

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

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

# Agenda

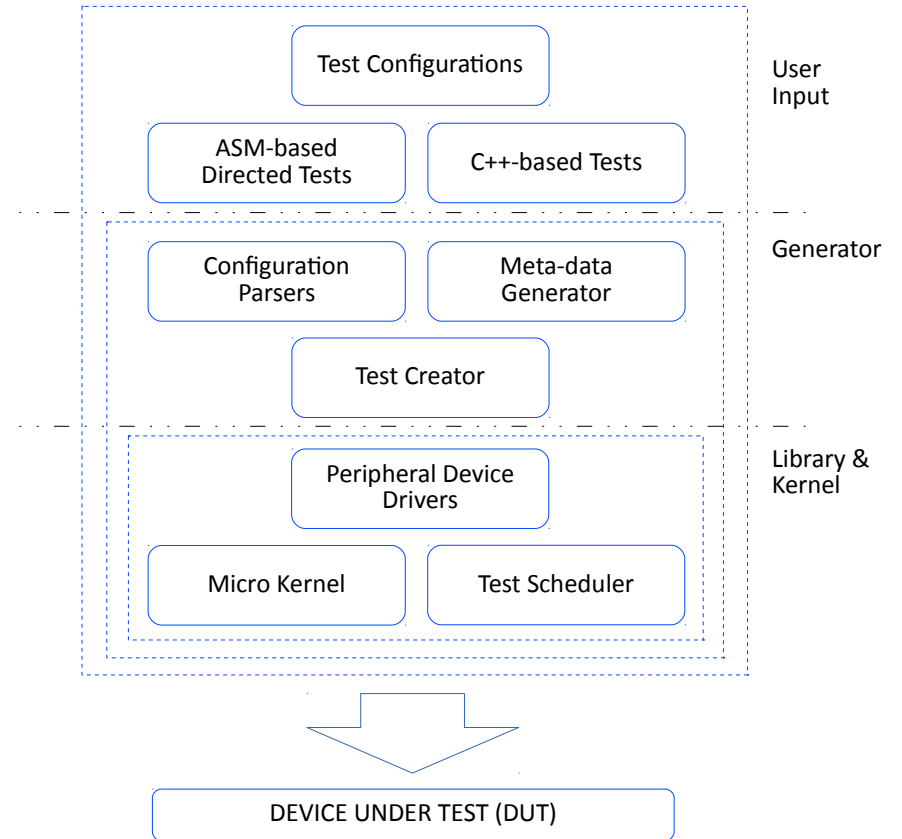
- Vector and bit manipulation support – Where we are?
- Overview of STING
- Verification strategy
- Results/Bugs Found
- Future work

# 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

# 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



# 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

# Rendering of Snippets

- 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

```
21 resource
22 cpu_t x
23 base_t rbl[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 (rbl) ; 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 (rbl)
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 (rbl)
52 blt rx1, rx2, do_branch
53
54
55 snip_check
56 ; Empty
```

```
1769 fsgnfn.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)
```

```
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)
```

# 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.

# 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

```
1 #####
2 #
3 # VALTRIX TECHNOLOGIES PVT. LTD. CONFIDENTIAL
4 #
5 #
6 # [2016] - Valtrix Technologies Private Limited
7 # All Rights Reserved.
8 #
9 # NOTICE: All information contained herein
10 # property of Valtrix Technologies Pvt. Ltd.
11 # and technical concepts contained herein
12 # Valtrix Technologies Pvt. Ltd. and may be
13 # U.S. and Foreign Patents, patents in process
14 # by trade secret or copyright law. Dis
15 # information or reproduction of this ma
16 # forbidden unless prior written permis
17 # Valtrix Technologies Pvt. Ltd.
18 #
19 #####
20
21 # The dictionary given below defines the datab
22 # by the instructions defined under the bit m
23 #
24 # SRC_1 contains the data to be loaded into t
25 # an instruction supports another source oper
26 # field defined in the dictionary. The expect
27 # be specified using EXPECTED_RESULT field
28
29 ISA_CLZ_CONF = {
30
31     DATA_SET : [
32         { SRC_1 : 0xFFFFFFFFFFFFFFFFUL
33           EXP
34         { SRC_1 : 0x0000000000000000UL
35           EXP
36         { SRC_1 : 0x0100000000000000UL
37           EXP
38         { SRC_1 : 0x0001000000000000UL
39           EXP
40         { SRC_1 : 0x0000100000000000UL
41           EXP
42         { SRC_1 : 0x0000010000000000UL
43           EXP
44         { SRC_1 : 0x0000001000000000UL
45           EXP
46         { SRC_1 : 0x0000000100000000UL
47           EXP
48         { SRC_1 : 0x0000000010000000UL
49           EXP
50         { SRC_1 : 0x0000000001000000UL
51           EXP
52         { SRC_1 : 0x0000000000100000UL
53           EXP
54         { SRC_1 : 0x0000000000010000UL
55           EXP
56         { SRC_1 : 0x0000000000001000UL
57           EXP
58         { SRC_1 : 0x0000000000000100UL
59           EXP
60         { SRC_1 : 0x0000000000000010UL
61           EXP
62         { SRC_1 : 0x0000000000000001UL
63           EXP
64         { SRC_1 : 0x0000000000000000UL
65           EXP
66     ]
67
68     EXPECTED_RESULT : [
69         { SRC_1 : 0xFFFFFFFFFFFFFFFFUL
70           EXP
71         { SRC_1 : 0x0000000000000000UL
72           EXP
73         { SRC_1 : 0x0100000000000000UL
74           EXP
75         { SRC_1 : 0x0001000000000000UL
76           EXP
77         { SRC_1 : 0x0000100000000000UL
78           EXP
79         { SRC_1 : 0x0000010000000000UL
80           EXP
81         { SRC_1 : 0x0000001000000000UL
82           EXP
83         { SRC_1 : 0x0000000100000000UL
84           EXP
85         { SRC_1 : 0x0000000010000000UL
86           EXP
87         { SRC_1 : 0x0000000001000000UL
88           EXP
89         { SRC_1 : 0x0000000000100000UL
90           EXP
91         { SRC_1 : 0x0000000000010000UL
92           EXP
93         { SRC_1 : 0x0000000000001000UL
94           EXP
95         { SRC_1 : 0x0000000000000100UL
96           EXP
97         { SRC_1 : 0x0000000000000010UL
98           EXP
99         { SRC_1 : 0x0000000000000001UL
100          EXP
101         { SRC_1 : 0x0000000000000000UL
102          EXP
103     ]
104
105     }
106
107     }
108
109     }
110
111     }
112
113     }
114
115     }
116
117     }
118
119     }
120
121     }
122
123     }
124
125     }
126
127     }
128
129     }
130
131     }
132
133     }
134
135     }
136
137     }
138
139     }
140
141     }
142
143     }
144
145     }
146
147     }
148
149     }
150
151     }
152
153     }
154
155     }
156
157     }
158
159     }
160
161     }
162
163     }
164
165     }
166
167     }
168
169     }
170
171     }
172
173     }
174
175     }
176
177     }
178
179     }
180
181     }
182
183     }
184
185     }
186
187     }
188
189     }
190
191     }
192
193     }
194
195     }
196
197     }
198
199     }
200
201     }
202
203     }
204
205     }
206
207     }
208
209     }
210
211     }
212
213     }
214
215     }
216
217     }
218
219     }
220
221     }
222
223     }
224
225     }
226
227     }
228
229     }
230
231     }
232
233     }
234
235     }
236
237     }
238
239     }
240
241     }
242
243     }
244
245     }
246
247     }
248
249     }
250
251     }
252
253     }
254
255     }
256
257     }
258
259     }
260
261     }
262
263     }
264
265     }
266
267     }
268
269     }
270
271     }
272
273     }
274
275     }
276
277     }
278
279     }
280
281     }
282
283     }
284
285     }
286
287     }
288
289     }
290
291     }
292
293     }
294
295     }
296
297     }
298
299     }
300
301     }
302
303     }
304
305     }
306
307     }
308
309     }
310
311     }
312
313     }
314
315     }
316
317     }
318
319     }
320
321     }
322
323     }
324
325     }
326
327     }
328
329     }
330
331     }
332
333     }
334
335     }
336
337     }
338
339     }
340
341     }
342
343     }
344
345     }
346
347     }
348
349     }
350
351     }
352
353     }
354
355     }
356
357     }
358
359     }
360
361     }
362
363     }
364
365     }
366
367     }
368
369     }
369
370     }
371
372     }
373
374     }
375
376     }
377
378     }
379
380     }
381
382     }
383
384     }
385
386     }
387
388     }
389
390     }
391
392     }
393
394     }
395
396     }
397
398     }
399
400     }
401
402     }
403
404     }
405
406     }
407
408     }
409
410     }
411
412     }
413
414     }
415
416     }
417
418     }
419
420     }
421
422     }
423
424     }
425
426     }
427
428     }
429
430     }
431
432     }
433
434     }
435
436     }
437
438     }
439
440     }
441
442     }
443
444     }
445
446     }
447
448     }
449
450     }
451
452     }
453
454     }
455
456     }
457
458     }
459
460     }
461
462     }
463
464     }
465
466     }
467
468     }
469
470     }
471
472     }
473
474     }
475
476     }
477
478     }
479
480     }
481
482     }
483
484     }
485
486     }
487
488     }
489
490     }
491
492     }
493
494     }
495
496     }
497
498     }
499
500     }
501
502     }
503
504     }
505
506     }
507
508     }
509
510     }
511
512     }
513
514     }
515
516     }
517
518     }
519
520     }
521
522     }
523
524     }
525
526     }
527
528     }
529
530     }
531
532     }
533
534     }
535
536     }
537
538     }
539
540     }
541
542     }
543
544     }
545
546     }
547
548     }
549
550     }
551
552     }
553
554     }
555
556     }
557
558     }
559
560     }
561
562     }
563
564     }
565
566     }
567
568     }
569
570     }
571
572     }
573
574     }
575
576     }
577
578     }
579
580     }
581
582     }
583
584     }
585
586     }
587
588     }
589
590     }
591
592     }
593
594     }
595
596     }
597
598     }
599
600     }
601
602     }
603
604     }
605
606     }
607
608     }
609
610     }
611
612     }
613
614     }
615
616     }
617
618     }
619
620     }
621
622     }
623
624     }
625
626     }
627
628     }
629
630     }
631
632     }
633
634     }
635
636     }
637
638     }
639
640     }
641
642     }
643
644     }
645
646     }
647
648     }
649
650     }
651
652     }
653
654     }
655
656     }
657
658     }
659
660     }
661
662     }
663
664     }
665
666     }
667
668     }
669
670     }
671
672     }
673
674     }
675
676     }
677
678     }
679
680     }
681
682     }
683
684     }
685
686     }
687
688     }
689
690     }
691
692     }
693
694     }
695
696     }
697
698     }
699
700     }
701
702     }
703
704     }
705
706     }
707
708     }
709
710     }
711
712     }
713
714     }
715
716     }
717
718     }
719
720     }
721
722     }
723
724     }
725
726     }
727
728     }
729
730     }
731
732     }
733
734     }
735
736     }
737
738     }
739
740     }
741
742     }
743
744     }
745
746     }
747
748     }
749
750     }
751
752     }
753
754     }
755
756     }
757
758     }
759
760     }
761
762     }
763
764     }
765
766     }
767
768     }
769
770     }
771
772     }
773
774     }
775
776     }
777
778     }
779
780     }
781
782     }
783
784     }
785
786     }
787
788     }
789
790     }
791
792     }
793
794     }
795
796     }
797
798     }
799
800     }
801
802     }
803
804     }
805
806     }
807
808     }
809
810     }
811
812     }
813
814     }
815
816     }
817
818     }
819
820     }
821
822     }
823
824     }
825
826     }
827
828     }
829
830     }
831
832     }
833
834     }
835
836     }
837
838     }
839
840     }
841
842     }
843
844     }
845
846     }
847
848     }
849
850     }
851
852     }
853
854     }
855
856     }
857
858     }
859
860     }
861
862     }
863
864     }
865
866     }
867
868     }
869
870     }
871
872     }
873
874     }
875
876     }
877
878     }
879
880     }
881
882     }
883
884     }
885
886     }
887
888     }
889
890     }
891
892     }
893
894     }
895
896     }
897
898     }
899
900     }
901
902     }
903
904     }
905
906     }
907
908     }
909
910     }
911
912     }
913
914     }
915
916     }
917
918     }
919
920     }
921
922     }
923
924     }
925
926     }
927
928     }
929
930     }
931
932     }
933
934     }
935
936     }
937
938     }
939
940     }
941
942     }
943
944     }
945
946     }
947
948     }
949
950     }
951
952     }
953
954     }
955
956     }
957
958     }
959
960     }
961
962     }
963
964     }
965
966     }
967
968     }
969
970     }
971
972     }
973
974     }
975
976     }
977
978     }
979
980     }
981
982     }
983
984     }
985
986     }
987
988     }
989
990     }
991
992     }
993
994     }
995
996     }
997
998     }
999
1000    }
```



# 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 checkpointed and compared with the expected output
- Error reported in case a failure is detected during comparison

```
19 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
20
21 ; Fixed pattern directed test for RV32B CLZ instruction
22
23
24 import "src/snippets/rv/rv_param_blocks.pb"
25
26 resource
27   cpu_t   x
28   base_t  rbl[8]@8
29   axreg_t rx1, rx2
30   unum_t  rno = snp_urandrange(0, ISA_CLZ_CONF.NUM_DATA_SET)
31
32
33 snip_setup
34   ; Empty
35
36
37 snip_init
38 @cpu: x
39   write_reg_to_mem(reg

43 snip_run
44
45 @cpu: x
46   construct_inst_reg(inst = clz, imm = ISA_CLZ_CONF.DATA_SET[rno].SRC_1,
47                       regx = rx1,
48                       regy = rx2,
49                       regb = rbl)
50
51
52 snip_check
53
54 @cpu: x
55   check_mem(val = ISA_CLZ_CONF.DATA_SET[rno].EXPECTED_RESULT, regx = rx1,
56            regy = rx2,
57            regb = rbl)

43 snip_run
44
45 @cpu: x
46   for index in (0, ISA_CLZ_CONF.NUM_DATA_SET):
47     construct_inst_reg(inst = clz, imm = ISA_CLZ_CONF.DATA_SET[index].SRC_1,
48                       regx = rx1,
49                       regy = rx2,
50                       regb = rbl)
51
52     check_mem(val = ISA_CLZ_CONF.DATA_SET[index].EXPECTED_RESULT, regx = rx1,
53             regy = rx2,
54             regb = rbl)
55
56   endfor
```

# 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

# 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)
```

# 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
```

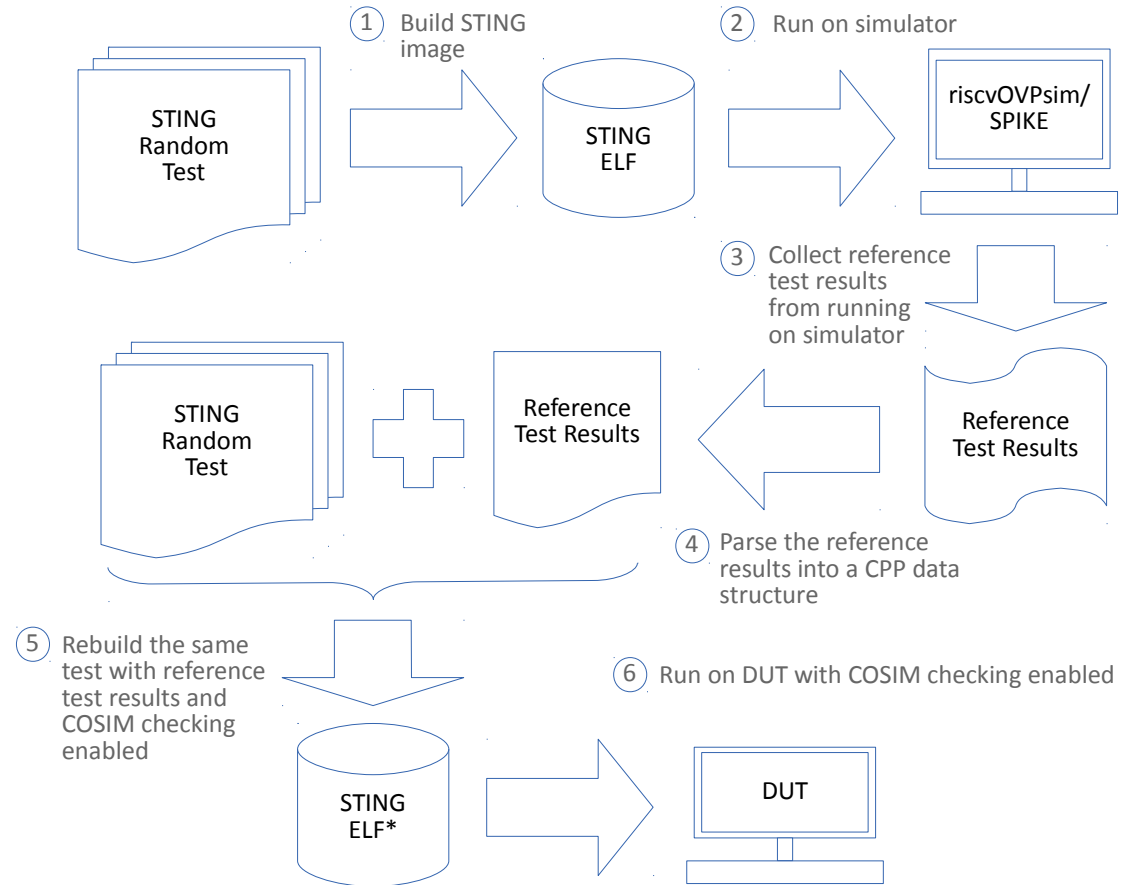
1. Generation of random input
2. Instruction execution
3. Checkpointing of output

Simulated result

Comparison with DUT output  
checkpointed at an earlier point

# 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.



# 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 vfgsnj.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 vwsu.v 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 srlw 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
72 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

# 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

# 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



# 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