Running the Architectural Compatibility Tests on your Model

By: Neel Gala & S Pawan Kumar
InCore Semiconductors Pvt. Ltd.

@incoresemi

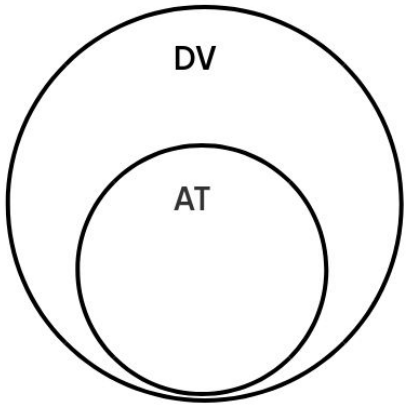
THE LINUX FOUNDATION

Prerequisite

- Targeted at test engineers/verification engineers
- Basic RISC-V assembly Knowledge
- Basic RTL implementation Knowledge
- Basic Python Knowledge

ACT: What, Why and When?

- Definition of ACT : Architectural Compatibility Tests
  - Used to test whether the designers interpreted and implemented the specification correctly.
  - Minimal positive testing to provide confidence in correctness of implementation
  - No negative testing.
    - Reason: Undefined behavior in most cases when some aspect is not implemented.
    - Does not mean that alternative behaviors are not tested. Example: misaligned memory accesses.
  - Signature based
    - Memory region to be dumped out at the end of each test and to be compared with the same generated by the reference (RISCV SAIL Model)
- ACT is not a substitute for DV
- Necessary for branding (Self certification).
- Ideally after verification
  - Potentially used as litmus during design



Note: The tests can be run at any point in the design-verification pipeline provided the environment has the necessary features to support it.

ACT Suite - https://github.com/riscv-non-isa/riscv-arch-test

Assumptions of ACT tests

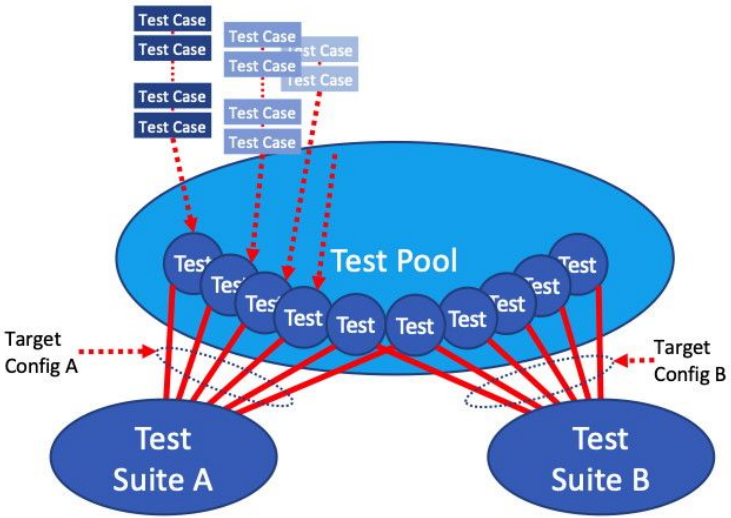
- Ability to use a custom trap handler
  - XTVEC writable to point to any 64B aligned region
  - Region pointed to by xtvec has read/write/execute permissions
- PMA Assumptions of current suite(minimum required)

	R/W	Xeq
Code	(4B)	Req
Data	1B,2B,4B,(8B)	-
Signature	1B,2B,4B,(8B)	-

( ) : Dependent on ISA Configuration
- : Don't care
N/A : Not Applicable;lack of tests

- Support exists only for implementations with homogenous behavior on misaligned accesses currently
- RVWMO memory model

ACT Structure



Owing to the configurable nature of the tests a framework/tool is required to select relevant tests and test-cases for a DUT - **RISCOF**

Necessary Tools

- Toolchain
  - riscv-gnu-toolchain for the reference
  - Custom allowed for the dut
- SAIL Reference Model for RISCV
- RISCOF as a python package
  - Requires Python 3.6.0+

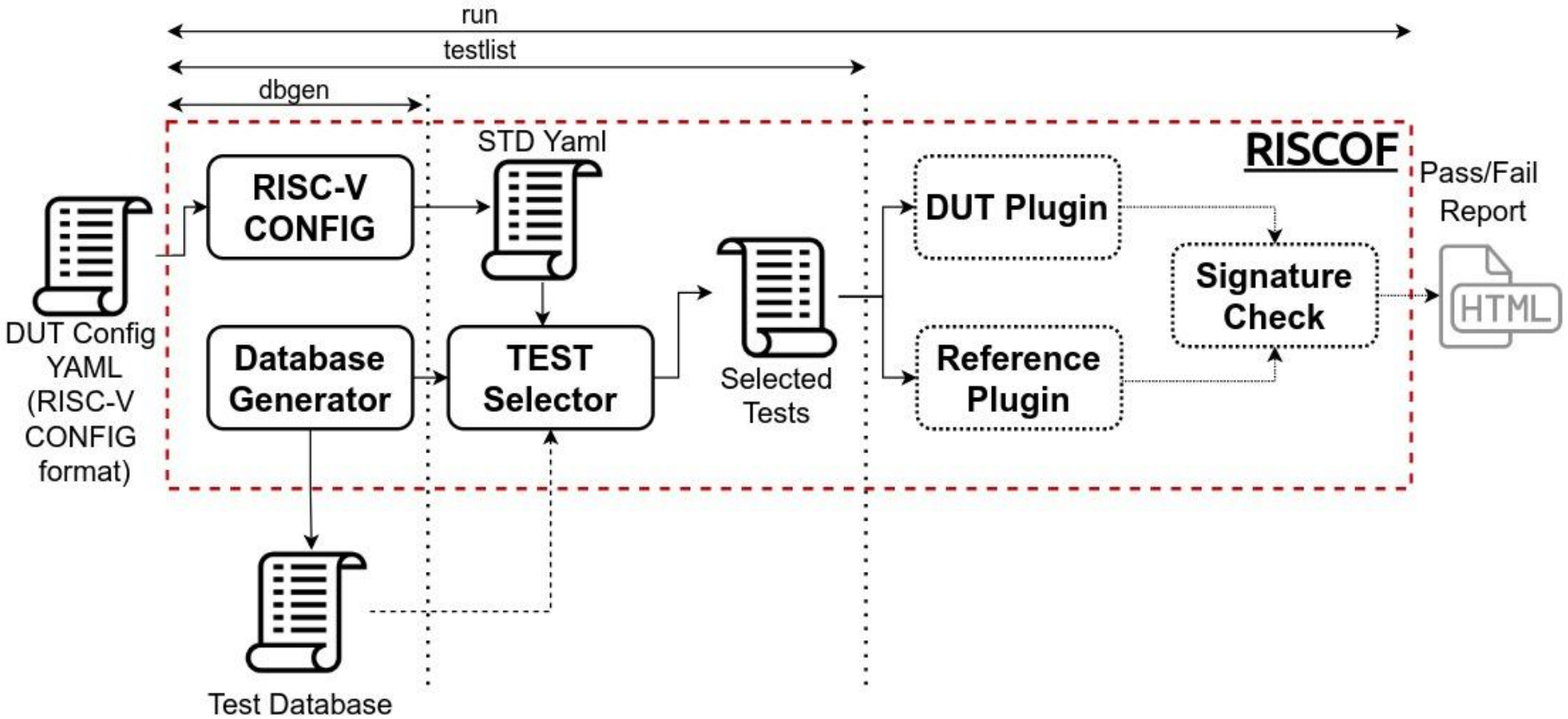
RISC-V GNU Toolchain - https://github.com/riscv-collab/riscv-gnu-toolchain
RISCV SAIL Model - https://github.com/riscv/sail-riscv
RISCOF Quickstart - https://riscof.readthedocs.io/en/stable/installation.html

Frequently Encountered Issues during installation*

- Requires sudo access to install various dependencies for toolchain binaries and SAIL model
- RHEL based systems
  - Z3 solver required by SAIL is not available as a standard package
  - Opam is not available as a standard package.
- Older python versions in the distro
- Scripts to automate installing all dependencies (link)*
  - Scripts which require sudo access - install_scripts/<distro>/sudo_<script_func>.sh
- SAIL is available as a docker container
  - Plugin for SAIL can use this to generate reference signatures
  - Get container using: docker pull registry.gitlab.com/incoresemi/docker-images/compliance

* - Courtesy Marc Karasek, InspireSemi, Inc

RISCOF: Overview



RISCOF Documentation - https://riscof.readthedocs.io/
RISCV-CONFIG Documentation - https://riscv-config.readthedocs.io/

RISCOF: Dbgen & Testlist*

dbgen

```
Generate the database of tests for the given suite
> riscof gendb --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir ./work
> tree -L 1 ./work
└── database.yaml ← Database yaml
```

testlist

```
Generate the testlist for the current DUT configuration
> riscof testlist --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir
./work
> tree -L 1 ./work/
├── database.yaml
├── riscv-test-suite ← Work Directory created by riscof
├── test_list.yaml ← Filtered list of tests for the given configuration
├── testdut_isa_checked.yaml
└── testdut_platform_checked.yaml ← Checked configuration files
```

Database Generation - https://riscof.readthedocs.io/en/stable/commands.html#gendb
Test List Generation - https://riscof.readthedocs.io/en/stable/commands.html#testlist

* - Discussed in detail in upcoming slides

RISCOF: Run

Run tests using a custom database file on the reference model

Run tests on reference model only

run

```
> riscof run --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir ./work --dbfile
./work/database.yaml --no-dut-run
> riscof run --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir ./work --no-dut-
run
> riscof run --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir ./work --testfile
./work/test_list.yaml --no-dut-run  <- Run the tests on the reference model using an existing testlist
> tree -L 1 ./work
├── Makefile.Reference-sail_c_simulator  <- Makefile from reference model
├── database.yaml
├── riscv-test-suite/
├── testdut_isa_checked.yaml
├── testdut_platform_checked.yaml
└── test_list.yaml
```

Run Command - https://riscof.readthedocs.io/en/stable/commands.html#run

RISCOF: Run

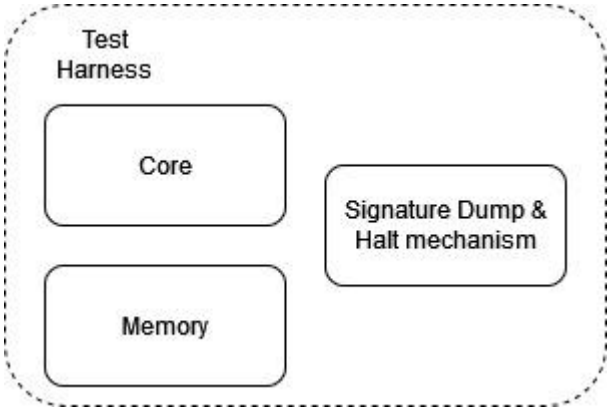
run

```
Run the tests on both the DUT and implementation and compare signatures
> riscof run --suite ./riscv-arch-test/ --env ./riscv-arch-test/riscv-test-suite/env/ --work-dir ./work
> tree -L 1 ./work
├── Makefile.DUT-testdut <── Makefile from the implementation
├── Makefile.Reference-sail_c_simulator
├── database.yaml
├── report.html <── Report generated for the run (self certify)
├── riscv-test-suite/
├── style.css <── Style file for the Report
├── testdut_isa_checked.yaml
├── testdut_platform_checked.yaml
└── test_list.yaml
```

Run Command - https://riscof.readthedocs.io/en/stable/commands.html#run

Recommended Test Harness

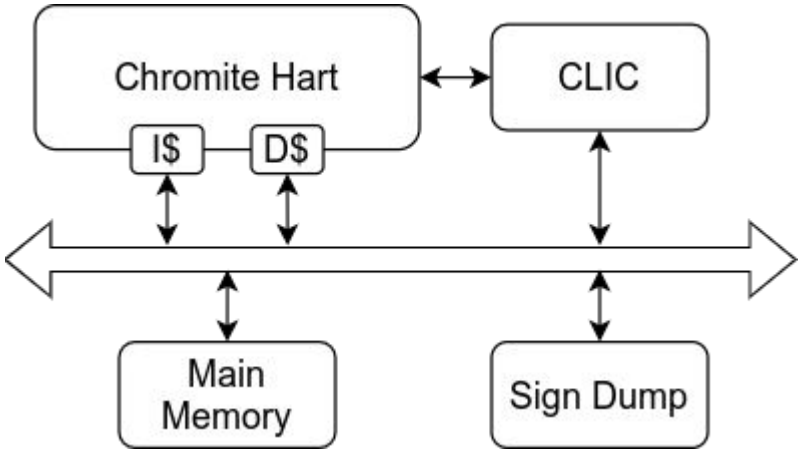
- Recommendations for testing environment
  - Test in RTL simulation with minimal additional modules.
    - HART + memory + Signature dumping
  - Tests target only the core and not the platform
    - Some platform features influence tests
      - Misaligned memory access
      - PMA of different regions
- **Disclaimer:** As long as the minimum requirements are met, it can be run in any environment but the tests are crafted with the above scenario in mind.



Test Harness Requirements

- Physical Memory Attributes (PMAs)
- Minimum memory requirements per section
  - Total - 1.14 GB (approx) (reduce to 21MB in future)
  - Code – 1.132 GB (jump tests) (reduce to 16MB in future)
  - Data – 2.3 MB
  - Sig – 1.6 MB
- Ability to dump memory region designated as signature
  - Bounded by rvtest_sig_begin and rvtest_sig_end labels
  - Output format should have 4 Bytes per line in little endian format.

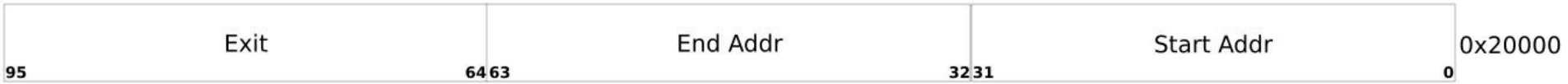
Harness example: Chromite signature plugin



- Memory Mapped @ (0x20000)
- Non synthesizable module
- Uses Language file I/O constructs(fwrite) to dump

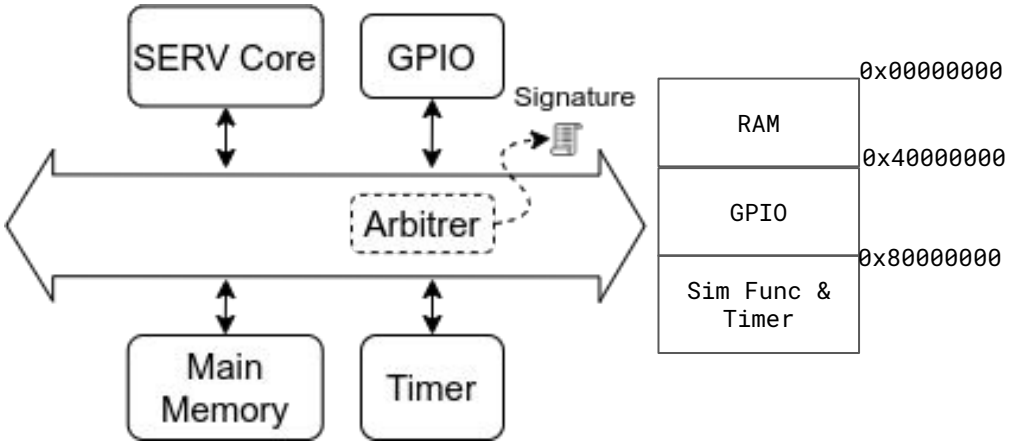
RISC-V ASM Code

```
1 sim_end:
2   fence;
3   li t6, 0x20000;
4   la t5, begin_signature;
5   sw t5, 0(t6);
6   la t5, end_signature;
7   sw t5, 8(t6);
8   sw t5, 12(t6);
```



RISCOF Plugin for Chromite - https://gitlab.com/incoresemi/riscof-plugins/-/tree/master/chromite
Chromite Implementation - http://core-generators.pages.incoresemi.com/chromite/
Signature Dump module - https://gitlab.incoresemi.com/core-generators/chromite/-/blob/master/test_soc/sign_dump.bsv

Harness Example: Signature Dump on SERV*



Pseudocode for halt

```
1 la a0, begin_signature;
2 la a1, end_signature;
3 li a2, 0x80000000;
4 while(a0 < a1){
5 convert byte in mem(a0) to hex ascii chars(a3)
6 sw a3, 0(a2); addi a0,a0,1; }
7 lui a0,0x90000000>>12;
8 sw a3,0(a0);
```

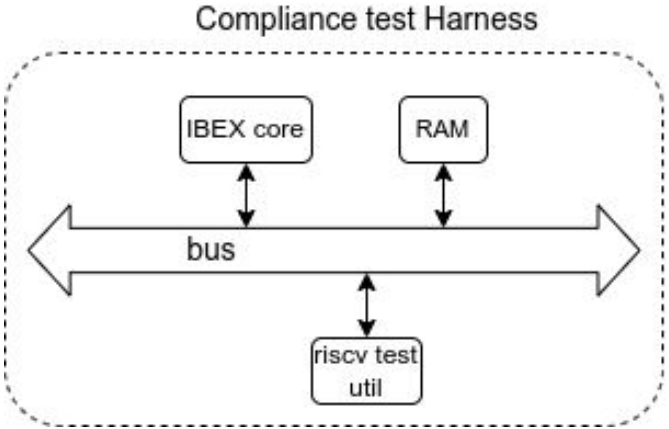
- Sim Only behavior
  - Redirects writes to 0x8XXXXXXX into files
  - A write to 0x9XXXXXXX halts sim
- Non synthesizable Code
- Uses Language file I/O constructs(fwrite) to dump the ascii character

- NEORV32 uses a sim only UART to achieve the same results.
- Similar pseudocode for halt.
- More information - NEORV32 Plugin

RISCOF Plugin for SERV - https://github.com/Abdulwadoodd/serv/blob/main/verif/plugin-serv/riscof_serv.py
SERV Implementation - https://github.com/olofk/serv
Signature Dump Logic - https://github.com/olofk/serv/blob/main/servant/servant_mux.v

* - Ported by Abdul Wadoodd (10x Engineers)

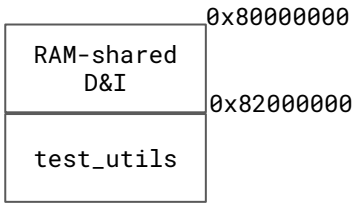
Harness Example: Ibex*




```
#define TESTUTIL_BASE 0x82000000
#define TESTUTIL_ADDR_BEGIN_SIGNATURE (TESTUTIL_BASE + 0x4)
#define TESTUTIL_ADDR_END_SIGNATURE (TESTUTIL_BASE + 0x8)

#define RVMODEL_HALT                                              \
  /* tell simulation about location of begin_signature */         \
        la t0, begin_signature;                                   \
        li t1, TESTUTIL_ADDR_BEGIN_SIGNATURE;                     \
        sw t0, 0(t1);                                             \
        /* tell simulation about location of end_signature */     \
        la t0, end_signature;                                     \
        li t1, TESTUTIL_ADDR_END_SIGNATURE;                       \
        sw t0, 0(t1);                                             \
        /* dump signature and terminate simulation */             \
        li t0, 1;                                                 \
        li t1, TESTUTIL_BASE;                                     \
        sw t0, 0(t1);
```

- riscv_testutil: Memory mapped @ 0x82000000
- Non synthesizable Code
- Uses Language IO constructs(display) followed by shell commands (grep, sed) to dump the signature



Plugin for Ibex - https://github.com/Abdulwadoodd/ibex/tree/compliance/dv/riscv_compliance

* - Courtesy of Abdul Wadoodd (10x Engineers)

DUT: Testing Environment

Setup

Setup reference plugin and templates for DUT plugin

```
> riscof setup --dutname testdut --work-dir ./work
> riscof arch-test --clone
> tree -L 1
.
├── config.ini
├── riscv-arch-test
├── sail_cSim
└── testdut
```

Clone the repo from github

RISCOF Configuration file

Test Suite from Github

Plugin Directories

config.ini

```
[RISCOF]
ReferencePlugin=sail_cSim
ReferencePluginPath=/demo/sail_cSim
DUTPlugin=testdut
DUTPluginPath=/demo/testdut

[testdut]
pluginpath=/demo/testdut
ispec=/demo/testdut/testdut_isa.yaml
pspec=/demo/testdut/testdut_platform.yaml

[sail_cSim]
pluginpath=/demo/sail_cSim
docker=True
image=compliance
```

Passed to plugin

Required only if installed via docker

testdut_isa.yaml

```
hart_ids: [0]
hart0:
  ISA: RV64IMCZicsr_Zifencei
  physical_addr_sz: 32
  User_Spec_Version: '2.3'
  supported_xlen: [64]
```

Plugin Directory contents

```
1 > tree -L 2 ./testdut/
2 ├── env
3 │   ├── link.ld
4 │   └── model_test.h
5 ├── riscof_testdut.py
6 ├── testdut_isa.yaml
7 └── testdut_platform.yaml
```

Linker file

Model specific macros

Sail Plugin documentation - https://gitlab.com/incoresemi/riscof-plugins/-/tree/master/sail_cSimt

DUT: Model specific Macros

- Purpose
  - Define model specific behaviors for various operations required by the tests
  - Compile tests to run on the DUT based on its configuration
- RVMODEL_BOOT
  - Entry label points to the start of this macro
  - Contains the code for the booting process
    - Entirely model specific - Can be empty
    - Any sort of initialisation routine can be implemented here
  - CSRs at reset state
  - RISCV-ISA-SIM and SAIL define this as an empty macro
  - Trap handler requirements
    - If xTVEC not writable: Region pointed to by xTVEC has read/write/execute permissions

Macro documentation - https://github.com/riscv-non-isa/riscv-arch-test/blob/main/spec/TestFormatSpec.adoc#432-required-model-defined-macros

DUT: Model Specific Macros

RVMODEL_DATA_BEGIN - RISCV-ISA-SIM

Custom Section for FESVR

```
1 #define RVMODEL_DATA_BEGIN\
2   .pushsection .tohost,"aw",@progbits;\
3   .align 8; .global tohost; tohost: .dword 0;\
4   .align 8; .global fromhost; fromhost: .dword 0;\
5   .popsection;\
6   .word 4;\
7   .align ALIGNMENT;\
8   .global begin_signature; begin_signature:
```

RVMODEL_DATA_END - RISCV-ISA-SIM

```
1 #define RVMODEL_DATA_END\
2   .global end_signature; end_signature:
```

Custom labels for signature boundary

- RVMODEL_DATA_BEGIN
  - Demarcates the start of the signature section
- RVMODEL_DATA_END
  - Demarcates the end of the signature section

Model header file for RISCV-ISA-SIM - https://gitlab.com/incoresemi/riscof-plugins/-/blob/master/spike_parallel/env/model_test.h

DUT: Model Specific Macros

- RVMODEL_HALT
  - Before entry to this macro the state is restored to what the RVMODEL_BOOT requires (sticky states are not restored)
  - Test jumps to this location with an expectation to end the sim/run at any point.
  - Perform any necessary operations such as IO to dump signature
  - Halt/Terminate at the end
    - Model specific halt sequence

RISCV-ISA-SIM

```
1 #define RVMODEL_HALT \
2   li x1, 1;\
3   write_tohost:\
4     sw x1,tohost,t2;\
5     j write_tohost;
```

Exit Sim

Chromite

```
1 #define RVMODEL_HALT \
2 sim_end:\
3   fence;\
4   li t6, 0x20000;\
5   la t5, begin_signature;\
6   sw t5, 0(t6);\
7   la t5, end_signature;\
8   sw t5, 8(t6);\
9   sw t5, 12(t6);
```

Dump Signature

Model header file for RISCV-ISA-SIM - https://gitlab.com/incoresemi/riscof-plugins/-/blob/master/spike_parallel/env/model_test.h

DUT: Linker script

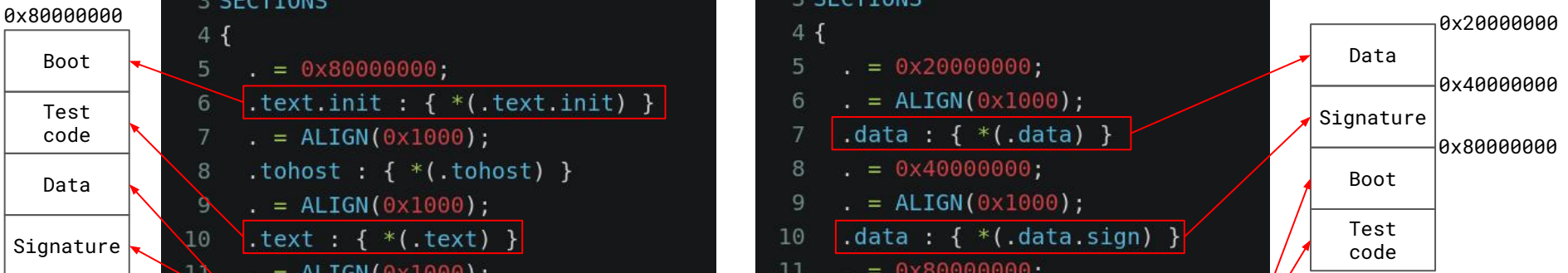
- Map sections to memory locations with appropriate PMAs
- Entry point should be set to “rvtest_entry_point”

Linker Script - RISCV-ISA-SIM & SAIL

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

Linker Script - Hypothetical

```
1 OUTPUT_ARCH( "riscv" )
2 ENTRY(rvtest_entry_point)
3 SECTIONS
4 {
5   . = 0x20000000;
6   . = ALIGN(0x1000);
7   .data : { *(.data) }
8   . = 0x40000000;
9   . = ALIGN(0x1000);
10  .data : { *(.data.sign) }
11  . = 0x80000000;
12  .text.init : { *(.text.init) }
13  . = ALIGN(0x1000);
14  .text : { *(.text) }
15  _end = .;
16 }
```



Linker file for RISCV-ISA-SIM - https://gitlab.com/incoresemi/riscof-plugins/-/blob/master/spike_parallel/env/link.ld

DUT: RISCOF Plugin Template

Rename to match input in config.ini and name the python file as riscof_<dutname>.py

Plugin

```python
from riscof.pluginTemplate import pluginTemplate
class dutname(pluginTemplate):
    __model__ = <name_of_dut>
    __version__ = <version_number>

    def __init__(self, *args, **kwargs):
        sclass = super().__init__(*args, **kwargs)
        <Your Plugin Code>
        return sclass

    def initialise(self, suite, workdir, env):
        '''
            :param suite: Absolute path to the suite directory
            :param workdir: The absolute path to the work directory.
            :param env: The directory containing the header files.
         '''
        // Initialise any local variables needed and store the arguments
```

Plugin

```python
    def build(self, isa_yaml, platform_yaml):
        """
            :param isa_yaml: Path to the checked isa specs yaml.
            :param platform_yaml: Path to the checked platform specs yaml.
        """
        // Configure run commands and other necessary options based
        // on input yamls

    def runTests(self, testlist):
        """ Use the model to run the tests and produce signatures.
            The signature files generated should be
            named-*self.name[:-1]+".signature"*.
            :param testlist: A dictionary of tests and
                other information about them(like macros,work_dir and isa).
        """
        // Compile and run the tests on your model
        // At the end of this function signature files should
        // be present in the appropriate test directory
```

Examples of RISCOF Plugins - https://gitlab.com/incoresemi/riscof-plugins/

DUT: Plugin Functions

Get config node passed in the ini file

Setup paths to the yaml files. (can be defined statically)

```
1    def __init__(self, *args, **kwargs):
2        config = kwargs.get('config')
3        self.dut_exe = "spike"
4        self.isa_spec = abspath(config['ispec'])
5        self.platform_spec = abspath(config['pspec'])
6        self.pluginpath = abspath(config['pluginpath'])
7
8    def initialise(self, suite, work_dir, archtest_env):
9        self.work_dir = work_dir
10       self.compile_cmd = 'riscv{1}-unknown-elf-gcc \
11        -march={0} -static -mcmodel=medany \
12       -fvisibility=hidden -nostdlib -nostartfiles \
13        -T '+self.pluginpath+'/env/link.ld \
14        -I '+self.pluginpath+'/env/ -I ' + archtest_env
```

Define Compile Command

Linker file for the implementation

Directory with the implementation header files

Checked Configuration Files

Configure spike based on input configuration

```
1  def build(self, isa_yaml, platform_yaml):
2      ispec = utils.load_yaml(isa_yaml)['hart0']
3      self.xlen = str(max(spec['supported_xlen']))
4      self.isa = 'rv' + self.xlen
5      if "64I" in ispec["ISA"]:
6          self.compile_cmd = \
7              self.compile_cmd+' -mabi='+'lp64 '
8      elif "64E" in ispec["ISA"]:
9          self.compile_cmd = \
10             self.compile_cmd+' -mabi='+'lp64e '
11     elif "32I" in ispec["ISA"]:
12         self.compile_cmd = \
13             self.compile_cmd+' -mabi='+'ilp32 '
14     elif "32E" in ispec["ISA"]:
15         self.compile_cmd = \
16             self.compile_cmd+' -mabi='+'ilp32e '
17     if "I" in ispec["ISA"]:
18         self.isa += 'i'
19     if "M" in ispec["ISA"]:
20         self.isa += 'm'
21     if "C" in ispec["ISA"]:
22         self.isa += 'c'
```

Plugin file for RISCV-ISA-SIM - https://gitlab.com/incoresemi/riscof-plugins/-/blob/master/spike_parallel/riscof_spike_parallel.py

DUT: Plugin Run Function

Entries in the database file have the same keys at the root level

Testlist format

```
/demo/riscv-arch-test/riscv-test-suite/rv64i_m/M/src/div-01.S:
  work_dir: /demo/riscof_work/src/div-01.S
  macros:
  - TEST_CASE_1=True
  - XLEN=64
  isa: RV64IM
  coverage_labels:
  - div
  test_path: /demo/riscv-arch-test/riscv-test-suite/rv64i_m/M/src/div-01.S
```

Testlist format

```
<name of assembly file>:
  work_dir: <absolute path>
  macros:
    [ <list of macros and values> ]
  isa: <isa for test compilation>
  coverage_labels:
    [ <covergroups for the test> ]
  test_path: <path to assembly file>
```

Constructing the command

```
def runTests(self, testList, cgf_file=None):
  for file in testList:
    testentry = testList[file]
    test = testentry['test_path']
    test_dir = testentry['work_dir']
    command= "@cd "+testentry['work_dir']+";" + = self.compile_cmd.format(testentry['isa'].lower(), self.xlen) \
           + ' ' + test + ' -o ' + 'dut.elf' + ' -D' + " -D".join(testentry['macros'])+";"
    sig_file = os.path.join(test_dir, self.name[:-1] + ".signature")
    command += self.dut_exe + ' --isa={0} +signature={1} +signature-granularity=4 {2};'.format(
ISA from config  self.isa, sig_file, 'dut.elf')
```

Construct compile command

Define macros

Set march

Signature file name

Command to execute spike

Test List Format - https://riscof.readthedocs.io/en/stable/testlist.html

DUT: Plugin Run Function

Using the make utility

```
make = utils.makeUtil(makefilePath=os.path.join(self.work_dir, "Makefile." + self.name[:-1]))
make.makeCommand = self.make + ' -j' + self.num_jobs
<Construct command>
make.add_target(command[,tname='target'])
make.execute("target")
make.execute_all(self.work_dir)
```

Path to generate the makefile at

Modify to run parallel jobs(optional)

Add a target

Run single target

Run all targets

Directory to execute the make command in

Target generated in Makefile

```
.PHONY : target
target :
    @cd /demo/riscof_work/riscv-test-suite/rv64i_m/M/src/div-01.S/dut; \
    riscv64-unknown-elf-gcc -march=rv64im -static -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles \
    -T /demo/riscof-plugins/spike_parallel/env/link.ld \
    -I /demo/riscof-plugins/spike_parallel/env/ \
    -I /demo/riscof/riscv-arch-test/riscv-test-suite/env \
    -mabi=lp64  /demo/riscv-arch-test/riscv-test-suite/rv64i_m/M/src/div-01.S \
    -o dut.elf -DTEST_CASE_1=True -DXLEN=64; \
    spike --isa=rv64im +signature-granularity=4 \
    +signature=/demo/riscof_work/riscv-test-suite/rv64i_m/M/src/div-01.S/dut/DUT-spike.signature dut.elf;
```

ISA from test

Work Directory for Test

Custom header and linker files

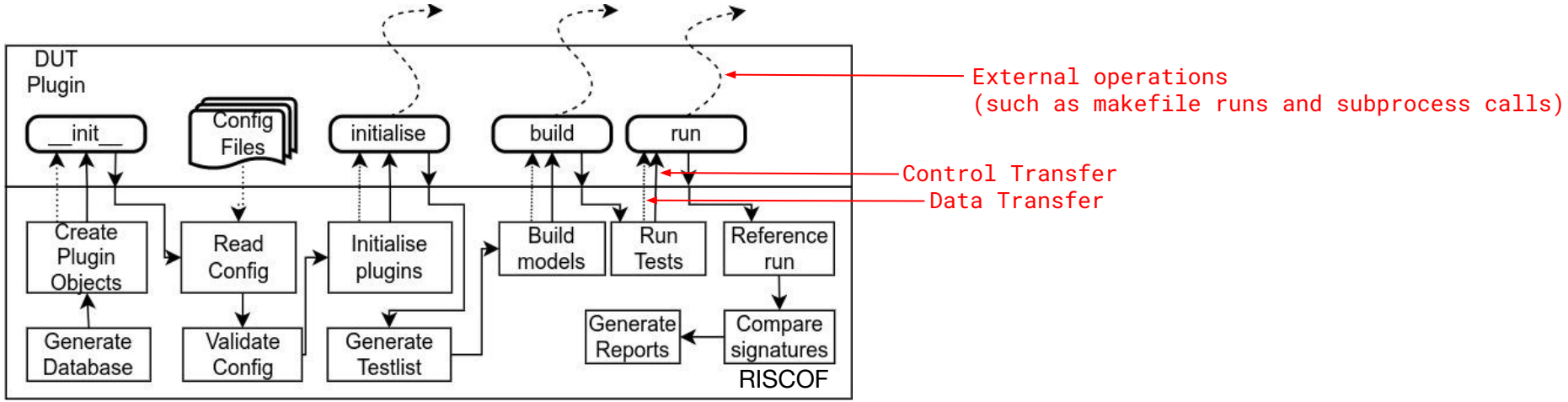
ACT Header file location

Macros

Signature file

Make Utility API documentation - https://riscof.readthedocs.io/en/stable/code-doc.html#riscof.utils.makeUtil

DUT: Alternate method to execute commands



- No difference as long as signature files are present at the correct location
- simple subprocess call to execute the command.
  - *shellCommand* utility to ease running shell commands as a subprocess.

Using subprocesses

```
1 utils.shellCommand(command).run()
```

- SAIL plugin supports reference signature generation via docker images (link)

DUT: Debugging Tests.

- Test Features for easier debugging
  - Each instance in the test is preceded by a label.
  - Use execution log and disassembly to correlate.

Test

```
1 inst_4:
2 // rs1==x9, rd==x16, rs1_val < 0 and imm_val == (xlen-1), rs1_val == -4194305
3 // opcode: slli ; op1:x9; dest:x16; op1val:-0x400001;  immval:0x1f
4 TEST_IMM_OP( slli, x16, x9, 0x80000000, -0x400001, 0x1f, x3, 16, x7)
```

Disassembly

```
1 00000000800003c8 <inst_4>:
2     800003c8:   e00007b7           lui a5,0xe0000
3     800003cc:   fff7879b           addiw   a5,a5,-1
4     800003d0:   03f79513           slli    a0,a5,0x3f
5     800003d4:   02a1b023           sd  a0,32(gp)
```

DUT: Debugging Tests

- How to Run a single test using riscof
  - Edit the dbfile generated to contain only the node corresponding to the test and pass to riscof using –dbfile argument & –no-clean
  - Edit the testlist generated to contain only the node corresponding to the test and pass to riscof using –testlist argument & –no-clean
  - Isolate the test in a separate directory and use that as argument to suite
  - Edit test source file to include only the faulting instance (not recommended).
    - Identify faulty instance by looking at the disassembly (contains labels which show which test failed).
- How to Run a single test(if using makefiles)

DUT: Debugging Tests

- Pitfall: Danger of an infinite trap loop or worse
- Enabling trap handler for the test
  - Add and initialise a default trap handler in RVMODEL_BOOT
  - Might need to include zicsr in the ISA while compiling the tests depending on the toolchain

Sample Trap Handler

```
1 #define RVMODEL_BOOT \
2   j entry; \
3   handler: \
4     j shutdown; \
5   entry: \
6     la x2, handler; \
7     csrw mtvec, x2;
8
9 #define RVMODEL_HALT \
10  shutdown: \
11    li x1, 1; \
12    sw x1,tohost, t2; \
13  inf_loop: \
14    j inf_loop;
```

Future of Architectural Testing Ecosystem

- Arch-test: CSR tests
- Arch-test: CLINT tests
- Arch-test: Memory Model tests
- RISCOF: Ability to extract labels of the test instance which failed using the elf and signature files.
- Tools/Interfaces for Async interrupt testing

Takeaways

- Brief overview of ACT
- Understanding about riscof
- Getting the DUT ready for testing
  - Signature Dumping mechanism
- Running ACT on DUT using riscof
- Getting the testing environment ready
  - RISCOF Plugin
  - Model Specific Macros
  - Linker File
- Debugging strategies

For any feedback/questions, please file issues on github(suite, framework)

Questions?

Additional Resources

- Test Suite (link)
- Test Format Specification (link)
- Setup Environment for using riscof (link)
- Installation instructions for reference model (link)
- Using reference model via docker(link)
- Running Architectural Tests on Spike as DUT (link)
- Guide to writing riscof plugins for a DUT (link)
- Example plugins for various hardware cores & simulators (link)

Examples of nuances between ACT and DV.

- F Extension
  - DV: Check result is correct for all possible QNaN values
  - ACT
    - Check result for all classes of floating point inputs(NaN, 0, inf)
    - Nan-Boxing of lesser width results in presence of D extension
    - Check NaN propagation
- SV48 Virtualisation scheme
  - DV: Test whether translation works for all VA to PA mappings
  - ACT: Test whether translation works for a mapping from VA to PA where all VPN != PPN