Verifying RISC-V Vector and Bit Manipulation Extensions using STING Design Verification Tool

Shubhodeep Roy Choudhury, Shajid T, Jevin John, George S

Valtrix SYSTEMS

Agenda

- Vector and bit manipulation support – Where we are?
- Overview of STING
- Verification strategy
- Results/Bugs Found
- Future work

Valtrix SYSTEMS

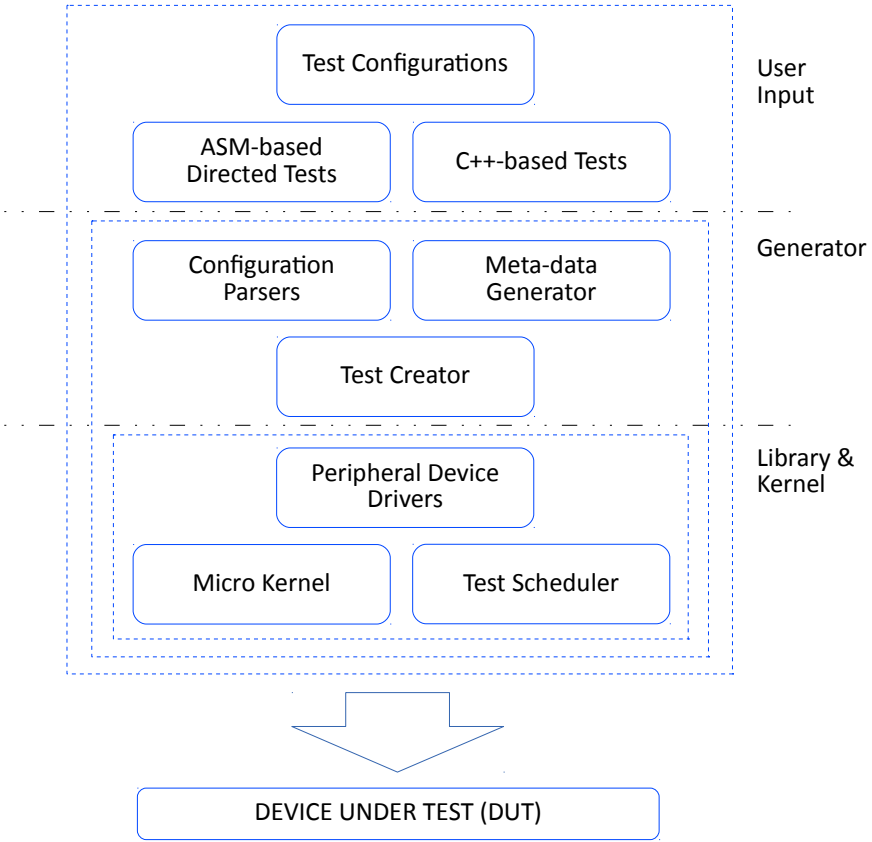
Status of Vector and Bit Manipulation Support

- Vector specification version 0.8-draft-20190906
- Bit manipulation specification version 0.9.2
- Support for all the 650+ vector and 100 bit manipulation instructions added to STING
- Work in progress

Valtrix
SYSTEMS

Overview of STING

- Highly configurable and flexible bare metal program
- Intelligent and architecturally correct instruction sequences complete with results checking
- Supports constrained random, directed and, graph based test generation methodologies
- Portable stimulus – simulations, emulation, FPGA or silicon
- MP and SoC ready



Valtrix SYSTEMS

Stimulus Development Framework - Snippets

- Programming framework for RISC-V test development
- High level language constructs mixed with inline assembly
- Allows reservation of registers, memory and hardware threads
- Enables users to install interrupt and exception handlers
- Embedded into the instruction stream after resolving the resource allocation constraints
- Easily integrate existing assembly and C/C++ tests

Valtrix
SYSTEMS

Rendering of Snippets

```
21 resource
22 cpu_t    x
23 base_t   rb1[4]@4
24 axreg_t rx1, rx2
25
26
27 snip_setup
28     ; Empty
29
30
31 snip_init
32
33 @cpu: x
34     sw      x0, 0 (rb1)     ; Initialize the shared memory with 0
35
36
37 snip_run
38
39 @cpu: x
40     li      rx1, 1          ; Master cpu writes a value of 1 in the
41                             ; shared memory. Use any one of the store
42                             ; operations for this
43
44     sw rx1, 0 (rb1)
45
46 @cpu: *
47     li      rx2, 1          ; Slave cpu keeps looping till it sees
48                             ; the flag update from master
49
50 do_branch:
51     lw rx1, 0 (rb1)
52     blt     rx1, rx2, do_branch
53
54
55 snip_check
56     ; Empty
```

- Rendering is the process of embedding the snippet into instruction stream
- Rendering of test intent along with different stimulus patterns leads to increased cross product coverage

```
1769     fsgnjn.d  f28, f10, f11
1770     lwu     x14, 1624(x28)
1771     amoor.d.rl    x24, x24, (x29)
1772     amoor.d.aq    x30, x30, (x27)
1773     enable_rvc "c.sw      x12, 60(x8)"
1774     // ++snip start: snippet_id: 00000005 SNP_MP_SimpleWait
1775     addi  x23, x23, 16
1776     .global start_SNP_MP_SimpleWait_run_block0_120_5_0
1777 start_SNP_MP_SimpleWait_run_block0_120_5_0:
1778     li      x10, 0x1
1779     sw      x10, 0(x23)
1780     .global end_SNP_MP_SimpleWait_run_block0_121_5_0
1781 end_SNP_MP_SimpleWait_run_block0_121_5_0:
1782     .global start_SNP_MP_SimpleWait_run_block1_122_5_0
1783 start_SNP_MP_SimpleWait_run_block1_122_5_0:
1784     li      x7, 0x1
1785 do_branch_119_5_0:
1786     lw      x10, 0(x23)
1787     blt x10, x7, do_branch_119_5_0
1788     .global end_SNP_MP_SimpleWait_run_block1_123_5_0
1789 end_SNP_MP_SimpleWait_run_block1_123_5_0:
1790     // --snip end: SNP_MP_SimpleWait
1791     lw      x12, 1516(x18)
1792     lh      x14, 1658(x27)
build/gen/tests/machine_test_0.s [+]          1752,5          7%
```

```
1601     // X26 <- [3101]
1602     lbu     x24, 1053(x26)
1603     fence     r, io
1604     enable_rvc "c.lw      x14, 32(x15)"
1605     sb      x13, -1648(x29)
1606     fcvt.w.d    x11, f11, rdn
1607     enable_rvc "c.ld      x13, 72(x15)"
1608     enable_rvc "c.sw      x13, 36(x15)"
1609     amominu.w.aq  x5, x5, (x20)
1610     // ++snip start: snippet_id: 01000005 SNP_MP_SimpleWait
1611     addi  x23, x23, 16
1612     .global start_SNP_MP_SimpleWait_run_block1_122_5_1
1613 start_SNP_MP_SimpleWait_run_block1_122_5_1:
1614     li      x7, 0x1
1615 do_branch_119_5_1:
1616     lw      x5, 0(x23)
1617     blt x5, x7, do_branch_119_5_1
1618     .global end_SNP_MP_SimpleWait_run_block1_123_5_1
1619 end_SNP_MP_SimpleWait_run_block1_123_5_1:
1620     // --snip end: SNP_MP_SimpleWait
1621     amoor.d       x6, x12, (x22)
1622     ld      x11, -2000(x22)
1623     enable_rvc "c.lw      x14, 24(x15)"
1624     amoadd.w.aq   x14, x24, (x29)
build/gen/tests/machine_test_1.s [+]
```

Valtrix SYSTEMS

Verification Strategy

- Test plan based verification
- Pattern based tests – Fixed and Random
- Constrained random and graph based test generation
- Snippets for directed scenarios
- Cross product with PMP, virtual memory, privilege mode of execution etc.

Valtrix
SYSTEMS

Fixed Pattern Directed Tests

- Define data sets of fixed input and output patterns for every instruction
- Cover the corner case and interesting values
- Useful for basic testing of ISA implementation

```
#################################################################
#
# VALTRIX TECHNOLOGIES PVT. LTD. CONFIDENTIAL
# _____________________________________________
#
# [2016] - Valtrix Technologies Private Limited
# All Rights Reserved.
#
# NOTICE:  All  information  contained herein is, and remains the
# property  of  Valtrix  Technologies  Pvt. Ltd. The intellectual
# and  technical  concepts  contained  herein  are proprietary to
# Valtrix  Technologies  Pvt.  Ltd. and may be covered by Indian,
# U.S. and Foreign Patents, patents in process, and are protected
# by  trade  secret  or  copyright  law.  Dissemination  of  this
# information  or  reproduction  of  this  material  is  strictly
# forbidden  unless  prior  written  permission  is obtained from
# Valtrix Technologies Pvt. Ltd.
#
#################################################################

# The dictionary given below defines the database of fixed patterns supported
# by the instructions defined under the bit manipulation extension.
#
# SRC_1 contains the data to be loaded into the first source operand. In case
# an instruction supports another source operand, the data set will have SRC_2
# field defined in the dictionary. The expected result for the input data will
# be specified using EXPECTED_RESULT field

ISA_CLZ_CONF = {

    DATA_SET : [
        { SRC_1 : 0xFFFFFFFFUL    EXPECTED_RESULT : 0x00000000 },
        { SRC_1 : 0x00000000UL    EXPECTED_RESULT : 0x00000020 },
        { SRC_1 : 0x01000000UL    EXPECTED_RESULT : 0x00000007 },
        { SRC_1 : 0x00100000UL    EXPECTED_RESULT : 0x0000000B },
        { SRC_1 : 0x00010000UL    EXPECTED_RESULT : 0x0000000F },
        { SRC_1 : 0x00001000UL    EXPECTED_RESULT : 0x00000013 },
        { SRC_1 : 0x00000100UL    EXPECTED_RESULT : 0x00000017 },
        { SRC_1 : 0x00000010UL    EXPECTED_RESULT : 0x0000001B },
        { SRC_1 : 0x00000001UL    EXPECTED_RESULT : 0x0000001F },
    ]
}
```

```
#################################################################
#
# VALTRIX TECHNOLOGIES PVT. LTD. CONFIDENTIAL
# _____________________________________________
#
# [2016] - Valtrix Technologies Private Limite
# All Rights Reserved.
#
# NOTICE:  All  information  contained herein
# property  of  Valtrix  Technologies  Pvt. Lt
# and  technical  concepts  contained  herein
# Valtrix  Technologies  Pvt.  Ltd. and may be
# U.S. and Foreign Patents, patents in process
# by  trade  secret  or  copyright  law.  Diss
# information  or  reproduction  of  this  mat
# forbidden  unless  prior  written  permissio
# Valtrix Technologies Pvt. Ltd.
#
#################################################################

# The dictionary given below defines the datab
# by the instructions defined under the bit ma
#
# SRC_1 contains the data to be loaded into th
# an instruction supports another source opera
# field defined in the dictionary. The expecte
# be specified using EXPECTED_RESULT field

ISA_CLZ_CONF = {

    DATA_SET : [
        { SRC_1 : 0xFFFFFFFFFFFFFFFFUL    EXPE
        { SRC_1 : 0x0000000000000000UL    EXPE
        { SRC_1 : 0x0100000000000000UL    EXPE
        { SRC_1 : 0x0010000000000000UL    EXPECTED_RESULT : 0x000000000000000b },
        { SRC_1 : 0x0001000000000000UL    EXPECTED_RESULT : 0x000000000000000f },
        { SRC_1 : 0x0000100000000000UL    EXPECTED_RESULT : 0x0000000000000013 },
        { SRC_1 : 0x0000010000000000UL    EXPECTED_RESULT : 0x0000000000000017 },
        { SRC_1 : 0x0000001000000000UL    EXPECTED_RESULT : 0x000000000000001b },
        { SRC_1 : 0x0000000100000000UL    EXPECTED_RESULT : 0x000000000000001f },
        { SRC_1 : 0x0000000010000000UL    EXPECTED_RESULT : 0x0000000000000023 },
        { SRC_1 : 0x0000000001000000UL    EXPECTED_RESULT : 0x0000000000000027 },
        { SRC_1 : 0x0000000000100000UL    EXPECTED_RESULT : 0x000000000000002b },
    ]
}
```

Valtrix SYSTEMS

Fixed Pattern Directed Tests (contd)

- Once data sets are ready, snippets are developed to program the input patterns and execute the instruction
- Results from the execution check-pointed and compared with the expected output
- Error reported in case a failure is is detected during comparison

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Fixed pattern directed test for RV32B CLZ instruction


import "src/snippets/rv/rv_param_blocks.pb"

resource
cpu_t    x
base_t   rbl[8]@8
axreg_t  rx1, rx2
unum_t   rno = snp_urandrange(0, ISA_CLZ_CONF.NUM_DATA_SET)


snip_setup
    ; Empty


snip_init

@cpu: x
    write_reg_to_mem(reg


snip_run

@cpu: x
    construct_inst_reg(inst = clz, imm  = ISA_CLZ_CONF.DATA_SET[rno].SRC_1,
                                   regx = rx1,
                                   regy = rx2,
                                   regb = rbl)


snip_check

@cpu: x
    check_mem(val = ISA_CLZ_CONF.DATA_SET[rno].EXPECTED_RESULT, regx = rx1,
                                                                regy = rx2,
                                                                regb = rbl)
```

```
snip_run

@cpu: x

    for index in (0, ISA_CLZ_CONF.NUM_DATA_SET):
        construct_inst_reg(inst = clz, imm  = ISA_CLZ_CONF.DATA_SET[index].SRC_1,
                                       regx = rx1,
                                       regy = rx2,
                                       regb = rbl)

        check_mem(val = ISA_CLZ_CONF.DATA_SET[index].EXPECTED_RESULT, regx = rx1,
                                                                      regy = rx2,
                                                                      regb = rbl)
    endfor

```

Valtrix SYSTEMS

Fixed Pattern Directed Tests (contd)

- In case of vectors, the data sets are tested with different configuration settings, which include

  - Supported SEW values

  - Supported LMUL values

  - Different combinations of VSTART and VL

  - Different combinations of mask register (V0)

  - Different combinations of VLEN, ELEN, SLEN and SELEN

Valtrix SYSTEMS

Random Pattern Directed Tests

- Fixed pattern tests not adequate to test all input/source values
- Limitation addressed by random pattern tests, where randomly chosen values are programmed into the source register
- The expected output is simulated and compared with the result obtained from the DUT

```
20 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
21
22 ; Random pattern directed test for RV32B CLZ instruction
23
24 import "src/snippets/rv/rv_param_blocks.pb"
25
26 resource
27 cpu_t     x
28 unum_t    inp
29 unum_t    exp_res
30 axreg_t   rx1, rx2, rx3
31 memory_t mem_chkp[8]@8
32
33 snip_setup
34     ; Empty
35
36 snip_init
37
38 @cpu: x
39
40     inp = snp_randnum()
41
42     la rx2, mem_chkp
43     write_val_to_mem(regx = rx1, regb = rx2, val = 0)
44
45 snip_run
46
47 @cpu: x
48
49     li rx1, inp
50
51     clz rx2, rx1
52
53     la rx3, mem_chkp
54     write_reg_to_mem(regx = rx2, regb = rx3, offst = 0)
55
56 snip_check
57
58 @cpu: x
59
60     exp_res = snp_sim_clz(inp)
61
62     la rx3, mem_chkp
63     check_mem(val = exp_res, regx = rx1, regy = rx2, regb = rx3)
```

Valtrix SYSTEMS

Random Pattern Directed Tests (contd)

```
1446     // ++snip start: snippet_id: 00000004 SNP_ISA_SIM_CLZ
1447     // Snippet memory : block = DEFAULT_SNIPPET_MEMORY offset = 0x2808 size = 0x8 page_size = PAGE_1G
1448     .global start_SNP_ISA_SIM_CLZ_run_block0_112_4_0
1449 start_SNP_ISA_SIM_CLZ_run_block0_112_4_0:
1450     li      x5, 0x1c1c0a761e692273
1451     clz     x6, x5
1452     li      x10, 0xffffc00080003808
1453     sd      x6, 0(x10)
1454     .global end_SNP_ISA_SIM_CLZ_run_block0_113_4_0
1455 end_SNP_ISA_SIM_CLZ_run_block0_113_4_0:
1456     // --snip end: SNP_ISA_SIM_CLZ
sup_test_0.s [+]
18449     // ++snip check start: snippet_id: 00000004 SNP_ISA_SIM_CLZ
18450     // Snippet memory : block = DEFAULT_SNIPPET_MEMORY offset = 0x2808 size = 0x8 page_size = PAGE_1G
18451     .global start_SNP_ISA_SIM_CLZ_check_block0_115_4_0
18452 start_SNP_ISA_SIM_CLZ_check_block0_115_4_0:
18453     li      x10, 0xffffc00080003808
18454     li      x5, 0x3
18455     ld      x6, 0(x10)
18456     beq x5, x6, check_mem_pass_114_4pb_check_mem
18457     la      x5, addr_0_0_snippet_fail
18458     ld      x5, 0(x5)
18459     j addr_0_0_snippet_fail_end
18460     .option rvc
18461     .option relax
18462     .balign 8
18463 addr_0_0_snippet_fail:
18464     .dword snippet_fail
18465 addr_0_0_snippet_fail_end:
18466     .option norvc
18467     .option norelax
18468     jalr x5
18469 check_mem_pass_114_4pb_check_mem:
18470     .global end_SNP_ISA_SIM_CLZ_check_block0_116_4_0
18471 end_SNP_ISA_SIM_CLZ_check_block0_116_4_0:
18472     // ++snip check end: snippet_id: 00000004 SNP_ISA_SIM_CLZ
```

1. Generation of random input
2. Instruction execution
3. Checkpointing of output

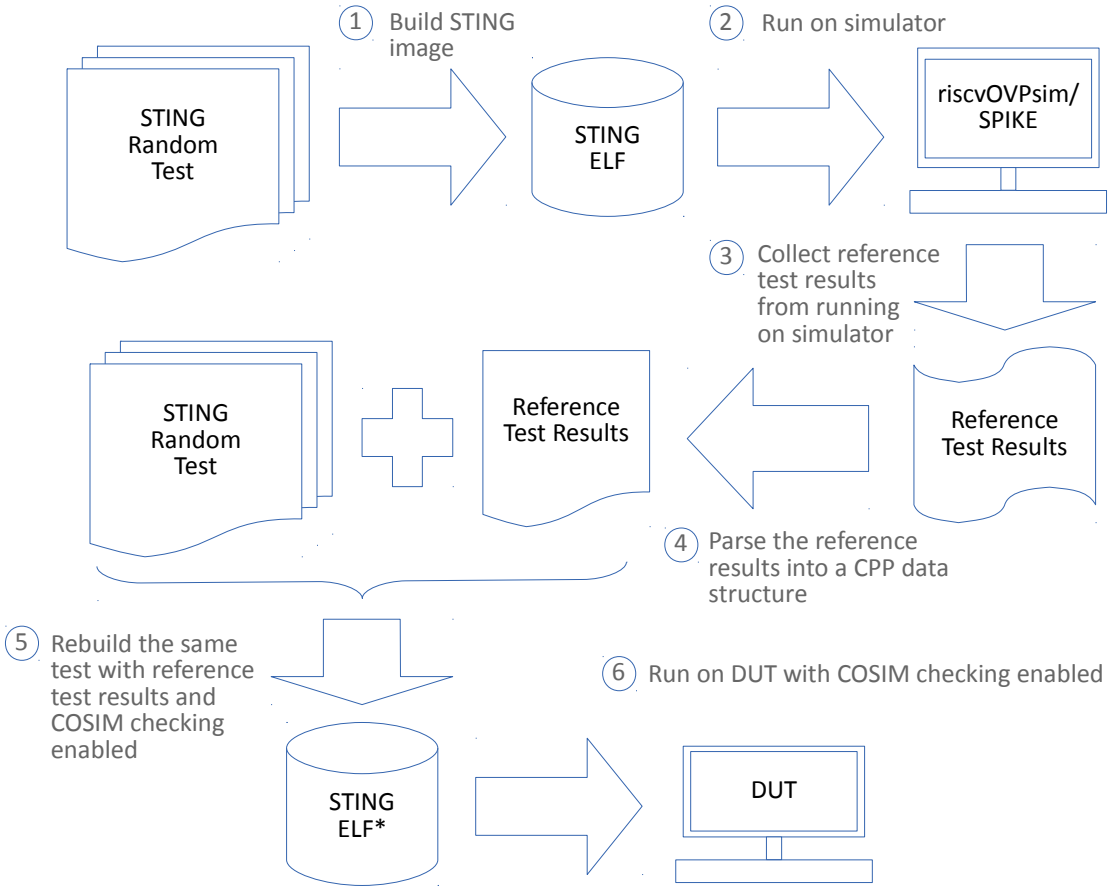
Simulated result

Comparison with DUT output checkpointed at an earlier point

Valtrix SYSTEMS

Random Tests with Checking

- Pattern based tests are great spot checks but tend to add lot of noise in random tests
- Random tests for increased coverage and interaction with other elements
- Correctness of test execution using cosim checking, multi pass results checking etc.



Valtrix SYSTEMS

Random Tests with Checking (contd)

```
8184    vlseg6bu.v v25, (x28) , v0.t
8185    vsb.v v11, (x20)
8186    lwu     x31, -1592(x15)
8187    vsb.v v8, (x18)
8188    vlsseg3b.v v20, (x28),x29, v0.t
8189    vsb.v v30, (x28) , v0.t
8190    vlseg2hff.v v24, (x15)
8191    vsb.v v8, (x20)
8192    enable_rvc "c.sw      x10, 36(x8)"
8193    vsb.v v9, (x18) , v0.t
8194    vlsseg6wu.v v22, (x28),x29, v0.t
8195    vsb.v v25, (x15)
8196    vlseg4hu.v v17, (x26)
8197    vsb.v v8, (x8)
8198    flw     f19, -1936(x28)
8199    vsb.v v30, (x28) , v0.t
8200    vfsgnj.vf v14, v11, f1
8201    vsb.v v29, (x27) , v0.t
8202    fsd     f15, -1992(x26)
8203    vsb.v v9, (x16) , v0.t
8204    amoxor.w.rl   x24, x24, (x26)
8205    vsb.v v16, (x15)
8206    vfnmsub.vv v17, v16, v16
8207    vsb.v v4, (x22) , v0.t
8208    vlxwu.v v6, (x20),v2, v0.t
8209    vsb.v v17, (x15)
8210    vssub.vv v19, v17, v16
8211    vsb.v v12, (x8)
8212    vlseg6hff.v v22, (x26)
8213    vsb.v v12, (x16)
8214    vlseg7e.v v7, (x16)
```

TEST – interleaved_vsb_v

```
2444    srlw   x7, x6, x7
45      vwsubu.vx v24, v30, x14
46      vlb.v v20, (x27) , v0.t
47      sb      x25, -1607(x28)
48      vwmul.vv v16, v30, v31
49      vlseg3b.v v10, (x20)
50      vwmul.vv v28, v19, v17
51      vssseg4w.v v19, (x27),x29, v0.t
52      fdiv.s   f17, f22, f13, rdn
53      enable_rvc "c.fld     f12, 88(x15)"
54      fclass.d     x12, f5
55      enable_rvc "c.sd      x11, 104(x8)"
56      vamoxore.v  v17, v3, (x27), v17
57      vlseg5bu.v v4, (x8)
58      sw      x31, 1792(x26)
59      vwmul.vv v30, v27, v21
60      vlseg2hu.v v7, (x16) , v0.t
61      enable_rvc "c.lw      x14, 24(x15)"
62      vsb.v v13, (x18)
63      vwmul.vv v14, v4, v12
64      vwmul.vv v14, v12, v10
65      srliw x11, x10, 22
66      vwmul.vv v28, v21, v18
67      vwmaccsu.vv v10, v7, v5
68      vwmul.vv v24, v27, v21
69      vlhuff.v v10, (x18)
70      vslideup.vi v10, v4, 5
71      vmulhsu.vx v28, v30, x31
2472    vlseg4eff.v v22, (x28) , v0.t
        vwmul.vv v18, v26, v21
```

TEST – more_vwmul_vv

```
791     vlseg4b.v v10, (x16) , v0.t
792     vlseg4b.v v4, (x22)
793     vlseg4b.v v25, (x28) , v0.t
794     vlseg4b.v v10, (x22)
795     vlseg4b.v v11, (x16)
796     vlseg4b.v v5, (x16)
797     vlseg4b.v v19, (x27) , v0.t
798     vlseg4b.v v4, (x22) , v0.t
799     vlseg4b.v v4, (x20)
800     vlseg4b.v v17, (x28) , v0.t
801     vlseg4b.v v20, (x26)
802     vlseg4b.v v27, (x28) , v0.t
803     vlseg4b.v v22, (x26)
804     vlseg4b.v v16, (x26)
805     vlseg4b.v v28, (x28) , v0.t
806     vlseg4b.v v10, (x8)
807     vlseg4b.v v11, (x22) , v0.t
808     vlseg4b.v v6, (x18)
809     vlseg4b.v v8, (x20) , v0.t
810     vlseg4b.v v7, (x8)
```

TEST – only_vlseg4b_v

Valtrix SYSTEMS

Miscellaneous Tests

- Snippets developed for a large number of directed scenarios to check compliance with the specification

  - Vector register overlap for widening/narrowing instructions and LMUL > 1
  - Verifying if current SEW or LMUL is legal for the executing instruction

- Cross products with existing configurations for PMA, PMP, virtual memory, privileged mode of execution

- Configurations to generate different biased random sequences involving vector and bit manipulation instructions

Valtrix
SYSTEMS

Results/Bugs Found

- Register overlap check failure for LMUL > 1
- Unexpected load/store address misaligned exceptions on randomizing SEW
- Incorrect decodes of several vector and bit-manipulation instructions
- No trap exception on executing vector atomics with unsupported SEW setting
- Unexpected VL setting at the end of vector fault-only-first load instructions
- Violation of rules in setting VL due to randomization of SEW and AVL
- Incorrect results with vector arithmetic operations
- Irregularities in sign extension of result of vector atomic operations
- Unexpected output of saturating arithmetic instructions on randomization of vector rounding mode (VXRM)
- VXRM/VXSAT not part of FCSR

Valtrix
SYSTEMS

Future Work

- Add support for the changes introduced by newer revisions of specification
- Improved LMUL randomization
- Handle different random combinations of striping length (SLEN)
- Enhance STING coverage manager for vector and bit manipulation instructions
- Tests for Vector EDIV extension

Valtrix
SYSTEMS