AZADI Verification Plan

S. No.	Testing feature	Test name covering the mentioned feature	Instruction generator options
		These are constrained random tests, generating instructions on a random seed	
1)	Arithmetic instruction, no load/store/branch instructions	riscv_arithmetic_basic_test	+instr_cnt=10000 +num_of_sub_program=0 +no_fence=1 +no_data_page=1 +no_branch_jump=1 +boot_mode=m
2)	Enable floating point instructions	riscv_floating_point_arithmetic_test	+instr_cnt=10000 +num_of_sub_program=0 +no_fence=1 +no_data_page=1 +no_branch_jump=1 +enable_floating_point=1 +boot_mode=m
3)	Machine mode random instruction test	riscv_machine_mode_rand_test	+instr_cnt=10000 +num_of_sub_program=5 +boot_mode=m
4)	Random instruction stress test	riscv_rand_instr_test	+instr_cnt=10000 +num_of_sub_program=5
5)	Enable floating point instructions **a)** A random mix of load/store instructions and other instructions (riscv_load_store_rand_instr_stream) **b)** Loop test (riscv_loop_instr) **c)** test data TLB switch and replacement (riscv_multi_page_load_store_instr_stream) **d)** Access the different locations of the same memory regions **e)** Stress back to back jump instruction	riscv_floating_point_rand_test	+enable_floating_point=1 +instr_cnt=10000 +num_of_sub_program=5 +directed_instr_0=riscv_load_store_rand_instr_stream,4 +directed_instr_1=riscv_loop_instr,4 +directed_instr_2=riscv_multi_page_load_store_instr_stream,4 +directed_instr_3=riscv_mem_region_stress_test,4 +directed_instr_4=riscv_jal_instr,4
6)	Jump among large number of sub-programs, stress testing iTLB operations. **a) A random mix of load/store instructions and other instructions**	riscv_rand_jump_test	+instr_cnt=15000 +num_of_sub_program=20 +directed_instr_0=riscv_load_store_rand_instr_stream,8
7)	**Stress back to back jump instruction**	riscv_jump_stress_test	+instr_cnt=5000 +num_of_sub_program=5 +directed_instr_1=riscv_jal_instr,20
8)	**Loop test**	riscv_loop_test	+instr_cnt=10000 +num_of_sub_program=5 +directed_instr_1=riscv_loop_instr,20
9)	Test with different patterns of load/store instructions, stress test MMU operations. **a)** A random mix of load/store instructions and other instructions (riscv_load_store_rand_instr_stream) **b)** hazard handling of load store unit.(riscv_load_store_hazard_instr_stream) **c) Test data TLB switch and replacement (riscv_multi_page_load_store_instr_stream)** **d)** Random load/store sequence to full address range (riscv_load_store_rand_addr_instr_stream) - The address range is not preloaded with data pages, use store instruction to initialize first	riscv_mmu_stress_test	+instr_cnt=10000 +num_of_sub_program=5 +directed_instr_0=riscv_load_store_rand_instr_stream,40 +directed_instr_1=riscv_load_store_hazard_instr_stream,40 +directed_instr_2=riscv_multi_page_load_store_instr_stream,40 +directed_instr_3=riscv_load_store_rand_addr_instr_stream,40
10)	Illegal instruction test, verify the processor can detect illegal instruction and handle corresponding exception properly. An exception handling routine is designed to resume execution after illegal instruction exception.	riscv_illegal_instr_test	+illegal_instr_ratio=25
11)	HINT instruction test, verify the processor can detect HINT instruction treat it as NOP. No illegal instruction exception is expected	riscv_hint_instr_test	+hint_instr_ratio=5
12)	Random instruction test with ebreak instruction enabled. Debug mode is not enabled for this test, processor should raise ebreak exception.	riscv_ebreak_test	+instr_cnt=6000 +no_ebreak=0
13)	Randomly assert debug_req_i, random instruction sequence in debug_rom section	riscv_debug_basic_test	+require_signature_addr=1 +gen_debug_section=1 +no_ebreak=1 +no_branch_jump=1 +instr_cnt=6000 +no_csr_instr=1 +no_fence=1 +num_of_sub_program=0 +randomize_csr=1
14)	Randomly assert debug_req_i more often, debug_rom is empty, with only a dret instruction	riscv_debug_stress_test	+require_signature_addr=1 +no_ebreak=1 +instr_cnt=6000 +no_csr_instr=1 +no_fence=1
15)	Randomly assert debug_req_i, insert branch instructions and subprograms into debug_rom to make core jump around within the debug_rom	riscv_debug_branch_jump_test	+require_signature_addr=1 +gen_debug_section=1 +no_ebreak=1 +instr_cnt=6000 +no_csr_instr=1 +no_fence=1 +num_of_sub_program=0 +num_debug_sub_program=2 +randomize_csr=1
16)	At a high level, this test checks that Ibex can correctly respond after receiving debug stimulus while every supported instruction is in its decoding stage. e.g. If the test detects a LUI instruction in the decoding stage, a debug request is generated and sent into Ibex. We never send a debug request if we see a LUI instruction again, as we don't want to send in debug requests for every instruction that is executed. **a)** A random mix of load/store instructions and other instructions (riscv_load_store_rand_instr_stream) **b)** b) hazard handling of load store unit.(riscv_load_store_hazard_instr_stream)	riscv_debug_instr_test	+require_signature_addr=1 +gen_debug_section=1 +randomize_csr=1 +no_csr_instr=0 +no_ebreak=1 +no_fence=0 +no_wfi=0 +directed_instr_0=riscv_load_store_rand_instr_stream,40 +directed_instr_1=riscv_load_store_hazard_instr_stream,40
17)	Assert debug_req while core is in WFI sleep state, should jump to debug mode	riscv_debug_wfi_test	+require_signature_addr=1 +gen_debug_section=1 +no_ebreak=1 +instr_cnt=6000 +no_csr_instr=1 +no_fence=1 +no_wfi=0 +randomize_csr=1 +num_of_sub_program=0
18)	Dret instructions will be inserted into generated code, ibex should treat these like illegal instructions.	riscv_dret_test	+require_signature_addr=1 +no_dret=0 +instr_cnt=6000
19)	A directed ebreak sequence will be inserted into the debug rom, upon encountering it, ibex should jump back to the beginning of debug mode. The sequence is designed to avoid an infinite loop.	riscv_debug_ebreak_test	+require_signature_addr=1 +gen_debug_section=1 +enable_ebreak_in_debug_rom=1 +no_csr_instr=1 +no_fence=1 +no_wfi=1 +no_ebreak=1 +instr_cnt=10000 +randomize_csr=1
20)	dcsr.ebreakm will be set at the beginning of the test upon the first entry into the debug rom. From then on, every ebreak instruction should cause debug mode to be entered.	riscv_debug_ebreakmu_test	+require_signature_addr=1 +gen_debug_section=1 +set_dcsr_ebreak=1 +no_ebreak=0 +no_csr_instr=1 +no_fence=1 +no_wfi=1 +instr_cnt=6000 +randomize_csr=1 +num_of_sub_program=0
21)	Inject debug stimulus during writes to xSTATUS and xIE	riscv_debug_csr_entry_test	+require_signature_addr=1 +gen_debug_section=1 +randomize_csr=1 +no_csr_instr=1 +enable_dummy_csr_write=1 +boot_mode=m
22)	Send interrupts while the core is executing in debug mode, should ignore everything	riscv_irq_in_debug_mode_test	+require_signature_addr=1 +gen_debug_section=1 +enable_interrupt=1 +randomize_csr=1 +no_csr_instr=1 +no_fence=1
23)	Send debug stimulus while core is in an interrupt handler	riscv_debug_in_irq_test	+require_signature_addr=1 +gen_debug_section=1 +enable_interrupt=1 +randomize_csr=1 +no_csr_instr=1 +no_fence=1
24)	Random instruction test with complete interrupt handling	riscv_single_interrupt_test	+instr_cnt=10000 +require_signature_addr=1 +enable_interrupt=1 +randomize_csr=1
25)	Random instruction test with complete interrupt handling	riscv_multiple_interrupt_test	+instr_cnt=10000 +require_signature_addr=1 +enable_interrupt=1 +randomize_csr=1
26)	Assert interrupt, and then assert another interrupt during the first irq_handler routine	riscv_nested_interrupt_test	+instr_cnt=10000 +require_signature_addr=1 +enable_interrupt=1 +enable_nested_interrupt=1 +randomize_csr=1 +no_csr_instr=1
27)	At a high level, this test checks that Ibex can correctly respond after receiving interrupt stimulus while every supported instruction is in its decoding stage. e.g. If the test detects a LUI instruction in the decoding stage, an interrupt is generated and sent into Ibex. We never send an interrupt if we see a LUI instruction again, as we don't want to send interrupts for every instruction that is executed. **a)** A random mix of load/store instructions and other instructions (riscv_load_store_rand_instr_stream) **b)** b) hazard handling of load store unit.(riscv_load_store_hazard_instr_stream)	riscv_interrupt_instr_test	+instr_cnt=6000 +require_signature_addr=1 +enable_interrupt=1 +enable_timer_irq=1 +randomize_csr=1 +no_csr_instr=0 +no_ebreak=1 +no_fence=0 +no_wfi=0 +directed_instr_0=riscv_load_store_rand_instr_stream,40 +directed_instr_1=riscv_load_store_hazard_instr_stream,40

28)	Inject interrupts after a encountering wfi instructions.	riscv_interrupt_wfi_test	+instr_cnt=6000 +require_signature_addr=1 +enable_interrupt=1 +randomize_csr=1 +no_wfi=0	
29)	Inject interrupts during dummy writes to xSTATUS and xIE	riscv_interrupt_csr_test	+require_signature_addr=1 +enable_interrupt=1 +randomize_csr=1 +enable_dummy_csr_write=1 +boot_mode=m	
30)	Normal random instruction test, but randomly insert instruction fetch or memory load/store errors	riscv_mem_error_test	+require_signature_addr=1 +instr_cnt=10000 +randomize_csr=1 +enable_unaligned_load_store=1	
31)	Dump performance counters at EOT for any analysis	riscv_perf_counter_test	+require_signature_addr=1 +instr_cnt=10000 +num_of_sub_program=5	
32)	Randomly assert debug_req_i, and set dcsr.step to make ibex execute one isntruction and then re-enter debug mode	riscv_debug_single_step_test	+require_signature_addr=1 +gen_debug_section=1 +no_ebreak=1 +no_branch_jump=0 +instr_cnt=10000 +no_csr_instr=1 +no_fence=1 +num_of_sub_program=2 +randomize_csr=1 +enable_debug_single_step=1	
33)	Randomly reset the core once in the middle of program execution	riscv_reset_test	+num_of_sub_program=5 +enable_unaligned_load_store=1 +directed_instr_0=riscv_load_store_rand_instr_stream,10	
34)	Random instruction test without compressed instructions	riscv_rv32im_instr_test	+disable_compressed_instr=1	
35)	User mode random instruction test	riscv_user_mode_rand_test	+instr_cnt=10000 +num_of_sub_program=5 +boot_mode=u	
36)	Set mstatus.tw and enable generation of WFI instructions. Boot into U-mode, upon encountering WFI instruction, core should trap to M-mode due to illegal instruction exception.	riscv_umode_tw_test	+require_signature_addr=1 +gen_debug_section=1 +no_ebreak=1 +instr_cnt=6000 +no_csr_instr=1 +no_fence=1 +no_wfi=0 +set_mstatus_tw=1 +randomize_csr=1 +num_of_sub_program=0 +boot_mode=u	
37)	Boot core into random priv mode and generate csr accesses to invalid CSRs (at a higher priv mode)	riscv_invalid_csr_test	+require_signature_addr=1 +instr_cnt=6000 +num_of_sub_program=0 +no_csr_instr=0 +enable_access_invalid_csr_level=1 +boot_mode=u	
38)	Test with different patterns of load/store instructions, stress test MMU operations.	riscv_floating_point_mmu_stress_test	+instr_cnt=5000 +num_of_sub_program=5 +enable_floating_point=1 +directed_instr_0=riscv_load_store_rand_instr_stream,40 +directed_instr_1=riscv_load_store_hazard_instr_stream,40 +directed_instr_2=riscv_multi_page_load_store_instr_stream,10 +directed_instr_3=riscv_mem_region_stress_test,10	
39)	riscv_illegal_csr_instruction_test	Enable illegal CSR instructions	+instr_cnt=3000 +num_of_sub_program=0 +no_fence=1 +no_data_page=1 +no_branch_jump=1 +enable_illegal_csr_instruction=1	fails
40)	riscv_misaligned_instr_test	Enable jumps to misaligned instruction addresses	+instr_cnt=3000 +num_of_sub_program=0 +no_fence=1 +no_data_page=1 +no_branch_jump=1 +enable_misaligned_instr=1	pass
41)	riscv_no_load_store_test	No load store instructions	+instr_cnt=3000 +num_of_sub_program=0 +no_load_store=1	pass
42)	riscv_fix_sp_test	Use SP as stack pointer	+instr_cnt=3000 +num_of_sub_program=0 +fix_sp=1	pass
43)	riscv_mstatus_mprv_test	Enable users to set mstatus.mprv to enable privilege checks on memory accesses.	+instr_cnt=3000 +num_of_sub_program=0 +set_mstatus_mprv=1	pass

Note: We can enable floating point in all 38 test so FPU instructions can be tested with other instruction as well.

Iteration