# オープンソースCPU「RISC-V」の研究

## 第12回 Rocket Chipがプログラムをロードして RTLシミュレーションを実行する仕組み

@msyksphinz

Rocket Chipをシミュレーションするとき，特に何も考えずにmakeを実行すると，シミュレーションが始まります．

```
make CONFIG=DefaultConfig output/rv64ui-p-add.out↵
```

しかし，具体的にどのようにして動作しているのかはこれだけではよく分かりません．これを理解するためには，Rocket Chipがどのようにコンパイルされ，実行されているのかについて知る必要があります．これは，ひいては，Rocket Chipの構造を理解し，どのようにプログラムの実行を制御しているのかを理解するのに役立ちます．

まず，Rocket Chipのプログラム実行方法を理解するためには，RISC-V FrontEnd Server（fesvr）について理解する必要があります．

## RISC-V Rocket Chipが動作するまでの仕組み

Rocket Chipの環境としては，シノプシスのVCSおよびフリーのRTLシミュレータであるVerilatorの両方で実行できる環境が用意されています．Verilatorシミュレータで実行する環境を見てみると，Makefileが置いてあるので中身を観察してみます．以下のように実行すると，幾つかのMakefileが生成されます．

```
cd emulator↵
make CONFIG=DefaultConfig↵
```

以下のMakefileが生成されました．

```
Makefile
Makefrag-verilator
```

Makefragを参照すると，**リスト1**に示すようなファイルがコンパイルに必要なことが分かります．jtagはとりあえず無視しますが，SimDTM.v，SimDTM.ccというのはRocketChipの全体制御を管理しているファイルになります．

また，Verilatorのコンパイル・オプションに，RISC-Vのライブラリが指定されています．これがRISC-Vのフロントエンド・サーバfesvrになります．

**リスト1　Rocket Chipのemulatorディレクトリでmakeを実行すると生成されるMakefileとMakefrag-verilatorの中身**
これらのファイルがRocket Chipの実行に必要

Makefileの中身

```
CXXSRCS := emulator SimDTM
```

Makefrag-verilatorの中身

```
verilog = ¥
  $(generated_dir)/$(long_name).v ¥
  $(generated_dir)/$(long_name).behav_srams.v ¥
⋮
cppfiles = $(addprefix $(base_dir)/csrc/, $(addsuffix .cc, $(CXXSRCS)))   ← SimDTM.cc
headers = $(wildcard $(base_dir)/csrc/*.h)

model_header = $(generated_dir)/$(long_name)/V$(MODEL).h
model_header_debug = $(generated_dir_debug)/$(long_name)/V$(MODEL).h

$(emu): $(verilog) $(cppfiles) $(headers) $(INSTALLED_VERILATOR)
        mkdir -p $(generated_dir)/$(long_name)
        $(VERILATOR) $(VERILATOR_FLAGS) -Mdir $(generated_dir)/$(long_name) ¥
        -o $(abspath $(sim_dir))/$@ $(verilog) $(cppfiles) -LDFLAGS "$(LDFLAGS)" ¥
        -CFLAGS "-I$(generated_dir) -include $(model_header)"
        $(MAKE) VM_PARALLEL_BUILDS=1 -C $(generated_dir)/$(long_name) -f V$(MODEL).mk
```

第1回　登場の背景（2018年2月号）
第2回　アーキテクチャの基礎知識（2018年3月号）
第3回　ハード＆ソフト開発環境あれこれ（2018年4月号）