmon会依次按照autocmd环境变量中的语句来执行，如果没有设置autocmd或者设置的不是启动linux的命令，PMON会加载如下环境变量，并按照环境变量启动内核

- 1| #rd为文件系统路径；al1为内核路径；append为启动参数
- 2| set rd (wd0,0)/boot/initrd.cpio
- 3| set al1 (wd0,0)/boot/vmlinuz
- 4|set append "g console=ttyS0,115200 rdinit=/sbin/init" （该方案的加载顺序为先执行rd环境变量参数来加载文件系统再使用al1环境变量的参数加载内核，若失败，会使用环境变量al的参数来加载内核）

该方案支持U盘、硬盘、网络方式启动。

2. 接口配置与使用

接口的各个配置主要位于PMON下，文件位置：

Targets/soc/conf/1s2k300

以下内容默认情况下所述配置项均位于该文件中。

2.1 内存

内存的常用配置相关项

```
option CONFIG_DDR_32BIT    //内存是否使用32位模式，不使用该宏定义表示使用64位模式

option MC0_MEMSIE=1024    //内存总容量设置，单位为MB，如果不设置该宏定义，程序默认为2GB

option DDR_RESET_REVERT    //内存的复位设置，需要硬件确认DDR复位引脚是否使用反向器，如果不设置该宏定义，表示硬件未使用反向器

option AUTO_DDR_CONFIG    //内存是否使用内存条，如果不定义该宏，表示内存为板载颗粒。
```

除以上常用配置外，针对板载的DDR颗粒，可能还会使用到的配置位于：

Targets/soc/include/mem_ctrl.h

```
#define MC0_BA_NUM          //内存颗粒的bank 数量，0表示有4个bank，1表示有8个bank

#define MC0_CSMAP           //内存颗粒所选片选，总工8个bit，对应cs7~cs0,一般硬件链接cs0，该宏置1即可

#define MC0_DRAM_WIDTH     //内存颗粒的数据宽度，0表示x4，1表示x8，2表示x16

#define MC0_SDRAM_CAPACITY //单个内存颗粒的总容量，单位为512MB。

#define MC0_COL_NUM        //内存颗粒列地址数x与12的差值，宏数值为12-x

#define MC0_ROW_NUM        //内存颗粒行地址数y与18的差值，宏数值为18-y
```

2.2 GMAC

常用配置项

```
syn0    at localbus0 base 0x8000000016020000
syn1    at localbus0 base 0x8000000016030000
select   gmac

option   USE_ENVMAC        //MAC 地址使用环境变量存储

option   USE_GMAC1_FUNCTION //使用GMAC1的引脚复用
```

功能测试

pmon下两个gmac的名称分别为syn0、syn1,测试时，尽量不要同时配置两个网口，若要同时使用两个网口请确认IP为不同网段。

*2k300的GMAC1引脚与其他接口有复用关系，若需要使用GMAC1功能，需要打开USE_GMAC1_FUNCTION,同时其他复用功能将不可用（具体见2k300引脚复用关系）

pmon下查看网络设备和配置网口的方法：

注意：PMON下若已经为syn0配置IP且要使用同网段验证另一网口，需要

`ifconfig syn0 remove`后才能对syn1配置IP。

kernel下若已经为eth0配置IP且要使用同网段验证另一网口，需要

`ifconfig eth0 down`后才能对eth1配置IP。


```
PMON> ping 192.168.1.50
PING 192.168.1.50 (192.168.1.50): 56 data bytes
64 bytes from 192.168.1.50: icmp_seq=0 ttl=255 time=0.099 ms
64 bytes from 192.168.1.50: icmp_seq=1 ttl=255 time=0.048 ms

--- 192.168.1.50 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.048/0.060/0.099 ms
PMON> ifconfig syn0 remove
===ioctl sifflags
PMON> ifconfig syn1 192.168.1.50
synopGMAC_linux_open called
Version = 0xd137
MacAddr = 0x30 0x30 0x30 0x30 0x31 0x31

===phy HALFDUPLEX MODE
DMA status reg = 0x0 before cleared!
DMA status reg = 0x0 after cleared!
register poll interrupt: gmac 0
==arp_ifinit done
PMON> No Link: 00000000
Link is up in FULL DUPLEX mode

===phy FULLDUPLEX MODE
Link is with 1000M Speed

PMON> ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10): 56 data bytes
64 bytes from 192.168.1.10: icmp_seq=0 ttl=64 time=0.488 ms
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.852 ms
```

```
PMON> ifconfig syn0 192.168.1.50
==arp_ifinit done
PMON> ifconfig syn1 192.168.2.50
synopGMAC_linux_open called
Version = 0xd137
MacAddr = 0x30 0x30 0x30 0x30 0x31 0x31
Link is up in FULL DUPLEX mode

===phy FULLDUPLEX MODE
Link is with 1000M Speed
DMA status reg = 0x0 before cleared!
DMA status reg = 0x0 after cleared!
register poll interrupt: gmac 0
==arp_ifinit done
PMON> ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10): 56 data bytes
64 bytes from 192.168.1.10: icmp_seq=0 ttl=64 time=1.649 ms
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=1.167 ms

--- 192.168.1.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.167/1.400/1.649 ms
PMON> ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10): 56 data bytes
64 bytes from 192.168.2.10: icmp_seq=0 ttl=64 time=0.637 ms
64 bytes from 192.168.2.10: icmp_seq=1 ttl=64 time=0.854 ms

--- 192.168.2.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
```

使用环境变量设置设置MAC地址

配置文件选上 `option USE_ENVMAC` , pmon启动后在命令行设置环境变量：

```
set ethaddr 00:11:22:33:44:55
```

修改生成的二进制文件gzrom-dtb.bin中的mac地址：

```
#!/bin/python3
import struct
ethaddr = [0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc]
a = 0x12
b = 0x15
c = 0x16
d = 0x17
e = 0x18
```



```
with open('gzrom-dtb.bin', "rb") as f:
    f_content = f.read()
    content = struct.unpack("B" * len(f_content), f_content)
    with open('gzrom-dtb_1.bin', 'wb') as f_w:
        f_w.write(f_content)
        f_w.seek(0xff1ee)
        f_w.write(struct.pack("B" * 6, ethaddr[0], ethaddr[1], ethaddr[2], ethaddr[3], ethaddr[4], ethaddr[5]))
```

功能调试方法：

a、目前支持的phy芯片

phy型号：
88E1111、88E151X、YT8511、YT8521

其他型号的phy芯片，可能使用通用设置即可使用，具体以实际测试为准。

b、phy芯片寄存器的调试方法

为方便适配新板卡或者新phy，pmon下预留了调试phy寄存器的命令：

1、读phy寄存器

命令名称：phyrd

命令参数：ext gmac_num reg

命令说明：

ext：0/1表示操作普通/扩展寄存器

gmac_num：要操作的gmac的编号0或1

reg：要操作的phy寄存器的地址

举例说明：

phyrd 1 0 0xa003 //读gmac 0上phy扩展寄存器0xa003的值

2、写phy寄存器

命令名称：phywr

命令参数：ext gmac_num reg data

命令说明：

ext：0/1表示操作普通/扩展寄存器

gmac_num：要操作的gmac的编号0或1

reg：要操作的phy寄存器的地址

data：想要设置的寄存器的值

举例说明：

phywr 0 1 0x3 5 //设置gmac 1上phy普通寄存器0x3的值为5

3、phy复位

phy复位属于写phy寄存器的一种特殊操作，命令形式如下：

phywr 0 gmac_num -1 -1 //gmac_num为0/1

c、phy芯片回环测试的方法

待补充

d、gmac 链接FPGA的方法

待补充

注：

pmon下的gmac驱动目录位于：

sys/dev/gmac/

内核下的gmac驱动目录位于：

2.3 UART

常用配置项

option USE_UART_FUNTION //2K300 使能全部10个串口，如果想单独使能其中一个，可以根据代码自行修改

功能测试

a、寄存器说明

2K300的所有串口均支持小数分频，且驱动已经加入小数分频功能。

UART0 ~ 9寄存器基址依次为：

0x1610,0400、0x1610,0800、0x1610,0c00.....0x1610,2400

b、pmon下的数据发送和接受

1) 设置波特率

命令名称：set_baud

命令参数：freq baud reg

命令说明：

freq为uart的参考时钟，在2K300上，uart0的freq为100000000,其他串口的freq为50000000

baud为波特率设置，常用设置为115200,其他波特率可自行设置

reg 为所操作串口的寄存器基址

2) 数据发送

命令名称：m1

命令参数：reg date

命令说明：

reg 为所操作串口的寄存器基址

date 为想发送的数据

3) 数据接受

命令名称：d1

命令参数：reg

命令说明：

reg 为所操作串口的寄存器基址

c、内核下数据发送与接收

1) 内核代码常用配置

Device Drivers --->

Character devices --->

Serial drivers --->

(16) Maximum number of 8250/16550 serial ports

(16) Number of 8250/16550 serial ports to register at runtime

2) 设置波特率

命令名称：stty

命令参数：-F /dev/ttySx speed buad_num cs8 -echo

参数说明：

/dev/ttySx : x是要操作的串口号 (0~9)

buad_num : 表示要设置的波特率

举例说明：

```
stty -F /dev/ttyS1 speed 115200 cs8 -echo //设置uart1的波特率为115200，8位数据位，关闭回显
```

3) 数据发送

命令名称：echo

命令参数："date" >>/dev/ttySx

参数说明：

/dev/ttySx : x是要操作的串口号（0~9）

"date" : 表示要发送的数据

举例说明：

```
echo "123456" >>/dev/ttyS3 //设置uart3发送123456
```

4) 数据接收

命令名称：cat

命令参数：cat /dev/ttySx

参数说明：

/dev/ttySx : x是要操作的串口号（0~9）

举例说明：

```
cat /dev/ttyS3 //监听uart3,显示接收数据
```

2.4 SPI

常用配置

LS2K300有4个spi，目前不支持4线模式，没有复用关系，无需配置。

功能测试

a、pmon下的测试命令

1) spi 初始化

命令名称：spi_base

命令参数：spi_base spi_num

命令说明：

spi_num：ls2k300有两个spi控制器，其中spi0和spi1是spi-flash控制器，该参数取值范围0~1。

举例说明：

```
spi_base 1 //初始化spi控制器1
```

2) spi 写数据

命令名称：write_pmon_byte

命令参数：addr data

命令说明：

addr：flash 地址

data：spi控制器要发送的数据

举例说明：

```
write_pmon_byte 0x10 5 //向spi外接设备的地址0x10写数字5
```

3) spi 读数据

命令名称：read_pmon

命令参数：addr num

命令说明：

addr: 读取的地址

num：读取的数量

举例说明：

```
read_pmon 0x10 10 //从spi外接flash的0x10地址读取10个字节地址
```

4) spi擦除数据

命令名称：erase_pmon

命令参数：无

命令说明：无

举例说明：

```
erase_pmon //例如擦除spi外接flash，擦除完之后全为ff
```

b、内核下测试

不同板卡SPI片选对应的器件可能不一样，如果外接了spi的flash，内核可以测试mtd设备：

1) 对与nand flash执行

```
mount -t yaffs2 /dev/mtdblk0 /mnt //若失败，查看mtd设备是否存在，或者确认yaffs2文件系统是否存在
```

2) 对于nor flash执行

```
mount -t jffs2 /dev/mtdblk0 /mnt //若失败，查看mtd设备是否存在，或者确认jffs2文件系统是否存在
```

注：

对与mtd设备，第一次使用挂载前，最好先进行一次擦除，命令为：

```
flash_erase /dev/mtdx 0 0 //x为设备编号，需要自行确认要擦除的设备
```

2.5 OTG

常用配置

```
select mod_usb_otg //是能OTG控制器
```

功能测试

a、pmon下的测试命令

1、主模式

命令名称：devls、usbtree

命令参数：无

命令说明：

举例说明：otg接口插上u盘后启动系统，pmon启动时会枚举该设备，devls会显示usb0设备，usbtree会显示该usb设备。

2、从模式

pmon下暂不支持

b、内核下的测试命令

1) 主设备测试

linux内核下，有比较完善的i2c-tools来进行i2c操作:

linux内核下，打开下述配置：

```
CONFIG_USB_DWC2_HOST、  
CONFIG_USB_STORAGE
```

3) 从设备测试

打开下述配置：

```
CONFIG_USB_OTG_WHITELIST、  
CONFIG_USB_DWC2_PERIPHERAL、
```

```
CONFIG_USB_G_PRINTER ;
关闭下述配置：
CONFIG_USB_CONFIGFS。
示例中模拟成打印设备，otg从模式同时只可作为一种从设备
```

2.6 IIC

常用配置

```
option USE_I2C_FUNCTION //使能I2C0 ~ I2C3
```

功能测试

a、pmon下的测试命令

1、i2c 主功能初始化

```
命令名称：i2c_init_host
命令参数：bus_num clk
命令说明：
    bus_num：ls2K300有四个i2c控制器，该参数取值范围0 ~ 3。
    clk      ：i2c控制器输出时钟的频率，取值范围：800~6250K (该范围只表示理论可分频范围)。
举例说明：
    i2c_init_host 0 300000 //使能i2c0的主功能，输出频率为300000Hz。
```

2、i2c 数据发送

```
命令名称：i2c_write
命令参数：bus_num dev_addr date_reg data
命令说明：
    bus_num：ls2K300有四个i2c控制器，该参数取值范围0 ~ 3。
    dev_addr：i2c外接设备的地址，请确认实际地址，程序已经对地址进行了右移1bit。
    date_reg：数据写入设备的位置或者寄存器地址。
    data     ：要写入的数据
举例说明：
    i2c_write 0 0x1 0 0x10 2 //i2c0向地址为1的外设的0x10处写2。（实际意义查看对应的外设手册）
```

3、i2c数据接收

```
命令名称：i2c_read
命令参数：bus_num dev_addr date_reg count
命令说明：
    bus_num：ls2K300有四个i2c控制器，该参数取值范围0 ~ 3。
    dev_addr：i2c外接设备的地址，请确认实际地址，程序已经对地址进行了右移1bit。
    date_reg：数据读入设备的位置或者寄存器地址。
    count   ：要读入的数据的数量
举例说明：
    i2c_read 0 0x1 0 0x10 2 //i2c0向地址为1的外设的0x10处读取2个值。（实际意义查看对应的外设手册）
```

4、i2c 从设备功能使能

```
命令名称：i2c_init_slave
命令参数：bus_num dev_addr
命令说明：
    bus_num：ls2K300只有i2c2可以使能从设备功能，该参数只能取2。
    dev_addr：设置i2c从设备的地址。
举例说明：
    i2c_init_slave 2 0x1 //设置i2c2为从设备功能，地址为0x1。（从设备的可访问位置和数据请查看cpu用户手册）
```

b、内核下的测试命令

1) 主设备测试

```
linux内核下，有比较完善的i2c-tools来进行i2c操作:
i2cdetect -l //查看i2c总线的busnum
```

```
i2cdetect -y busnum //探测具体busnum上的外设地址。
i2cget           //读取外设数据，可通过i2cget -h查看使用说明。
i2cset           //写入i2c外设数据，可通过i2cset -h查看使用说明。
```

4) 从设备测试
待补充

2.7 CAN

功能测试

a、pmon下测试--暂不支持

b、内核下测试

1、内核配置CAN

配置方法：在内核源码目录下输入命令“make menuconfig ARCH=loongarch”，确保以下配置被选择

CONFIG_CAN_DEV、CONFIG_CAN_LSCAN、CONFIG_CAN_LSCAN_PLATFORM、CONFIG_CAN_RAW、
CONFIG_CAN_BCM、CONFIG_CAN_GW

配置成功：被选择的项，界面中显示是“*”号，或者对应项后面是y

2、以can1和CANFD盒（USBCAN-FD-200U）连接，执行can_test.sh脚本，在执行前修改参数，内容如下：

```
#!/bin/sh
n=1
bit=1000000
dbit=1000000
drate=0.8

ip link set can$n down
sleep 1
set +x
sleep 1
ip link set can$n type can bitrate $bit sample-point $drate dbitrate $dbit dsample-point $drate fd on restart-ms 1000

ip link set can$n up

#-f:send -g:delay time /ms -l:backage long -D:send data -n:send times
cangen -f -g 2 -L 32 -D 112233445566 can$n -n 1

#candump can0 &
```

4、两个CAN设备互相连接，测试

```
#!/bin/sh
n1=0
n2=1
bit=1000000
dbit=1000000
drate=0.8

ip link set can$n1 down
ip link set can$n2 down
sleep 1
set +x
sleep 1
ip link set can$n1 type can bitrate $bit sample-point $drate dbitrate $dbit dsample-point $drate fd on restart-ms 1000
ip link set can$n2 type can bitrate $bit sample-point $drate dbitrate $dbit dsample-point $drate fd on restart-ms 1000

sleep 1
ip link set can$n1 up
ip link set can$n2 up

#candump can$n1 &

#-f:send -g:delay time /ms -l:backage long -D:send data -n:send times
cangen -f -g 2 -L 32 -D 112233445566 can$n2 -n 1
```

测试成功的现象：串口端会打印32字节的字符串 1122334455660000....（0补齐）

注：

以上脚本需要修改四个参数：

n:测试的can序号，例如can注册的名称是can1.则n=1；
bit：仲裁域波特率，例如1Mbps,则bit=1000000；
dbir：数据域波特率，例如1Mbps.则dbit=1000000，注意dbit<=bit；
dtate：设置比分比，例如80%，则dbit=0.8；
candump：接收命令；

2.8 GPIO

功能测试

a、pmon下测试

注：以下命令的描述均不涉及修改复用，复用关系的修改请自行查看处理器手册

1、gpio初始化

命令名称：gpio_init
命令参数：num dir
命令说明：
num：ls2K300的node上有106个gpio，取值范围0~105。
dir：gpio的使能方向，0表示输出，1表示输入。
举例说明：
gpio_init 76 1 //node gpio 76 设置为输入

2、gpio 输出设置

命令名称：gpio_set
命令参数：num val
命令说明：
num：ls2K300的 node上有106个gpio，取值范围0~105。
val：gpio输出电平，0表示低电平，1表示高电平。
举例说明：
gpio_set 76 1 // gpio 5 输出高电平

3、gpio 输入值获取

命令名称：gpio_get
命令参数：num
命令说明：
num：ls2K300的node上有106个gpio，取值范围0~106。
举例说明：
gpio_set 76 //读取gpio 76 的输入值。

b、内核下测试

内核下已经加入了gpio的驱动，可直接使用linux通用方式进行控制：

1、内核配置

```
Device Drivers --->
  *- GPIO Support --->
    [*] /sys/class/gpio/... (sysfs interface)
```

2、使用说明

/sys/class/gpio：控制gpio的目录
/sys/class/gpio/export：文件用于通知系统需要导出控制的GPIO引脚编号
/sys/class/gpio/unexport：用于通知系统取消导出
/sys/class/gpio/gpiochipX：目录保存系统中GPIO寄存器的信息，包括每个寄存器控制引脚的起始编号base，label记录了控制器的名称，node的gpio名称为：ls-gpio，base从0开始，brige的gpio名称为：ls7a-gpio，base从32开始，即编号0~31为node的gpio，编号32~95为brige的gpio。

举例说明：

```
echo 12 > /sys/class/gpio/export //生成/sys/class/gpio/gpio12目录，如果没有出现相应的目录，说明此引脚不可导出
echo "out" > /sys/class/gpio/gpio12/direction //direction可设置为in、out、low、high。（high/low设置时方向同时为输出）
echo 1 > /sys/class/gpio/gpio12/value //输出value文件是端口的数值，为1或0。（gpio输出时操作）
```

```
cat /sys/class/gpio/gpio12/value //gpio端口的输入值（gpio输入时操作）
```

3、内核下的gpio中断处理测试
待补充

2.9 PWM

PWM功能测试

a、pmon下测试

1、pwm控制

命令名称：pwm_control

命令参数：num low full

命令说明：

num：ls2K300上有4个pwm，取值范围0~3。

low：表示低电平持续的周期。

full：pwm波形总持续周期。（高电平持续周期= full - low）

举例说明：

```
pwm_control 2 50 90 //pwm2 输出50倍周期的低电平，40倍周期的高电平（pwm模块的输入参考时钟为50Mhz，即每周期为1/50Mhz）
```

2、pwm 脉冲测量

命令名称：pwm_measure

命令参数：num l

命令说明：

num：ls2K300上有4个pwm，取值范围0~3。

举例说明：

```
pwm_measure 2 //pwm2 测量脉冲周期
```

b、内核下测试

1、内核下的配置

```
CONFIG_PWM_LS
```

2、以测试pwm3为例

```
echo 0 > /sys/class/pwm/pwmchip3/export
```

```
echo 0 > /sys/class/pwm/pwmchip3/pwm0/enable //失能PWM
```

```
echo 10000 > /sys/class/pwm/pwmchip3/pwm0/period //设置周期
```

```
echo 5000 > /sys/class/pwm/pwmchip3/pwm0/duty_cycle //设置低脉冲
```

```
echo 1 > /sys/class/pwm/pwmchip3/pwm0/enable //使能PWM
```

注意：这里设置的周期和低脉冲的参数，并不是实际填充到低脉冲和脉冲周期寄存器的值，根据驱动中的计算，寄存器中值会缩小10倍。

2.10 EMMC

常用配置

```
OPTION EMMC
```

```
emmc0 at localbus0 flags 0 //使能emmc pmon下访问
```

功能测试

a、pmon下测试

1、pmon下emmc是以设备的形式存在，设备名称为emmc0。

可使用pmon下命令 devls 查看。

a、内核下测试

1、内核下配置

配置方法：在内核源码目录下输入命令“make menuconfig ARCH=loongarch”进入配置界面，在界面中搜索要配置项的名称，使处于被选择的状态。SDIO需要配置的项如下：


```
CONFIG_MMC_BLOCK、CONFIG_MMC_LS2K、CONFIG_LS_APBDMAC、CONFIG_DMADEVICES NLS_CODEPAGE_936。
```

2、内核下的emmc是以mmcblock的形式存在，可以使用fdisk -l 命令查看

3、读写测试

1) 给设备mmcblock0分区：fdisk /dev/mmcblock0

根据提示依次选择“n, p, w”，得到分区 mmcblock0p1

2) 格式化分区为ext4格式：mkfs.ext4 /dev/mmcblock0p1

3) 挂载分区到指定文件夹：mount /dev/mmcblock0p1 /mnt

读写挂载的文件夹，就是访问emmc设备

读EMMC：ls /mnt/； 测试正确现象：调试串口打印emmc中的内容

写EMMC：cp file.txt /mnt； 测试正确现象：文件file.txt被成功复制到emmc中，并且内容不变

4) 卸载分区：umount 挂载的文件夹

4、测试性能

读性能：dd if=/dev/mmcblk0 of=/dev/zero bs=1M count=400

写性能：dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=400 其中bs是块大小，count是一次读写数据大小B

2.11 SDIO

常用配置

```
OPTION SDCARD
```

```
tfcard0 at localbus0 flags 0 //使能sdio pmon下访问
```

功能测试

a、pmon下测试

1、pmon下SDIO是以设备的形式存在，设备名称为tfcard0。

可使用pmon下命令 devls 查看。

b、内核下测试

1、内核配置

配置方法：在内核源码目录下输入命令“make menuconfig ARCH=loongarch”进入配置界面，在界面中搜索要配置项的名称，使处于被选择的状态。SDIO需要配置的项如下：

```
CONFIG_MMC_BLOCK、CONFIG_MMC_LS2K、CONFIG_LS_APBDMAC、CONFIG_DMADEVICES NLS_CODEPAGE_936。
```

2、查看设备

内核下的sdio是以设备mmcblock的形式存在，可以使用fdisk -l 命令查看

查看设备：fdisk -l

设备存在则会出现打印：/dev/mmcblock0

注意：当SDIO设备和EMMC设备同时存在时，fdisk -l会出现两个mmcblock设备，注意区分

3、读写测试

1) 给设备mmcblock0分区：fdisk /dev/mmcblock0

根据提示依次选择“n, p, w”，得到分区 mmcblock0p1

2) 格式化分区为ext4格式：mkfs.ext4 /dev/mmcblock0p1

3) 挂载分区到指定文件夹：mount /dev/mmcblock0p1 /mnt

读写挂载的文件夹，就是访问emmc设备

4) 卸载分区：umount 挂载的文件夹

2.12 LIO

功能测试

a、pmon下测试

连接nvram，使用m2、d2等命令读写LIO地址

b、内核下测试

同pmon，使用devmem等测试

2.13 I2S

接口描述

龙芯 2K0300 集成一个 I2S 控制器,数据宽度是 32 位,支持 DMA 传输,支持多家公司的 codec 芯片。I2S 控制器支持主或从模式。主模式时由 I2S 控制器产生位时钟信号、左右声道选择时钟信号和数据信号,从模式时 I2S 控制器接收位时钟信号、左右声道选择时钟信号和数据信号。

常用配置

暂不支持

功能测试

a、pmon下测试

暂不支持

b、内核下测试

1、内核下的i2s总线做为音频子系统的一部分，支持ALSA（ASOC）规范

2、内核下打开如下配置

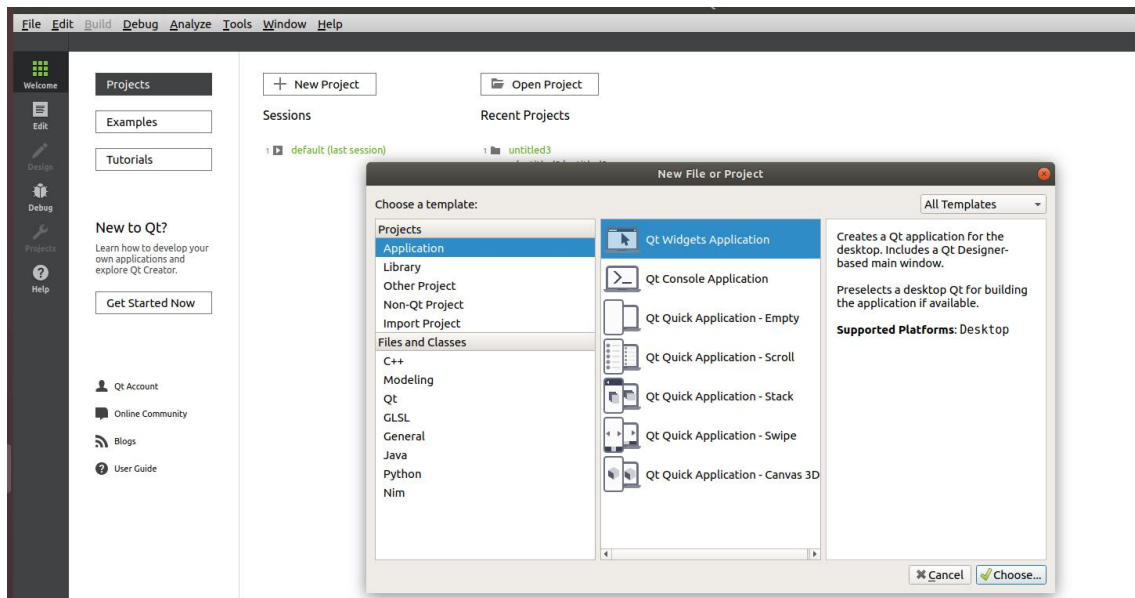
```
CONFIG_SND_LS_I2S ,  
CONFIG_SND_LS_PCM ,  
CONFIG_SND_LS_SOUND ,  
SND_SOC_ES8323_I2C  
CONFIG_SND_LS_2K300
```

3、/dev/snd目录下会枚举alsa设备。

4、使用aplay和arecord命令测试放音与录音

3. QT开发

（1）通过IDE-qtcreator构建需要的QT功能



打开qtcreator，点击file->New File or Project,选择需要的工程后点击
choose，进行下一步，如下图所示，填入你需要的工程名称，选择桌面构建，
点击next，则构建完成，则可以编写想要的项目代码了。

（下图是构建一个mainwindow工程）

Qt Widgets Application

Location
Kits
Details
Summary

Introduction and Project Location

This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

Name:

Create in:

☐ Use as default project location


Qt Widgets Application

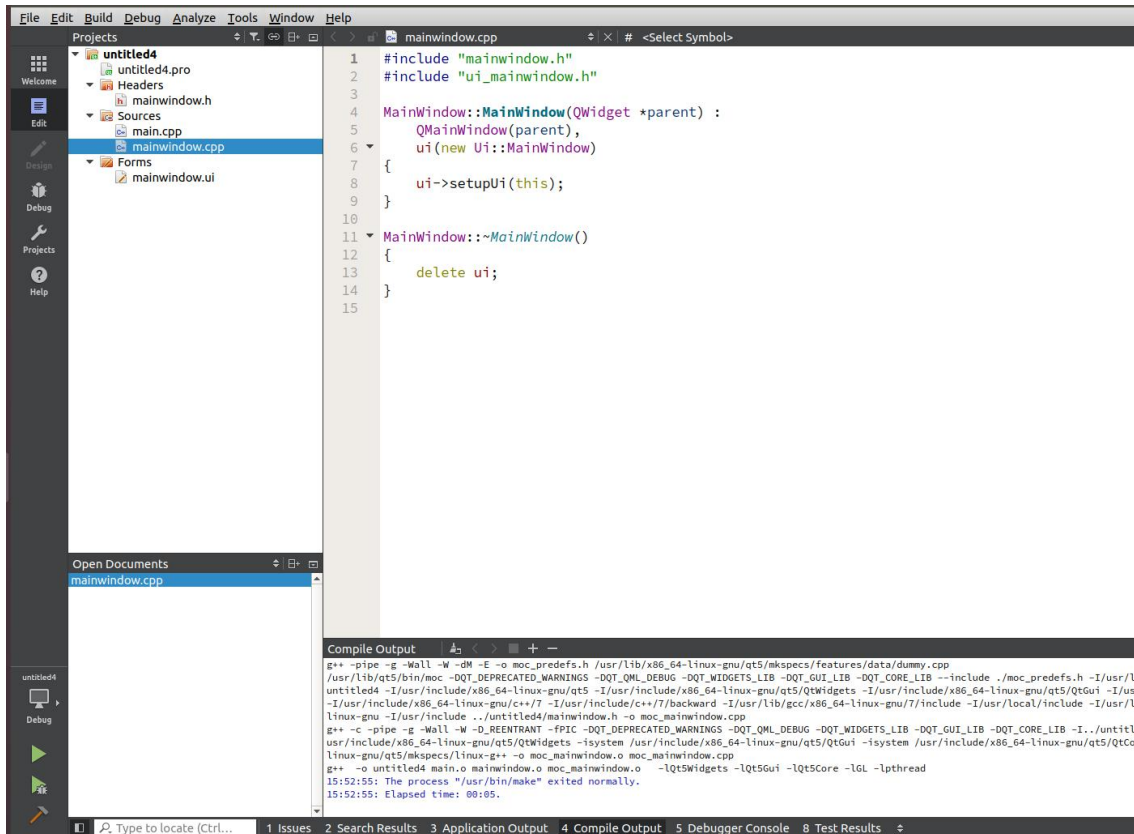
Location
Kits
Details
Summary

Kit Selection

The following kits can be used for project **untitled4**:

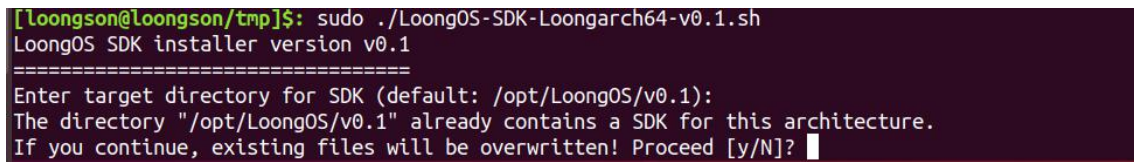
☒ Select all kits

☒  Desktop



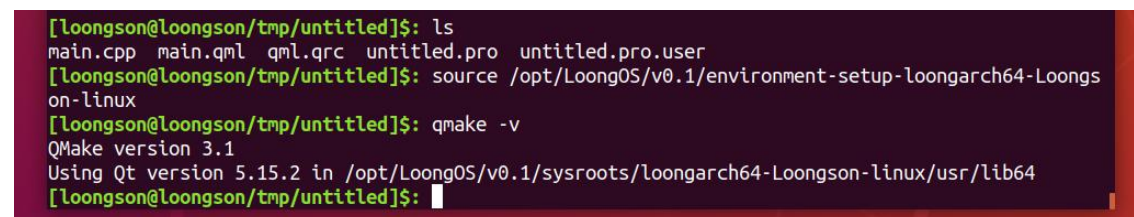
(2) 开始对编写结束的代码进行交叉编译。

首先安装我们的交叉编译工具链，回车选择默认，在选择y，则开始安装。



打开我们的代码工程目录，导出我们安装的交叉编译工具环境变量，如下图所示，（注：环境变量仅在当前窗口有效，窗口关闭则失效，再次重新导入可。）

输入`qmake -v`，查看环境变量是否设置成功，如下图路径版本，则设置成功。



开始编译代码，输入`qmake`构建我们工程的`MakeFile`文件，在输入`make`进行编译，等待编译结束，则可看到我们的二进制文件，把二进制文件，拷贝传输到我们的板卡上，输入`./文件名 -platform linuxfb` 则可在屏幕显示我们的工程

项目界面。无UI界面项目，则直接运行二进制即可，不需要加后面制定fb参数。

```
[loongson@loongson/tmp/untitled]$: qmake
Info: creating stash file /tmp/untitled/.qmake.stash
[loongson@loongson/tmp/untitled]$: make
loongarch64-Loongson-linux-g++ -march=loongarch64 -mtune=la264 -mabi=lp64d -mstrict-align -fst
ack-protector-strong -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security
--sysroot=/opt/LoongOS/v0.1/sysroots/loongarch64-Loongson-linux -c -pipe -O2 -pipe -g -felimin
ate-unused-debug-types --sysroot=/opt/LoongOS/v0.1/sysroots/loongarch64-Loongson-linux -O2 -std
=gnu++11 -Wall -Wextra -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_QUICK_LIB -DQT_GUI_LIB -DQT_QMLMODE
```

```
gOS/v0.1/sysroots/loongarch64-Loongson-linux/usr/lib64/libQt5Network.so /opt/LoongOS/v0.1/sysroo
ts/loongarch64-Loongson-linux/usr/lib64/libQt5Core.so -lGL -lpthread
[loongson@loongson/tmp/untitled]$: ls
main.cpp main.qml qml.qrc qrc_qml.o untitled.pro
main.o Makefile qrc_qml.cpp untitled untitled.pro.user
[loongson@loongson/tmp/untitled]$: ldd untitled
not a dynamic executable
[loongson@loongson/tmp/untitled]$:
```