

如何运行 sodor 的 emulator

本文是关于 ucb 的 The Sodor Processor 的 emulator 的说明, 旨在能让读者成功运行 sodor.
写于 2021 年 6 月 20 日星期日. 有问题可以发送邮件到 1263810658@qq.com

1. 克隆仓库

ucb 的仓库 <https://github.com/ucb-bar/riscv-sodor> 融入 chipyard 中了, 因此没有 emulator, 只有 verilator.

我们需要

```
$git clone https://github.com/passlab/riscv-sodor.git --recursive
```

```
$ cd riscv-sodor
```

```
$ git submodule update --init --recursive
```

```
$ cd riscv-fesvr
```

```
$ ./configure --prefix=/usr/local
```

这一步是为了产生 makefile 文件.

```
$ make
```

这里可能有的错误:

Error during sbt execution: Error retrieving required libraries

(see /home/hadoop/.sbt/boot/update.log for complete log)

Error: Could not retrieve sbt 1.3.8

2. 下载 sbt

首先查看自己的 sbt 环境. `sbt sbt-version`

可能会需要几分钟

getting org.scala-sbt sbt 0.13.0 (this may take some time)...他会尝试下载,但是由于 sbt 在国外所以大概率下载失败.

最后显示[error] [launcher] error during sbt launcher: error retrieving required libraries.

我们需要手动安装 sbt. 网上许多手动安装 sbt 的教程我尝试后都是行不通的. 下面这个我可以安装成功.

第一步, 下载 sbt 压缩包

打开 <http://www.scala-sbt.org/download.html>, 下载最后的 0.13.18 版本比较保险. 因为接近需要的 0.13.0 环境. 下载下来后传到虚拟机里可以用 vmware tool 拖动, 服务器可以用 lrzsz 或者 scp. 这里不一一赘述.

第二步, 建立目录, 解压文件到所建立目录

```
sudo tar zxvf sbt-0.13.5.tgz -C /opt/scala/
```

第三步, 建立启动 sbt 的脚本文件

/*选定一个位置, 建立启动 sbt 的脚本文本文件, 如/opt/scala/sbt/ 目录下新建文件名为 sbt 的文本文件*/

```
$ cd /opt/scala/sbt/
$ vim sbt
/*在 sbt 文本文件中添加
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -
XX:MaxPermSize=256M"
java $SBT_OPTS -jar /opt/scala/sbt/bin/sbt-launch.jar "$@"
然后按 esc 键 输入 :wq 保存退出，注意红色字体中的路径是定位到解压的 sbt 文件包
中的 sbt-launch.jar 文件的绝对路径*/
/*修改 sbt 文件权限*/
$ chmod u+x sbt
```

第四步 配置 PATH 环境变量，保证在控制台中可以使用 sbt 命令

```
$ vim ~/.bashrc
/*在文件尾部添加如下代码后，保存退出*/
export PATH=/opt/scala/sbt/:$PATH

/*使配置文件立刻生效*/
$ source ~/.bashrc
```

第五步，设置源

```
vim ~/.sbt/repositories
[repositories]
local
huaweicloud-maven: https://repo.huaweicloud.com/repository/maven/
maven-central: https://repo1.maven.org/maven2/
huaweicloud-ivy: https://repo.huaweicloud.com/repository/ivy/,
[organization]/[module]/(scala_[scalaVersion]/)(sbt_[sbtVersion]/)[revi
sion]/[type]s/[artifact](-[classifier]).[ext]
```

第六步，测试 sbt 是否安装成功

```
/*第一次执行时，会下载一些文件包，然后才能正常使用，要确保联网了，安装成功后显
示如下*/
$ sbt sbt-version
[info] Set current project to sbt (in build file:/opt/scala/sbt/)
[info] 0.13.5
```

3. 安装 jdk

安装好 sbt 后再次 make。可能依旧 make 失败了

要修改 build.properties，让他和我们安装的版本号一致。

```
build.properties U X
project > build.properties
1 sbt.version=0.13.18
2
```

error: scala.reflect.internal.MissingRequirementError: object java.lang.Object in compiler mirror not found.

因为需要 jdk8

Jdk8 可能需要 oracle 账号, 可以从镜像源下载, 然后传到虚拟机或服务器

<https://repo.huaweicloud.com/java/jdk/>

安装 jdk 这里不再赘述.

安装好后使用 alternatives 命令进行 jdk 版本的切换 1500 表示优先级

alternatives --install /usr/bin/java java /usr/local/java/jdk1.8.0_162/bin/java 1500

alternatives --config java (会显示多个版本的 jdk 让你选择, 直接根据数字进行切换)

java -version 如果显示 1.8 那就就可以了 java1.8 就是 java8

到这里应该 make 就可以成功了.

可以尝试

\$ make run-emulator

或者一个个 run

cd emulator/rv32_1stage 然后 make run .

make report 可以看到

```
t/*.out
ne.riscv.out:#----- Tracer Data -----
ne.riscv.out:#
ne.riscv.out:#      CPI    : 1.00
ne.riscv.out:#      IPC    : 1.00
ne.riscv.out:#      cycles: 219798
ne.riscv.out:#
ne.riscv.out:#      Bubbles    : 0.000 %
ne.riscv.out:#      Nop instr   : 0.000 %
ne.riscv.out:#      Arith instr : 42.814 %
ne.riscv.out:#      Ld/St instr : 33.050 %
ne.riscv.out:#      branch instr: 15.032 %
ne.riscv.out:#      misc instr  : 9.103 %
ne.riscv.out:#-----
riscv.out:#----- Tracer Data -----
riscv.out:#
riscv.out:#      CPI    : 1.00
riscv.out:#      IPC    : 1.00
```

vim output/vvadd.riscv.out 可以看每个周期的指令在干嘛.

```
1 Loaded memory.
2 Instantiated HTIF.
3 Cyc=      0 Op1=[0x00000000] Op2=[0x00000000] W[W, 1= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x00000200 DASM(00000093)
4 Cyc=      1 Op1=[0x00000000] Op2=[0x00000000] W[W, 2= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x00000204 DASM(00000113)
5 Cyc=      2 Op1=[0x00000000] Op2=[0x00000000] W[W, 3= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x00000208 DASM(00000193)
6 Cyc=      3 Op1=[0x00000000] Op2=[0x00000000] W[W, 4= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x0000020c DASM(00000213)
7 Cyc=      4 Op1=[0x00000000] Op2=[0x00000000] W[W, 5= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x00000210 DASM(00000293)
8 Cyc=      5 Op1=[0x00000000] Op2=[0x00000000] W[W, 6= 0x00000000] Mem[_ 0: 0x464c457f
  ] PC= 0x00000214 DASM(00000313)
9 Cyc=      6 Op1=[0x00000000] Op2=[0x00000000] W[W, 7= 0x00000000] Mem[_ 0: 0x464c4000
  ] PC= 0x00000218 DASM(00000313)
NORMAL | master output/vvadd.riscv.out utf-8[unix] 0% 4: 1
```

grep \# output/*.riscv.out 选出重点.

下面附一个 emulator 的原理

make run 做了什么:

运行 sbt, Scala 构建工具, 选择 rv32 1stage 项目, 并运行生成处理器的 Verilog RTL 描述的 Chisel 代码。生成的 Verilog 代码可以在 emulator/rv32_1stage/generated-src/ 中找到。

I 运行 verilator, 一种将 Verilog 编译为周期精确的 C++ 仿真代码的开源工具

I 将生成的 C++ 代码编译为称为模拟器的二进制文件。

I 运行仿真器二进制文件, 将提供的 RISC-V 二进制文件加载到模拟内存中。所有 RISC-V 测试和基准测试都将在调用 \make run 时执行。

如下图

