

果壳 FPGA 平台  
陈煦豪

测试软件包括 Vivado 和 minicom。

Vivado 软件负责与目标机的 JTAG 接口交互，顺序完成三部分工作：

- 向目标机的 JTAG 接口传输 bitstream，完成 FPGA 上逻辑的烧写
- 将 linux.bin 通过 JTAG 接口传输给 DDR4 SODIMM 存储
- 通过 JTAG 接口配置 vio 软复位，解除 FPGA 上核(CPU)的复位状态

minicom 软件负责与目标机的 UART 接口交互，其功能为，在核执行目标代码时，输出串口打印的信息。

工程资源：[https://github.com/ssdfghhhhhh/NutShell\\_U250.git](https://github.com/ssdfghhhhhh/NutShell_U250.git)

环境：32 服务器

## 一：生成 Bitstream

1:需要：verilog 代码，位于/NutShell\_U250/build。在文件夹中替换 Topmain.v 后在 vivado 中刷新文件。

配置：修改 Makefile 文件

```
DATAWIDTH ?= 64
BOARD ?= pynq # sim pynq axu3cg
CORE ?= inorder # inorder ooo embedded
```

修改\src\main\scala\top\Settings.scala

```
object PynqSettings {
  def apply() = Map(
    "FPGAPlatform" -> true,
    "NrExtIntr" -> 3,
    "ResetVector" -> 0x80000000L,
    "MemMapBase" -> 0x00000000080000000L,
    "MemMapRegionBits" -> 28,
    "MMIOBase" -> 0x00000000040600000L,
    "MMIOSize" -> 0x0000000001000000L
  )
}
```

2:Generate Bitstream

3:最终需要两个文件：位于：

path/fpga/board/pynq/build/cxhU250\_pynq/cxhU250\_pynq.runs/impl\_1/

(1) System\_top\_wrapper.bit

(2) System\_top\_wrapper.ltx

4:使用 vivado 内 hardware manager 打开一块 U250 板卡，右键 xcu250 选择 program device，烧写上述两个文件。注：避免使用 21330409X021A 板卡，这张卡有用户在使用。

## 二：编译程序

1：目标机上运行的目标代码为\*.bin。编译\*.bin 需要使用果壳可用的 Riscv 工具链。具体参考 compile\_bbl.pdf

2：编辑 DDR4 装载的数据 data.txt

目标机上运行的目标代码为\*.bin，需要通过 JTAG 接口传输到 DDR4 SODIMM 中，在 FPGA 平台的逻辑设计中，包括了 jtag\_axi 的 IP，其连接 DDR4 MIG IP，主机根据写地址和写数据信息发送 jtag\_axi 的写事务请求，完成 DDR4 SODIMM 数据的装载。

具体的实现为：使用 bin2fpgadata，将\*.bin 转为\*.txt

./bin2fpgadata -i a.bin -o a.txt

▼ 使用方法 Plain Text 复制代码

```
1 usage: ./bin2fpgadata --in=string [options] ...
2 options:
3     -i, --in            input binary file (string)
4     -o, --out           output data text file (string [=data.txt])
5     -b, --burst         burst length (unsigned int [=1024])
6     --noeof             disable end of file char 0a
7     --tcl               generate tcl script
8     --offset            address offset (unsigned int [=0])
9     --hole-begin        address hole begin (unsigned int [=0])
10    --hole-end           address hole end (unsigned int [=0])
11    -?, --help           print this message
```

例子：./bin2fpgadata -i a.bin -o a.txt

图 1: bin2fpgadata 使用方法

注：bin2fpgadata 默认以 0 为基地址，需在 cpp 文件中修改“offset”为 80000000 后重新编译，与工程的内存映射相匹配。



图 2: data.txt 内容

三：写入 data，复位 CPU，观察串口

1：命令行 sudo minicom 打开串口观察工具，调整至 fpga 板卡对应接口 (21330409X021A 对应 ttyUSB2)，波特率 115200

```
Welcome to minicom 2.6.2

OPTIONS: I18n
Compiled on Jun 10 2014, 03:20:53.
Port /dev/ttyUSB2, 05:51:52

Press CTRL-A Z for help on special keys
```

```
CTRL-A Z for help | 115200 8S1 | NOR | Minicom 2.6.2 | VT102 | Offline
```

图 3：minicom 界面

2: 通过 jtag 向 DDR 写入 data.txt

jtagmv.tcl 功能为提取 data.txt 内地址数据信息，写入 DDR4 SODIMM。

在 vivado 的 tcl 窗口下输入：

```
source /path/jtagmv.tcl
```

注：完整的\*.bin 的数据写入 DDR4 后，会报告 ERROR: bad hexStr for Cse\_NumberUtil::hexStrBinStr，如图 4。如果需要再一次用 jtag 传输\*.bin 的内容到 DDR4，则需要先在 tcl 窗口输入一条命令 delete\_hw\_axi\_txn wr\_txn，这是因为上一条 ERROR 指示使得建立的 wr\_txn 事务未重置，需要手动删除该事务信息。

```
ERROR: [Xicom 50-38] xicom: Internal error: bad hexStr for Cse_NumberUtil::hexStr2BinStr
INFO: [Labtoolstcl 44-481] WRITE DATA is: 00000000
ERROR: [Common 17-39] 'run_hw_axi' failed due to earlier errors.

while executing
"run_hw_axi wr_txn"
("while" body line 7)
invoked from within
"while {[eof $fdata] != 1} {
  gets $fdata aline
  set AddrString [lindex $aline 0]
  gets $fdata dline
  set DataString [lindex $dline 0]
  ..."
(file "jtagmv.tcl" line 4)
```

图 4：Jtag 向 DDR 写入完成后的提示信息

3: 用 vio 设置 CPU 复位状态，解复位后 CPU 启动，串口输出信息。如图 5&6

```

file Edit View Search Terminal Help
[0.000000] Zone ranges:
[0.000000] DMA32 empty:
[0.000000] Normal [mem 0x000000008020000-0x0000000081ffffff]
[0.000000] Movable zone start for each node
[0.000000] Early memory node ranges
[0.000000] node 0: [mem 0x000000008020000-0x0000000081ffffff]
[0.000000] Initmem setup node 0 [mem 0x000000008020000-0x0000000081ffffff]
[0.000000] Cannot allocate SWIOTLB buffer
[0.000000] elf hwcap is 0x122a
[0.000000] Built 1 zonelists, mobility grouping on. Total pages: 7575
[0.000000] Kernel command line: root=/dev/mmcblk0 rootfstype=ext4 ro rootwait
[0.000000] Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
[0.000000] Inode-cache hash table entries: 2048 (order: 2, 16384 bytes)
[0.000000] Sorting _ex table...
[0.000000] Memory: 291408/307200k available (666K kernel code, 88K rwdata, 1)
[0.000000] SLUB: galloc=64, order=3, Win0jects=3, Win0jects=3, CPU=1, Node=1
[0.000000] NR_IRQS: 0 nr_irqs: 0, preallocated irqs: 0
[0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles
[0.000000] console [hvc0] enabled
[0.000000] console [hvc0] enabled
[0.000000] bootconsole [early0] disabled
[0.000000] bootconsole [early0] disabled
[0.000000] Calibrating delay loop (skipped), value calculated using timer f)
[0.000000] pid_max: default: 4096 minimum: 301
[0.000000] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[0.000000] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[0.010000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, ms
[0.010000] clocksource: Switched to clocksource riscv_clocksource
[0.010000] Unpacking initramfs...
[0.020000] workingset: timestamp bits=62 max_order=13 bucket_order=0
[0.020000] random: get_random bytes called from 0xfffffffff8001998a with crn0
[0.020000] Freeing unused kernel memory: 88K
[0.020000] This architecture does not have kernel memory protection.

Hello, RISC-V World!
[0.020000] init[1]: unhandled signal 4 code 0x1 at 0x0000000000100cc in he]
[0.020000] CPU: 0 PID: 1 Comm: init Not tainted 4.18.0-00046-g2ba394515c9-3
[0.020000] sepc: 000000000000100cc ra: 00000000000010c4 sp: 0000003fffdcbec0
[0.020000] gp: 00000000000013e0 tp: 0000000000000000 t0: 0000000000000004
[0.030000] t1: 0000000000000000 t2: 0000000000000001 s0: 0000000000000000

```

图 5、6：启动 linux 系统