BREKER™ THE LEADER IN PORTABLE STIMULUS

Automating Verification Checks

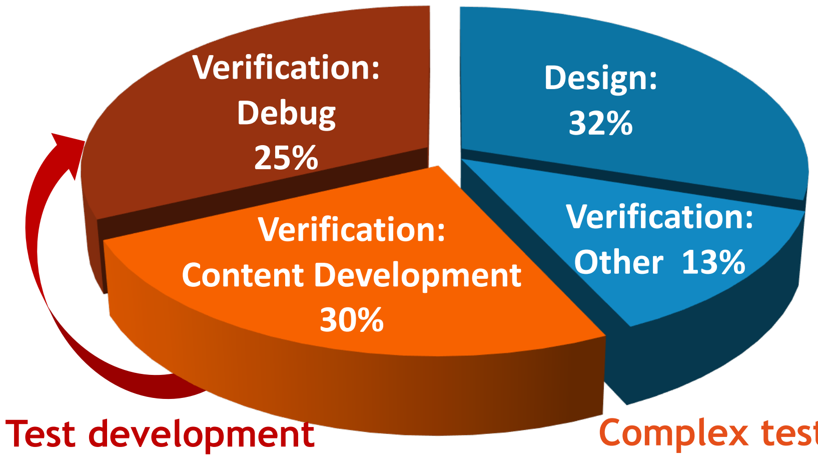
Synthesizing Self-Checking Test Content

David Kelf

DVClub Europe October 2022

Explosive Verification Cost

Project Resource Deployment

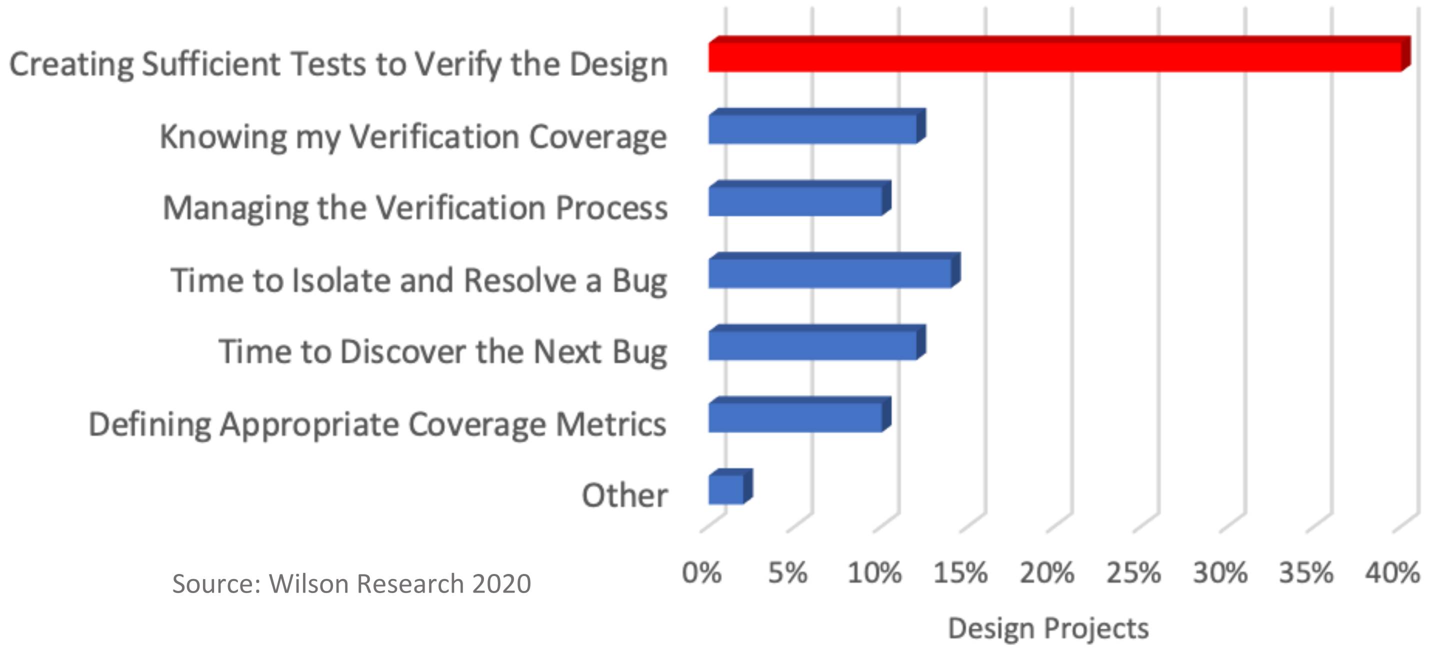


Test development drives debug

Complex tests hard to get right

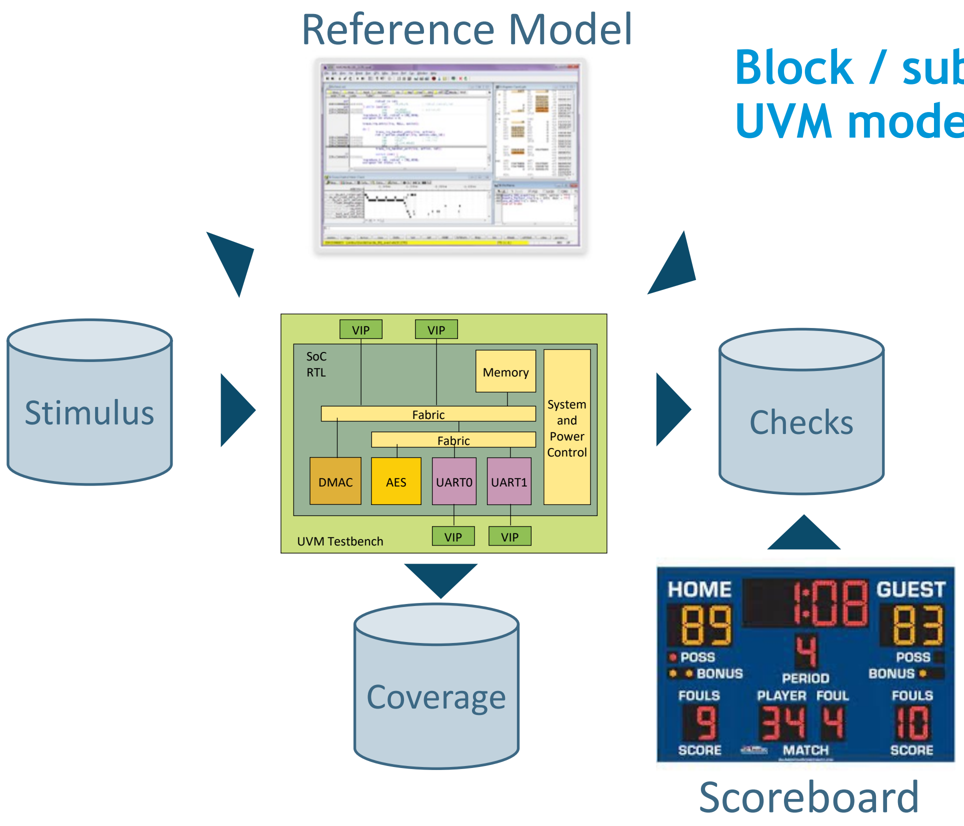
Source: Wilson Research 2020

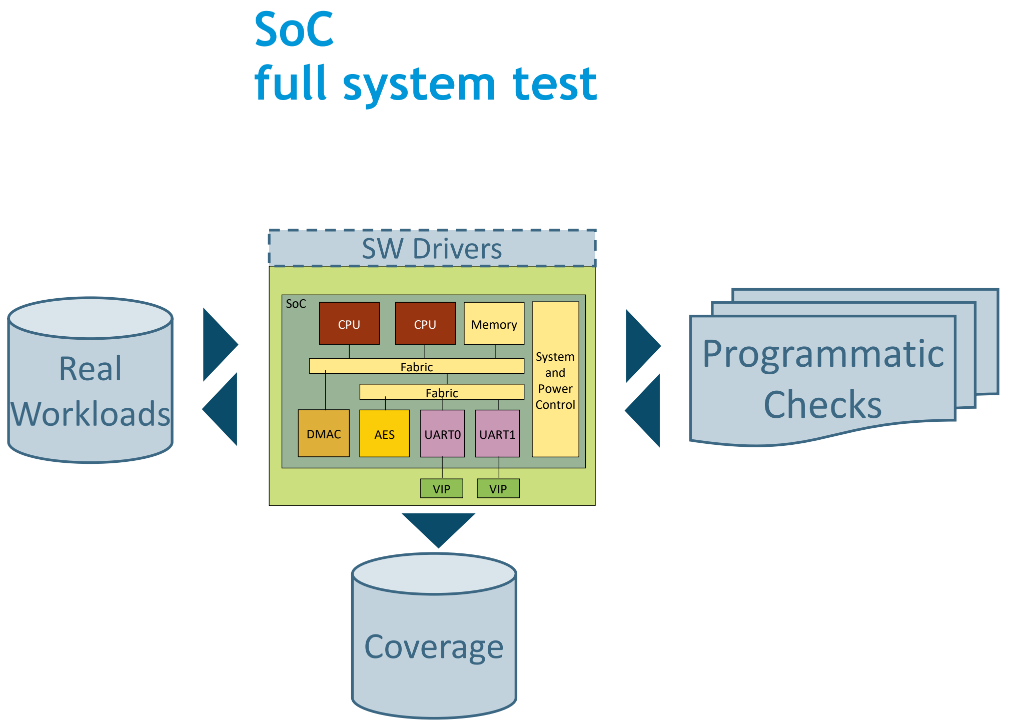
Largest Functional Verification Challenge



Source: Wilson Research 2020

UVM & SoC Verification Check Alternatives Today



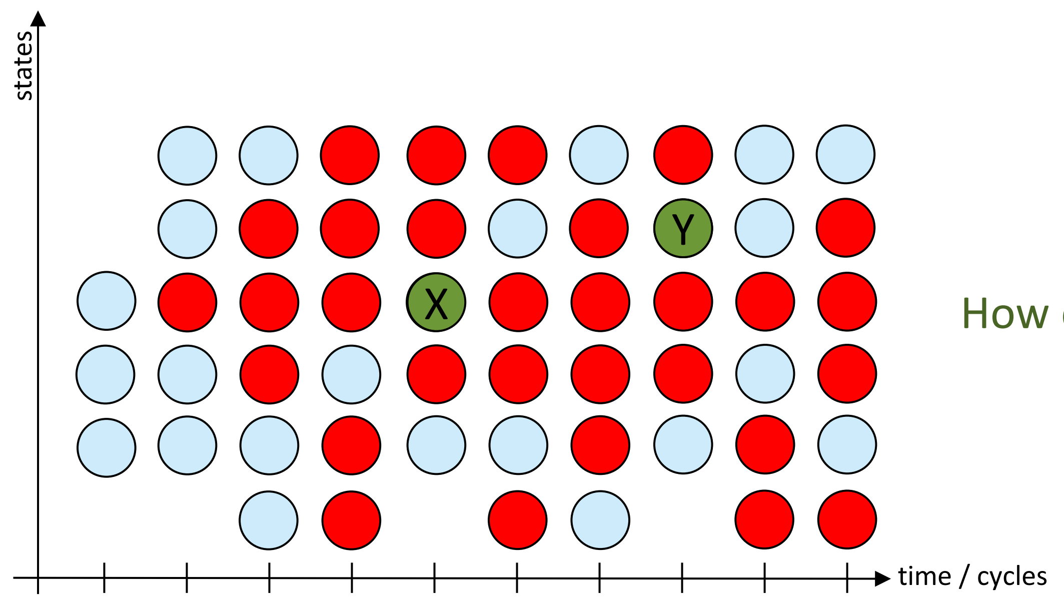


Composing checks (and coverage models) can be onerous and error prone

Formal Approach

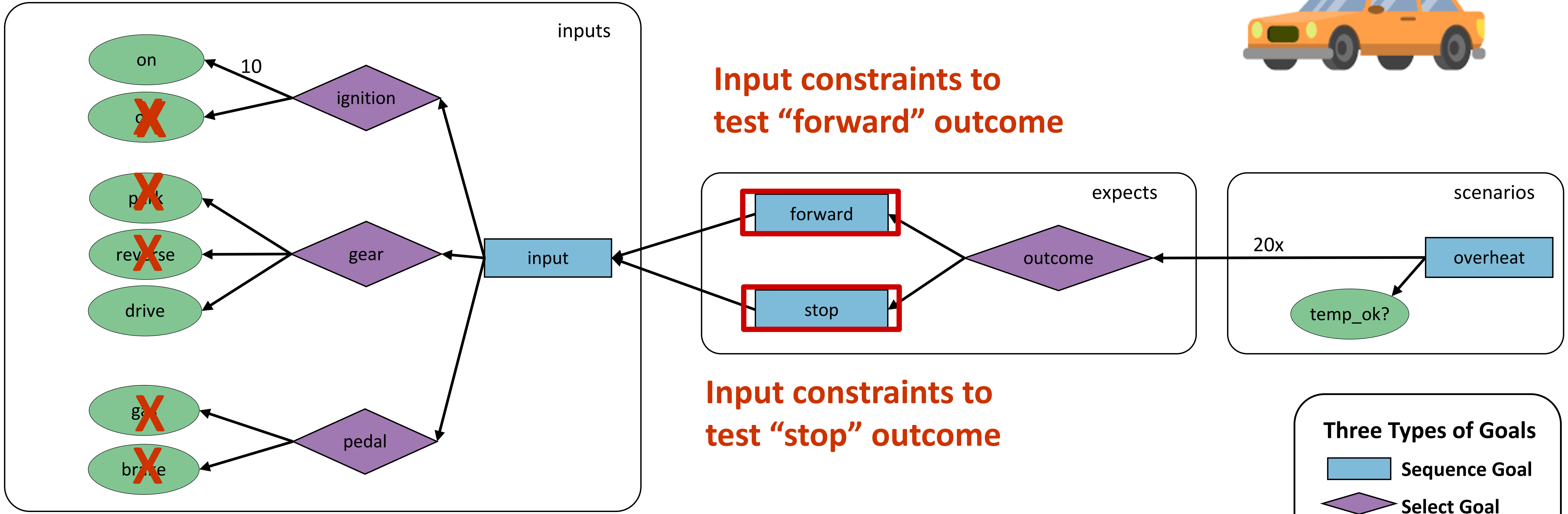
- Traditional dynamic test method:
  Send stimulus, write expected results, check coverage of test
- How do Formal Verification tools approach this?

  Propose check and see if it can happen based on entire state space
- Can we do the same in dynamic verification?



How does Y happen after X happens?

Start with the end in mind



Input constraints to test "forward" outcome

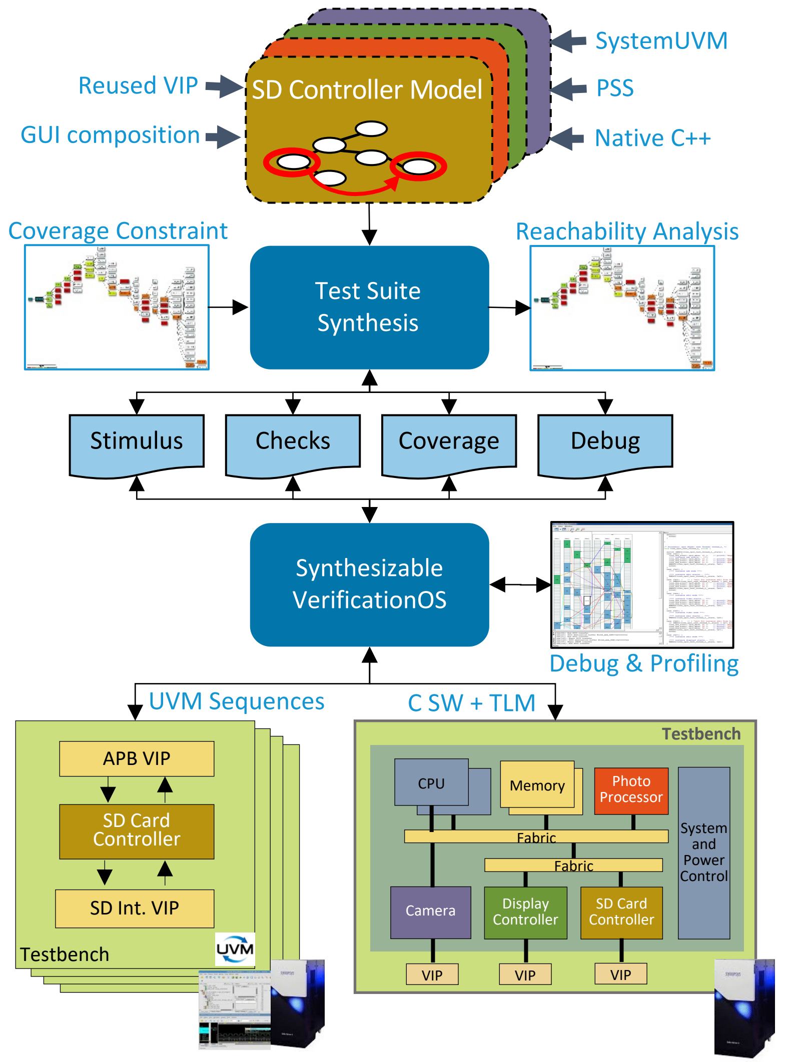
Input constraints to test "stop" outcome

Three Types of Goals

🟦	Sequence Goal
◆	Select Goal
🟢	Leaf Goal

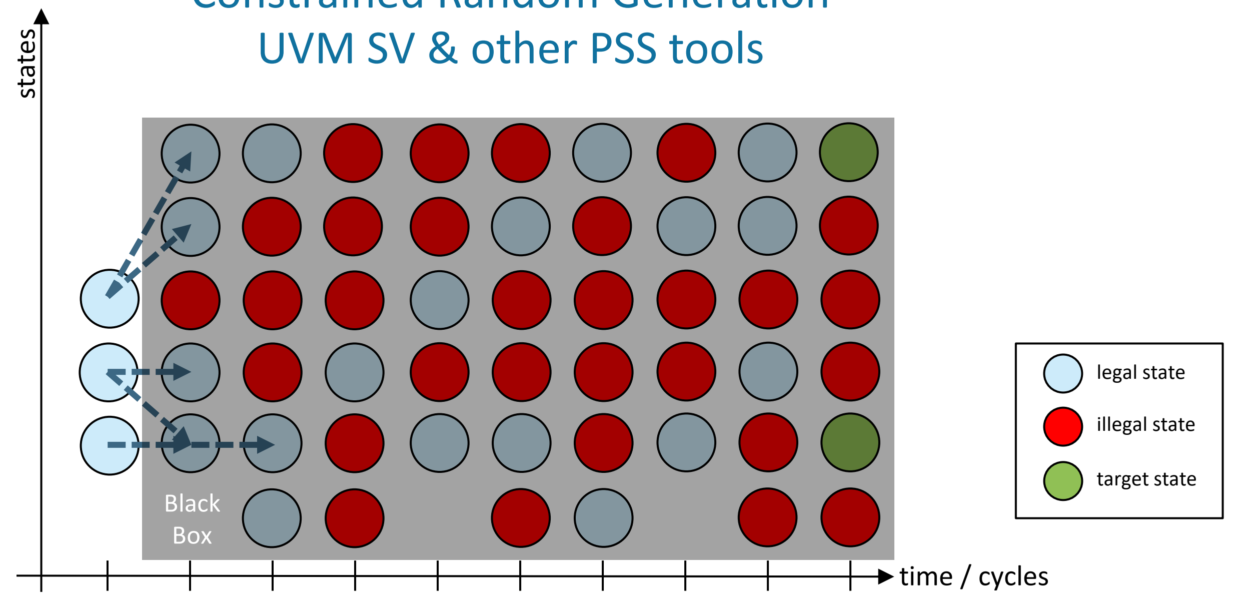
Synthesis approach to generate test content?

- Create a specification scenario model that shows how the device is supposed to work
  - Could be composed in PSS, C++, graphically, or using SystemVIP
- Synthesize model based on coverage constraints
  - Set coverage up front
- Generate entire test content automatically
  - Stimulus, checks, coverage models, debug detail
- Map to verification phase and execution platform



Constrained Random vs AI Planning Algorithm Synthesis

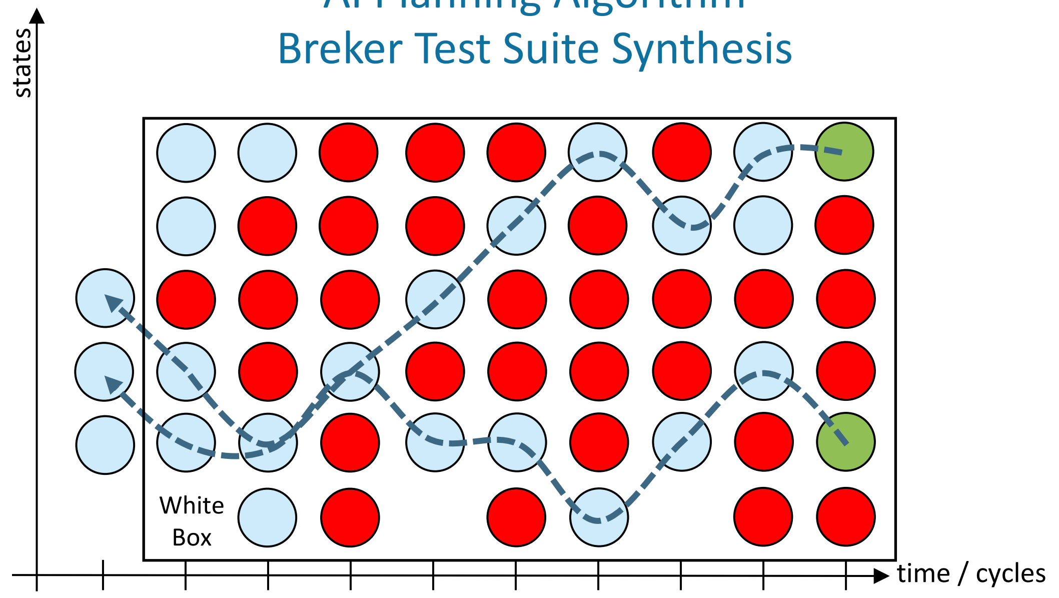
Constrained Random Generation
UVM SV & other PSS tools



Design black box, shotgun tests to search for key state

Low probability of finding complex bug

AI Planning Algorithm
Breker Test Suite Synthesis

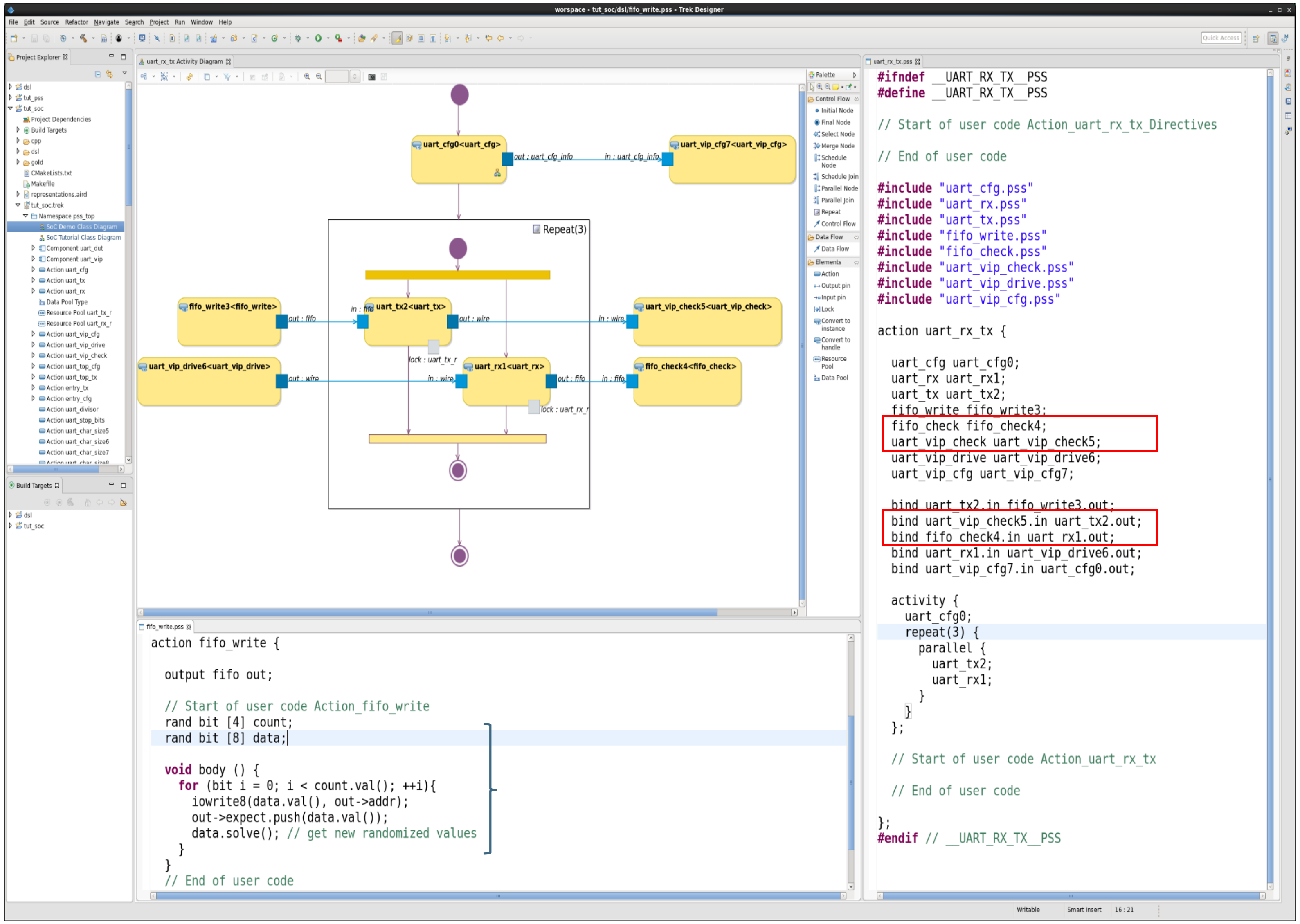


Starts with key state and intelligently works backward through space

Deep sequential, optimized test discovers complex corner-cases

White Paper Discussing AI Planning Algorithm Test Generation on Breker Website

Block level embedded checks



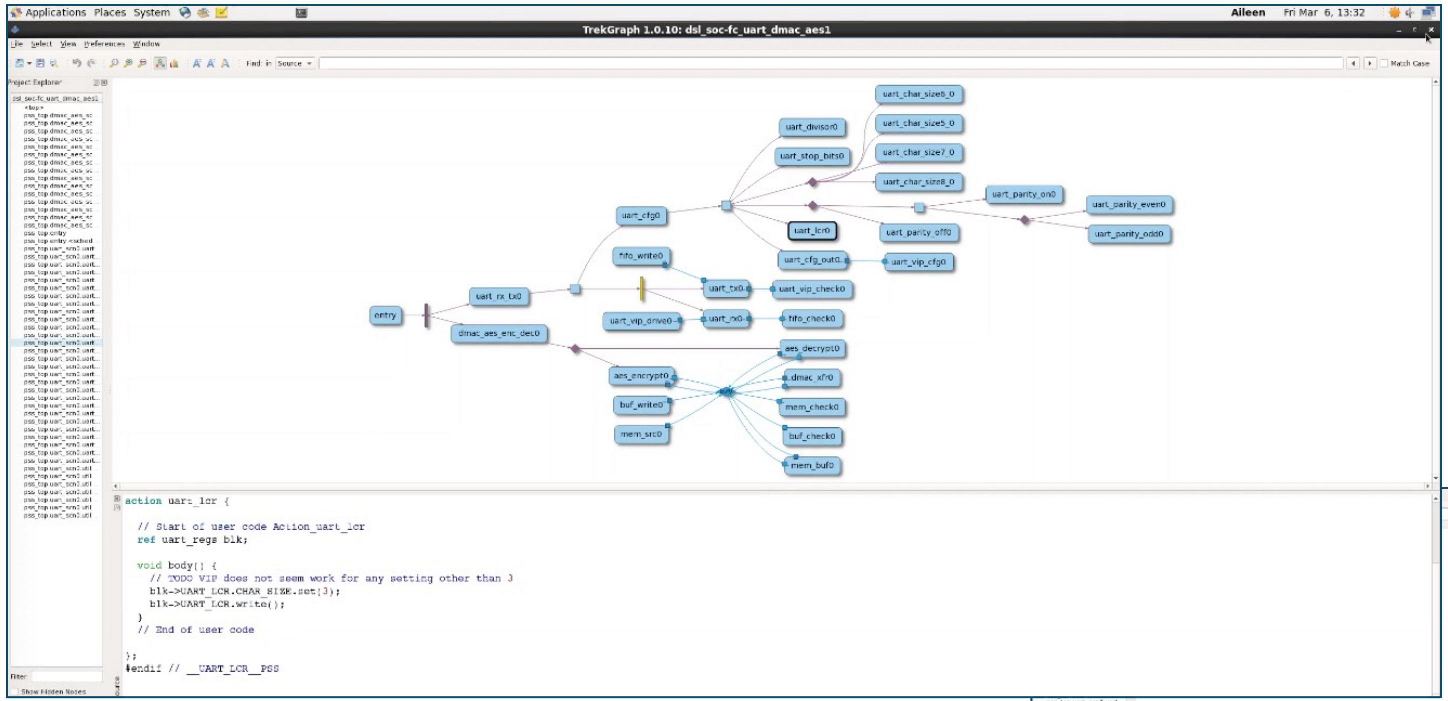
```
action fifo_write {

  output fifo out;

  // Start of user code Action_fifo_write
  rand bit [4] count;
  rand bit [8] data;

  void body () {
    for (bit i = 0; i < count.val(); ++i){
      iowrite8(data.val(), out->addr);
      out->expect.push(data.val());
      data.solve(); // get new randomized values
    }
  }
  // End of user code
```

```
#ifndef __UART_RX_TX__PSS
#define __UART_RX_TX__PSS

// Start of user code Action_uart_rx_tx_Directives

// End of user code

#include "uart_cfg.pss"
#include "uart_rx.pss"
#include "uart_tx.pss"
#include "fifo_write.pss"
#include "fifo_check.pss"
#include "uart_vip_check.pss"
#include "uart_vip_drive.pss"
#include "uart_vip_cfg.pss"

action uart_rx_tx {

  uart_cfg uart_cfg0;
  uart_rx uart_rx1;
  uart_tx uart_tx2;
  fifo_write fifo_write3;
  fifo_check fifo_check4;
  uart_vip_check uart_vip_check5;
  uart_vip_drive uart_vip_drive6;
  uart_vip_cfg uart_vip_cfg7;

  bind uart_tx2.in fifo_write3.out;
  bind uart_vip_check5.in uart_tx2.out;
  bind fifo_check4.in uart_rx1.out;
  bind uart_rx1.in uart_vip_drive6.out;
  bind uart_vip_cfg7.in uart_cfg0.out;

  activity {
    uart_cfg0;
    repeat(3) {
      parallel {
        uart_tx2;
        uart_rx1;
      }
    }
  };

  // Start of user code Action_uart_rx_tx

  // End of user code

};
#endif // __UART_RX_TX__PSS
```

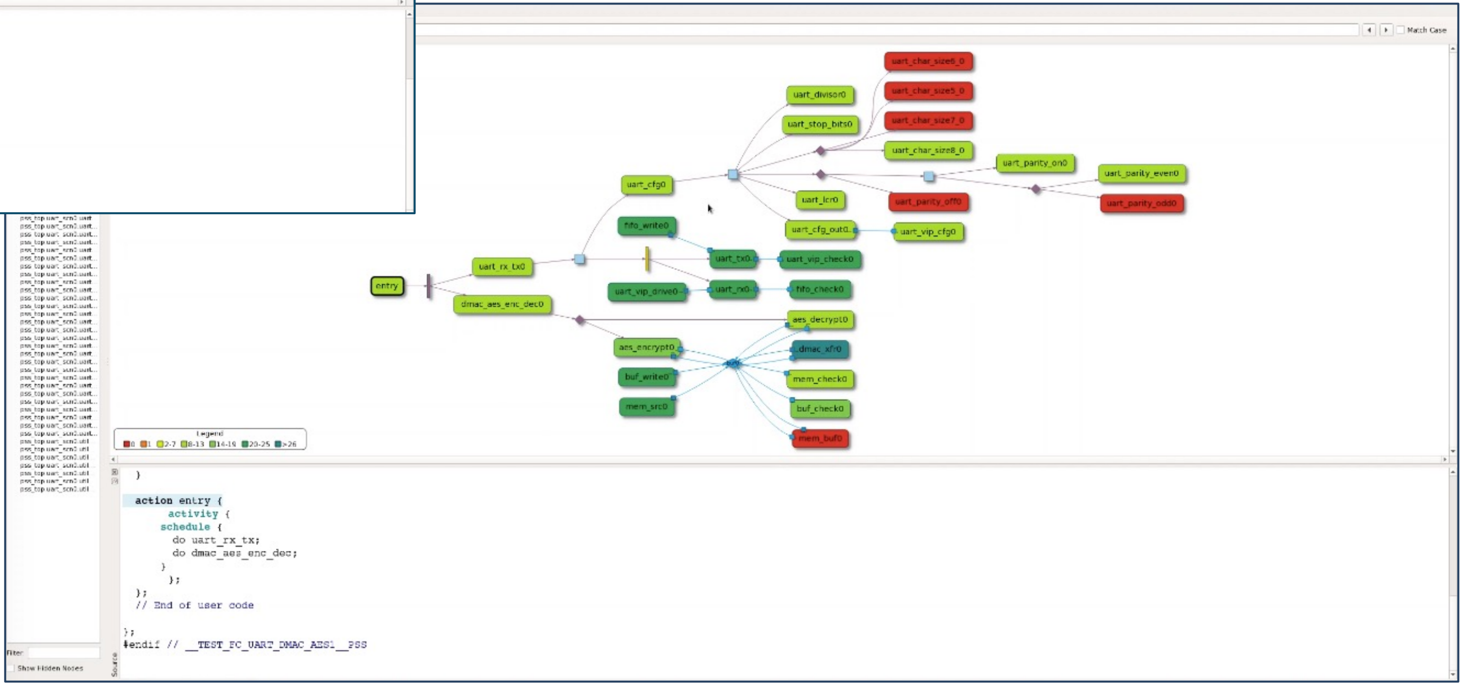
Path Coverage Analysis

BREKER™

Portable

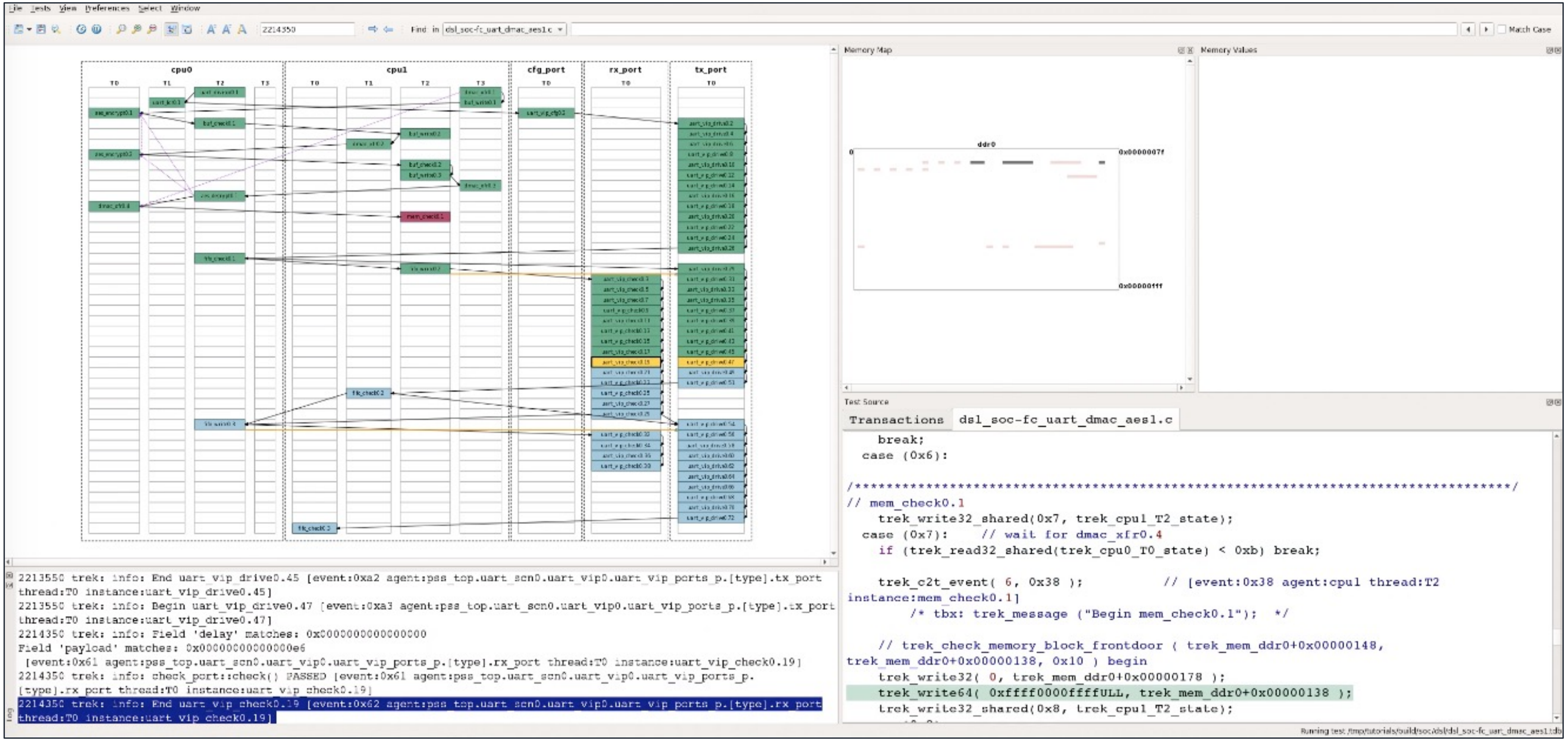


```
action uart_lcr {

  // Start of user code Action_uart_lcr
  ref uart_regs blk;

  void body() {
    // TODO VIP does not seem work for any setting other than 3
    blk->UART_LCR.CHAR_SIZE.set(3);
    blk->UART_LCR.write();
  }
  // End of user code

};
#endif // __UART_LCR__PSS
```



```
  }

  action entry {
        activity {
      schedule {
        do uart_rx_tx;
        do dmac_aes_enc_dec;
      }
        };
  };
  // End of user code

};
#endif // __TEST_FC_UART_DMAC_AES1__PSS
```

Self-Checking Test Execution

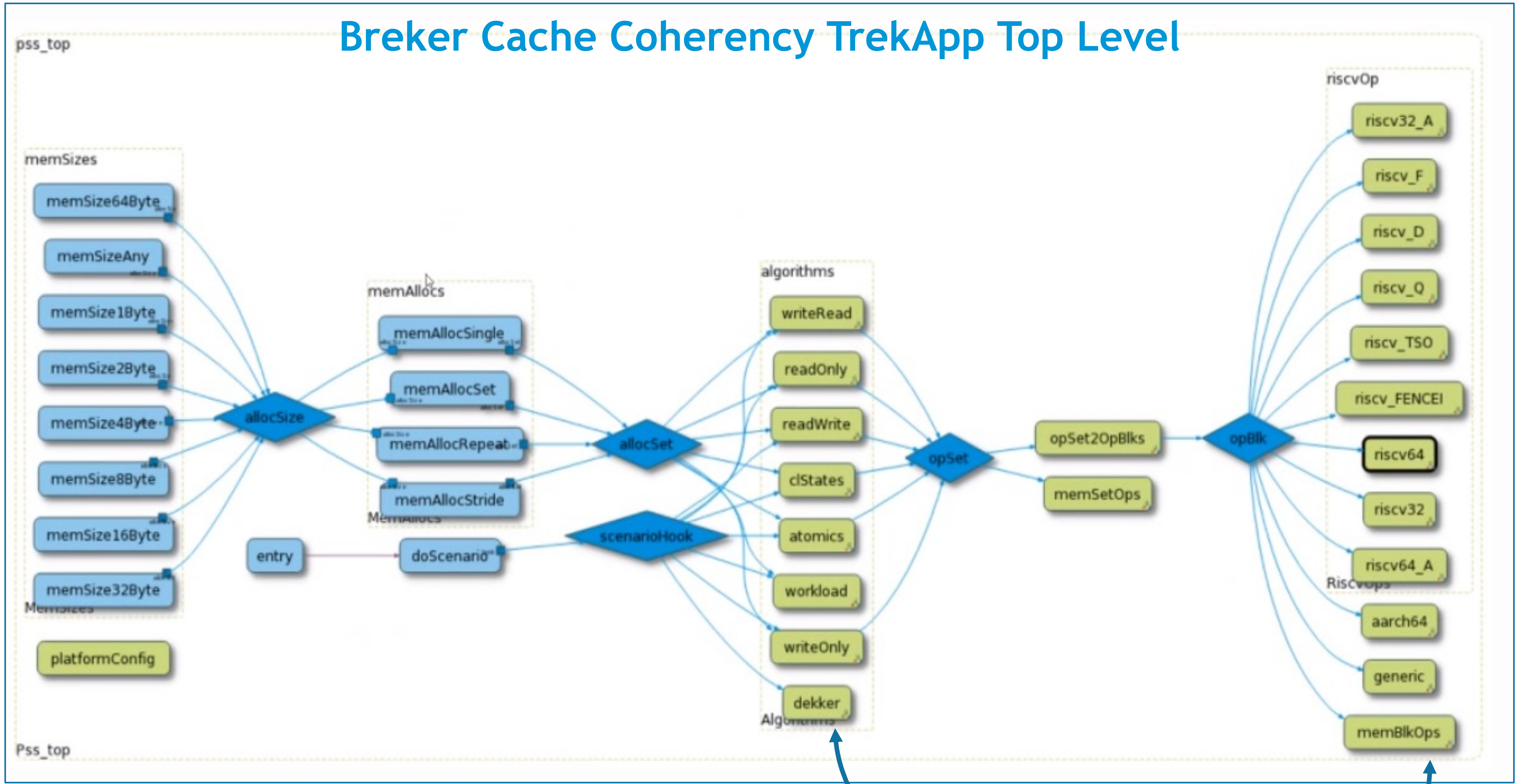

File Tests View Preferences Select Window
2214350
Find in dsl_soc-fc_uart_dmac_aes1.c
Match Case
cpu0
cpu1
cfg_port
rx_port
tx_port
T0
T1
T2
T3
Memory Map
Memory Values
ddr0
0x0000007f
0x000000ff
Test Source
Transactions
dsl_soc-fc_uart_dmac_aes1.c
Log
Running test /tmp/tutorials/build/soc/dsl/dsl_soc-fc_uart_dmac_aes1.tdb

```
2213550 trek: info: End uart_vip_drive0.45 [event:0xa2 agent:pss_top.uart_scn0.uart_vip0.uart_vip_ports_p.[type].tx_port
thread:T0 instance:uart_vip_drive0.45]
2213550 trek: info: Begin uart_vip_drive0.47 [event:0xa3 agent:pss_top.uart_scn0.uart_vip0.uart_vip_ports_p.[type].tx_port
thread:T0 instance:uart_vip_drive0.47]
2214350 trek: info: Field 'delay' matches: 0x0000000000000000
Field 'payload' matches: 0x00000000000000e6
 [event:0x61 agent:pss_top.uart_scn0.uart_vip0.uart_vip_ports_p.[type].rx_port thread:T0 instance:uart_vip_check0.19]
2214350 trek: info: check_port::check() PASSED [event:0x61 agent:pss_top.uart_scn0.uart_vip0.uart_vip_ports_p.
[type].rx_port thread:T0 instance:uart_vip_check0.19]
2214350 trek: info: End uart_vip_check0.19 [event:0x62 agent:pss_top.uart_scn0.uart_vip0.uart_vip_ports_p.[type].rx_port
thread:T0 instance:uart_vip_check0.19]
```

```
    break;
  case (0x6):

/*************************************************************************************/
// mem_check0.1
    trek_write32_shared(0x7, trek_cpu1_T2_state);
  case (0x7):    // wait for dmac_xfr0.4
    if (trek_read32_shared(trek_cpu0_T0_state) < 0xb) break;

    trek_c2t_event( 6, 0x38 );          // [event:0x38 agent:cpu1 thread:T2
instance:mem_check0.1]
        /* tbx: trek_message ("Begin mem_check0.1");  */

    // trek_check_memory_block_frontdoor ( trek_mem_ddr0+0x00000148,
trek_mem_ddr0+0x00000138, 0x10 ) begin
    trek_write32( 0, trek_mem_ddr0+0x00000178 );
    trek_write64( 0xffff0000ffffULL, trek_mem_ddr0+0x00000138 );
    trek_write32_shared(0x8, trek_cpu1_T2_state);
```

SoC Test Content

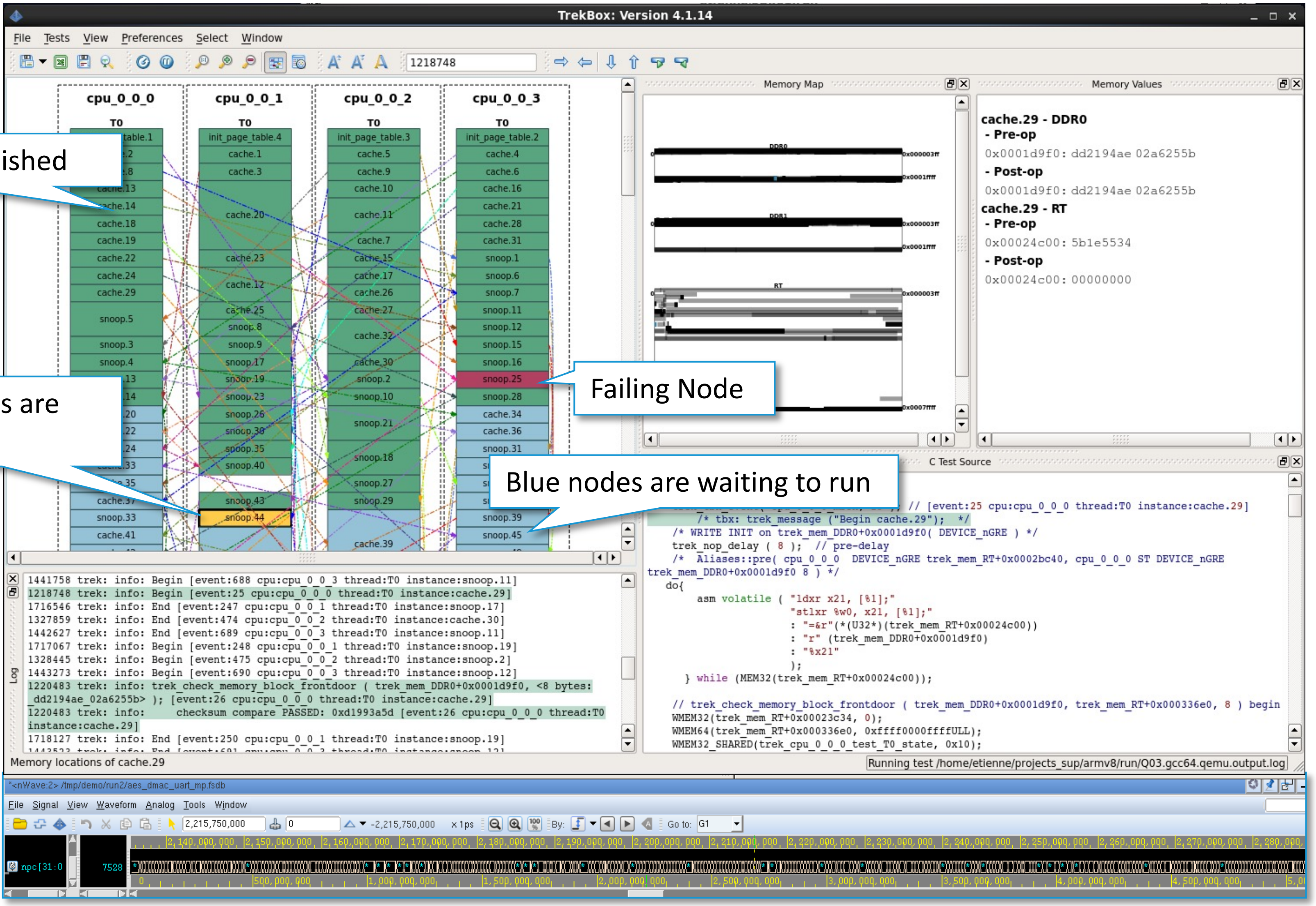
Breker Cache Coherency TrekApp Top Level



Exec body { memory.check () }

High-level Test Debug Driving Issue Resolution

Portable



cache.29 - DDR0
- Pre-op
0x0001d9f0: dd2194ae 02a6255b
- Post-op
0x0001d9f0: dd2194ae 02a6255b
cache.29 - RT
- Pre-op
0x00024c00: 5b1e5534
- Post-op
0x00024c00: 00000000

```
1441758 trek: info: Begin [event:688 cpu:cpu_0_0_3 thread:T0 instance:snoop.11]
1218748 trek: info: Begin [event:25 cpu:cpu_0_0_0 thread:T0 instance:cache.29]
1716546 trek: info: End [event:247 cpu:cpu_0_0_1 thread:T0 instance:snoop.17]
1327859 trek: info: End [event:474 cpu:cpu_0_0_2 thread:T0 instance:cache.30]
1442627 trek: info: End [event:689 cpu:cpu_0_0_3 thread:T0 instance:snoop.11]
1717067 trek: info: Begin [event:248 cpu:cpu_0_0_1 thread:T0 instance:snoop.19]
1328445 trek: info: Begin [event:475 cpu:cpu_0_0_2 thread:T0 instance:snoop.2]
1443273 trek: info: Begin [event:690 cpu:cpu_0_0_3 thread:T0 instance:snoop.12]
1220483 trek: info: trek_check_memory_block_frontdoor ( trek_mem_DDR0+0x0001d9f0, <8 bytes:
_dd2194ae_02a6255b> ); [event:26 cpu:cpu_0_0_0 thread:T0 instance:cache.29]
1220483 trek: info:     checksum compare PASSED: 0xd1993a5d [event:26 cpu:cpu_0_0_0 thread:T0
instance:cache.29]
1718127 trek: info: End [event:250 cpu:cpu_0_0_1 thread:T0 instance:snoop.19]
```

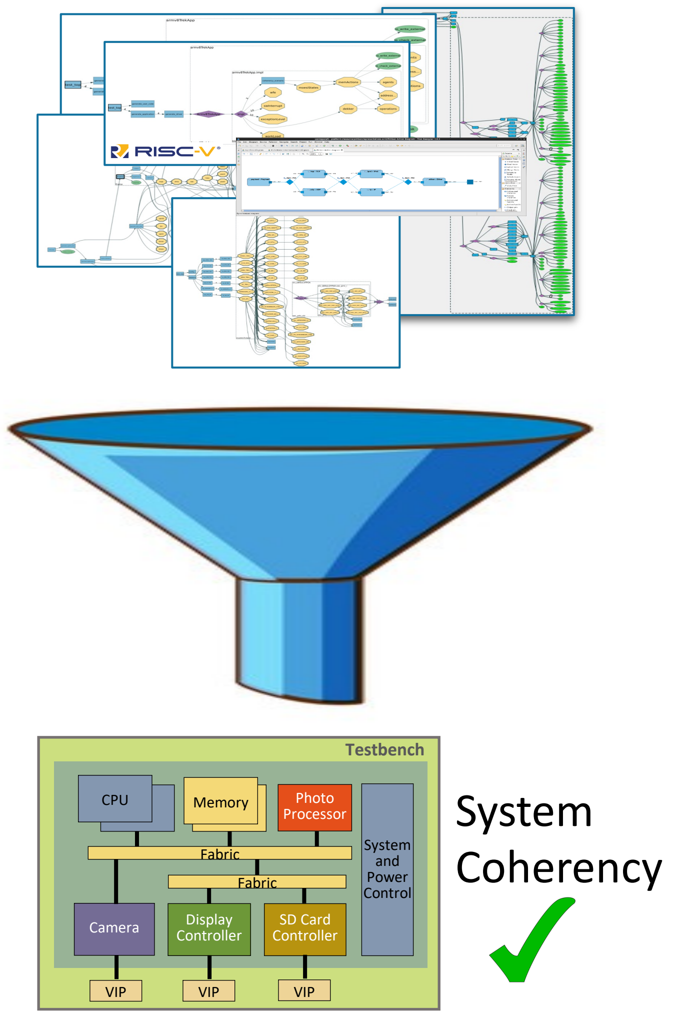
Memory locations of cache.29

```
                                          // [event:25 cpu:cpu_0_0_0 thread:T0 instance:cache.29]
         /* tbx: trek_message ("Begin cache.29");  */
     /* WRITE INIT on trek_mem_DDR0+0x0001d9f0( DEVICE_nGRE ) */
     trek_nop_delay ( 8 );  // pre-delay
     /*  Aliases::pre( cpu_0_0_0  DEVICE_nGRE trek_mem_RT+0x0002bc40, cpu_0_0_0 ST DEVICE_nGRE
trek_mem_DDR0+0x0001d9f0 8 ) */
   do{
         asm volatile ( "ldxr x21, [%1];"
                        "stlxr %w0, x21, [%1];"
                        : "=&r"(*(U32*)(trek_mem_RT+0x00024c00))
                        : "r" (trek_mem_DDR0+0x0001d9f0)
                        : "%x21"
                        );
     } while (MEM32(trek_mem_RT+0x00024c00));

     // trek_check_memory_block_frontdoor ( trek_mem_DDR0+0x0001d9f0, trek_mem_RT+0x000336e0, 8 ) begin
     WMEM32(trek_mem_RT+0x00023c34, 0);
     WMEM64(trek_mem_RT+0x000336e0, 0xffff0000ffffULL);
     WMEM32_SHARED(trek_cpu_0_0_0_test_T0_state, 0x10);
```

Running test /home/etienne/projects_sup/armv8/run/Q03.gcc64.qemu.output.log

*<nWave:2> /tmp/demo/run2/aes_dmac_uart_mp.fsdb

Breker TrekApp SystemVIP Library



The Breker Configurable TrekApp Library

- The ***Cache Coherency TrekApp 2.0*** verifies cache and system-level coherency in a multiprocessor SoC
- The ***ARM TrekApp*** automated integration testing of ARM based systems
- The ***RISC-V TrekApp*** handles typical processor integration issues for the RISC-V open ISA
- The ***Power Management TrekApp*** automates power domain switching verification
- The ***Security TrekApp*** automates testing of hardware access rules for HRoT fabrics
- The ***Networking TrekApp*** automates packet generation, dissection and prediction

Breker: Your Verification GPS

- Effective test content composition is the toughest verification challenge and checks/coverage is one of the most onerous activities
- Checks and coverage models may be automated via test suite synthesis leveraging an abstract executable specification
- Test Suite Synthesis automation for both UVM and SoC verification has been proven to save 5X resources while increasing coverage significantly

For more Information:

www.brekersystems.com

Thanks for Listening!
Any Questions?

brekersystems.com/resources/case-studies