自作 RISC-V CPUのデバッグ手法

kanataso@kanapipopipo
第3回 自作CPUを語る会
2024/06/02

自作RISC-V CPU、どうやってデバッグしてますか

大前提

assertを書く

テストを書く

基本：riscv-tests

RISC-VのCPU向けのテストセット

命令1つ1つに対してのテストが揃っている

・物理アドレス(p), 仮想アドレス(v)向けのテストがある

iverilog, verilator

OSSのVerilogのシミュレータ

verilatorは高速、でも制限あり

どちらも変なバグがある(気がする)

GitHub Actionsでコミットごとにテストを実行

riscv-tests

.tohostに結果が書き込まれる

・1なら成功

・gpレジスタにも書き込まれている

↓

メモリステージで.tohostに書き込まれるのを確認したら、書き込み内容を見て$finish

```
1   OUTPUT_ARCH( "riscv" )
2   ENTRY(_start)
3
4   SECTIONS
5   {
6     . = 0x80000000;
7     .text.init : { *(.text.init) }
8     . = ALIGN(0x1000);
9     .tohost : { *(.tohost) }
10    . = ALIGN(0x1000);
11    .text : { *(.text) }
12    . = ALIGN(0x1000);
13    .data : { *(.data) }
14    .bss : { *(.bss) }
15    _end = .;
16  }
```

riscv-tests

ページング込みのテスト(-v-)が終わらない

・.tohostが仮想アドレスになっている

ページングユニットを越した先で判定するべきだった

riscv-testsが落ちたらどうする？

vcdを見る？

→ GTKWave

波形を見てもいいが、ちょっと面倒

→簡単なログを$displayで出力すると楽

$displayの問題点

always @(posedge clk) begin

　　$display(“〜”);

end

が複数あると、displayされる順番がクロック毎に入れ替わったりする

&

$displayが増えると、目的のログを探すのが大変！

・波形を見るのとそこまで変わらない気がしてくる

解決法：$displayの形式を正規化して出力、処理

正規化したログを出力, 保存 → pythonで処理 → 見やすいログ！

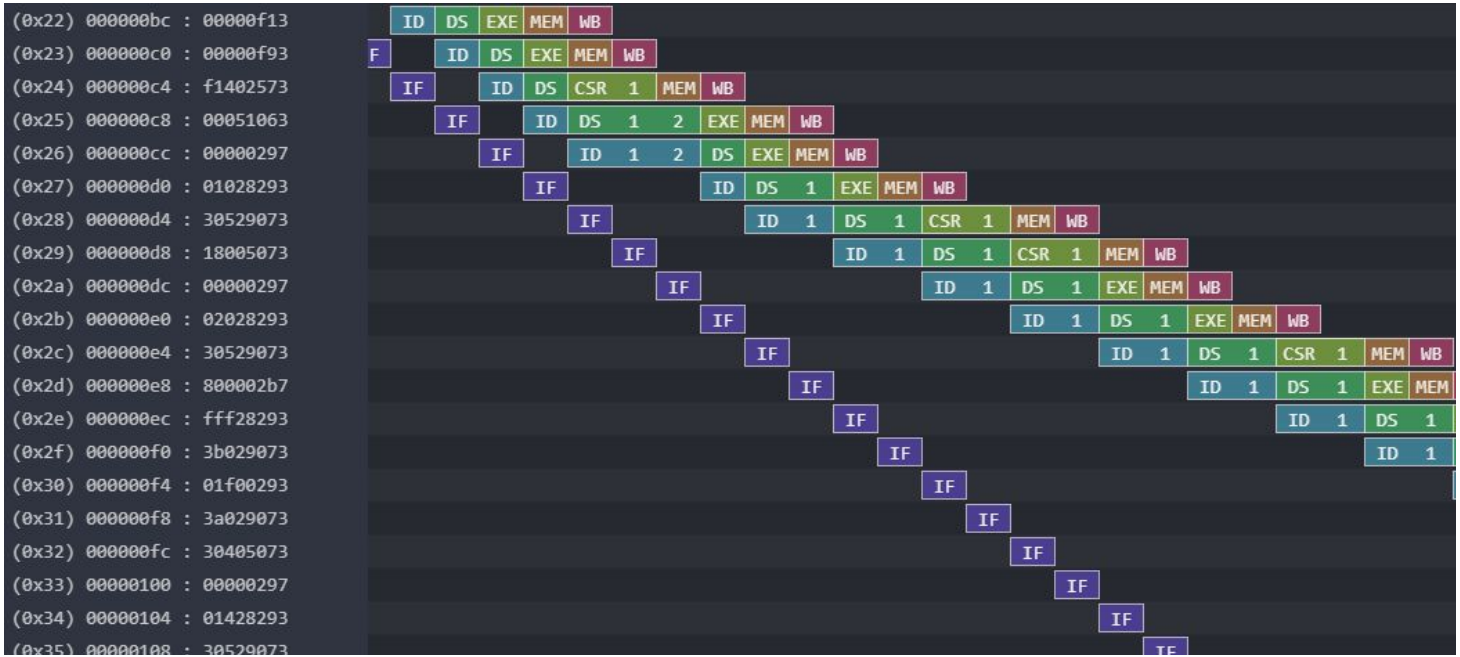
```
60 `ifdef PRINT_DEBUGINFO
61 always @(posedge clk) if (util::logEnabled()) begin
62     $display("data,wbstage.valid,b,%b", valid);
63     $display("data,wbstage.inst_id,h,%b", valid ? info.id.id : iid::X);
64     if (valid) begin
65         $display("data,wbstage.pc,h,%b", info.pc);
66         $display("data,wbstage.inst,h,%b", info.inst);
67         $display("data,wbstage.wb_addr,d,%b", reg_addr);
68         $display("data,wbstage.wb_data,h,%b", wdata);
69         $display("data,wbstage.inst_count,d,%b", inst_count);
70     end
71 end
```

見やすいログができた

```
344 # 26 clock ------------------------
345 IF Stage --------------------
346 pc                      : 0x00000054
347 next_pc                 : 0x00000000
348 requested_pc            : 0x00000050
349 requesting_pc           : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
350 ireq.valid              : 0
351 iresp.valid             : 0
352 ID Stage --------------------
353 valid                   : 0
354 inst_id                 : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
355 DS Stage --------------------
356 valid                   : 0
357 inst_id                 : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
358 EXE Stage --------------------
359 ll_last_inst_id         : 0x1
360 valid                   : 1
361 inst_id                 : 0x0000000000000000
362 pc                      : 0x00000000
363 inst                    : 0x0500006f
364 i_exe                   : 1
365 br_exe                  : 0
366 op1_data                : 0x00000000
367 op2_data                : 0x00000050
368 calc_stall              : 0
369 ismulticyc              : 0
370 jmp_flg                 : 1
371 branch_taken            : 1
372 branch_target           : 0x00000050
373 CSR Stage --------------------
```

Konataで見れるようにもできる

命令フェッチで命令にidを振っておくとKanata log formatに変換できる。



riscv-testsが落ちたら....

まず、Konataでどこでおかしくなったか確認

次に、おかしい部分をテキストログで確認

それでも分からなければ、vcdを確認

vcdだけよりもよくなった？

まだつらい

数十万~数千万命令実行後に何かがおかしくなっているかもしれない場合、

・ログを出力するだけで遅くなる

・ログの容量がとても大きくなる

・そもそもpcの遷移が正しいのかを確認するのがつらい

・どこかでストールしてしまうが、いつストールし始めたか分からん

→デバッグ速度が非常に遅くなる

ログを出力するだけで遅くなる

n命令目, nクロック後からログを出力するオプションを作成すればOK

・ログを表示するかどうかの変数をpackageに用意する

```
60 `ifdef PRINT_DEBUGINFO
61 always @(posedge clk) if (util::logEnabled()) begin
62     $display("data,wbstage.valid,b,%b", valid);
63     $display("data,wbstage.inst_id,h,%b", valid ? info.id.id : iid::X);
64     if (valid) begin
```

$dumpon, $dumpoff

pcの遷移が正しいのかを確認する

spikeと比べればOK

spike :　riscv-software-src/riscv-isa-sim

　　RISC-V公式のシミュレータ

実行したPCを出力する機能がある(トレース)

→これと比べられるログを生成して、比べる

不審なストールを検知する

nクロックの間、ずっと同じ命令を処理し続けているときに、

finishしてデバッグ用の情報を表示するオプションを作成

WFI (Wait For Interrupt)に注意

長いストールを表示するとKonataが重くなる

riscv-testsが落ちた, 変な挙動になったら....

おかしくなった場所がわからない場合、

・spikeと比較

・ストール検知

でおかしくなる場所を探る。

riscv-testsが落ちた, 変な挙動になったら....

おかしくなった場所がわかったので、

まず、Konataでどこでおかしくなったか確認

次に、おかしい部分をテキストログで確認

それでも分からなければ、vcdを確認

まだつらい！

割り込み系のバグ

・ISSと割り込みのタイミングが変わるので、比較ができなくなる

メモリのバグ

・たいへん

・メモリ操作を監視すると良いが、キャッシュにバグがあったらどうする

ChiselでSVをテスト

BlackBoxを使う

ただし、ポートにinterfaceとかstructが使えない