ZedBoard 上板说明

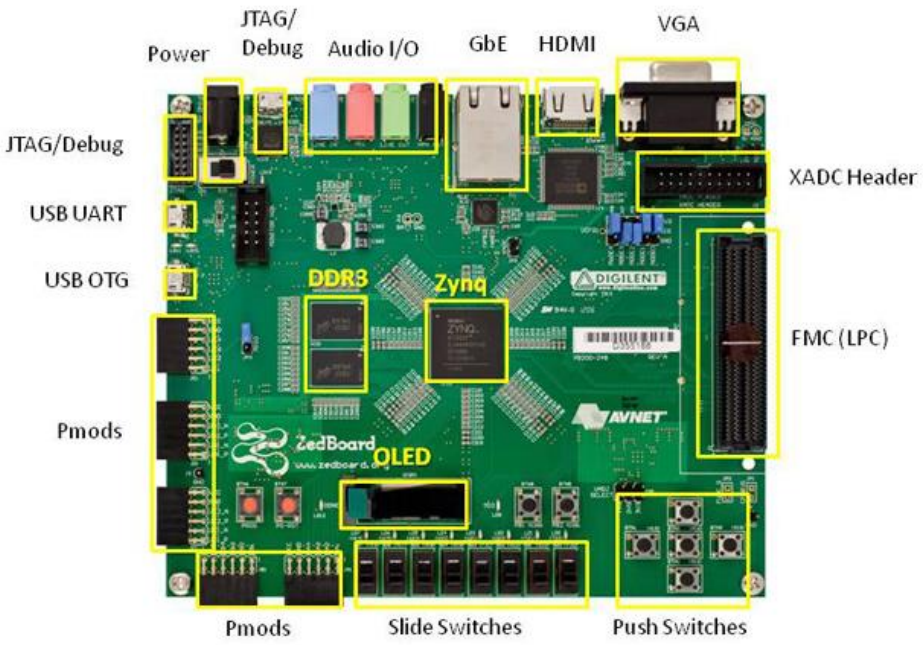
NOOP 上板

陈嘉鹏

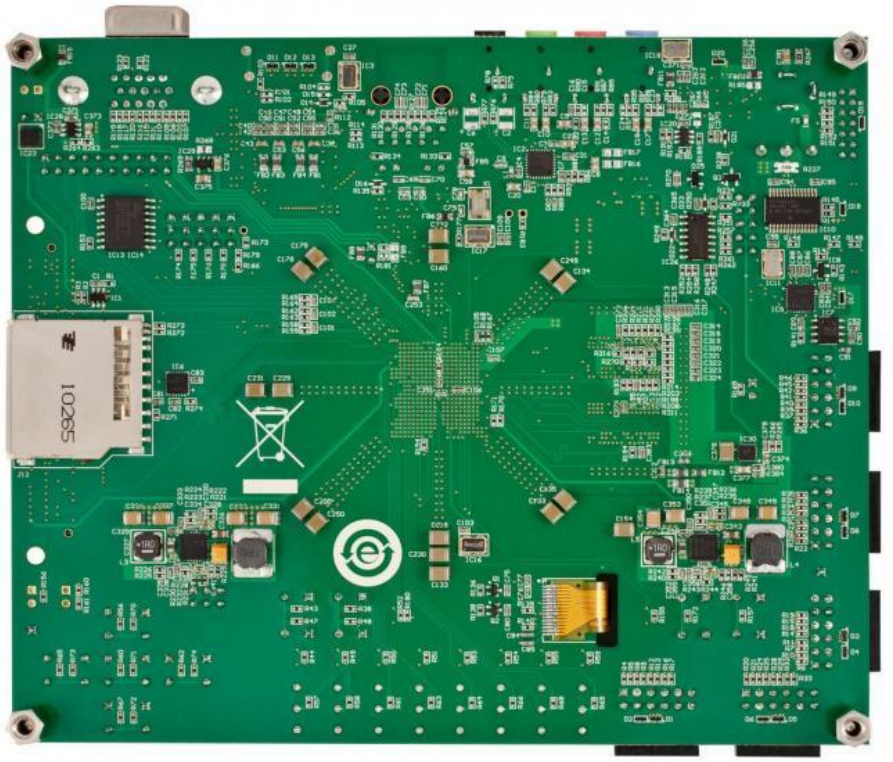
2017-12-20

目录

1、ZedBoard 介绍



* SD card cage and QSPI Flash reside on backside of board

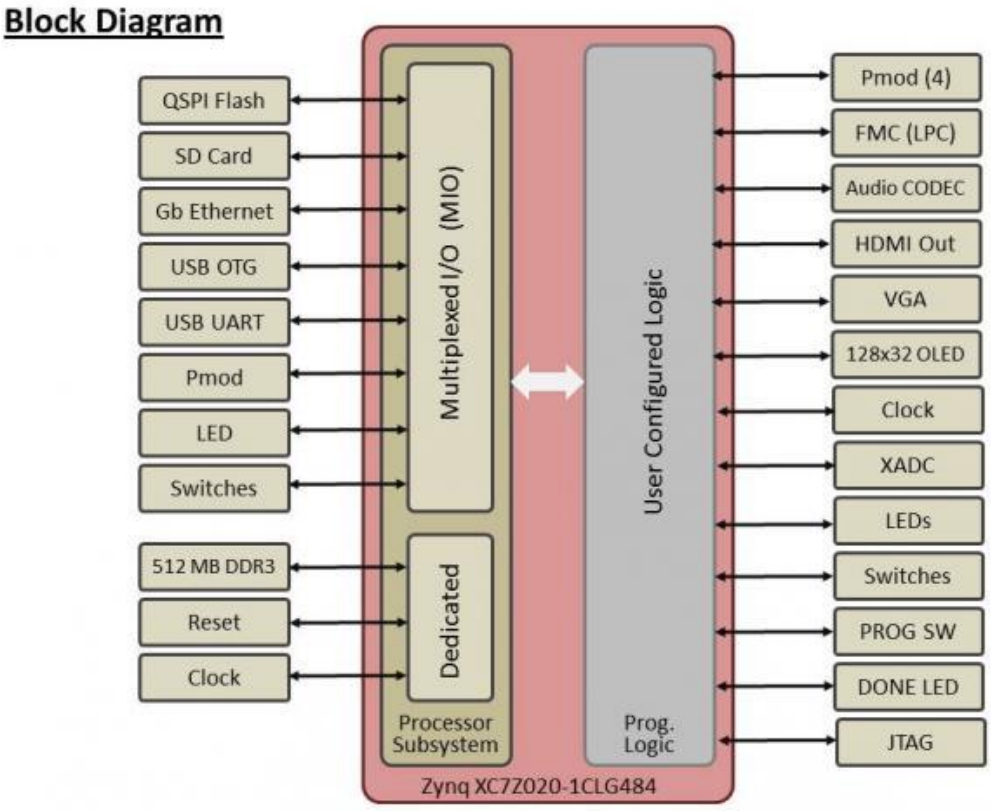


如上两图所示，这是我们这一次上板所使用的 ZedBoard 开发板，相比于其他开发板，ZedBoard 开发板在 SOC 中集成了一个 ARM CPU（ARM Cortex-A9 MPcore），使得在其上进行软件硬件开发可以借助这个 CPU 进行快速实现与测试。

我们在此次实验中所使用到的设备及接口分别是：

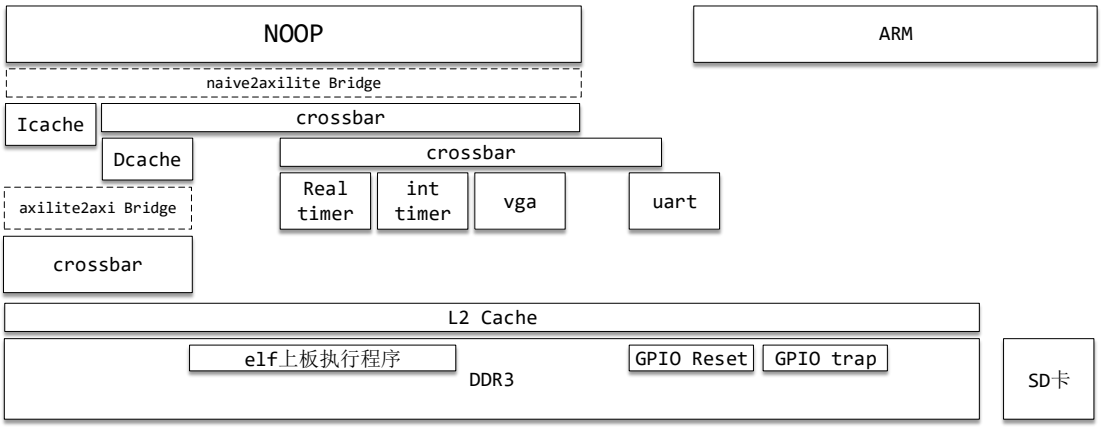
- ✧ Power　　电源接口
- ✧ USB-UART　　进行串口数据传输
- ✧ JTAG-Debug　　板子上写着(PROG)，用于烧板子以 debug 使用等
- ✧ VGA　　VGA 线接口
- ✧ GBE　　千兆以太网接口，连网线
- ✧ Zynq　　FPGA + ARM CPU
- ✧ DDR3　　内存
- ✧ SD 卡卡槽　　第二张图左侧，放 linux 系统，初始启动 ARM CPU

其内部连接结构大致如下所示：



Zynq 可以使用所有上图所示的接口或者外设，最中间部分是 ARM CPU 以及一些 FPGA 资源，两者可以通过配置以及电路连线设置使用所有以上的 IO，不过在使用的时候因为其在板子上的连接方式有所区别。

2、Uncore 及执行情况



上面说到 Zynq 内嵌了一个 ARM CPU，这个 CPU 通过 SD 卡进行启动，SD 卡上放置了 debian 系统以及相应的 VFS，ARM 通过读 SD 卡启动 debian，同时加载 VFS。这里可以 SD 卡看做是磁盘+BIOS，而 DDR3 就是了内存了。

启动后我们通过 shell 来与 ARM 交互，中间通过了 uart，host 通过 USB-UART 与 UART 交互再与 ARM 交互。ARM 要让 NOOP 开始执行程序，则需要使用 loader 加载，此时该程序位于 SD 卡中，ARM 使用 loader 程序将 SD 卡中 elf 上板执行程序加载到 ARM，解析后再放置到 DDR3 中相应的位置。

程序解析完毕，ARM 将 GPIO trap 设置为非 failed 也非 pass 状态，将 GPIO Reset 置为无效，则 NOOP 开始执行。从设定的起始位置读代码，这个代码就是刚刚 ARM 加载进去的。

通过 NOOP 接口发出请求。经过一个本地协议到 Axilite 协议转换桥到达 ICache，ICache 发起请求到 Axilite2axi 转换桥，桥将信号发送到 crossbar，crossbar 发现是送往 DDR3 的信号，则送到了 DDR3（L2 cache 暂且忽略掉）。

DDR3 得到数据请求，同时在 axi4 总线中有 brust 请求，DDR3 控制器处理这个 brust 请求，于是将该地址的数据，每次加 4 字节的方式 brust（一次请求多次数据发送）发送到 ICache，直到 ICache 一行全满。ICache 一旦拿到 NOOP 请求的 address 下的数据，立即发送到 NOOP（是否立即与实现相关），不继续等待 brust 完毕，这样 NOOP 就拿到了对应的指令。

数据的读取与指令相同，这里进行忽略。

继续说明对外设的访问，NOOP 像请求数据一样从 LSU 发出写或者读数据的请求，首先进行本地协议到 Axilite 的协议转换，到达 crossbar，crossbar 分析地址发现是外设的地址（如果是 DDR 的地址那么就像刚刚读取指令一样的步骤），于是将请求扔到连接外设的 crossbar，这个 crossbar 继续分析地址，发现是 Uart 的地址，于是就将数据请求扔到 Uart 的接口上（其他外设也是一样）。

在后面会说明数据写请求扔到这个 Uart 上会在 ssh+minicom 上显示相应写的字符。而读请求能从 ssh+minicom 接受数据的输入。因此实际上这里 Uart 起到了 ARM 与 NOOP 的连接通信作用，而实际的电路中自然就是连在一起的了。

程序执行完毕，NOOP 将结果写入 GPIO trap，ARM 监听 GPIO trap 发现变化了，知道 NOOP 执行完毕，就将 GPIO reset 置为有效，同时输出执行结果。

3、需要安装软件及配置

gtkterm ，串口终端，用于接收从 USB-UART 得到的信息，我们这里主要用于和 Zynq 中 ARM CPU 跑着的 linux 系统进行交互。

Cable driver，xilinx 烧板子线驱动。若能够正常烧板子则不需要安装，若存在问题，则到 vivado 目录下的 data/xicom/cable_drivers 目录下找 install_drivers 脚本进行安装。

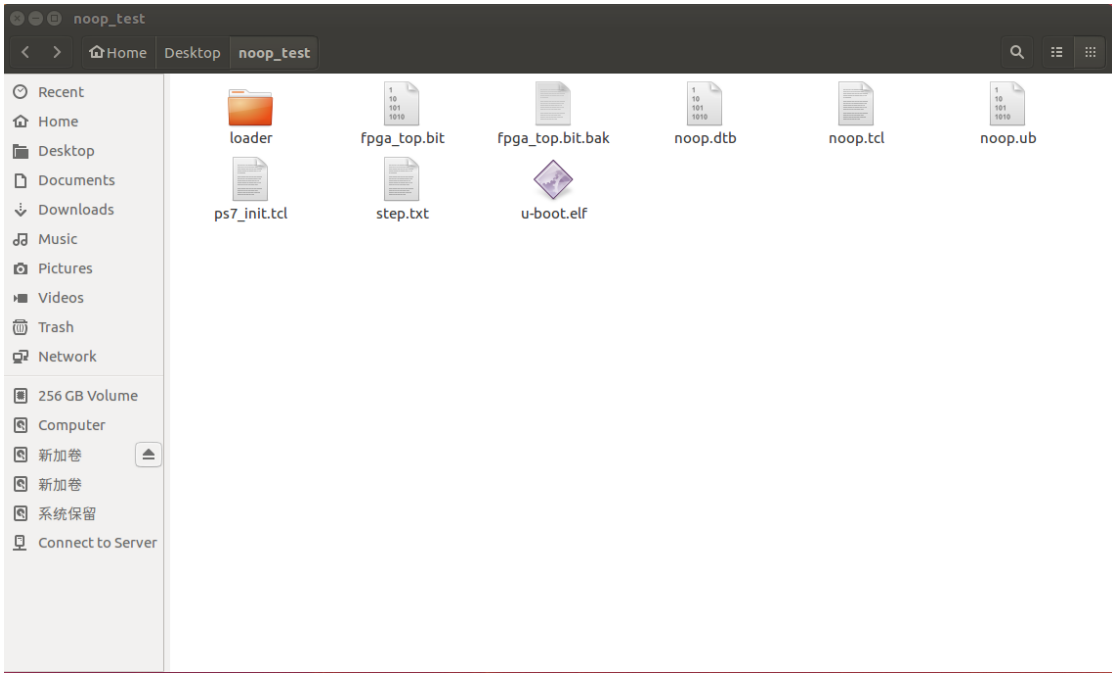
4、上板步骤

1、替换文件

将烧板子的文件夹下边文件 fpga_top.bit 以及 ps7_init.tcl 替换成工程刚生成好的 bitstream 文件以及相应目录下的 ps7_init.tcl 文件。（用 find 在工程目录找）

.bit 文件与 FPGA 最终如何连成电路有关。

ps7_init.tcl 与 clk 频率、DDR 初始化等有关。

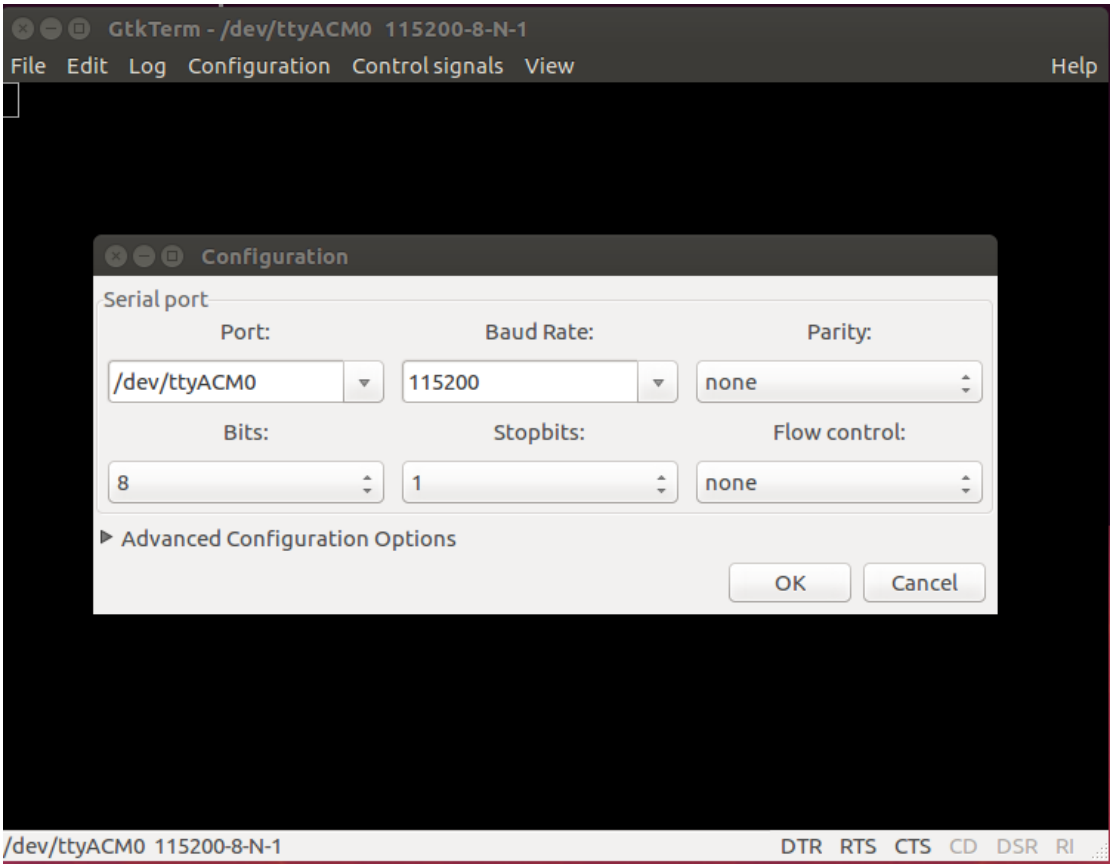


2、插线

插好 USB-UART、JTAG-Debug、Power 接口的连线，将 ZedBoard 网口用网线连接到路由器或者自己电脑，拨开 ZedBoard 开关接通电源。

3、烧板子

打开 gtkterm：sudo gtkterm

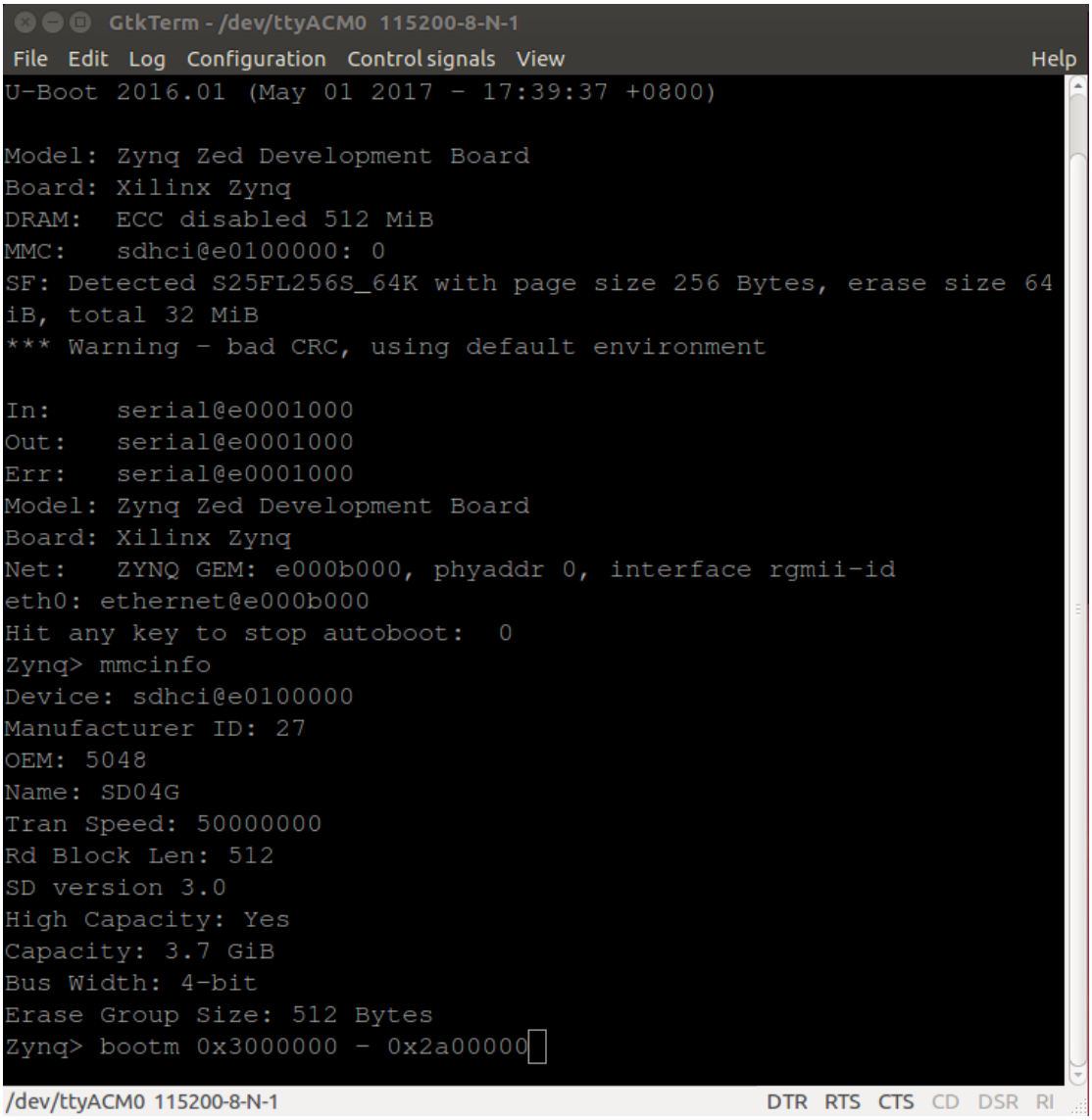


将端口调整为/dev/ttyACM0，波特率调整为 115200，如上图所示。（端口不正确后面的步骤无输出，波特率不正确显示乱码）

在烧板子的文件夹目录下打开 terminal，输入 xsdb，之后输入 source noop.tcl，开始烧入，gtkterm 产生相应的输出。

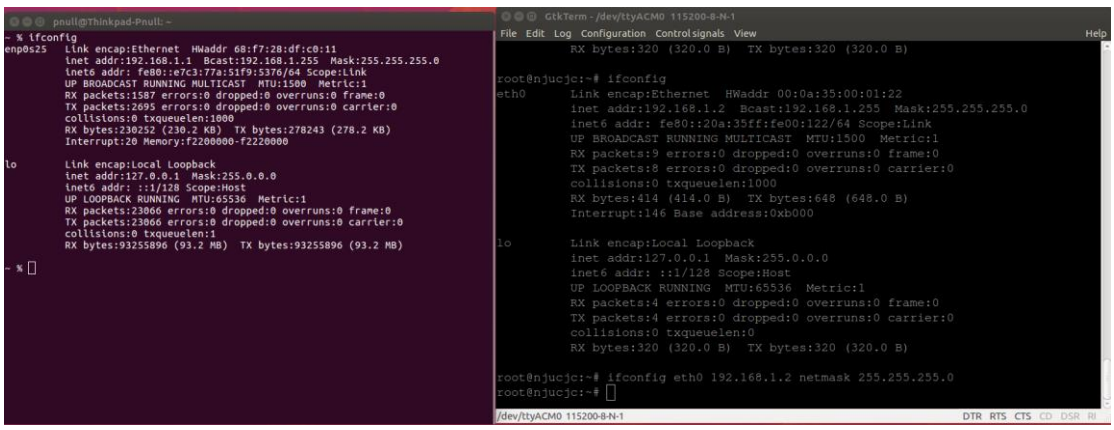
串口程序 gtkterm 中产生 Zynq 字样时，可以进行输入，输入 mmcinfo 查看系统信息，避免后序 VFS 加载失败。

之后输入 bootm 0x3000000 – 0x2a00000 系统开始启动，如下所示：



启动完毕，输入用户 root，密码 root 进入 shell 中。

将 Host PC 和 Zynq 配置到同一子网，如下



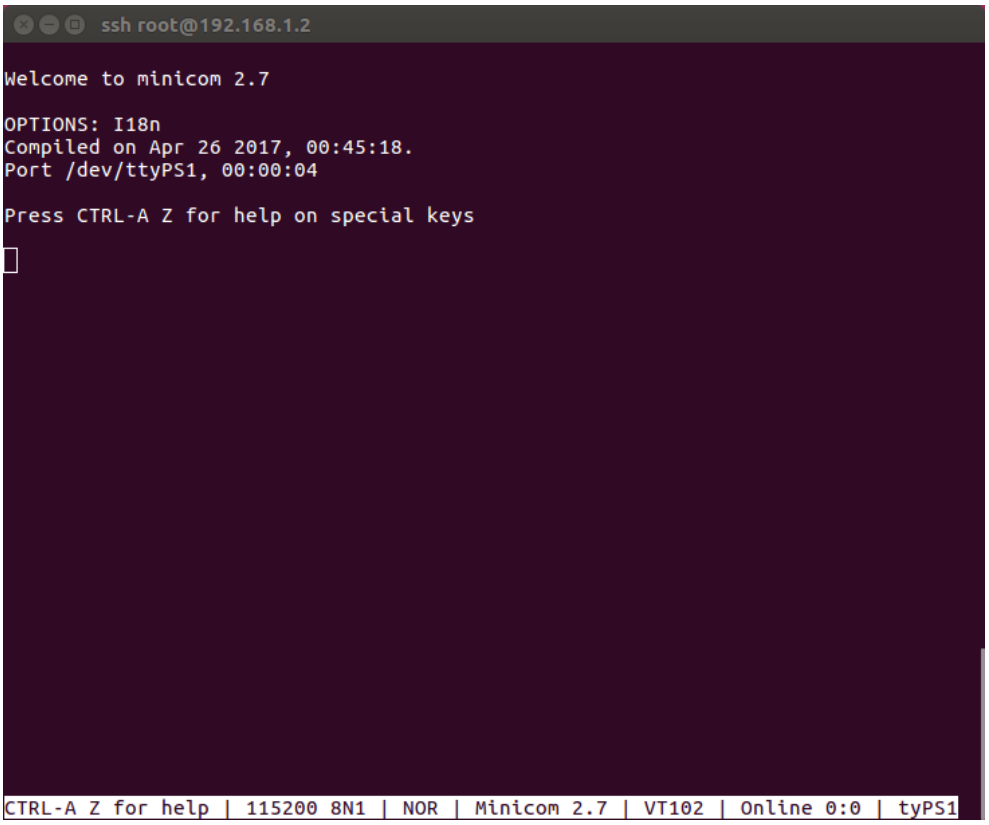
使用 Ping 测试两者之前能否通信

```
root@njucjc:~# ping -c 4 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.716 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.291 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.377 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.385 ms

--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.291/0.442/0.716/0.162 ms
root@njucjc:~#
```

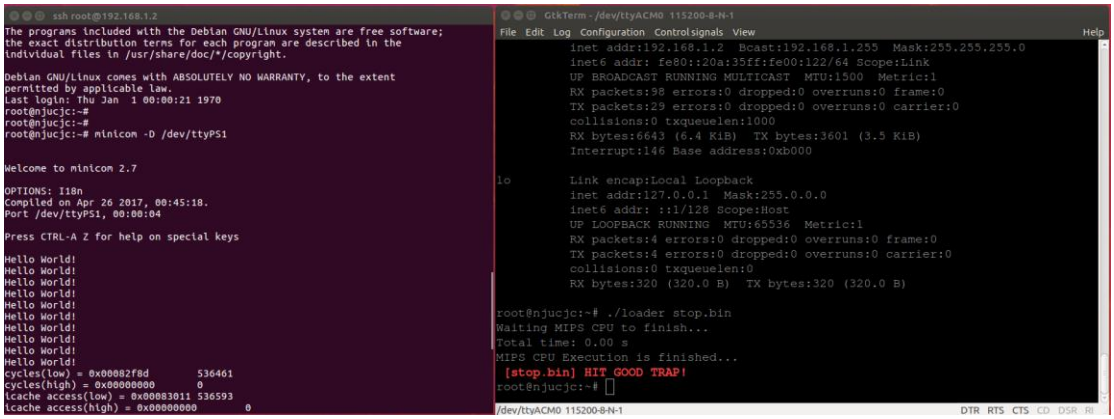
使用 ssh 命令远程登录到 shell 上，打开串口程序，准备从本地串口文件中接收信息。

```
Documents % ssh root@192.168.1.2
root@192.168.1.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jan  1 00:00:21 1970
root@njucjc:~#
root@njucjc:~#
root@njucjc:~# minicom -D /dev/ttyPS1
```

使用 ctrl A 之后按 z 修改配置，直到与最下面显示配置相同：



在 gtkterm 中使用由 arm_loader.c 本地编译（ARM+linux 编译）的可执行程序 loader 加载 NOOP CPU 上需要测试的 elf 程序，该 elf 可执行文件应该是在 host CPU 上进行交叉编译，并通过 scp 传输过来的文件。



如上图所示，ARM 加载了一个输出 hello world 的可执行程序到 DDR3 中，加载完毕让 NOOP 开始执行，左侧 ssh + minicom 输出了 NOOP CPU 执行的 printk（向串口写 ASCII 字符）打印的字符。

NOOP 执行完毕，ARM 在 gtkterm 中显示出红色的：[程序名] HIT GOOD TRAP!

若是产生了错误，则会输出：[程序名] HIT BAD TRAP!

到此上板步骤结束。

注意：有时想直接停止当前执行的程序，再另外加载其他程序。于是就在 gtkterm 中使用 Ctrl+C 使其停止工作，之后使用 loader 重新加载。这样做容易使得整个系统卡死（特别是 ctrl+c 后重新加载较大的程序），建议使用 ctrl + c 后使用 hello world 清理一下系统，之后再继续加载其他程序。

5、部分上板程序结果

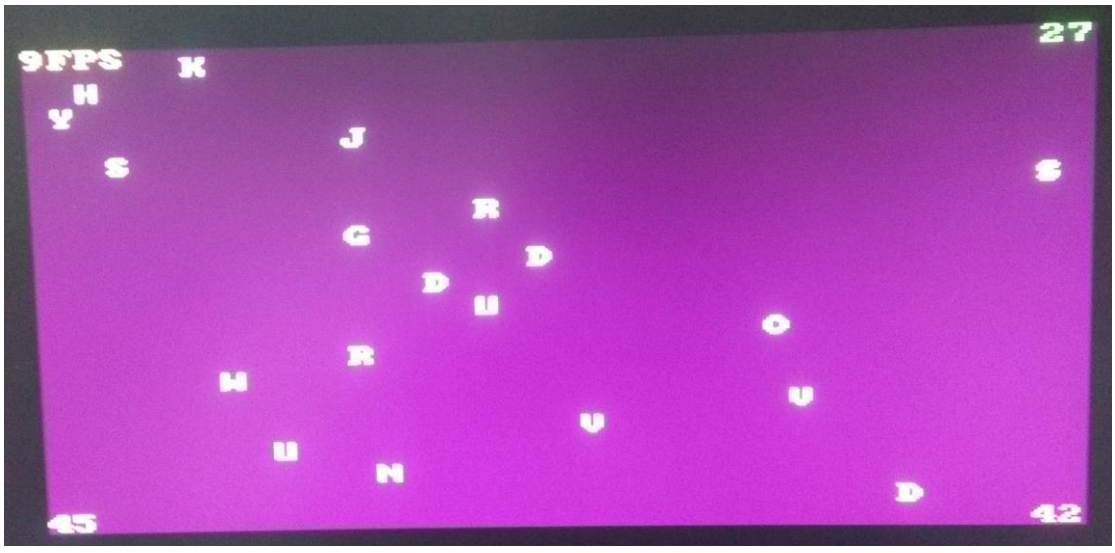
1、benchmark(microbench)

```
[qsort] Quick sort: * Passed.
  min time: 2287 ms [241]
[queen] Queen placement: * Passed.
  min time: 3826 ms [134]
[bf] Brainf**k interpreter: * Passed.
  min time: 13845 ms [189]
[fib] Fibonacci number: * Passed.
  min time: 63468 ms [45]
[sieve] Eratosthenes sieve: * Passed.
  min time: 29629 ms [143]
[15pz] A* 15-puzzle search: * Passed.
  min time: 6733 ms [86]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 7761 ms [174]
[lzip] Lzip compression: * Passed.
  min time: 12175 ms [217]
[ssort] Suffix sort: * Passed.
  min time: 3814 ms [155]
[md5] MD5 digest: * Passed.
  min time: 14004 ms [139]
==================================================
MicroBench PASS        152 Marks
                 vs. 100000 Marks (i7-6700 @ 3.40GHz)
Exited (0).
```

):

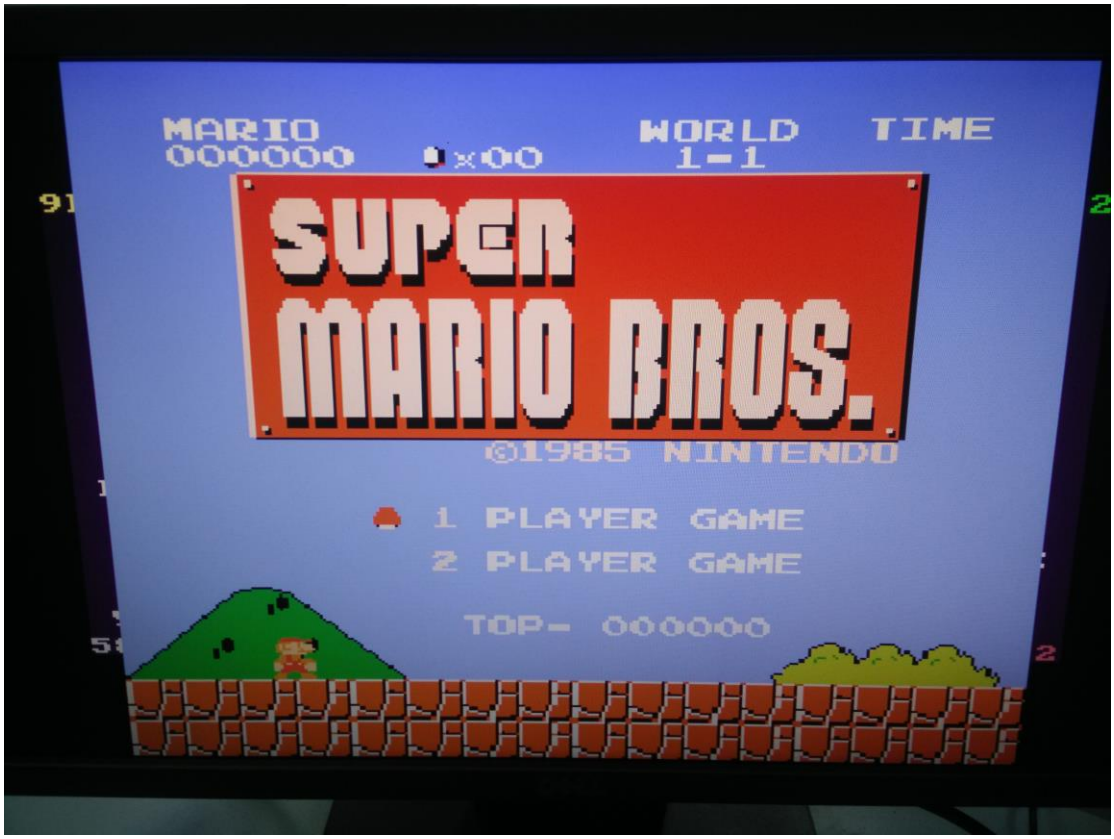
需要所有的测试均 pass，若有 failed 则说明 CPU 存在 bug。

2、打字小游戏（typing）：



能够正常在 ssh+minicom 中进行打字。

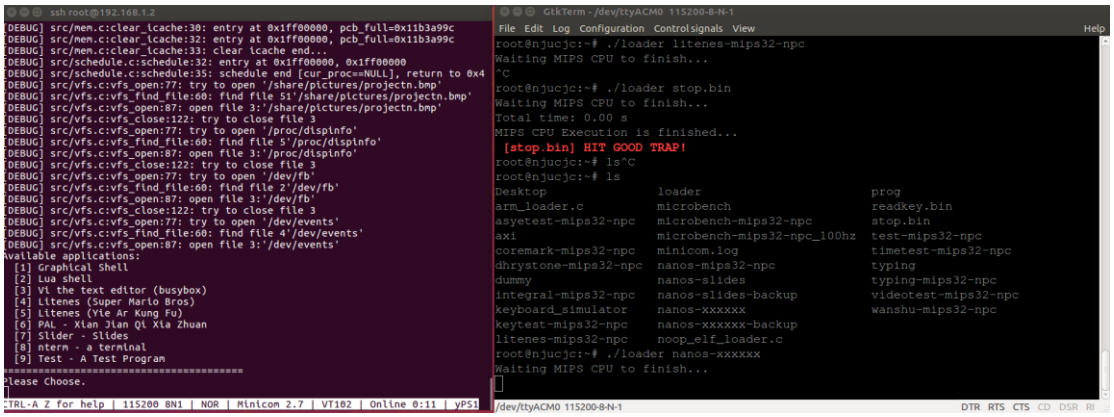
3、马里奥（litenes）：



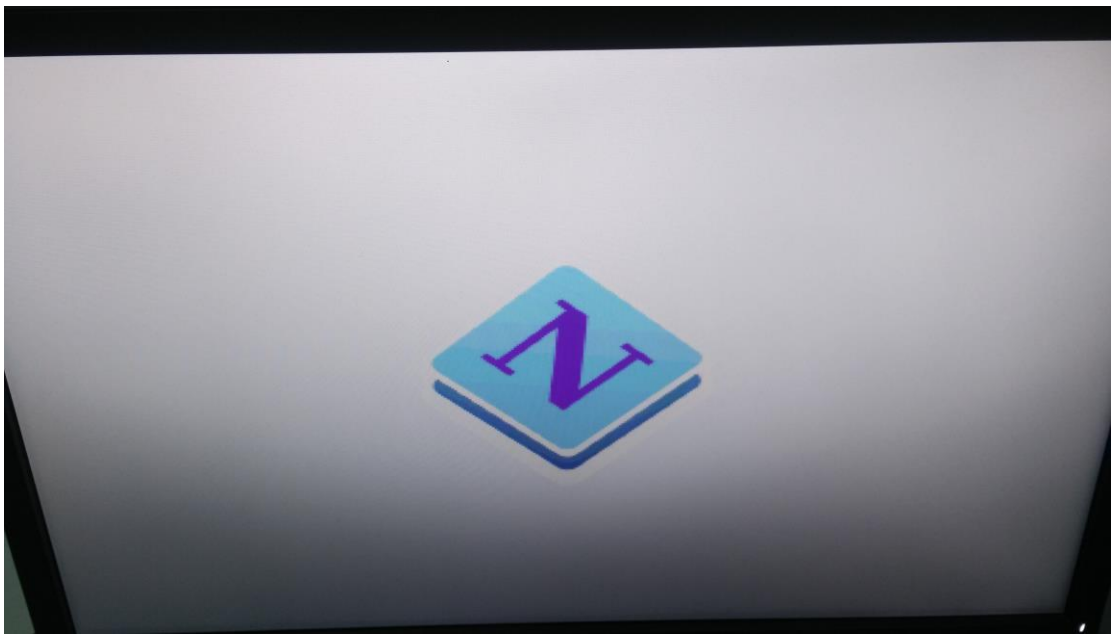
Y 进入游戏，wasd 上下左右。

此次实验最少需要达成以上三个目标。（同时通过 Nexus-am 的除 nanos-lite 之外的 app）

4、NANOS



VGA 显示如下：



选择 6 加载仙剑奇侠传：

回车进入新的故事：

第四个为 nanos-lite，需要在前面的基础上进一步完善 am 才能进一步执行。