BREKER™ THE LEADER IN PORTABLE STIMULUS

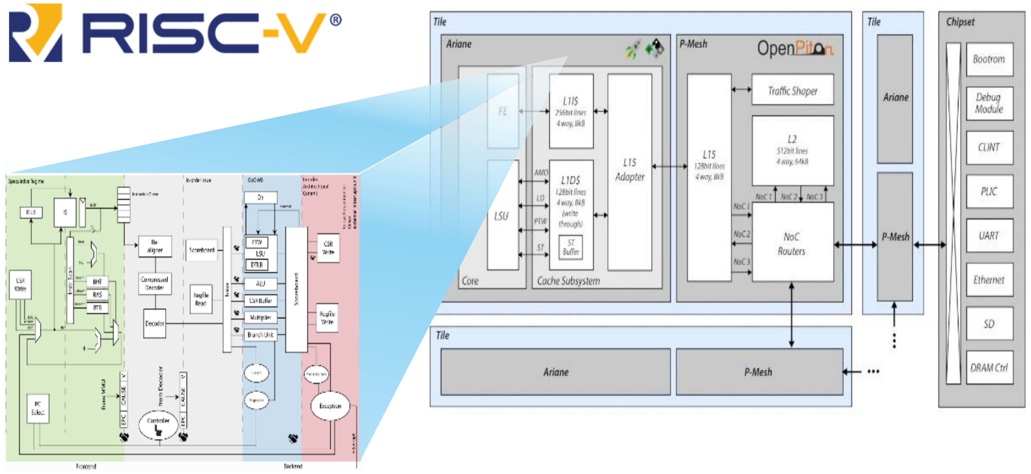
ALPINUM

A Layered, Graph-based Approach to RISC-V Verification

David Kelf, CEO, Breker Verification Systems

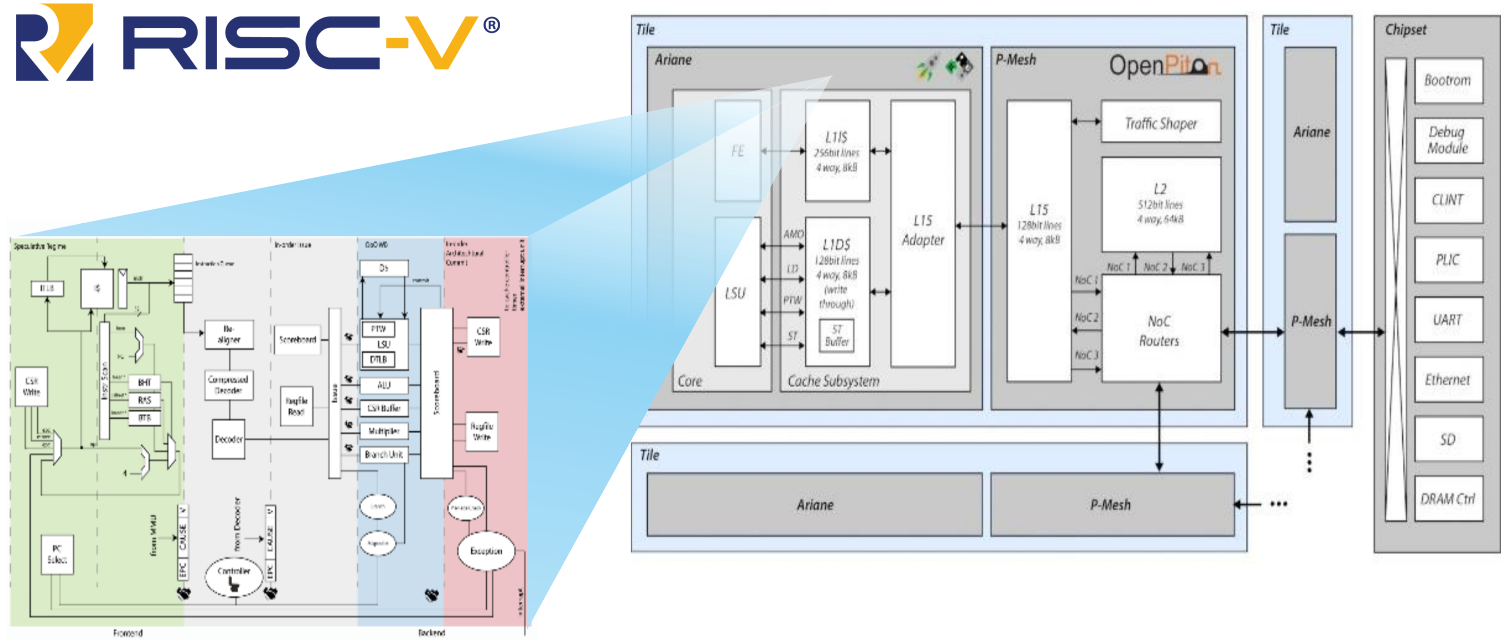
Processor Verification: An Alternative Verification Paradigm

- RISC-V Controllers are one thing, RISC-V Apps Processors entirely different
- Traditionally processors represent toughest verification challenges
  - E.G. ARM spends $150K+ per annum on verification, single core requires a quadrillion (10^{15}) cycles
  - Compounded by realities of RISC-V ISA environment
- Processor flexibility drives enormous, complex verification state search spaces
  - E.G. Opp code instruction sequences * varied arguments * varied values
  - Multitude of applications, SW options, compatibility requirements



Micro-Architectural Verification via Random Instructions Useful, But…

We must address system integrity
We must scale to a broader scenario combination
We must fully decipher the ISA

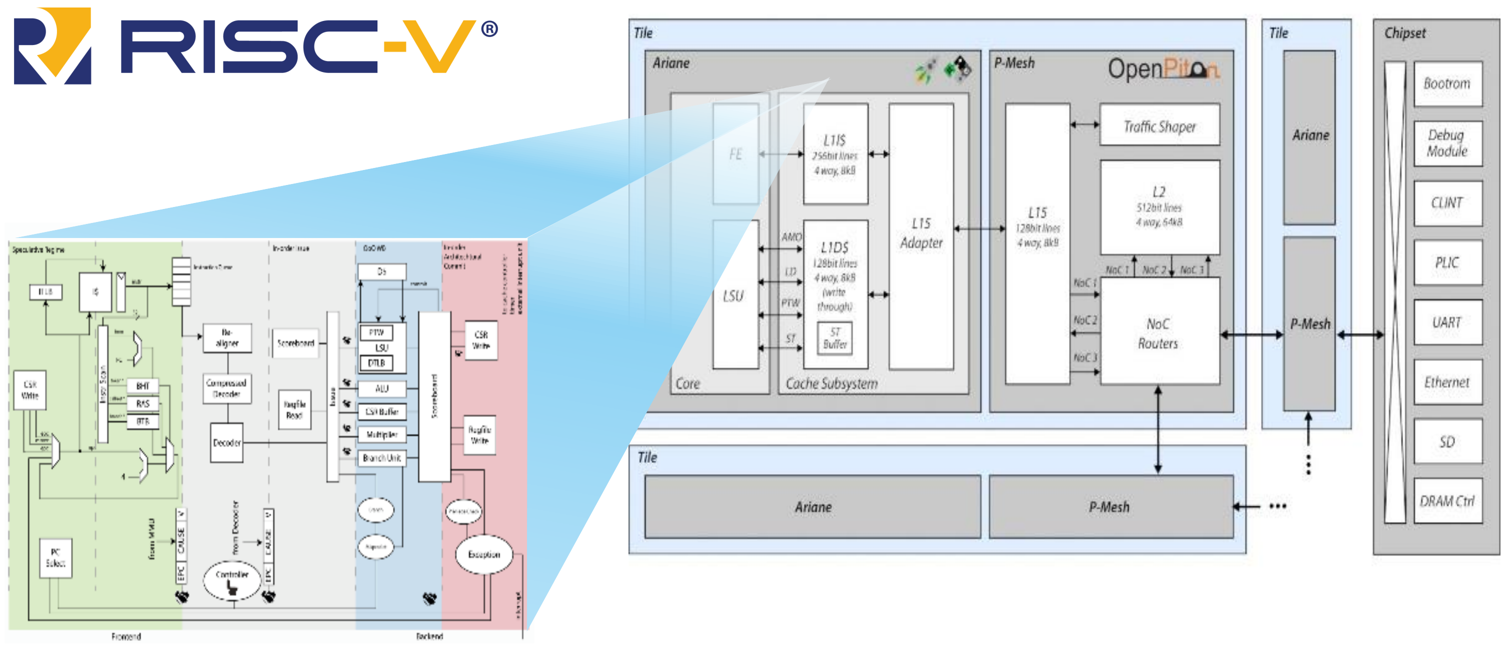


RISC-V Verification “Stack”: Layered System Integrity Verification

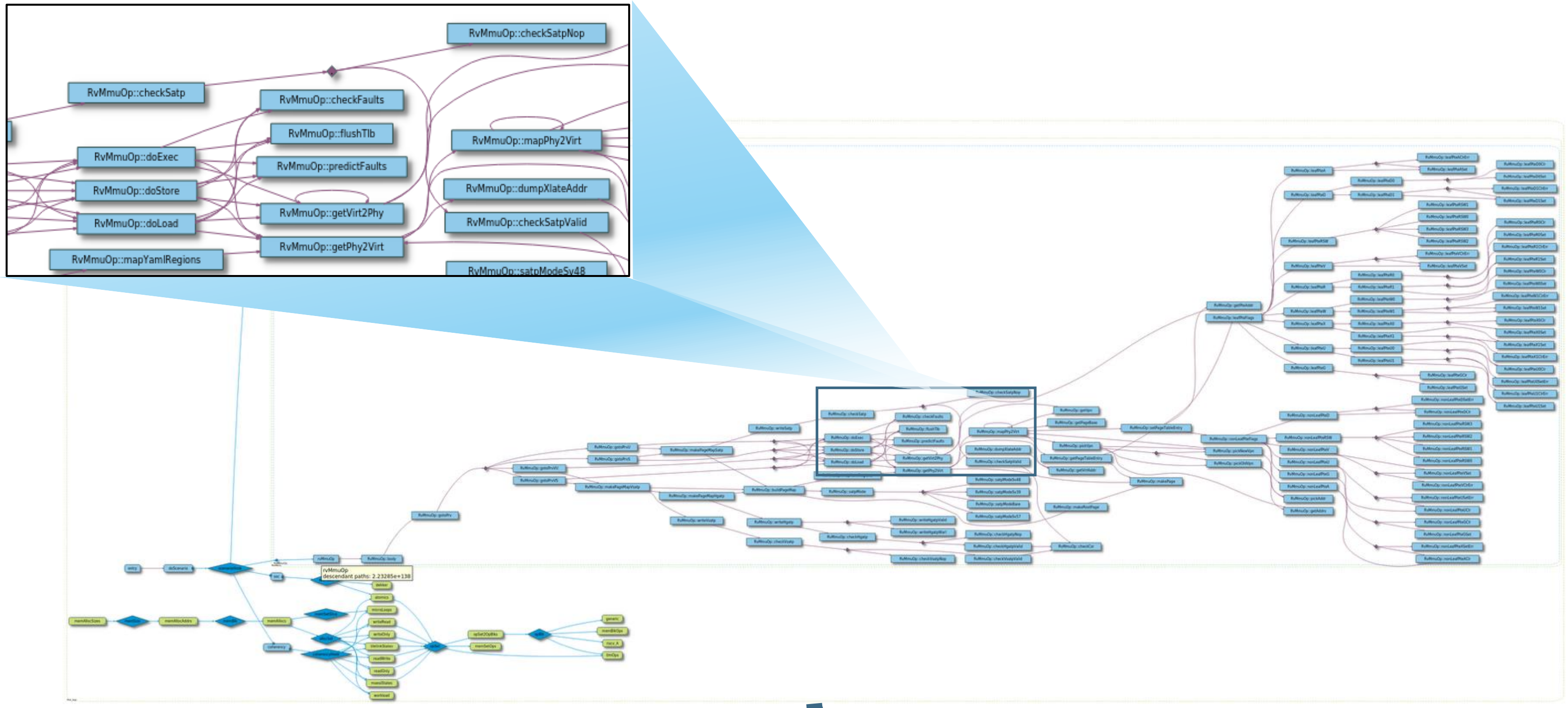
Suggested RISC-V verification “stack”

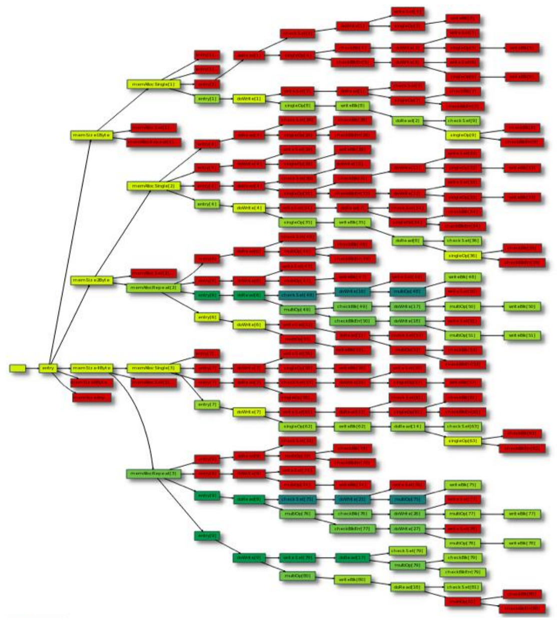
Complexity

- Performance/power profiling
- SW Execution, OS Boot
- System integration integrity
- Core operation integrity
- Micro-architecture functionality
- ISA compliance
- Up & running “Hello World”



Graph Based Model Captures System Scenarios





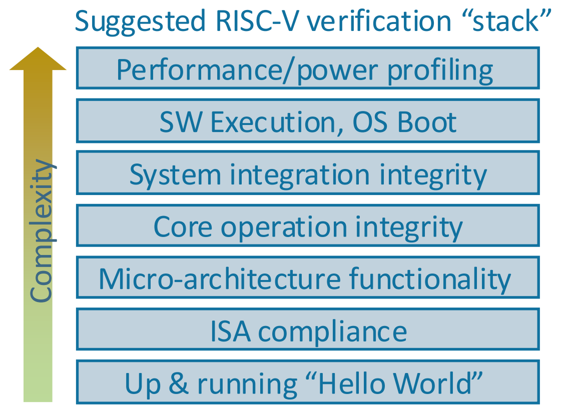
Walking graph into a tree, then crossing branches = high coverage

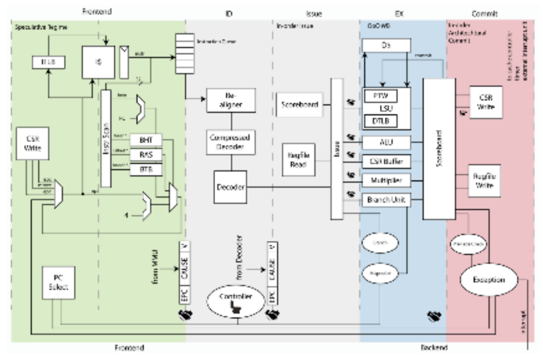
BREKER™

RISC-V Core-Assurance

RISC-V®

Random Instructions	Do instructions yield correct results
Register/Register Hazards	Pipeline perturbations dues to register conflicts
Load/Store Integrity	Memory conflict patterns
Conditionals and Branches	Pipeline perturbations from synchronous PC change
Exceptions	Jumping to and returning from ISR
Asynchronous Interrupts	Pipeline perturbations from asynchronous PC change
Privilege Level Switching	Context switching
Core Security	Register and Memory protection by privilege level
Core Paging/MMU	Memory virtualization and TLB operation
Sleep/Wakeup	State retention across WFI
Voltage/Freq Scaling	Operation at different clock ratios
Core Coherency	Caches, evictions and snoops


Suggested RISC-V verification "stack"
Complexity
Performance/power profiling
SW Execution, OS Boot
System integration integrity
Core operation integrity
Micro-architecture functionality
ISA compliance
Up & running "Hello World"

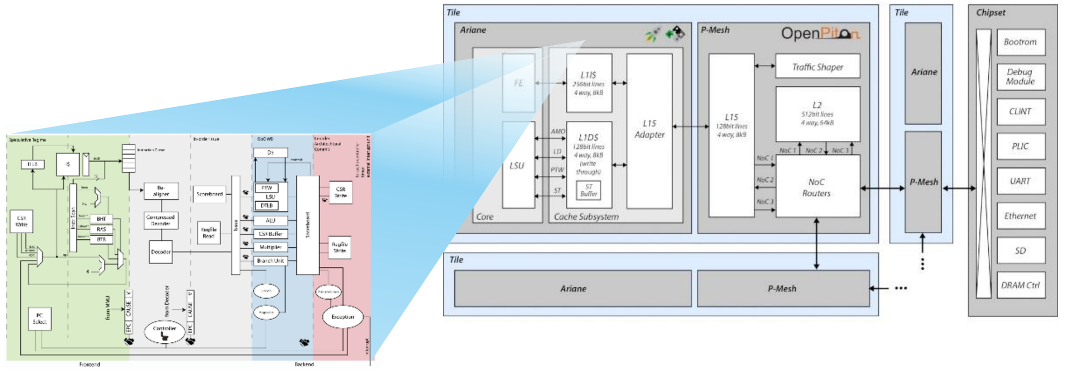


RISC-V SoC Integrity

Suggested RISC-V verification "stack"

Complexity

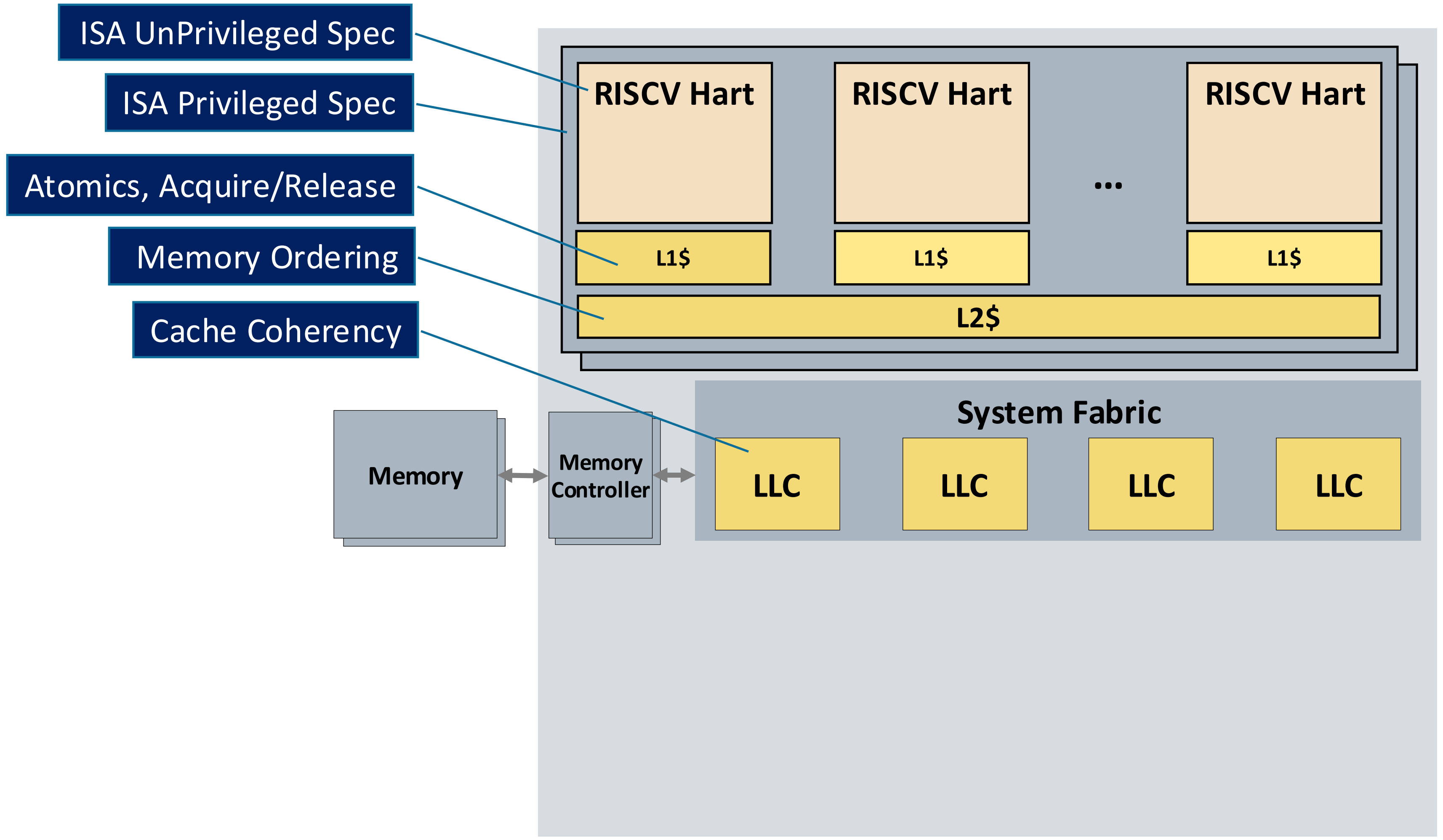
- Performance/power profiling
- SW Execution, OS Boot
- System integration integrity
- Core operation integrity
- Micro-architecture functionality
- ISA compliance
- Up & running "Hello World"



RISC-V SoC Integrity Functionality

Random Memory Tests	Test Cores/Fabrics/Memory controllers across DDR, OCRAM, FLASH, etc.
Random Register Tests	Read/write test to all uncore registers
System Interrupts	Randomized interrupts through CLINT
Multi-core Execution	Concurrent operations on fabric and memory
Memory Ordering	For weakly order memory protocols
Atomic Operation	Across all memory types
System Coherency	Cover all cache transitions, evictions, snoops
System Paging/IOMMU	System memory virtualization
System Security	Register and Memory protection across system
Power Management	System wide sleep/wakeup and voltage/freq scaling
Packet Generation	Generating networking packets for I/O testing
Interface Testing	Analyzing coherent interfaces including CXL & UCIe
SoC Profiling	Layering concurrent tests to check operation under stress
Firmware-First	Executing SW on block or sub-system without processor

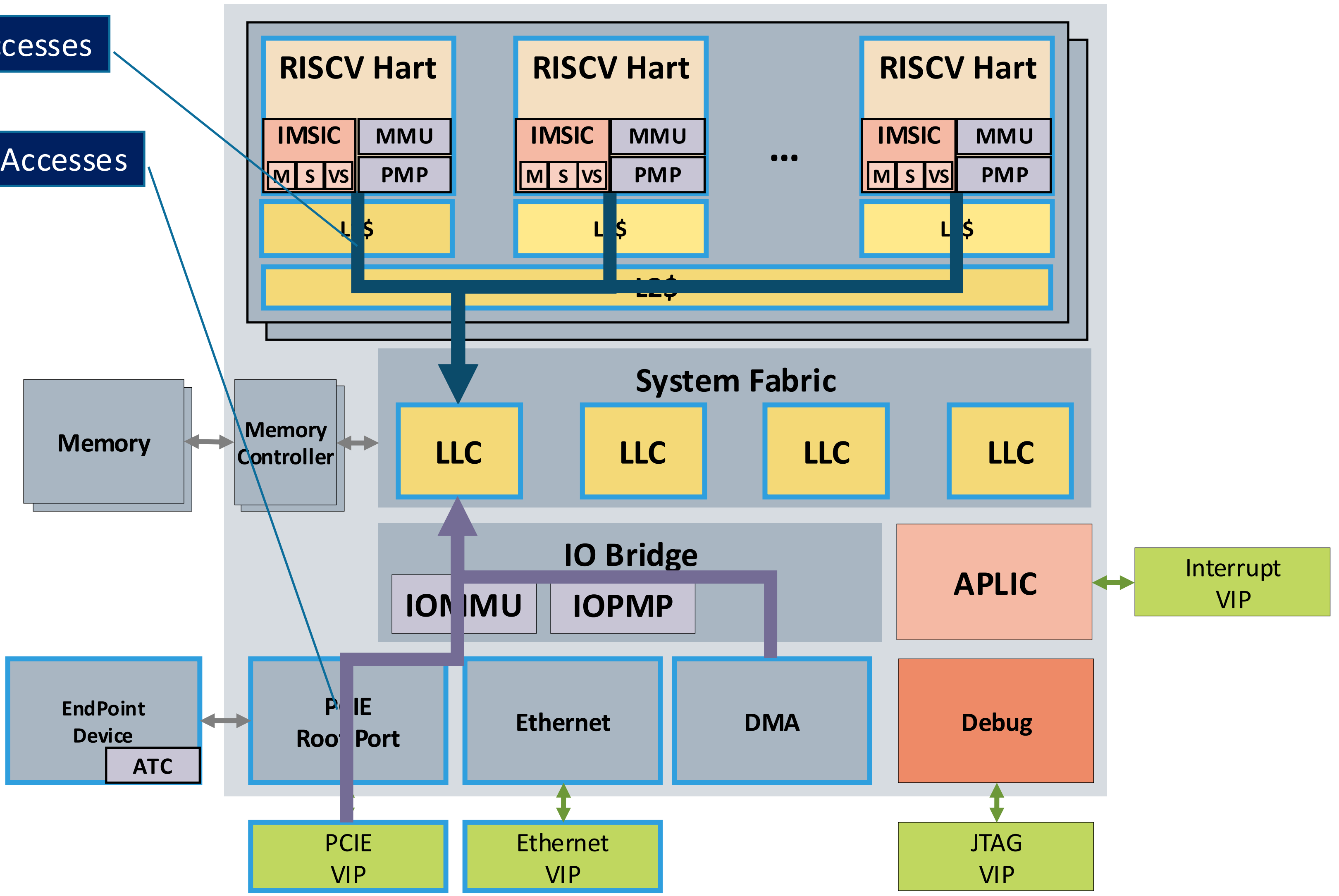
Atomics, Ordering & Cache Coherency: Example SoC Verification



Atomics, Ordering & Cache Coherency

Cache Coherent Concurrent Accesses

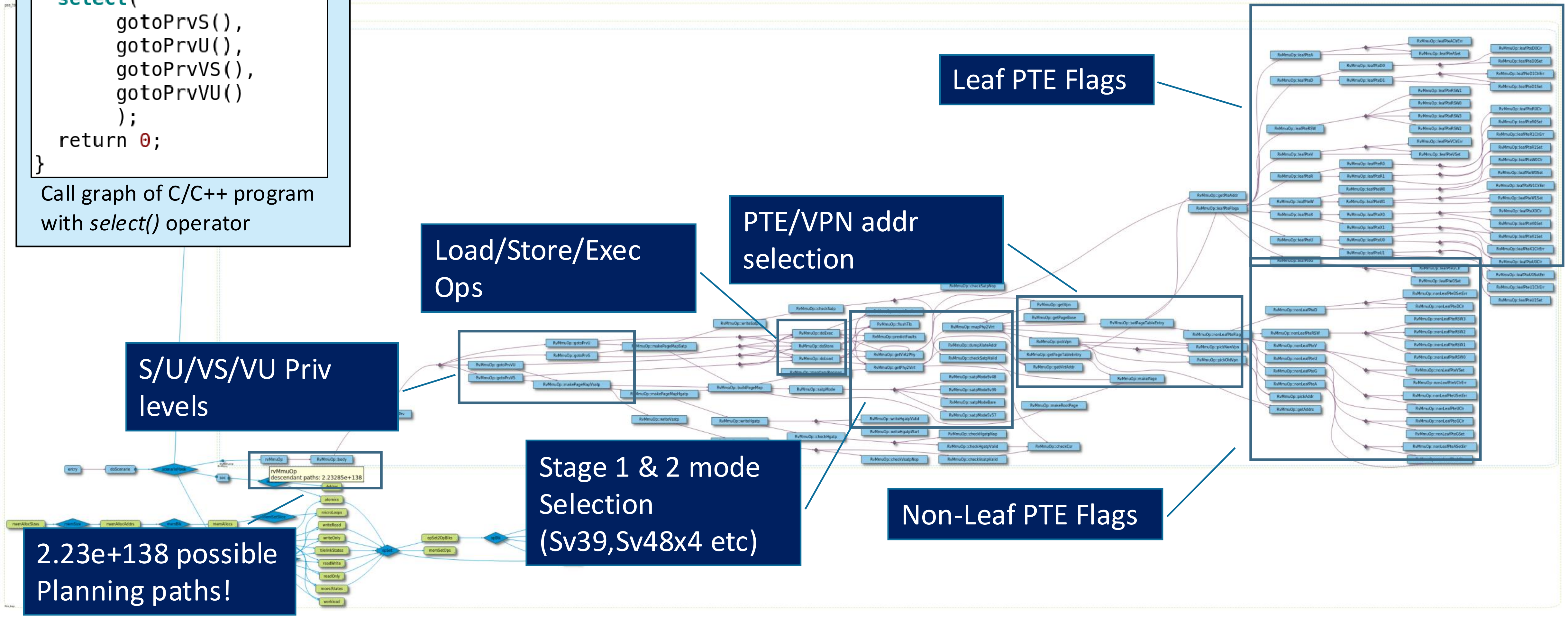
I/O Concurrent Accesses



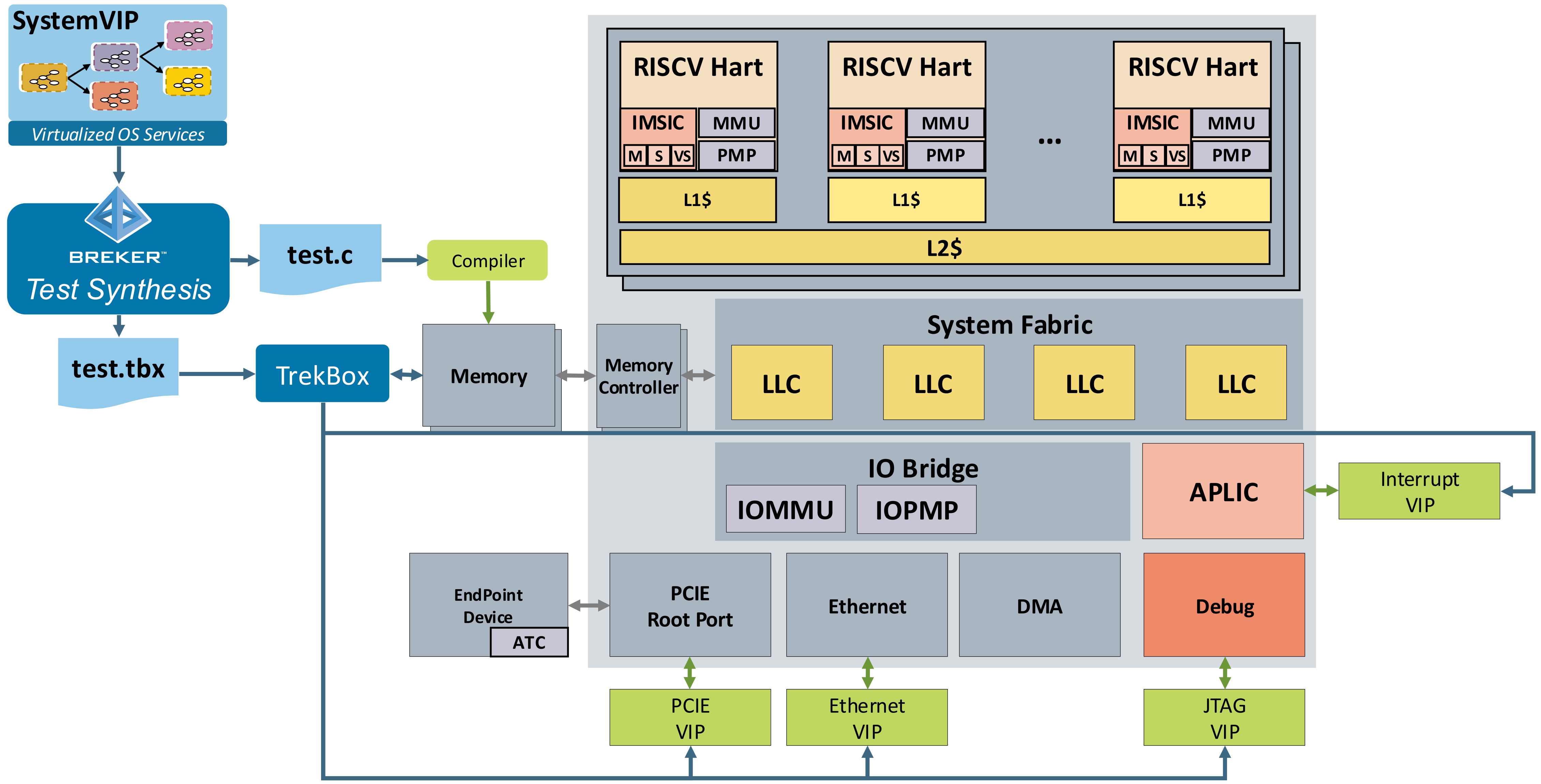
Graph Based Model: MMU & Hypervisor

```
int RvMmuOp::gotoPrv() {
  select(
       gotoPrvS(),
       gotoPrvU(),
       gotoPrvVS(),
       gotoPrvVU()
       );
  return 0;
}
```

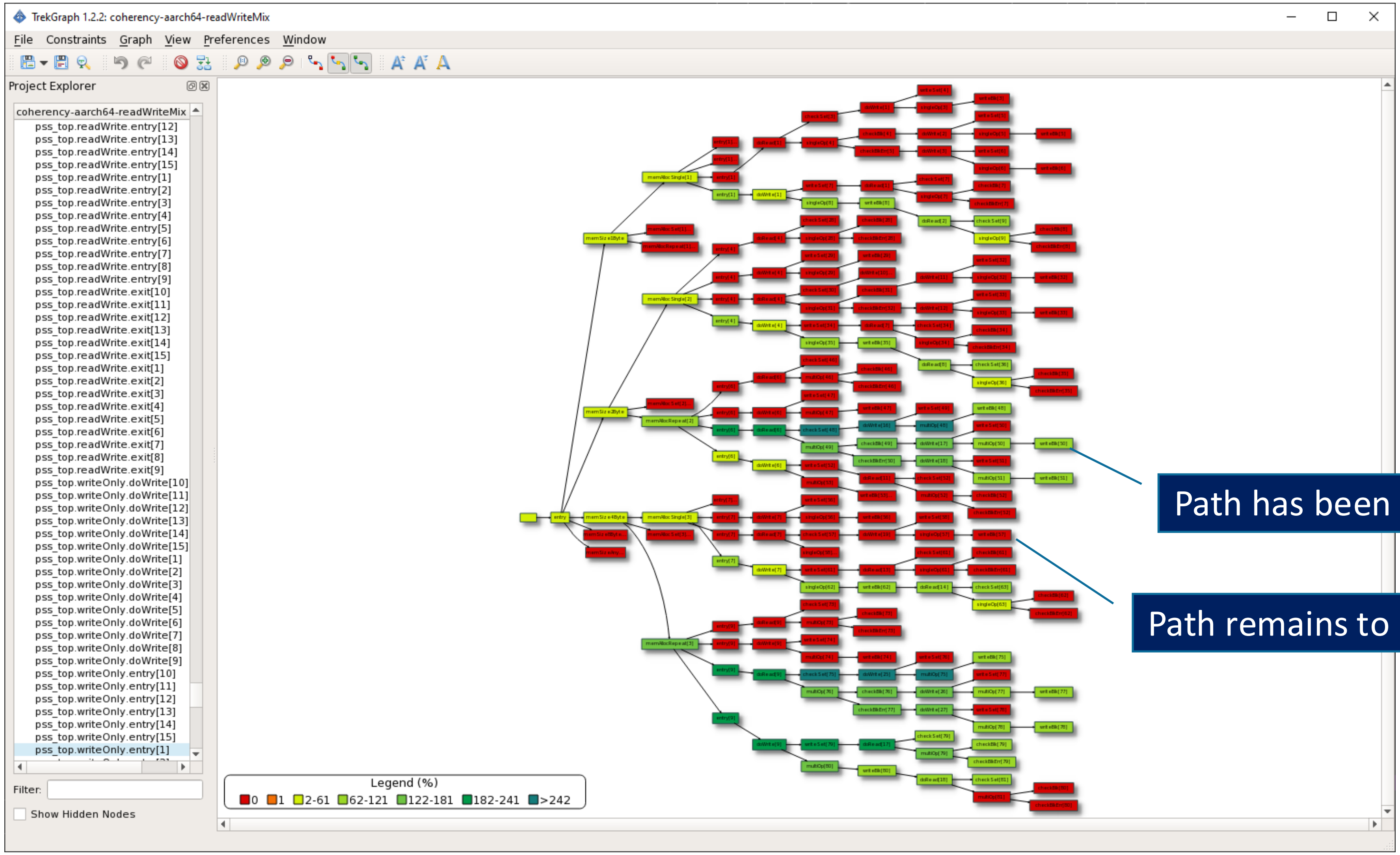
Call graph of C/C++ program with *select()* operator


Leaf PTE Flags
PTE/VPN addr selection
Load/Store/Exec Ops
S/U/VS/VU Priv levels
Stage 1 & 2 mode Selection (Sv39,Sv48x4 etc)
Non-Leaf PTE Flags
2.23e+138 possible Planning paths!
rvMmuOp descendant paths: 2.23285e+138

Test Suite Synthesis



Analyze & Close Scenario Model Coverage



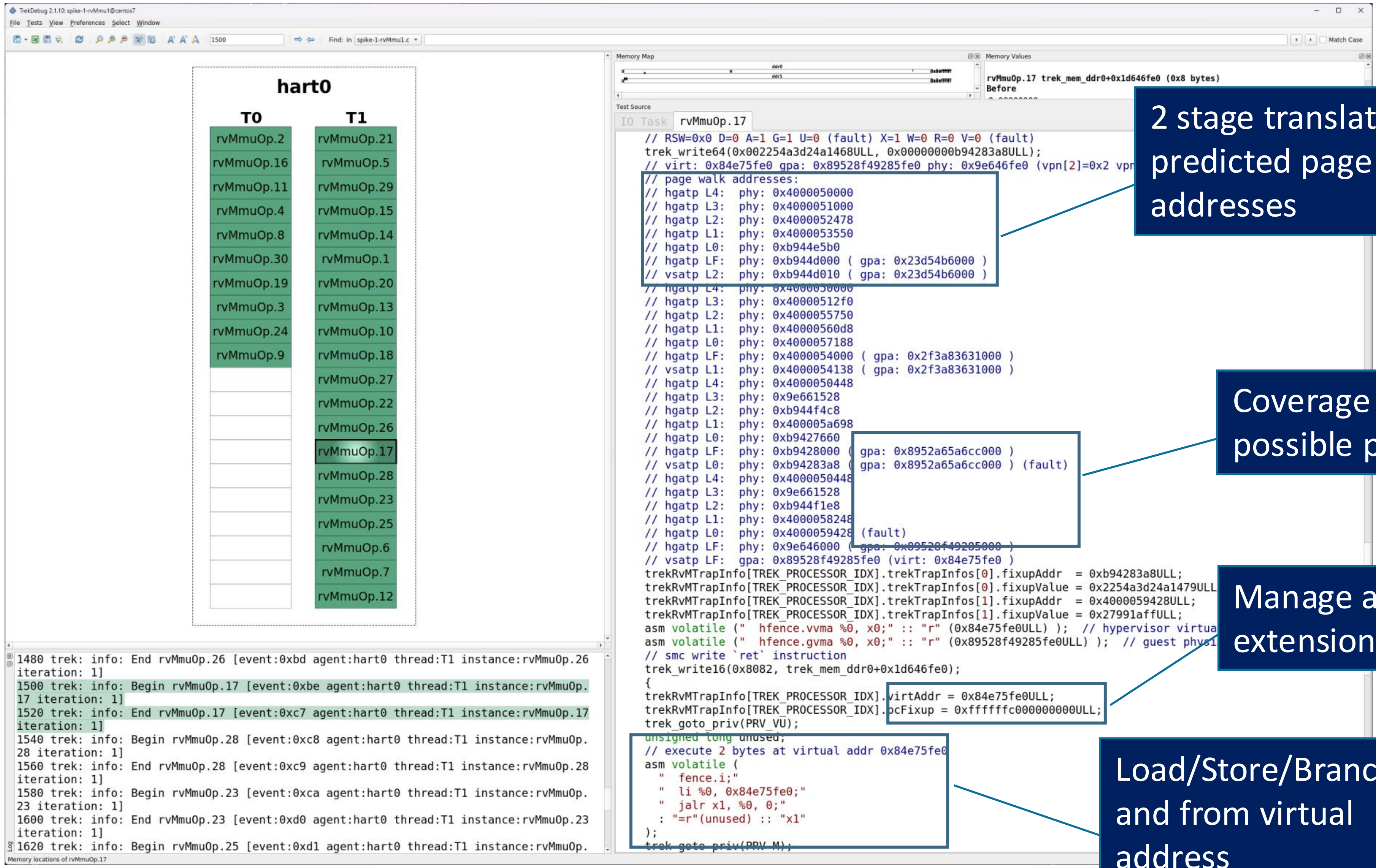
coherency-aarch64-readWriteMix
pss_top.readWrite.entry[12]
pss_top.readWrite.entry[13]
pss_top.readWrite.entry[14]
pss_top.readWrite.entry[15]
pss_top.readWrite.entry[1]
pss_top.readWrite.entry[2]
pss_top.readWrite.entry[3]
pss_top.readWrite.entry[4]
pss_top.readWrite.entry[5]
pss_top.readWrite.entry[6]
pss_top.readWrite.entry[7]
pss_top.readWrite.entry[8]
pss_top.readWrite.entry[9]
pss_top.readWrite.exit[10]
pss_top.readWrite.exit[11]
pss_top.readWrite.exit[12]
pss_top.readWrite.exit[13]
pss_top.readWrite.exit[14]
pss_top.readWrite.exit[15]
pss_top.readWrite.exit[1]
pss_top.readWrite.exit[2]
pss_top.readWrite.exit[3]
pss_top.readWrite.exit[4]
pss_top.readWrite.exit[5]
pss_top.readWrite.exit[6]
pss_top.readWrite.exit[7]
pss_top.readWrite.exit[8]
pss_top.readWrite.exit[9]
pss_top.writeOnly.doWrite[10]
pss_top.writeOnly.doWrite[11]
pss_top.writeOnly.doWrite[12]
pss_top.writeOnly.doWrite[13]
pss_top.writeOnly.doWrite[14]
pss_top.writeOnly.doWrite[15]
pss_top.writeOnly.doWrite[1]
pss_top.writeOnly.doWrite[2]
pss_top.writeOnly.doWrite[3]
pss_top.writeOnly.doWrite[4]
pss_top.writeOnly.doWrite[5]
pss_top.writeOnly.doWrite[6]
pss_top.writeOnly.doWrite[7]
pss_top.writeOnly.doWrite[8]
pss_top.writeOnly.doWrite[9]
pss_top.writeOnly.entry[10]
pss_top.writeOnly.entry[11]
pss_top.writeOnly.entry[12]
pss_top.writeOnly.entry[13]
pss_top.writeOnly.entry[14]
pss_top.writeOnly.entry[15]
pss_top.writeOnly.entry[1]

Legend (%)
🟥0 🟧1 🟨2-61 🟩62-121 🟩122-181 🟩182-241 🟩>242

Path has been exercised

Path remains to be exercised

MMU Hypervisor Two-Stage Address Translation Example


TrekDebug 2.1.10: spike-1-rvMmu1@centos7
File Tests View Preferences Select Window
Find: in spike-1-rvMmu1.c
Match Case
hart0
T0
T1
rvMmuOp.2 rvMmuOp.16 rvMmuOp.11 rvMmuOp.4 rvMmuOp.8 rvMmuOp.30 rvMmuOp.19 rvMmuOp.3 rvMmuOp.24 rvMmuOp.9
rvMmuOp.21 rvMmuOp.5 rvMmuOp.29 rvMmuOp.15 rvMmuOp.14 rvMmuOp.1 rvMmuOp.20 rvMmuOp.13 rvMmuOp.10 rvMmuOp.18 rvMmuOp.27 rvMmuOp.22 rvMmuOp.26 rvMmuOp.17 rvMmuOp.28 rvMmuOp.23 rvMmuOp.25 rvMmuOp.6 rvMmuOp.7 rvMmuOp.12
Memory Map
Memory Values
Test Source
IO Task
rvMmuOp.17
Log
Memory locations of rvMmuOp.17

```
rvMmuOp.17 trek_mem_ddr0+0x1d646fe0 (0x8 bytes)
Before
```

```
// RSW=0x0 D=0 A=1 G=1 U=0 (fault) X=1 W=0 R=0 V=0 (fault)
trek_write64(0x002254a3d24a1468ULL, 0x00000000b94283a8ULL);
// virt: 0x84e75fe0 gpa: 0x89528f49285fe0 phy: 0x9e646fe0 (vpn[2]=0x2 vpn
// page walk addresses:
// hgatp L4:  phy: 0x4000050000
// hgatp L3:  phy: 0x4000051000
// hgatp L2:  phy: 0x4000052478
// hgatp L1:  phy: 0x4000053550
// hgatp L0:  phy: 0xb944e5b0
// hgatp LF:  phy: 0xb944d000 ( gpa: 0x23d54b6000 )
// vsatp L2:  phy: 0xb944d010 ( gpa: 0x23d54b6000 )
// hgatp L4:  phy: 0x4000050000
// hgatp L3:  phy: 0x40000512f0
// hgatp L2:  phy: 0x4000055750
// hgatp L1:  phy: 0x40000560d8
// hgatp L0:  phy: 0x4000057188
// hgatp LF:  phy: 0x4000054000 ( gpa: 0x2f3a83631000 )
// vsatp L1:  phy: 0x4000054138 ( gpa: 0x2f3a83631000 )
// hgatp L4:  phy: 0x4000050448
// hgatp L3:  phy: 0x9e661528
// hgatp L2:  phy: 0xb944f4c8
// hgatp L1:  phy: 0x400005a698
// hgatp L0:  phy: 0xb9427660
// hgatp LF:  phy: 0xb9428000 ( gpa: 0x8952a65a6cc000 )
// vsatp L0:  phy: 0xb94283a8 ( gpa: 0x8952a65a6cc000 ) (fault)
// hgatp L4:  phy: 0x4000050448
// hgatp L3:  phy: 0x9e661528
// hgatp L2:  phy: 0xb944f1e8
// hgatp L1:  phy: 0x4000058248
// hgatp L0:  phy: 0x4000059428 (fault)
// hgatp LF:  phy: 0x9e646000 ( gpa: 0x89528f49285000 )
// vsatp LF:  gpa: 0x89528f49285fe0 (virt: 0x84e75fe0 )
trekRvMTrapInfo[TREK_PROCESSOR_IDX].trekTrapInfos[0].fixupAddr  = 0xb94283a8ULL;
trekRvMTrapInfo[TREK_PROCESSOR_IDX].trekTrapInfos[0].fixupValue = 0x2254a3d24a1479ULL
trekRvMTrapInfo[TREK_PROCESSOR_IDX].trekTrapInfos[1].fixupAddr  = 0x4000059428ULL;
trekRvMTrapInfo[TREK_PROCESSOR_IDX].trekTrapInfos[1].fixupValue = 0x27991affULL;
asm volatile ("  hfence.vvma %0, x0;" :: "r" (0x84e75fe0ULL) );  // hypervisor virtua
asm volatile ("  hfence.gvma %0, x0;" :: "r" (0x89528f49285fe0ULL) );  // guest physi
// smc write `ret` instruction
trek_write16(0x8082, trek_mem_ddr0+0x1d646fe0);
{
trekRvMTrapInfo[TREK_PROCESSOR_IDX].virtAddr = 0x84e75fe0ULL;
trekRvMTrapInfo[TREK_PROCESSOR_IDX].pcFixup = 0xffffffc000000000ULL;
trek_goto_priv(PRV_VU);
unsigned long unused;
// execute 2 bytes at virtual addr 0x84e75fe0
asm volatile (
  "  fence.i;"
  "  li %0, 0x84e75fe0;"
  "  jalr x1, %0, 0;"
  : "=r"(unused) :: "x1"
);
trek_goto_priv(PRV_M);
```

```
1480 trek: info: End rvMmuOp.26 [event:0xbd agent:hart0 thread:T1 instance:rvMmuOp.26
iteration: 1]
1500 trek: info: Begin rvMmuOp.17 [event:0xbe agent:hart0 thread:T1 instance:rvMmuOp.
17 iteration: 1]
1520 trek: info: End rvMmuOp.17 [event:0xc7 agent:hart0 thread:T1 instance:rvMmuOp.17
iteration: 1]
1540 trek: info: Begin rvMmuOp.28 [event:0xc8 agent:hart0 thread:T1 instance:rvMmuOp.
28 iteration: 1]
1560 trek: info: End rvMmuOp.28 [event:0xc9 agent:hart0 thread:T1 instance:rvMmuOp.28
iteration: 1]
1580 trek: info: Begin rvMmuOp.23 [event:0xca agent:hart0 thread:T1 instance:rvMmuOp.
23 iteration: 1]
1600 trek: info: End rvMmuOp.23 [event:0xd0 agent:hart0 thread:T1 instance:rvMmuOp.23
iteration: 1]
1620 trek: info: Begin rvMmuOp.25 [event:0xd1 agent:hart0 thread:T1 instance:rvMmuOp.
```

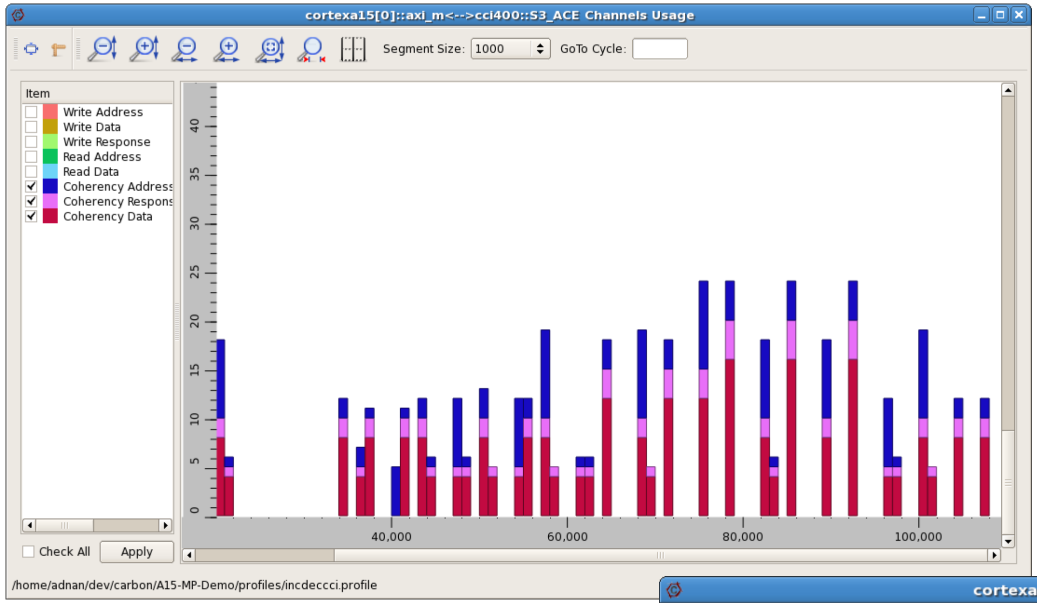
2 stage translation predicted page walk addresses

Coverage of all possible page faults

Manage address sign extension

Load/Store/Branch to and from virtual address

Massive Stress Testing of Coherency Architecture



cortexa15[0]::axi_m<-->cci400::S3_ACE Channels Usage

Segment Size: 1000 GoTo Cycle:

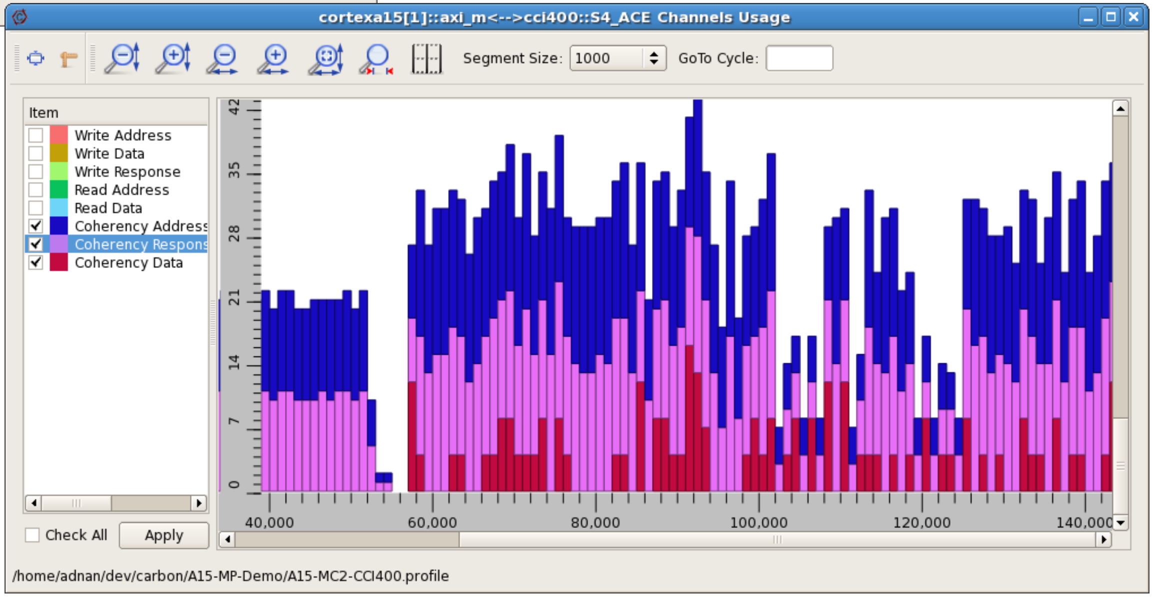
Item
- [ ] 🟥 Write Address
- [ ] 🟧 Write Data
- [ ] 🟩 Write Response
- [ ] 🟩 Read Address
- [ ] 🟦 Read Data
- [✓] 🟦 Coherency Address
- [✓] 🟪 Coherency Response
- [✓] 🟥 Coherency Data

Check All Apply

/home/adnan/dev/carbon/A15-MP-Demo/profiles/incdeccci.profile

Typical coherency test …

… vs Microloop based tests



cortexa15[1]::axi_m<-->cci400::S4_ACE Channels Usage

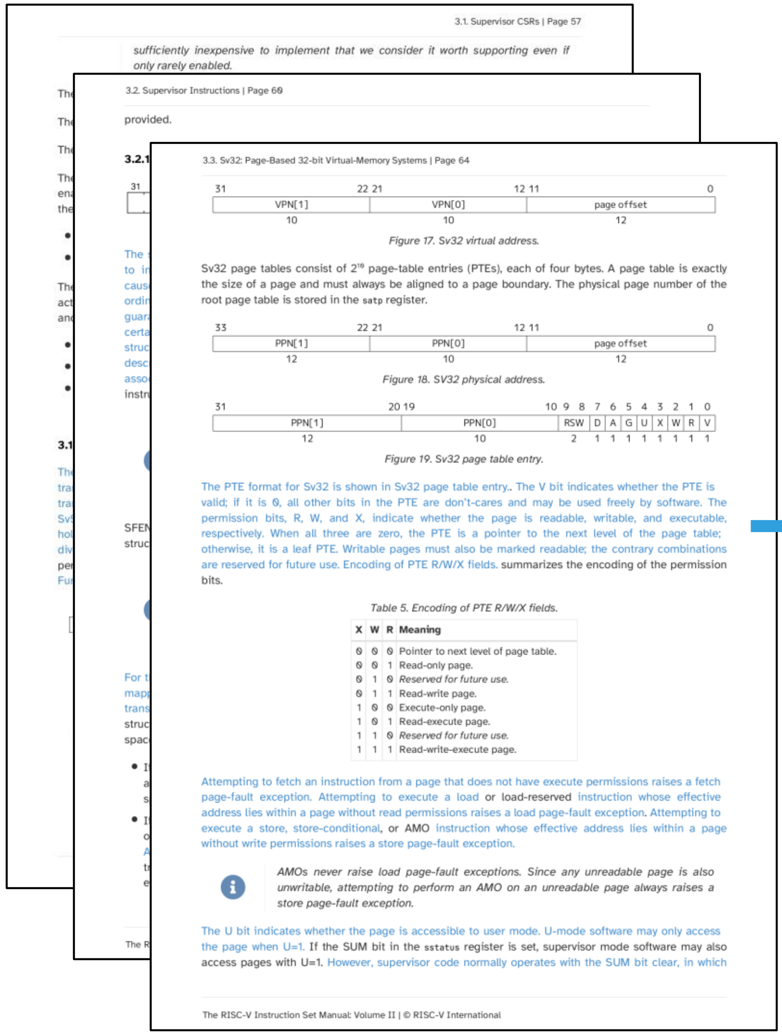
Segment Size: 1000 GoTo Cycle:

Item
- [ ] 🟥 Write Address
- [ ] 🟧 Write Data
- [ ] 🟩 Write Response
- [ ] 🟩 Read Address
- [ ] 🟦 Read Data
- [✓] 🟦 Coherency Address
- [✓] 🟪 Coherency Response
- [✓] 🟥 Coherency Data

Check All Apply

/home/adnan/dev/carbon/A15-MP-Demo/A15-MC2-CCI400.profile

RISC-V Specs, Test Plans and Coverage



3.1. Supervisor CSRs | Page 57

sufficiently inexpensive to implement that we consider it worth supporting even if only rarely enabled.

3.2. Supervisor Instructions | Page 60

provided.

3.3. Sv32: Page-Based 32-bit Virtual-Memory Systems | Page 64

31 — 22	21 — 12	11 — 0
VPN[1]	VPN[0]	page offset
10	10	12

Figure 17. Sv32 virtual address.

Sv32 page tables consist of 2^{10} page-table entries (PTEs), each of four bytes. A page table is exactly the size of a page and must always be aligned to a page boundary. The physical page number of the root page table is stored in the `satp` register.

33 — 22	21 — 12	11 — 0
PPN[1]	PPN[0]	page offset
12	10	12

Figure 18. SV32 physical address.

31 — 20	19 — 10	9 — 8	7	6	5	4	3	2	1	0
PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
12	10	2	1	1	1	1	1	1	1	1

Figure 19. Sv32 page table entry.

The PTE format for Sv32 is shown in Sv32 page table entry.. The V bit indicates whether the PTE is valid; if it is 0, all other bits in the PTE are don't-cares and may be used freely by software. The permission bits, R, W, and X, indicate whether the page is readable, writable, and executable, respectively. When all three are zero, the PTE is a pointer to the next level of the page table; otherwise, it is a leaf PTE. Writable pages must also be marked readable; the contrary combinations are reserved for future use. Encoding of PTE R/W/X fields. summarizes the encoding of the permission bits.

Table 5. Encoding of PTE R/W/X fields.

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

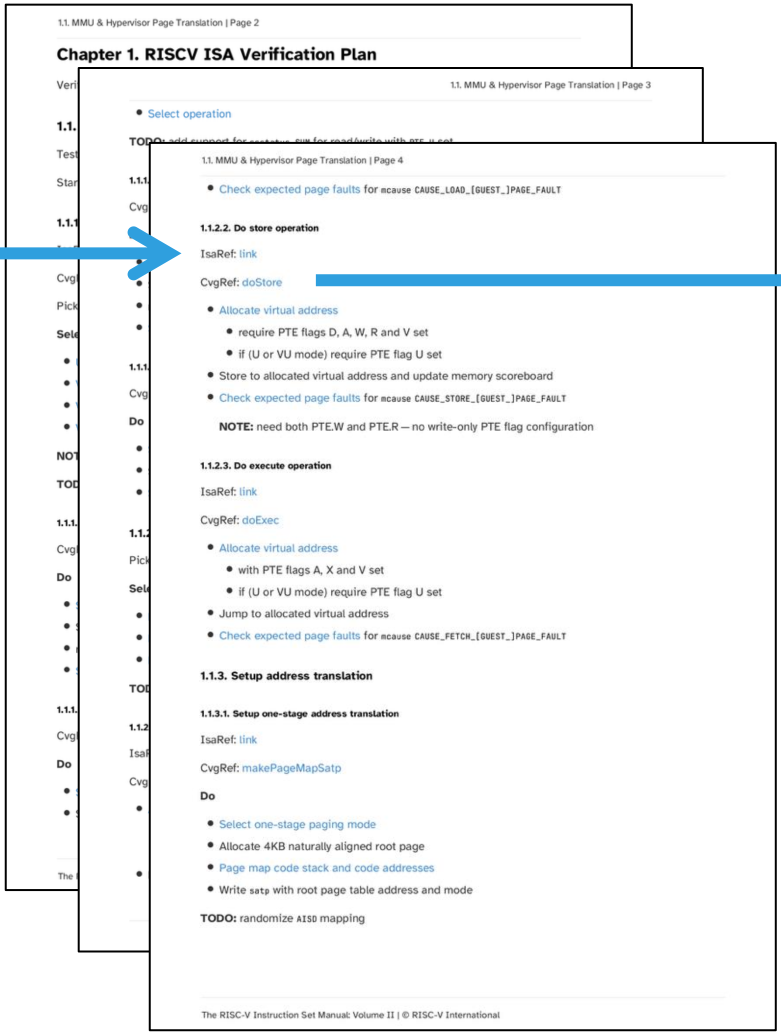
Attempting to fetch an instruction from a page that does not have execute permissions raises a fetch page-fault exception. Attempting to execute a load or load-reserved instruction whose effective address lies within a page without read permissions raises a load page-fault exception. Attempting to execute a store, store-conditional, or AMO instruction whose effective address lies within a page without write permissions raises a store page-fault exception.

AMOs never raise load page-fault exceptions. Since any unreadable page is also unwritable, attempting to perform an AMO on an unreadable page always raises a store page-fault exception.

The U bit indicates whether the page is accessible to user mode. U-mode software may only access the page when U=1. If the SUM bit in the `sstatus` register is set, supervisor mode software may also access pages with U=1. However, supervisor code normally operates with the SUM bit clear, in which

The RISC-V Instruction Set Manual: Volume II | © RISC-V International

ISA Specification



1.1. MMU & Hypervisor Page Translation | Page 2

Chapter 1. RISCV ISA Verification Plan

1.1. MMU & Hypervisor Page Translation | Page 3

- Select operation

1.1. MMU & Hypervisor Page Translation | Page 4

- Check expected page faults for `mcause CAUSE_LOAD_[GUEST_]PAGE_FAULT`

1.1.2.2. Do store operation

IsaRef: link

CvgRef: doStore

- Allocate virtual address
  - require PTE flags D, A, W, R and V set
  - if (U or VU mode) require PTE flag U set
- Store to allocated virtual address and update memory scoreboard
- Check expected page faults for `mcause CAUSE_STORE_[GUEST_]PAGE_FAULT`

NOTE: need both PTE.W and PTE.R — no write-only PTE flag configuration

1.1.2.3. Do execute operation

IsaRef: link

CvgRef: doExec

- Allocate virtual address
  - with PTE flags A, X and V set
  - if (U or VU mode) require PTE flag U set
- Jump to allocated virtual address
- Check expected page faults for `mcause CAUSE_FETCH_[GUEST_]PAGE_FAULT`

1.1.3. Setup address translation

1.1.3.1. Setup one-stage address translation

IsaRef: link

CvgRef: makePageMapSatp

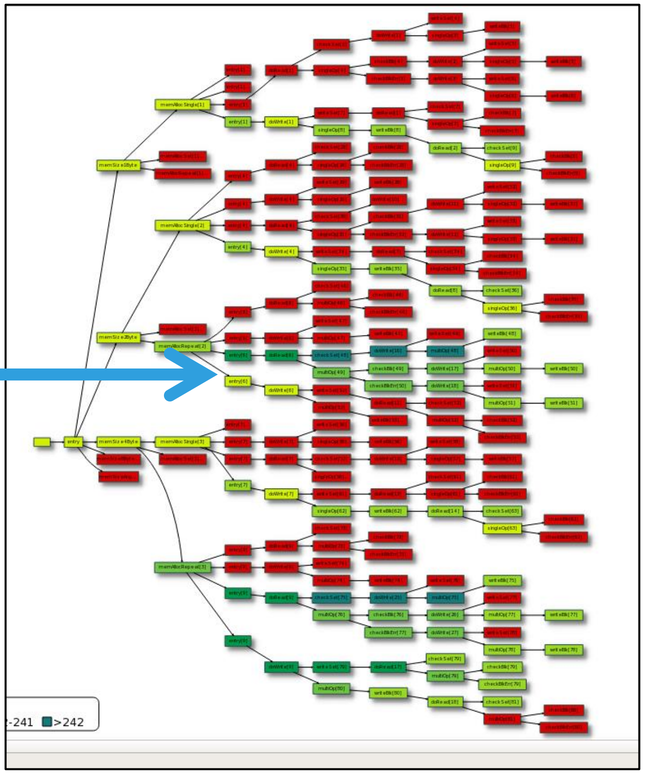
Do

- Select one-stage paging mode
- Allocate 4KB naturally aligned root page
- Page map code stack and code addresses
- Write `satp` with root page table address and mode

TODO: randomize `AISD` mapping

The RISC-V Instruction Set Manual: Volume II | © RISC-V International

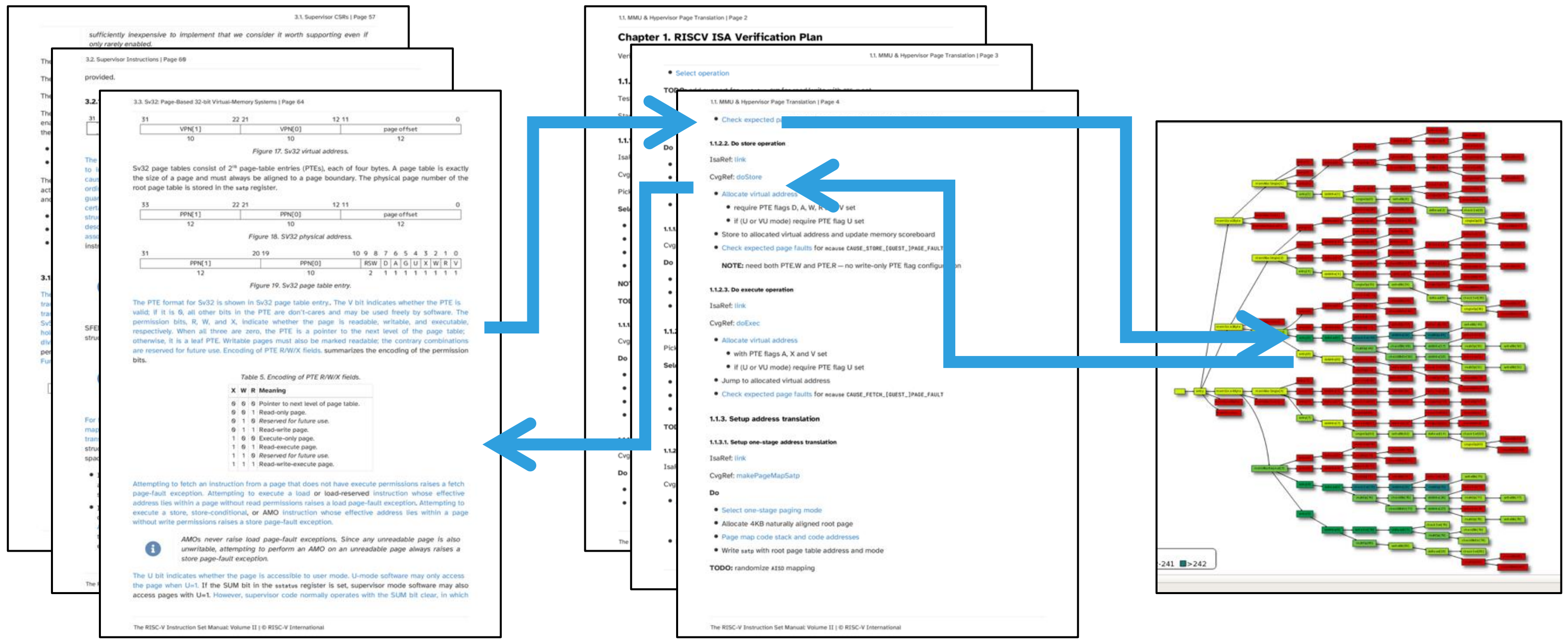
RISCV Test Plan



Coverage Reports

Leveraging AI For Spec to Test and Return

- Breker already leveraging Planning Algorithms
  - AI Expert System for advanced state space search
- New project using SLM to comprehend ISA spec and create plan/model
  - Include back-annotation to ensure spec coverage



A Word on Certification

	Phase 1	Phase 2	Phase 3	Phase 4
	Certified Microcontroller	**Certified Apps Processor**	**Certified Subsystem**	**Certified Server Platform**
Scope	Few ISA Features ● RV20 Base ISA ● Unpriv ISA ● Priv ISA	Many ISA Features ● RVA23 Profile ● Full Priv/Unpriv ● Superset of Microcontroller tests ● Some System Integrity	SoC Components ● AIA ● IOMMU ● PCIe Integration ● Peripherals ● System Integrity	Processor & SoC ● Certified Apps Processor ● Certified SoC Components
Testing	ISA compatibility	ISA compatibility System Integrity	ISA compatibility System Integrity	System Integrity

Thanks for Listening!
Any Questions?