



武汉飞思灵微电子有限公司
Wuhan FisiLink Microelectronics Technology Co., Ltd

FSL91030(M)芯片

测试板及其软件使用手册

版本: V1.6

武汉飞思灵微电子有限公司

2023 年 5 月

1. 概述	3
1.1. 前言	3
1.2. 软件包结构	3
2. BSP 介绍	4
2.1. 软件版本	4
2.2. 启动流程	4
3. 测试板使用	7
3.1. 启动	7
3.2. 登录及升级 SDK	8
3.3. 端口编号	9
4. 版本编译&烧写	12
4.1. 环境&烧写工具准备	12
4.2. 裸机 sdk 编译&烧写	13
4.2.1. 解压裸机 sdk 代码置 tools 的同级目录	13
4.2.2. 导入环境变量	13
4.2.3. 编译&烧写	13
4.3. 业务 sdk 编译	15
4.3.1. 内置 cpu	15
4.3.2. 外置 cpu	16
4.4. Liunx sdk 编译&烧写	17
4.4.1. 硬件 flash 要求	17
4.4.2. 分区	17
4.4.3. 编译准备	18
4.4.4. 通过 JTAG 烧写 freeloader.elf	19
4.4.5. ubifs 版本编译&烧写（demo 板默认编译烧写方式）	21
4.4.6. ramfs + jffs2 版本编译&烧写	22
4.4.7. ramfs + ubifs 版本编译&烧写	24
4.4.8. uboot 下常用指令	25
5. 配置软件使用	26
6. SDK 使用	28
7. 版本记录	29

1. 概述

1.1. 前言

本文档是 FSL91030M 芯片测试板及其配套软件包的使用手册，用于介绍 FSL91030M 芯片软件包功能组件的移植、说明和开发指导，以及测试板的使用方法。

1.2. 软件包结构

FSL91030M 芯片软件包结构框图如下：

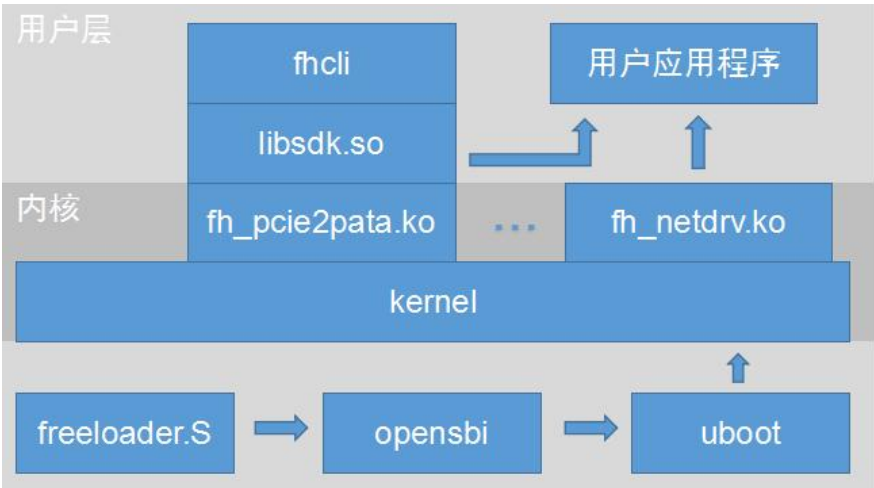


图 1-1 结构框图

FSL91030M 芯片软件包目录结构如下表所示：

表 1-1 软件包目录结构

软件包名	模块分类	模块名	说明
linux_sdk.tgz tools.zip	BSP	freeloder.elf	内部集成有 opensbi, uboot。启动后将引导进入 uboot, uboot 加载 kernel 和 rootfs 启动
		fdt.dtb	设备树
		kernel.bin	内核源码
		initrd.bin	根文件系统
	SDK	Kernel	内核模块驱动
		User	用户态代码
		Thd_code	使用工具源码
	编译脚本	Makefile	统一的编译脚本
	交叉编译工具链	riscv-nuclei-elf-gcc	编译 linux 系统镜像的工具
		riscv-nuclei-linux-gnu-gcc	编译软件包其它模块镜像工具

2. BSP 介绍

本芯片集成 SOC 子系统，内嵌 RISC-V 处理器，并提供丰富的外设接口，包括 DDR, uart, GPIO, I2C, NOR flash, SPI, MDIO, Watchdog 等。

2.1. 软件版本

软件版本：

序号	软件名称	软件包版本
1	freeloder	-
2	opensbi	v 0.7
3	u-boot	2020.07-rc2
4	Linux Kernel	5.8.0
5	rootfs（根文件系统）	busybox 版本： 1.31.1 glibc 版本： 2.29

2.2. 启动流程

芯片上电后启动流程分为如下 5 个步骤，分别是 **freeloder -> opensbi -> uboot -> kernel -> rootfs**，下面从五个阶段来进行分析：

freeloder 启动流程如下：

设置异常和中断跳转地址->分别搬运 **opensbi, uboot, kernel, rootfs, fdt** 到 **ddr** 对应的位置->跳转到 **opensbi**

Opensbi

RISC-V 架构中，存在着定义于操作系统之下的运行环境。这个运行环境不仅将引导启动 RISC-V 下的操作系统，还将常驻后台，为操作系统提供一系列二进制接口，以便其获取和操作硬件信息。RISC-V 给出了此类环境和二进制接口的规范，称为“操作系统二进制接口”，即“SBI”。Opensbi 为它的一种实现。

Opensbi 不仅提供引导作用，还提供了 M 模式转换到 S 模式的实现，同时在 S-Mode 下的内核可以通过这一层访问一些 M-Mode 的服务。

uboot

U-boot 提供命令式 shell 界面，类似 Linux 终端的命令行。支持的命令主要有：设置环境变量，网络测试指令（ping），TFTP 下载指令，NOR 操作指令（sp 等），内存操作指令（md, mw 等），启动内核指令等。

另外 U-boot 必须提供系统部署功能，通过网络传输（tftp）入内存，再将 DDR 中的镜像数据烧录至 Flash 上：

- ✧ 整个系统（包括 dtb, U-boot, Kernel, Rootfs 等镜像）的烧录工作；
- ✧ 单个分区中镜像的升级工作；

kernel

linux 内核在完成系统的初始化之后，加载各个模块控制器，然后挂载某个文件系统作为根文件系统。

1) TABLE DMA

支持通过 TABLE DMA 配置交换芯片表项，接口列表如下：

函数名	描述
fslral_dma_mem_read	通过 dma 读取表项
fslral_dma_mem_write	通过 dma 配置表项
dma_alloc	分配 dma 缓存
dma_free	释放 dma 缓存
get_dma_info	获取分配的 dma 缓存信息

2) PACKET DMA

支持通过 PACKET DMA 收发 CPU 和交换芯片之间的报文，接口列表如下：

函数名	描述
prepare_pkt	配置待发包的包头
fsl_common_tx	通过 pkt dma 发包
fsl_common_rx_register	注册收包处理函数
fsl_common_rx_unregister	注销指定的收包处理函数
fsl_common_rx_unregister_all	注销所有的收包处理函数
fsl_common_rx_start	开始接收包
fsl_common_rx_shutdown	停止接收包

3) I2C

通过/dev/i2c-x 标准字符设备访问，提供 IOCTL 等接口可以让应用程序访问并设置 I2C 寄存器。

4) MDIO

配置 CPU 温度，目前提供指令如下所示：

函数名	描述
mdio_mode_cfg	配置 mdio 是访问内部还是外部
mdio_master_read	通过 mdio 读
mdio_master_write	通过 mdio 写

rootfs

根文件系统是 Linux 系统中文件和数据的存储区域，通常它还包括系统配置文件，Busybox 等应用程序以及应用程序所需要的库。

当前文件系统为 tmpfs 格式，做的更改掉电丢失。如果要永久保存文件，可以先挂载 flash 设备，然后把文件放在 flash 设备中。包含的文件夹如下所示：

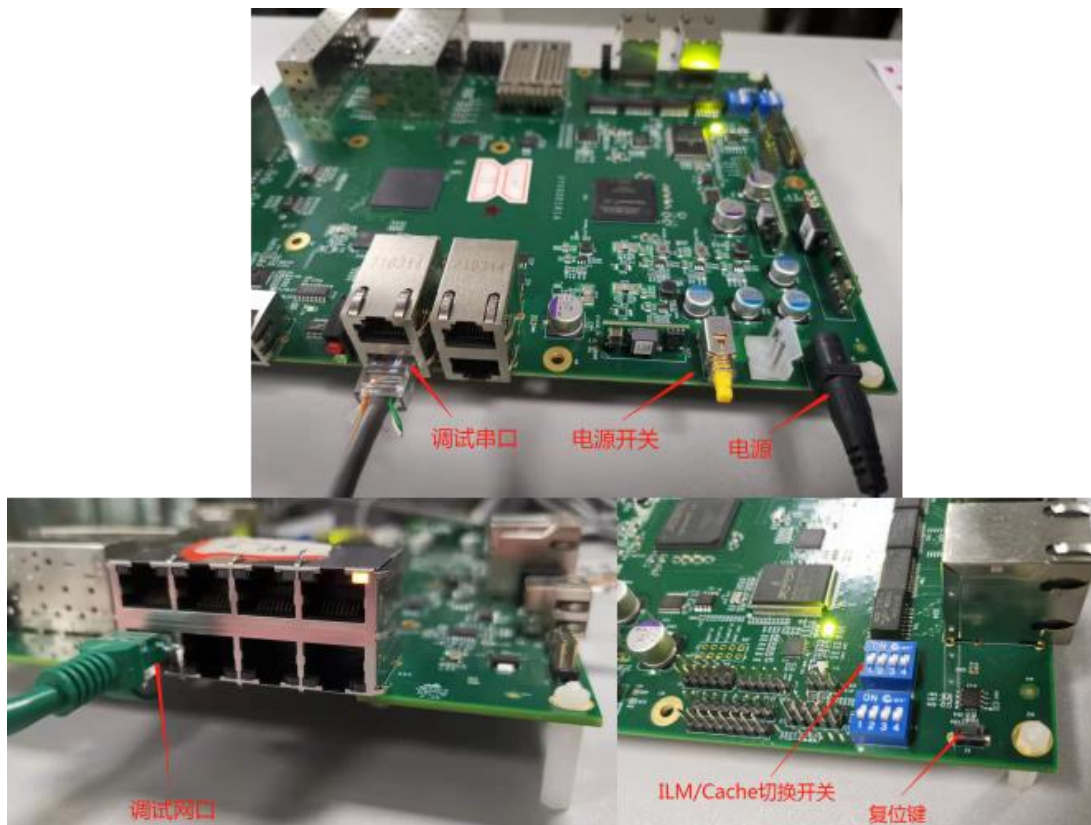
目录	描述
/etc	系统相关的配置文件
/etc/init	存放系统启动相关配置
/sbin /bin	存放系统相关的工具，由 Busybox 安装
/usr/bin /usr/sbin	存放用户的工具，由 Busybox 安装或者自定义
/lib	存放了很多库文件，应用程序会进去加载需要的动态链接库
/proc	系统的虚拟文件系统的挂载点，里面是虚拟的文件，表示了系统的一部分状态
/sys	系统的虚拟文件系统的挂载点，非常全面的表示了系统状态
/var	系统的虚拟文件系统的挂载点，用于存放一些数据
/dev	存放设备文件
/root	用户目录
/mnt	挂载 flash 设备

3. 测试板使用

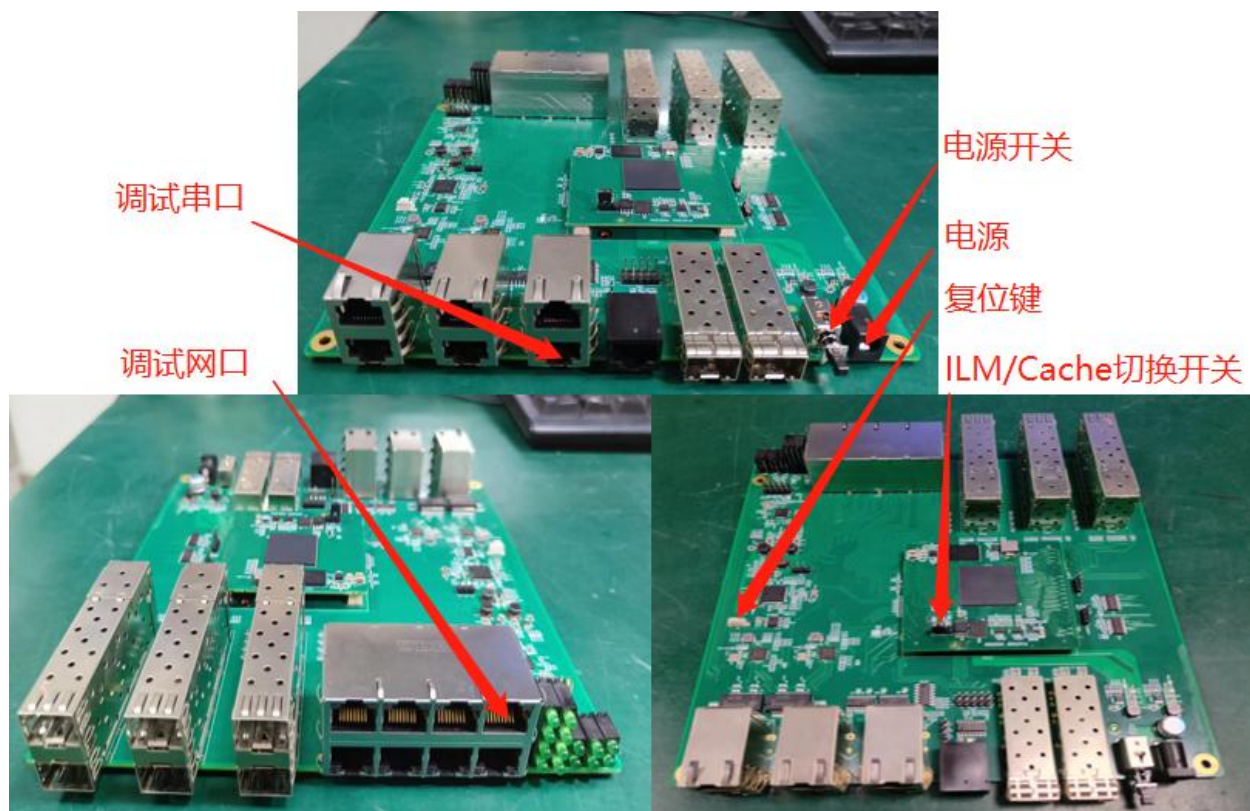
3.1. 启动

1) 组装连线

如图 3-1 所示，分别连接电源线、串口线（串口波特率 115200,8N1）和网线。ILM/Cache 切换开关关闭（选择为 Cache 模式）；然后打开电源开关。



无盖板测试板连接图



带盖板测试板连接图

2) 登录 linux 界面

等待 30s 后，系统自动进入 linux 登录界面。

```
Welcome to fsl xuanyuan System Technology
fsl login: █
```

3.2. 登录及升级 SDK

登录界面的账号：root，密码：fsl。

登录后，默认进入“/root”目录。该目录下的驱动文件（*.ko）在系统启动时已自动安装。运行“fhcli”命令，进入配置界面，相关的配置操作信息请详见第 5 章。

由于 CPU 的网口连接了交换芯片的端口 0（0 号电口），故需要先进入 fhcli 的命令行，将调试网口对应的网络设备设置为 eth0，才能正

常使用调试网口。相应的配置操作如下：

```
quit          //退出 fhcli
vi config.fhme //在 root 下
cpu_port_enable=1 //cpu 口使能
```



```
cpu_port=x    //0 号电口的逻辑端口号
wq 保存后 sync
重启 ./fhcli
```

可通过 `ifconfig` 命令，查看调试网口对应的网络设备为 `eth0`。

```
# ifconfig
[ 1268.940894] #enter get_stats
eth0      Link encap:Ethernet  HWaddr 00:02:AA:BB:CC:38
          inet addr:12.26.0.150  Bcast:12.26.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

网络可 `ping` 通后，可通过 `scp` 命令或 `tftp` 命令进行上传下载文件。

```
tftp -g -r libsdk.so 12.26.0.251
scp xxx@12.26.0.251:/home/libsdk.so    (注：更新该目录下的 libsdk.so 即可升级 sdk)
chmod 777 libsdk.so
```

（可选）可通过如下命令修改 `mac` 和 `ip`：

```
ifconfig eth0 down
ifconfig eth0 hw ether 00:02:AA:BB:CC:13
ifconfig eth0 up
ifconfig eth0 12.26.0.122 netmask 255.255.0.0
```

3.3. 端口编号

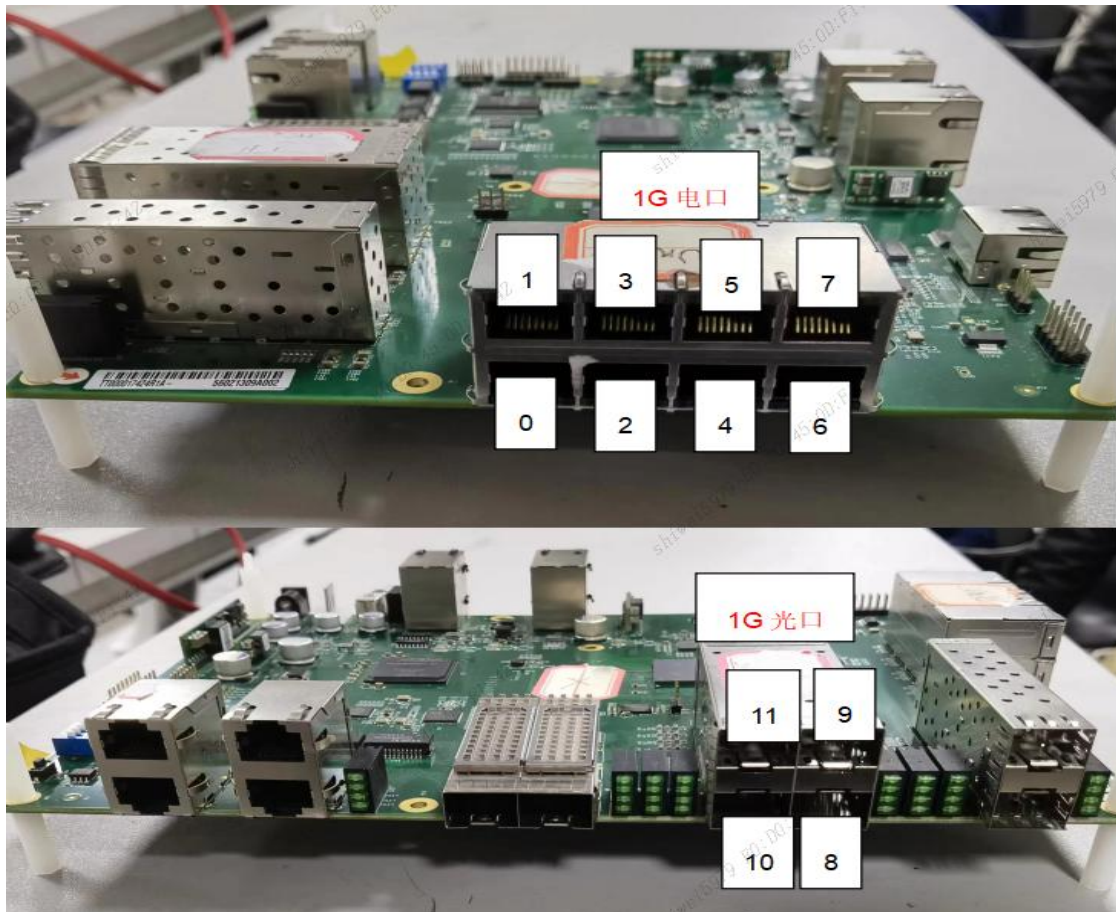
Demo 板上电后，通过串口登录，

```
login:root
```

```
password:fsl
```

```
运行 fhcli:  ./fhcli
```

将板子端口形态切换成 8+4 模式，8 个电口加 4 个 1G 的光口(芯片初始化必要执行)：

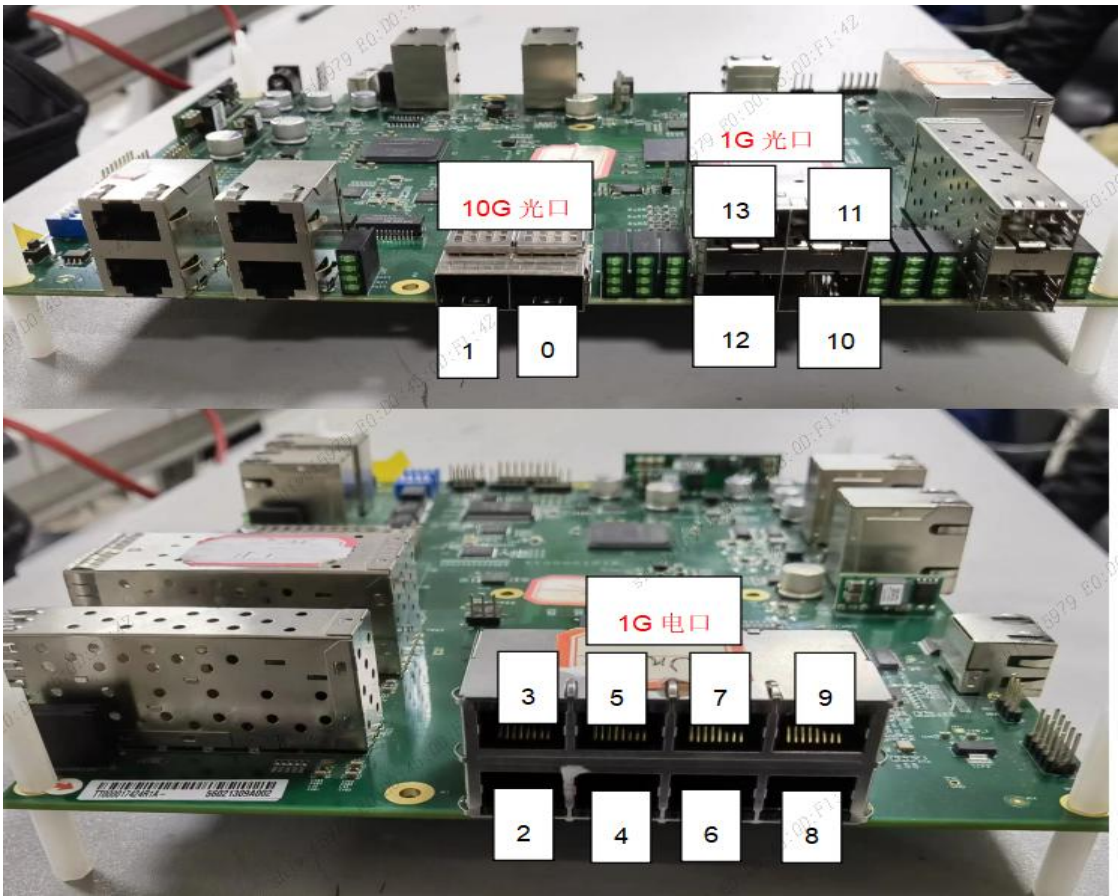


无盖板测试板连接图

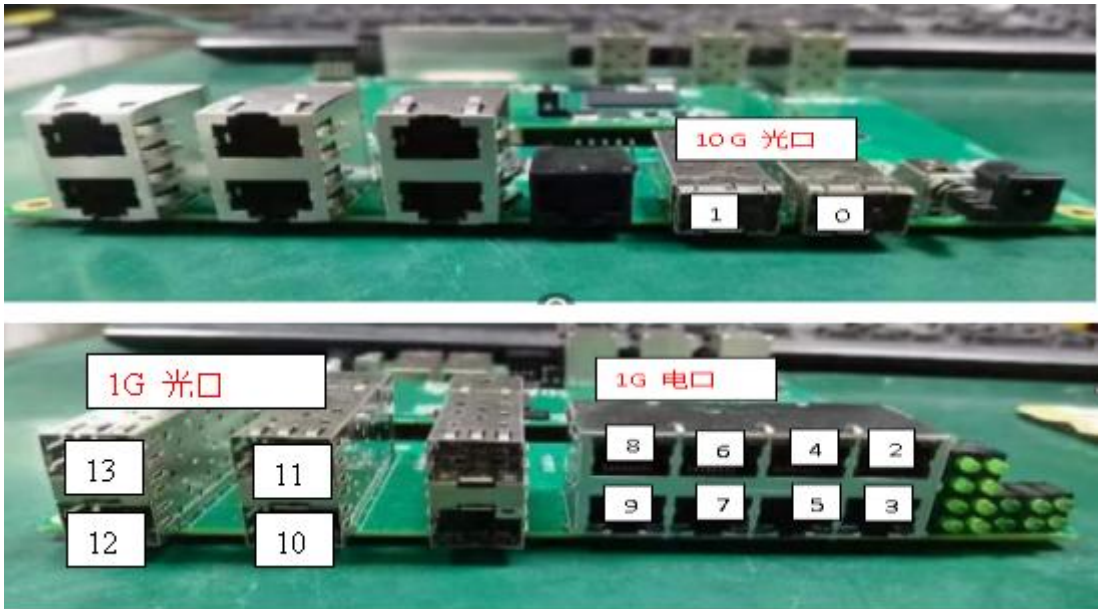


带盖板测试板连接图

将 demo 板端口形态切换成 8+4+2 模式, 8 个电口加 4 个 1G 光口 2 个 10G 的光口(芯片初始化必要执行):



无盖板测试板连接图



带盖板测试板连接图











4. 版本编译&烧写

4.1. 环境&烧写工具准备

虚拟机：Ubuntu 20.04 并安装以下工具

```
sudo apt-get install make
sudo apt install gcc
sudo apt install g++
sudo apt install git
sudo apt install bison
sudo apt install flex
sudo apt-get install device-tree-compiler
sudo apt-get install mtd-utils
```

在虚拟机中解压 tools.zip 工具包（可用于 sdk、linux_sdk、nuclei_sdk 的编译&烧写）：

 gcc.tgz	2023/1/4 17:31	WinRAR 压缩文件	127,015 KB
 HBird_Driver.zip	2023/2/17 10:02	WinRAR ZIP 压缩...	6,576 KB
 linux_gcc.tgz	2023/1/4 17:33	WinRAR 压缩文件	379,691 KB
 openocd_nuclei.cfg	2022/11/1 15:41	CFG 文件	2 KB
 openocd_nuclei_ilm.cfg	2022/11/1 15:41	CFG 文件	2 KB
 openocd-2022.12-linux-x64.tgz	2023/1/4 19:02	WinRAR 压缩文件	3,674 KB
 openocd-2022.12-win32-x32.zip	2023/2/17 10:04	WinRAR ZIP 压缩...	7,186 KB
 sdktoolchain_cfg.sh	2022/11/29 16:44	SH 文件	1 KB
 setup.sh	2022/11/24 14:59	SH 文件	1 KB
 toolchain_cfg.sh	2022/11/24 14:59	SH 文件	1 KB

gcc.tgz、linux_gcc.tgz 为编译工具链，需解压。

setup.sh 用于编译 nuclei_sdk 前导入工具链。sdktoolchain_cfg.sh 用于编译业务 sdk 前导入工具链。toolchain_cfg.sh 用于编译 linux_sdk 前导入工具链。

解压 HBird_Driver.zip，按照内部《安装说明.txt》进行安装。解压 openocd-2022.12-linux-x64.tgz 或 openocd-2022.12-win32-x32.zip。本文档后续烧写介绍以 linux 版 openocd 为例。

openocd_nuclei.cfg & openocd_nuclei_ilm.cfg 分别是针对 flash 和 RAM 的 openocd 配置文件。

在上位机建立一个 tftp 服务器或使用 tftp64.exe。以便在 uboot 下通过 tftp 下载更新内核和根文件系统。

4.2. 裸机 sdk 编译&烧写

4.2.1. 解压裸机 sdk 代码置 tools 的同级目录

```
tar -zxvf nuclei_sdk_v x_x_xx.tgz
```

4.2.2. 导入环境变量

修改 ~/tools/setup.sh 中的工具链路径

在 ~/tools/下 source setup.sh

```
NUCLEI_TOOL_ROOT=/home/sven/xy1030/Nuclei/gcc/bin
NMSIS_ROOT=../NMSIS

# Create your setup_config.sh
# and define NUCLEI_TOOL_ROOT like below
# NUCLEI_TOOL_ROOT=/home/develop/Software/Nuclei
# SETUP_CONFIG=setup_config.sh

[ -f $SETUP_CONFIG ] && source $SETUP_CONFIG

[ -f .ci/build_sdk.sh ] && source .ci/build_sdk.sh
[ -f .ci/build_applications.sh ] && source .ci/build_applications.sh

echo "Setup Nuclei SDK Tool Environment"
echo "NUCLEI_TOOL_ROOT=$NUCLEI_TOOL_ROOT"

export PATH=$NUCLEI_TOOL_ROOT:$PATH
```

4.2.3. 编译&烧写

在 ~/nuclei-sdk-fh/nuclei_sdk_soc_test_cases/fh/RELEASE

```
make clean dasm/bin CORE=ux600f DOWNLOAD=flash HUBMODE_SW=0 BOARD_TYPE=1
LED_MODE=3 MANAGEMENT=1
```

HUBMODE_SW: 0(8+4)、1(8+2)、2(8+6)、3(8+4+2)

MANAGEMENT: 0(1030)、1(1030M)

BOARD_TYPE=1(COMMON) (FSL old demo boards(without pinch plate) Or user's boards)				
LED_MODE[1:0]=00	GEPHY	[Link/Act]		
	1G	[Link/Act]		

	10G	[Link/Act]		
LED_MODE[1:0]=01	GEPHY	[SPD1000/Link/Act]	[SPD100(10)/Link/Act]	
	1G	[SPD1000/Link/Act]	[Disable]	
	10G	[SPD1000/Link/Act]	[SPD10G/Link/Act]	
LED_MODE[1:0]=10	GEPHY	[SPD1000/Link/Act]	[SPD100/Link/Act]	[SPD10/Link/Act]
	1G	[SPD1000/Link/Act]	[Disable]	[Disable]
	10G	[SPD1000/Link/Act]	[SPD10G/Link/Act]	[Disable]
LED_MODE[1:0]=11	GEPHY	[Link/Act]	[SPD1000]	[SPD100]
	1G	[Link/Act]	[SPD1000]	[Disable]
	10G	[Link/Act]	[SPD1000]	[SPD10G]
BOARD_TYPE=0(OWNER) (FSL new demo boards(with pinch plate))				
LED_MODE[1:0]=00	GEPHY	[Link/Act]		
	1G	[Act]		
	10G	[Act]		
LED_MODE[1:0]=01	GEPHY	[SPD1000/Link/Act]	[SPD100(10)/Link/Act]	
	1G	[Act]	[Disable]	
	10G	[Act]	[Disable]	
LED_MODE[1:0]=10	GEPHY	[Link]	[Act]	[Disable]

	1G	[Act]	[Disable]	[Disable]
	10G	[Act]	[Disable]	[Disable]
LED_MODE[1:0]=11	GEPHY	[Link/Act]	[SPD1000]	[Disable]
	1G	[Act]	[Disable]	[Disable]
	10G	[Act]	[Disable]	[Disable]

烧写参考 4.4.4 中的 freeloader.elf 的烧写步骤。

4.3. 业务 sdk 编译

4.3.1. 内置 cpu

1) 导入环境变量

修改 ~/sdk/build_config.cfg 中的 SDK_PATH 为当前 sdk/ 所在路径

修改 ~/tools/sdktoolchain_cfg.sh 中的工具链路径

在 ~/tools/下 source sdktoolchain_cfg.sh

```
export PATH=/home/sven/xy1030/Nuclei/linux_gcc/gcc/bin:$PATH
export TOOLCHAIN_DIR=/home/sven/xy1030/Nuclei/linux_gcc/gcc
export CROSS_COMPILE=${TOOLCHAIN_DIR}/bin/riscv-nuclei-linux-gnu
export LD_LIBRARY_PATH=${TOOLCHAIN_DIR}/sysroot:${TOOLCHAIN_DIR}/sysroot/
lib64/ld64:${TOOLCHAIN_DIR}/sysroot/lib:${TOOLCHAIN_DIR}/sysroot/usr:
${TOOLCHAIN_DIR}/sysroot/usr/lib64/ld64
```

2) 编译

sdk 目录下:

make clean

make SDK_MODE=2 2>1.txt

3) 生成文件

sdk/out/lib/libsdk.so

sdk/out/bin/fhcli

4.3.2. 外置 cpu

1) 设置环境变量

```
export PATH=
```

```
export TOOLCHAIN_DIR=
```

```
export CROSS_COMPILE=
```

```
export LD_LIBRARY_PATH=
```

上面的路径使用自己编译工具链的路径

2) 修改编译工具链

打开 sdk 目录 make 文件夹下的 make.linux 文件,

```
26
27 $(info "ffffffff RUN_MODE is ${RUN_MODE}")
28 ifeq ($(R_SDK_MODE),810)
29 $(info "vvvvvvvvvv")
30 CC := ${TOOLCHAIN_DIR}/arm-wrs-linux-gnueabi-gcc
31 AR := ${TOOLCHAIN_DIR}/arm-wrs-linux-gnueabi-ar
32 RANLIB := ${TOOLCHAIN_DIR}/arm-wrs-linux-gnueabi-ranlib
33 LD:= ${TOOLCHAIN_DIR}/arm-wrs-linux-gnueabi-ld
34 CFLAGS += -march=armv7-a -mfloat-abi=hard -mfpu=neon -marm -mthumb-interwork -mtune=cortex-a7
--sysroot=/opt/windriver/wrlinux/8.0-fsl-ls10xx/sysroots/cortexa7hf-vfp-neon-wrs-linux-gnueabi
-std=c99 -Wextra -Wbad-function-cast -Wcast-align -Wcast-qual -Wchar-subscripts -Wmissing-proto
types -Wnested-externs -Wpointer-arith -Wredundant-decls -Wstrict-prototypes -Wparentheses -Wsw
itch -Wswitch-default -Wunused -Wuninitialized -Wunused-but-set-variable -Wno-unused-parameter
-Wno-missing-field-initializers -Wno-sign-compare -Wshadow -Wno-inline -MMD -MP -g -O2
35 endif
```

修改黄色高亮部分工具链为自己的编译工具链, 34 行 CFLAGS 根据工具链的不同, 可能需要加以调整。

3) 适配访问总线

适配外部 cpu 访问总线, 修改对应的 src/core/ral/ 目录下的 iic_fpga_func.c, mii_fpga_func.c, spi_fpga_func.c, 至少要保证一条访问通道能通。

4) 编译

sdk 目录下:

```
make clean
```

```
make SDK_MODE=1 2>1.txt
```

5) 生成文件

```
sdk/out/lib/libsdk.so
```

```
sdk/out/bin/fhcli
```


4.4. Liunx sdk 编译&烧写

以下默认为不低于 linux_sdk_v1_0_3 的版本，低于 linux_sdk_v1_0_3 的版本会额外注释。

4.4.1. 硬件 flash 要求

两个 spi 分别挂有 nor flash 和 nand flash。

nor flash

nor 应不小于 1M，用于放置启动引导程序 uboot，设备树 dtb，和 uboot 环境变量 env。

nor 如果大于 1M，也可以存放内核 kernel，根文件系统 rootfs（ramfs，jffs2）。

nandflash

nandflash 用于存放内核 kernel，根文件系统 rootfs（ubifs）。

kernel 和 rootfs 可以放在 nor，也可以放在 nand，只需要修改启动命令 bootcmd，从相应的地方加载即可。

如果 nor 足够大，可以存放 kernel 和 rootfs，也可以不要 nand。

4.4.2. 分区

本 linux sdk 以硬件有 64MB norflash，128MB nandflash 为例，提供 ramfs+jffs2、ramfs+ubifs、ubifs（demo 板默认）三种版本加载根文件系统。

ramfs+jffs2 都只需要 nor flash 即可，无需使用 nandflash

ramfs+ubifs 需要至少 1M 的 norflash 和 40M 的 nandflash。

ubifs 需要至少 1M 的 norflash 和 64M 的 nandflash。

用户可根据实际需要修改分区大小。请务必保证 linux_sdk/u-boot/include/configs/nuclei-hbird.h 与 linux_sdk/conf/nuclei_ux600fd.dts 中的分区信息正确。

同时修改 linux_sdk/u-boot/include/configs/nuclei-hbird.h 中相关的环境变量

Nor flash（64M）在 uboot 下分区如下：

名称	uboot	dtb	env	kernel	rootfs	rootfs_jffs2
分区大小(bytes)	896K	64K	64K	4M	7M	52M
分区首地址	0x0	0xE0000	0xF0000	0x100000	0x500000	0xc00000
作用	freeloder	设备树	环境变量	存放内核	存放 ramfs 的文件系统	存放并运行 jffs2 的根文件系统
uboot 分区	freeloder			kernel_nor	ramfs_nor	jffs2_nor
烧写文件	freeloder.elf / freeloder.bin			kernel.bin	initrd.bin	jffs2.img

Nand flash（128M）在 uboot 下分区如下：

名称	kernel	rootfs	rootfs_ubifs	reserved
分区大小(bytes)	4M	20M	16M / 40M	88M / 64M
分区首地址	0x0	0x400000	0x1800000	-
作用	存放内核	存放 ramfs 的文件系统	存放并运行 ubifs 的根文件系统	保留
uboot 分区	kernel_nand	ramfs_nand	ubifs_nand	-
烧写文件	kernel.bin	initrd.bin	ubifs.img	-

4.4.3. 编译准备

修改 ~/tools/toolchain_cfg.sh 中的工具链路径

在 ~/tools/下 source toolchain_cfg.sh

```
export PATH=$PATH:/home/sven/xy1030/Nuclei/gcc/bin:/home/sven/xy1030/Nuclei/
openocd/bin:/home/sven/xy1030/Nuclei/linux_gcc/gcc/bin
export TOOLCHAIN_DIR=/home/sven/xy1030/Nuclei/linux_gcc/gcc
export CROSS_COMPILE=${TOOLCHAIN_DIR}/bin/riscv-nuclei-linux-gnu
export LD_LIBRARY_PATH=${TOOLCHAIN_DIR}/sysroot:${TOOLCHAIN_DIR}/sysroot/
lib64/ld64:${TOOLCHAIN_DIR}/sysroot/lib:${TOOLCHAIN_DIR}/sysroot/usr:
${TOOLCHAIN_DIR}/sysroot/usr/lib64/ld64
```

确认 linux_sdk 下 ln -s freeloder_rootfs freeloder

Linux 系统开发工具包 `linux_sdk.tgz` 提供了编译 linux 系统的源码和编译环境。如果需要编译 linux 系统，可以解压并进入 `linux_sdk` 目录，然后执行相应的命令即可。

```
$make help          //查看命令

- buildroot_initramfs-menuconfig : run menuconfig for buildroot, configuration will be saved into conf/
- buildroot_initramfs_sysroot : generate rootfs directory using buildroot
- linux-menuconfig : run menuconfig for linux kernel, configuration will be saved into conf/
- buildroot_busybox-menuconfig : run menuconfig for busybox in buildroot
- uboot-menuconfig : run menuconfig for uboot
- initrd : generate initramfs cpio file
- bootimages : generate boot images for SDCard
- freeloader : generate freeloader(first stage loader) run in norflash
- freeloader4m : generate freeloader(first stage loader) 4MB version run in norflash
- upload_freeloader : upload freeloader into development board using openocd and gdb
- uboot : build uboot and generate uboot binary
- sim : run opensbi + linux payload in simulation using xl_spike
- clean : clean this full workspace
- cleanboot : clean generated boot images
- cleanlinux : clean linux workspace
- cleanbuildroot : clean buildroot workspace
- cleansysroot : clean buildroot sysroot files
- cleanuboot : clean u-boot workspace
- cleanfreeloader : clean freeloader generated objects
- cleanopensbi : clean opensbi workspace
- preboot : If you run sim target before, and want to change to bootimages target, run this to prepare environment
- presim : If you run bootimages target before, and want to change to sim target, run this to prepare environment
```

4.4.4. 通过 JTAG 烧写 freeloader.elf

其中 `freeloader.elf` 通过 `jtag` 烧写，`freeloader.elf` 烧写完成后，启动运行到 `uboot` 命令行，再烧写剩下的文件（由于 `freeloader.elf` 已经包含了 `fdt.dtb`，所以 `fdt.dtb` 可以不用再单独烧写）。

确保设备断电后，将设备拨码开关拨至下载模式。将测试板与电脑通过 `jtag` 连接好，测试板上电。

1> Linux 版 `openocd` 烧写 `freeloader.elf`（两种方式，任选其一）

如果是 linux 虚拟机，还能看到下图所示的设备：



在 `openocd_202212/openocd/bin` 下打开终端

```
sudo chmod 777 -R /dev/bus/usb //设置 jtag 可执行权限
./openocd -f openocd_nuclei.cfg
```

若连接成功，则会有如下打印：

```
Info : Nuclei SPI controller version 0x00000001
Info : Found flash device 'micron mt25ql512' (ID 0x0020ba20)
cleared protection for sectors 0 through 1023 on flash bank 0
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

保留以上终端，另外在 `linux_sdk/` 下打开终端

```
source toolchain_cfg.sh
riscv-nuclei-elf-gdb
target remote localhost:3333
load ./freeloader_rootfs/freeloader.elf
```

2> Windows 版 openocd 烧写 freeloader.elf（两种方式，任选其一）

修改 `openocd.bat` 第 1、2 行的路径为本地路径后将 `openocd.bat` 复制到桌面，双击运行第一个终端启动 `openocd`。若连接成功，则会有如下打印：

```
Info : Nuclei SPI controller version 0x00000001
Info : Found flash device 'micron mt25ql512' (ID 0x0020ba20)
cleared protection for sectors 0 through 1023 on flash bank 0
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

保留以上终端，另外在 `linux_sdk/` 下打开终端

```
source toolchain_cfg.sh
riscv-nuclei-elf-gdb
target remote xx:xx:xx:xx:3333 //PC 当前 IP 地址
load ./freeloader_rootfs/freeloader.elf
```

上述两种方式烧写成功后，如下图所示

```

Loading section .text, size 0xa0818 lma 0x20000000
Loading section .interp, size 0x21 lma 0x200a0818
Loading section .dynsym, size 0x18 lma 0x200a0840
Loading section .dynstr, size 0xb lma 0x200a0858
Loading section .hash, size 0x10 lma 0x200a0868
Loading section .gnu.hash, size 0x1c lma 0x200a0878
Loading section .dynamic, size 0x110 lma 0x200a0898
Loading section .got, size 0x8 lma 0x200a09a8
Start address 0x20000000, load size 657824
Transfer rate: 29 KB/sec, 13704 bytes/write.

```

4.4.5. ubifs 版本编译&烧写（demo 板默认编译烧写方式）

将 uboot、dtb、env 编成一 freeloader.elf，通过 jtag 下载到 nor flash，然后通过 uboot 将 kernel、ubifs 烧写到 nand flash。系统以 ubifs 方式启动，ubifs 可读可写，做的更改掉电不丢失。

该方式针对设备既有 nor flash，又有 nand flash 的应用场景。ubifs 保存需要修改的文件。

1>编译分区

修改 linux_sdk/u-boot/include/configs/nuclei-hbird.h：代码已经写好，选择 for ubifs boot

```

//for ubifs boot
#define MTDPARTS_DEFAULT
"mtdparts=spi_nor:896K(uboot),64K(dtb),64K(env);spi-nand:4M(kernel_nand),20M(ramfs_nand),40M(ubifs_nand)"

```

修改 linux_sdk/conf/nuclei_ux600fd.dts：代码已经写好，选择 for ubifs boot

```

//for ubifs boot
partition@01800000 {
    reg = <0x01800000 0x02800000>;
    label = "ubifs_nand";
};

```

2>编译 freeloader

```

make clean
make freeloader

```

生成 freeloader.elf、freeloader.bin、kernel.bin、initrd.bin 在 linux_sdk/freeloader_rootfs

3>制作 ubifs.img

```

cd ~/linux_sdk/kernel/led_164          //修改 Makefile 中的 TOOLCHAIN_DIR
make                                    //生成 led_164.ko

```

```
cd ~/linux_sdk/kernel/xy1000_net //修改 Makefile 中的 TOOLCHAIN_DIR
make //生成 xy1000_net.ko
```

拷贝 led_164.ko、xy1000_net.ko 和业务 sdk 编译出的 config.fhme、libsdk.so、fhcli 到 linux_sdk/driver\&lib/

拷贝文件到 rootfs：在 linux_sdk 下执行 ./driver\&lib/cp.sh ubifs

若要添加其它文件到 ubifs，可将文件放到 work/buildroot_initramfs_sysroot/ 下相应的路径，然后重新制作。

cp.sh 中集成了 mkfs.ubifs 指令，注意-c 的参数与实际保持一致，否则分区容量不对

4>下载

搭建好 openocd 与 gdb 环境，下载 freeloader.elf 文件到 nor flash，参考章节 4.4.4。

5>运行 uboot 烧写（更新）kernel、rootfs_ubifs

测试板的任意电口和串口和电脑的网口和 USB 口相连。上电后，按“asd”进入 uboot，第一次运行时，需要烧写 kernel、ubifs 到 nand flash。

uboot 中集成了相关烧写指令，将指定名称的文件放入服务器的 tftpboot 文件夹下（也可以使用软件 tftpd32）。

```
run updateos_nand //烧写 kernel.bin 到 nand flash 的 kernel_nand 分区
run updateubifs_boot //烧写 ubifs.img 到 nand flash 的 ubifs_nand 分区
setenv bootcmd run bootcmd_ubifs_boot //设置启动命令
saveenv //保存环境变量
boot //启动 linux
```

***若 linux_sdk 版本低于 linux_sdk_v1_0_3 则使用以下命令

```
run updateos //烧写 kernel.bin 到 nand flash 的 kernel_nand 分区
run updateubifs //烧写 ubifs.img 到 nand flash 的 ubifs_nand 分区
setenv bootcmd run bootcmd_ubifs //设置启动命令
saveenv //保存环境变量
boot //启动 linux
```

4.4.6. ramfs + jffs2 版本编译&烧写

将 uboot、dtb、env 编成一 freeloader.elf（freeloader.bin），通过 jtag 下载到 nor flash。然后通过 uboot 将 kernel、ramfs、jffs2 烧写到 nor flash。系统以 ramfs 方式启动，启动后再挂载 jffs2。ramfs 只读，做的更改掉电丢失；jffs2 可读可写，做的更改掉电不丢失。

该方式针对设备仅有一个 nor flash，无 nand flash 的应用场景。jffs2 保存需要修改的文件，由于 jffs2 包含文件越多，挂载时间越长，会影响使用体验，因此搭配 ramfs 使用。ramfs 保存不需要修改的文件，既能快速启动系统，又能减小 jffs2 大小及其挂载时间。

1>编译分区

修改 linux_sdk/u-boot/include/configs/nuclei-hbird.h：代码已经写好，选择 for ramfs + mount jffs2

```
//for ramfs + mount jffs2
define MTDPARTS_DEFAULT
"mtdparts=spi_nor:896K(uboot),64K(dtb),64K(env),4M(kernel_nor),20M(ramfs_nor),10M(jffs2_nor)"
```

2>编译 freeloader

```
make clean
make freeloader
```

生成 freeloader.elf、freeloader.bin、kernel.bin、initrd.bin 在 linux_sdk/freeloader_rootfs

3>制作 jffs2.img

```
cd ~/linux_sdk/kernel/led_164          //修改 Makefile 中的 TOOLCHAIN_DIR
make                                    //生成 led_164.ko
cd ~/linux_sdk/kernel/xy1000_net        //修改 Makefile 中的 TOOLCHAIN_DIR
make                                    //生成 xy1000_net.ko
```

拷贝 led_164.ko、xy1000_net.ko 和业务 sdk 编译出的 config.fhme、libsdk.so、fhcli 到 linux_sdk/driver\&lib/

拷贝文件到 rootfs：在 linux_sdk 下执行 ./driver\&lib/cp.sh ramfs

ramfs 为 freeloader 文件夹下编译出来的 initrd.bin 文件

若要添加其它文件到 ramfs，可将文件放到 work/buildroot_initramfs_sysroot/ 下相应的路径，然后重新制作。

打包 jffs2.img：mkfs.jffs2 -r rootfs_jffs2 -o jffs2.img -l -n -s 0x100 -e 0x10000 --pad=0x00A00000

//注意：-e --pad 的参数与实际保持一致，否则挂载会报错

4>下载

搭建好 openocd 与 gdb 环境，下载 freeloader.elf 文件到 nor flash，参考章节 4.4.4。

5>运行 uboot 烧写（更新）kernel、rootfs、rootfs_jffs2

测试板的任意电口和串口和电脑的网口和 USB 口相连。上电后，按“asd”进入 uboot，第一次运行时，需要烧写 kernel、ramfs、jffs2 到 nor flash。

uboot 中集成了相关烧写指令，将指定名称的文件放入服务器的 tftpboot 文件夹下（也可以使用软件 tftpd32）。

```
run updateos_nor           //烧写 kernel.bin 到 nor flash 的 kernel_nor 分区
run updateramfs_nor        //烧写 initrd.bin 到 nor flash 的 ramfs_nor 分区
run updatejffs2_nor        //烧写 jffs2.img 到 nor flash 的 jffs2_nor 分区
setenv bootcmd run bootcmd_nor //设置启动命令
saveenv                    //保存环境变量
boot                       //启动 linux
```

启动后系统将自动通过脚本/etc/init.d/S30initconfig.sh 挂载 jffs2。

4.4.7. ramfs + ubifs 版本编译&烧写

将 uboot、dtb、env 编成一 freeloader.elf，通过 jtag 下载到 nor flash。然后通过 uboot 将 kernel、ramfs、ubifs 烧写到 nand flash。系统以 ramfs 方式启动，启动后再挂载 ubifs。ramfs 只读，做的更改掉电丢失；ubifs 可读可写，做的更改掉电不丢失。

该方式针对设备既有 nor flash，又有 nand flash 的应用场景。ubifs 保存需要修改的文件。ramfs 保存不需要修改的文件。

1>编译分区

修改 linux_sdk/u-boot/include/configs/nuclei-hbird.h：代码已经写好，选择 for ramfs + mount ubifs

```
//for ramfs + mount ubifs
#define MTDPARTS_DEFAULT
"mtdparts=spi_nor:896K(uboot),64K(dtb),64K(env);spi-nand:4M(kernel_nand),20M(ramfs_nand),16M(ubifs_nand)"
```

修改 linux_sdk/conf/nuclei_ux600fd.dts：代码已经写好，选择 for ramfs + mount ubifs

```
//for ramfs + mount ubifs
partition@01800000 {
    reg = <0x01800000 0x01000000>;
    label = "ubifs_nand";
};
```

2>编译 freeloader

```
make clean
make freeloader
```

生成 freeloader.elf、freeloader.bin、kernel.bin 在 linux_sdk/freeloader_rootfs

3>制作 ubifs.img


```
cd ~/linux_sdk/kernel/led_164           //修改 Makefile 中的 TOOLCHAIN_DIR
make                                     //生成 led_164.ko
cd ~/linux_sdk/kernel/xy1000_net        //修改 Makefile 中的 TOOLCHAIN_DIR
make                                     //生成 xy1000_net.ko
```

拷贝 led_164.ko、xy1000_net.ko 和业务 sdk 编译出的 config.fhmc、libsdk.so、fhcli 到 linux_sdk/driver\lib/

拷贝文件到 rootfs：在 linux_sdk 下执行 ./driver\lib\cp.sh ramfs

ramfs 为 freeloader 文件夹下编译出来的 initrd.bin 文件

若要添加其它文件到 ramfs，可将文件放到 work/buildroot_initramfs_sysroot/ 下相应的路径，然后重新制作。

打包 ubifs.img：mkfs.ubifs -F -m 2048 -e 124KiB -c 128 -r rootfs_ubifs ubifs.img

//注意：-c 的参数与实际保持一致，否则分区容量不对

4>下载

搭建好 openocd 与 gdb 环境，下载 freeloader.elf 文件到 nor flash，参考章节 4.4.4。

5>运行 uboot 烧写（更新）kernel、rootfs、rootfs_ubifs

测试板的任意电口和串口和电脑的网口和 USB 口相连。上电后，按“asd”进入 uboot，第一次运行时，需要烧写 kernel、ramfs、ubifs 到 nand flash。

uboot 中集成了相关烧写指令，将指定名称的文件放入服务器的 tftpboot 文件夹下（也可以使用软件 tftpd32）。

```
run updateos_nand           //烧写 kernel.bin 到 nand flash 的 kernel_nand 分区
run updateramfs_nand        //烧写 initrd.bin 到 nand flash 的 ramfs_nand 分区
run updateubifs_nand        //烧写 ubifs.img 到 nand flash 的 ubifs_nand 分区
setenv bootcmd run bootcmd_nand //设置启动命令
saveenv                     //保存环境变量
boot                        //启动 linux
```

启动后系统将自动通过脚本/etc/init.d/S30initconfig.sh 挂载 jffs2

4.4.8. uboot 下常用指令

run updatedtb //烧写 fdt.dtb 到 nor flash 的 dtb 分区

run updatefreeloader //烧写 freeloader.bin 到 nor flash 的 uboot、dtb 分区，同时擦除 env 分区

5. 配置软件使用

系统启动后，在路径/root 下有 fhcli。fhcli 为配置软件，会调用 libsdk.so 提供的功能。在启动 fhcli 前，先检查/root/config.fhme 下的默认配置是否正确，尤其是访问寄存器的接口，以及交换模式的设置。Fhcli 在启动时，会根据 config.fhme 中的配置进行初始化设置。

fhcli 为命令行操作，可以通过输入命令完成各种功能。进入 fhcli 后如下图

```
FSLcli.0> [/root]
FSLcli.0> [/root] reglist TOP_CFG

mems below:
TOP_CFG_REG_AXI_CP_CFG
TOP_CFG_REG_CHIP_INFO_REG
TOP_CFG_REG_CHIP_INTR
TOP_CFG_REG_EFUSE_CSR
TOP_CFG_REG_INVISIBLE_REG
TOP_CFG_REG_PCS_SWITCH_MODE_CFG
TOP_CFG_REG_PLL_PD_CTRL
TOP_CFG_REG_PTP_PLL_DIVISOR
TOP_CFG_REG_RESET_GLOBAL
TOP_CFG_REG_RESET_MISC
TOP_CFG_REG_RESET_PCS_ADPT
TOP_CFG_REG_RESET_SERDES
TOP_CFG_REG_RESET_SERDES_PCS
TOP_CFG_REG_RESET_SWITCH_EPHY
TOP_CFG_REG_RGMII_ALM_CSR
TOP_CFG_REG_RGMII_CSR
TOP_CFG_REG_RGMII_DUPLEX
TOP_CFG_REG_SD1G_PLL_DIVISOR
TOP_CFG_REG_SDXG_PLL_DIVISOR
TOP_CFG_REG_SOC_PLL_DIVISOR
TOP_CFG_REG_SYNC_ETH_CFG
TOP_CFG_REG_SYS_PLL_DIVISOR
TOP_CFG_REG_TDC_CFG
TOP_CFG_REG_TI_MODE_CFG
TOP_CFG_REG_WORK_MODE_CFG
FSLcli.0> [/root] getreg TOP_CFG_REG_WORK_MODE_CFG
read addr 0x0
TOP_CFG_REG_WORK_MODE_CFG[0x0]=0x60fc2c: <SDXG_LED_MODE=0,SERIAL_CLK_EN=1,SERIAL_DATA_EN=1,CLOCK_CYCLE=0,BURST_CYCLE=0,BLINK_RATE=3,LED_ACTIVE_LOW=1,LED_MODE=7,REUSE_IND=0,SYNC_EN=0,SDXG1_EN=1,SDXG0_EN=3,GEPHY1_OFF=0,GEPHY0_OFF=0>
FSLcli.0> [/root]
```

进入 fhcli:

```
./fhcli
```

常用命令:

1.查询寄存器名（输入大写字符串，会输出带有该字符串的所有寄存器名）

```
reglist [字符串]
```

```
reglist I_NET
```

2.查询表项名（输入大写字符串，会输出带有该字符串的所有表项名）

```
memlist [字符串]
```

```
memlist I_NET
```

3.读寄存器

getreg [寄存器名]

getreg I_NET_DEF_VLAN_CTL

4.写寄存器

modreg [寄存器名] [域名 1]=val1 < [域名 2]=val2> ...

modreg I_NET_DEF_VLAN_CTL PORT_BMP=0

5.读表项

dump [表项名] <[索引]>

modify I_NET_PORT_SRM 0

6.写表项

modify [表项名] [索引] 1 [域名 1]=val1 <[域名 2]=val2> ...

modify I_NET_PORT_SRM 0 1 STP_CHK_EN=0 BRG_EN=0

7.配置交换芯片工作模式

modeswitch switch mode=[模式] //模式为 1, 2, 3, ...

8.查看交换芯片寄存器的访问接口

ext_intf get

9.设置交换芯片寄存器的访问接口

ext_intf set mode=3 addr=0x5c

6. SDK 使用

详见《FSL91030(M)芯片 SDK 接口文档_V1.4.pdf》、《FSL91030(M)芯片 SDK API 调用说明文档_V1.4.pdf》、《FSL91030(M)芯片 SDK 命令行配置业务示例_V1.32.pdf》

7. 版本记录

修改时间	版本号	修改记录	修改记录
2021.10.11	V1.0	初始版本。	施维
2022.12.01	V1.4	内容优化。	施维
2023.02.16	V1.5	补充测试板使用、sdk 编译；更新 linux_sdk 烧写。	施维
2023.05.21	V1.6	更新内容及勘误	施维