Getting to Work with OpenPiton

Princeton University

http://openpiton.org
Simulating OpenPiton RTL
Anatomy of a Simulation

• Simulation model
  – Design under test (DUT) RTL
  – Top-level test bench
  – Simulator compiler arguments
    • Verilog macros, include directories, monitor params, etc.

• Test stimuli
  – Assembly tests
  – C tests
  – Source/sink bit vectors
    • Based on infrastructure from Christopher Batten’s group at Cornell
# OpenPiton Simulation Models

## Table 3: OpenPiton Simulation Models

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>manycore</td>
<td>C/Assembly</td>
</tr>
<tr>
<td>chip_fpga_bridge</td>
<td>Unit Test</td>
</tr>
<tr>
<td>dmbr</td>
<td>Unit Test</td>
</tr>
<tr>
<td>dmbr_test</td>
<td>Unit Test</td>
</tr>
<tr>
<td>fpga_chip_bridge</td>
<td>Unit Test</td>
</tr>
<tr>
<td>fpga_fpga_hpc_bridge</td>
<td>Unit Test</td>
</tr>
<tr>
<td>fpga_fpga_lpc_bridge</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_counter</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_fsm</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_htsm</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_lfsr</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_rtsm</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_shiftreg</td>
<td>Unit Test</td>
</tr>
<tr>
<td>ifu_esl_stsm</td>
<td>Unit Test</td>
</tr>
<tr>
<td>jtag_testbench</td>
<td>Unit Test</td>
</tr>
</tbody>
</table>
Simulation Scripts/Tools

• sims
  – piton/tools/src/sims/sims,1.262
  – Adapted from OpenSPARC T1
  – Build and launch individual simulations
  – Regressions
    • Single simulation model
  – Supports Synopsys VCS, Cadence Incisive, and Icarus Verilog
    • Synopsys VCS recommended

• contint
  – piton/tools/src/contint/contint,1.0
  – Calls sims
  – Continuous integration bundles
    • Multiple simulation models
  – Currently only supports SLURM job scheduler and Synopsys VCS
Simulator Choice

• Synopsys VCS, Cadence Incisive and Icarus Verilog are all supported

• Commands shown are for VCS

• For Icarus Verilog, replace `vcs` with `icv`

• For Cadence Incisive, replace `vcs` with `ncv`
Building a Simulation Model

• Required `sims` arguments
  - `sys=<simulation model>`
  - `vcs_build`

• Other useful arguments
  - `vcs_build_args=<VCS arguments>`
  - `debug_all`
Simulation Model Build Outputs

• stdout and sims.log

• build/<simulation_model>/<build_id>/
  -build_id=<name>
  • Default is rel-0.1
Example: The manycore Model

- `sims -sys=manycore -vcs_build`
  - `sims.log`: check for build errors
  - Check for SIGDIE

- `build/manycore/rel-0.1/`
Hands-on: The manycore Model

- cd $PITON_ROOT/build
- sims -sys=manycore -icv_build
  - sims.log: check for build errors
  - build/manycore/rel-0.1/
Running a Simulation

• Required `sims` arguments
  - `sys=<simulation model>`
  - `vcs_run`
  `<test stimuli>`
    • Varies by simulation model type (assembly file, source/sink prefix)

• Other useful arguments
  - `build_id=<name>`
  - `gui`
    • Requires `-debug_all` during build
Simulation Outputs

• Depends on test type
• stdout and sims.log
  – PASS (HIT GOOD TRAP)
• Test binary (diag.exe)
• Memory image (mem.image)
• Assembler log (midas.log)
• Symbol table (symbol.tbl)
• Performance log (perf.log)
• Status log (status.log)
Example: Assembly Test Simulation

- sims -sys=manycore -vcs_run princeton-test-test-test.s
  - C tests have similar syntax
Hands-on: Assembly Test Simulation

• `sims -sys=manycore -icv_run princeton-test-test-test.s`
  - Should take 2-5mins
Example output
Debugging Simulations

• Monitors (\textit{manycore})
  – Non-synthesizeable Verilog modules
  – Instantiated in top-level test bench
  – X-module references DUT signals
    • Print useful output
    • Check properties

• Tools for parsing simulation output
  – \texttt{pc\_grep <log>}, \texttt{reg\_grep <log>}, etc.
Debugging Simulations
Debugging Simulations

• Waveforms - DVE
  – Build with \texttt{-debug\_all}
  – Run with \texttt{-gui}

• Example (again):
  – \texttt{sims -sys=manycore -vcs\_build -debug\_all}
  – \texttt{sims -sys=manycore -vcs\_run princeton-test-test-test.s -gui}
Debugging Simulations
Debugging Simulations
Exploring the assembly test suite

- `piton/verif/diag/assembly/`

- **Diaglists** (`piton/verif/diag/*_.diaglist`)
  - Groups of assembly tests and assembly test declarations
  - Assembly test declaration
    ```
    label testfile.s <sims arguments>
    ```
  - Assembly group definitions
    ```
    <groupname sys=mymodel sims args>
    test1 test1.s
    </groupname>
    ```
  - Groups can be nested
Diaglists

<tile1 sys=manycore -x_tiles=1 -y_tiles=1>
<cmp_default name=default>

<all_tile1_passing>
<all_tile1_passing_no rtl_csm>

<tile1_mini>
<preinceton-test>
  princeton-test-test princeton-test-test.s -finish_mask=3 -midas_args=-DTHREAD_COUNT=2 -midas_args=-DTHREAD
  STRIDE=1
  basic-io-test basic-io-test.s
  uart-hello-world uart-hello-world.s
</preinceton-test>

<tile1 mini icache>
  bypass win bypass win.s
  /// fail_perf_chase_l1hit chase_pal -midas_args=-pal_diag_args=-lenbytes=512 -midas_args=-pal_diag_args=-
  stride=16 -midas_args=-pal_diag_args=-nodisablel1 warmup -midas_args=-pal_diag_args=-expect_time=3 -midas_args=-pal
  diag_args=-expect_string=L1 ld hit
  dmiss_imiss Dmiss_imiss.s
  done_retry_trap done_retry_trap.s
  exu_alu exu_alu.s
  hp_reg_rdwr hp_reg_rdwr.s
  ihit_sameset Ihit_sameset.s
  imiss_branches Imiss_branches.s
  imiss_oddeven Imiss_oddeven.s
  intr_basic2 intr_basic2.s
Common Test Flags

- `–rtl_timeout=`
  - Number of cycles sims will wait before timing out the test

- `–sim_run_args=`
  - Arguments (e.g. `plusargs`) to Verilog simulator

- `–midas_args=`
  - Arguments to assembler, `midas`
  - Thread count, thread stride, and more

- `–finish_mask=`
  - Mask specifying threads to wait for
Types of tests

• Thousands of assembly tests
  – IFU, TLU, etc
  – arch
    • fp, exu, mem, trap, etc.
  – TSO tests
  – PAL-generated (randomized) tests
  – C tests
Running a Regression

• Groups of tests as defined in diaglists

• Tests utilize the same simulation model
  – One build, multiple test runs

• sims -sim_type=vcs -group=<regression name>
  – Simulation model specified by group declaration
  – -sim_type=vcs replaces -vcs_build and -vcs_run
Running a Regression

• Example:

```
sims -sim_type=vcs -group=tile1_mini
```
Regression Outputs

• Simulation model will be built as usual in
  \texttt{build/\langle simulation\_model \rangle/\langle build\_id \rangle/}

• Tests run sequentially
  – Test results stored in \texttt{build/\langle date \rangle/\langle id \rangle}

• Check results
  – \texttt{regreport \langle test results directory \rangle}
Regression Outputs

Summary for /tank/mmckewon/research/projects/piton/openpiton/build/2016_06_11_0

<table>
<thead>
<tr>
<th>Status</th>
<th>tile1ミニ</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS:</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>FAIL:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Diag Problem:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>License Problem:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MaxCycles Hit:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Socket Problem:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Timeout:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LessThreads:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simics Problem:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Performance:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Killed By Job Q:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unknown:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UnFinished:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>flexm error:</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Diag Count: 46 | 46

Cycles/Sec: 18358353 | 18358353
K Cycles: 2018733 | 2018733
#Diags Used: 46 | 46

Details for tile1ミニ

PASS:

Diag: basic-l0-test::default::tile1ミニ::job64993903 PASS
Sat Jun 11 02:27:09 EDT 2016
sim.log: 26118590: Simulation -> PASS (HIT GOOD TRAP)
Cyc= 26118590, Sec= 1.910, C/S=13674607.3
sims: group_name = tile1ミニ
sims: regress_date = 2016 06 11
sims: regress_time = 02_26 15
sims: /tank/mmckewon/research/projects/piton/openpiton/build/manycore/tile1_2016_06_11_0/simv +cpu_num=0 +downwarn
finish +docerrorfinish +spc pipe=0 +vcs+dumpwavossf +TIMEOUT=50000 +wait cycle to kill=10 +tg seed=0 +good trap=00
00000200:1000122000 +bad trap=0000082920:1000122020 +efuse data file=efuse.img +asm_diag_name=basic-l0-test.s +efuse
image name=default.dat +dv_root=/tank/mmckewon/research/projects/piton/openpiton/piton

Diag: bypass_win::default::tile1ミニ::job57238050 PASS
Sat Jun 11 02:27:43 EDT 2016
sim.log: 26643580: Simulation -> PASS (HIT GOOD TRAP)
Cyc= 26643580, Sec= 2.150, C/S=12392325.6
sims: group_name = tile1ミニ
sims: regress_date = 2016 06 11
Continuous Integration Bundles

• Infrastructure for large scale continuous integration testing

• Supports multiple different simulation models

• Specified by XML files
Continuous Integration Bundles

<bundles>

<bundle_name>

  <asm_test name="asm_test_name">
    <sys>sim_model</sys>
    <asm_diag_name>test.s</asm_diag_name>
  </asm_test>

  <asm_regress name="regress_name">
    <sys>sim_model</sys>
    <group>regression name</group>
  </asm_regress>

  <include>sub-bundle name</include>

  .
  .
  .

</bundle_name>

</bundles>
Continuous Integration Bundles

Table 5: OpenPiton Continuous Integration Bundles

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_push</td>
<td>a compact set of tests designed to run for every git commit</td>
</tr>
<tr>
<td>git_push_lite</td>
<td>a light version of git_push with fewer tests</td>
</tr>
<tr>
<td>nightly</td>
<td>a complete set of tests desired to run every night</td>
</tr>
<tr>
<td>pal_tests</td>
<td>a set of PAL tests</td>
</tr>
<tr>
<td>all_tile1_passing</td>
<td>All single tile tests</td>
</tr>
<tr>
<td>tile1_mini</td>
<td>a mini set of single tile tests</td>
</tr>
<tr>
<td>all_tile2_passing</td>
<td>All 2-tile tests</td>
</tr>
<tr>
<td>tile2_mini</td>
<td>a mini set of 2-tile tests</td>
</tr>
<tr>
<td>tile4</td>
<td>All 4-tile tests</td>
</tr>
<tr>
<td>tile8</td>
<td>All 8-tile tests</td>
</tr>
<tr>
<td>tile16</td>
<td>All 16-tile tests</td>
</tr>
<tr>
<td>tile36</td>
<td>All 36-tile tests</td>
</tr>
<tr>
<td>tile64</td>
<td>All 64-tile tests</td>
</tr>
</tbody>
</table>
Running a `contint` Bundle

- `contint` — continuous integration tool
  - Currently requires SLURM job scheduler

- `contint --bundle=<bundle name>`

- Example:
  - `contint --bundle=git_push`
contint Bundle Outputs

• All simulation models will be built and simulations submitted to scheduler

• Results will be aggregated and printed to stdout

• Individual simulation results located in
  – build/contint_<bundle_name>_<date>_<id>

• Re-process results
  – contint --bundle=<bundle_name> --check_results --contint_dir=<results directory>

• Example:
  – contint --bundle=git_push --check_results --contint_dir=${PWD}/contint_git_push_2016_6_19_0
**contint** Bundle Outputs

Contint: Checking results for bundle item 'nightly_dmbr_source_sink_18'
Contint: Total diags: 1

Contint: Checking results for bundle item 'nightly_dmbr_source_sink_19'
Contint: Total diags: 1

Contint: Checking results for bundle item 'nightly_dmbr_source_sink_20'
Contint: Total diags: 1

Summary for builds and runs

<table>
<thead>
<tr>
<th>Status</th>
<th>ASM REGRESS</th>
<th>ASM TEST</th>
<th>OTHER</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both PASS</td>
<td>14</td>
<td>52</td>
<td>39</td>
<td>105</td>
</tr>
<tr>
<td>Build FAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Run FAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Item Count</td>
<td>14</td>
<td>52</td>
<td>39</td>
<td>105</td>
</tr>
</tbody>
</table>

Summary for /tank/mmckeown/research/projects/piton/openpiton/build/contint_nightly_2016_06_10_0/ by bundle item types

<table>
<thead>
<tr>
<th>Status</th>
<th>ASM REGRESS</th>
<th>ASM TEST</th>
<th>OTHER</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS</td>
<td>2197</td>
<td>51</td>
<td>39</td>
<td>2287</td>
</tr>
<tr>
<td>FAIL</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Diag Problem</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>License Problem</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MaxCycles Hit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Socket Problem</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Timeout</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LessThreads</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sinics Problem</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Performance</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Killed By Job Q</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unknown</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UnFinished</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>flexlm error</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Diag Count</td>
<td>2288</td>
<td>52</td>
<td>39</td>
<td>2299</td>
</tr>
</tbody>
</table>

| Cycles/Sec      | 748682      | 2976763  | 140951 | 773704|
| K Cycles        | 762291179   | 4612272  | 33146  | 888445998|
| #Diags Used     | 2280        | 51       | 39     | 2290 |

Summary for /tank/mmckeown/research/projects/piton/openpiton/build/contint_nightly_2016_06_10_0/ assembly regressions only
contint Bundle Outputs

Details for nightly_tile4_dmbr_tile4_tile4
-------------------------------------------------------------------------------------
Details for tile4 group
-------------------------------------------------------------------------------------
FAIL:
----------------
Diag: br_stress:default:tile4:0:job149357       FAIL (HIT BAD TRAP)
Fri Jun 10 11:58:23 EDT 2016
sim.log: 253779500 : Simulation -> FAIL(HIT BAD TRAP)
Cyc= 253779500, Sec= 164.358, C/S=1544201.4
network config not specified, assuming 2dmesh configurationsims: group_name = tile4
sims: regress_date = 2016 06 10
sims: regress_time = 09 16 52
sims: /tank/mmckeown/research/projects/piton/openpiton/build/manycore/contint_nightly_tile4_dmbr_tile4_tile4_2016 06 10 0/simw +cpu num=0 +downwarningfinish+doerrorfinish +spc pipe=0 +asm err en +softint off=1 +inst check off=1 +vcs+lic+wait +vcs+dumpvaroff +finish mask=33 +TIMEOUT=500000 +wait cycle_to_kill=10 +max cycle=5000000 +tg seed=0 +good trap=0000002000:1000122000 +bad trap=0000002000:1000122000 -cm line+tg+cond+branch+asm -cm_name br_stress -cm_dir /tank/mmckeown/research/projects/piton/openpiton/build/manycore/contint_nightly_tile4_dmbr_tile4_tile4_2016 06 10 0 +efuse_data_file=efuse.img +asm_diag_name=br_stress.s +efuse_image_name=default.dat +fast_boot +dv_root=/tank/mmckeown/research/projects/piton/openpiton/piton

Details for ALL not in other groups
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
Details for nightly_dmbr_assembly_1_one_bin
-------------------------------------------------------------------------------------
Time"
Backup Slides
More Information

• More info can be found at:
  – OpenPiton Simulation Manual
    • http://openpiton.org or in OpenPiton download
  – OpenPiton discussion forum
  – Email openpiton@princeton.edu
PAL assembly test generators
Adding new tests

• Create new assembly file or copy existing

• Implement test functionality
Creating a new test from a template
Verilog Monitors

1. List of monitors

2. An example

3. Command line options
Why we have monitors

• Don’t freak out! They’re telling you something important
Commonly used monitors

Textual output for simulation debugging
sims.log
Sparc core, L1I/D, L1.5, L2, fake-memory, etc...
Monitors pass/fail traps
Monitor file locations

$PITON_ROOT/piton/verif/env/manycore/
  monitor.v.pyv
  lsu_mon.v.pyv, exu_mon.v, tlu_mon.v
  l2_mon.v.pyv, cmp_l15_messages_mon.v.pyv
Disabling monitors

Disable debug texts through plusargs to VCS:
- disable_l2_mon, disable_l15_mon
- turn_off_exu_monitor

Command line example:
```bash
sims -sys=manycore -vcs_run test.s -
sim_run_args=+disable_l2_mon -
sim_run_args=+disable_l15_mon -
sim_run_args=+turn_off_exu_monitor
```
Provided Testbenchs
Source/sink testbench interface
Unit tests
Running many tests
Running subsets of tests