Verification of a Cache Coherent system with an A53 cluster using ACE VIP with Graph Based Stimulus

Galen Blake
Perry Wobil
Altera Corporation

September 18, 2015
Austin, TX

Agenda

Verification of Cache Coherent Access from an FPGA

Graph Based Multi Context Stimulus Concepts

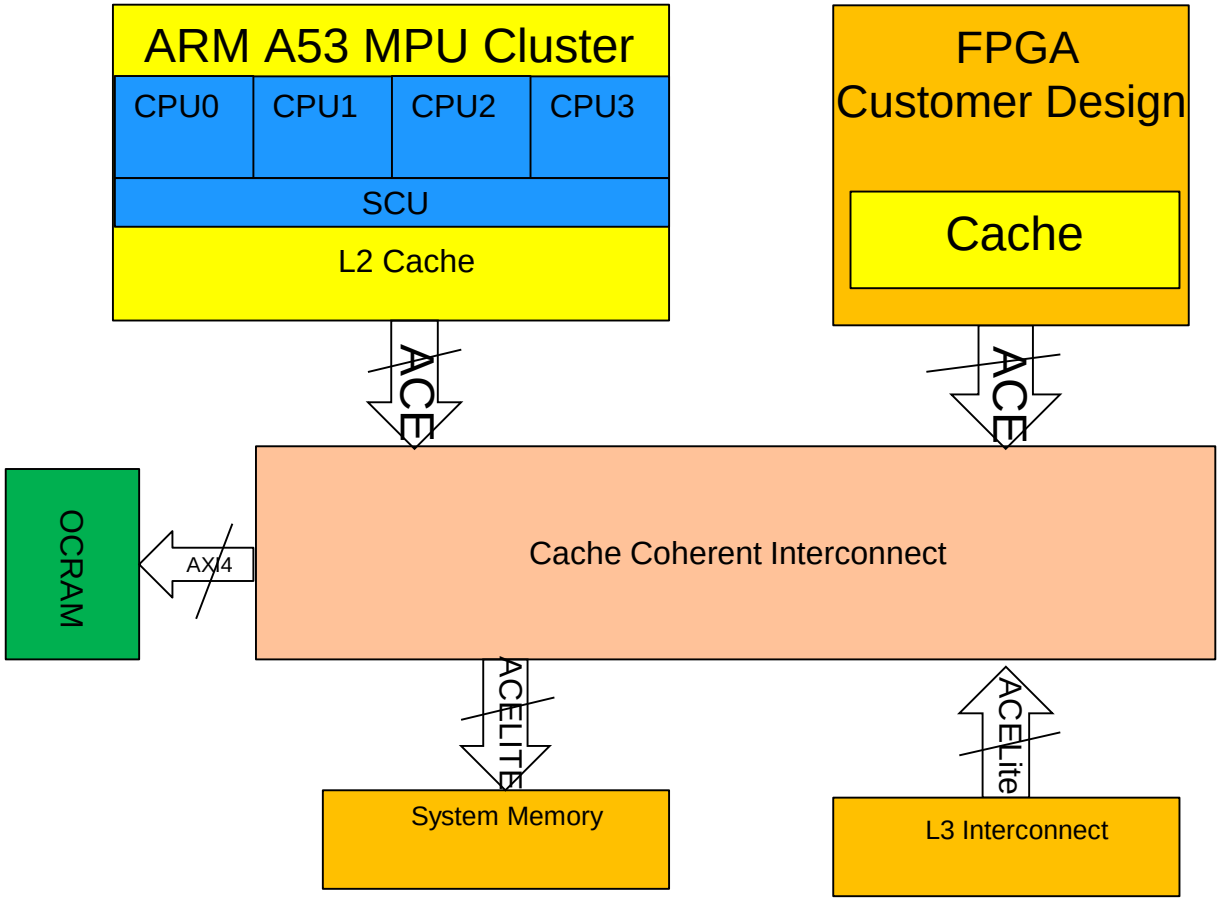
Application to a Cache Coherent SOC-FPGA system

Verification of Cache Coherent Access from an FPGA

System Description

- An ARM quad A53 cluster
- A Cache Coherent Interconnect to coordinate coherent transactions
- Customer Defined V8 compliant ACE agent synthesized into the FPGA soft Logic

Block Diagram of Test Bench & DUT



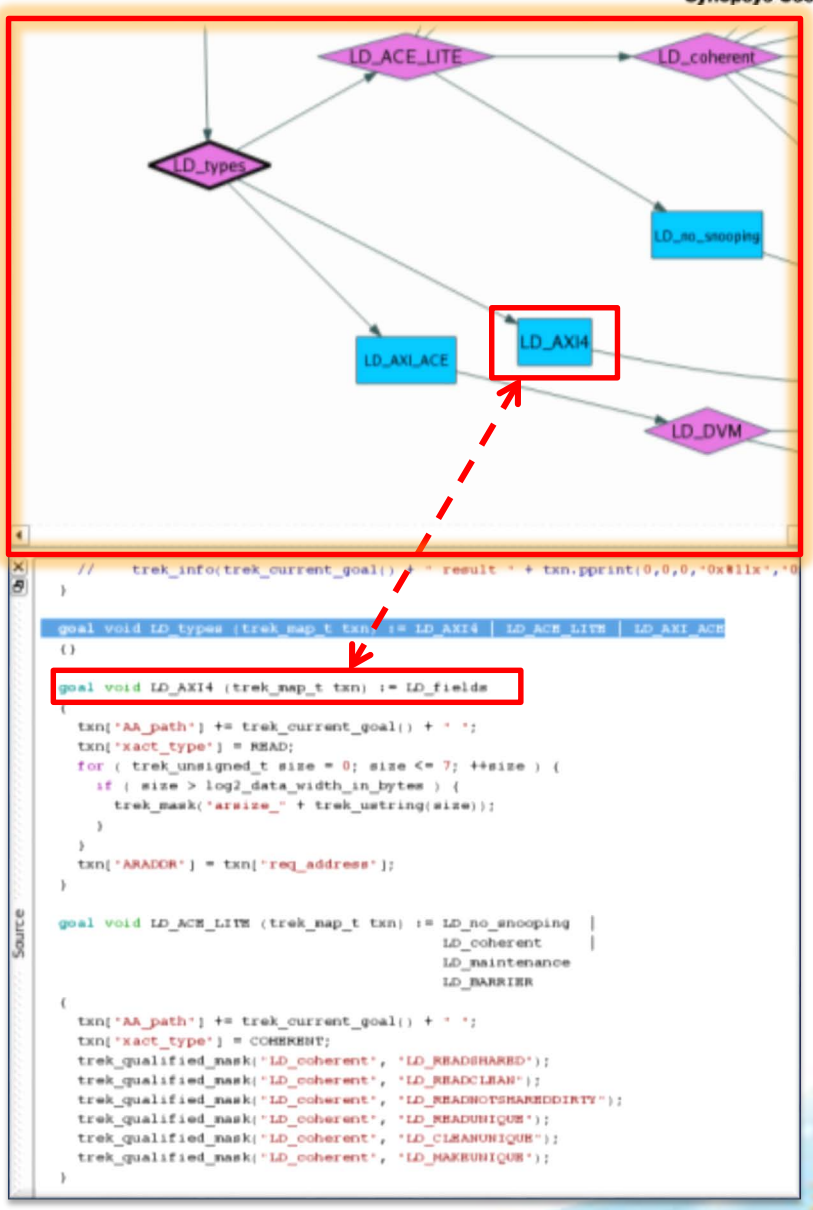
Verification Challenges

- How do we model a customer defined ACE agent
  - In our case we chose the Synopsys ACE VIP
  - It can be configured in many different ways to match potential customer configurations
  - A key feature needed is the local cache to support all ACE coherent transactions that require data allocation.
- How do we generate C and UVM stimulus, coordinate and check cacheable transactions
  - In our case we chose the Breker Trek-SoC tool
  - It is capable of generating C code for the A53 and UVM sequences for the ACE VIP and automatically coordinates and checks the coherent traffic from each of them

Graph Based Stimulus Concepts

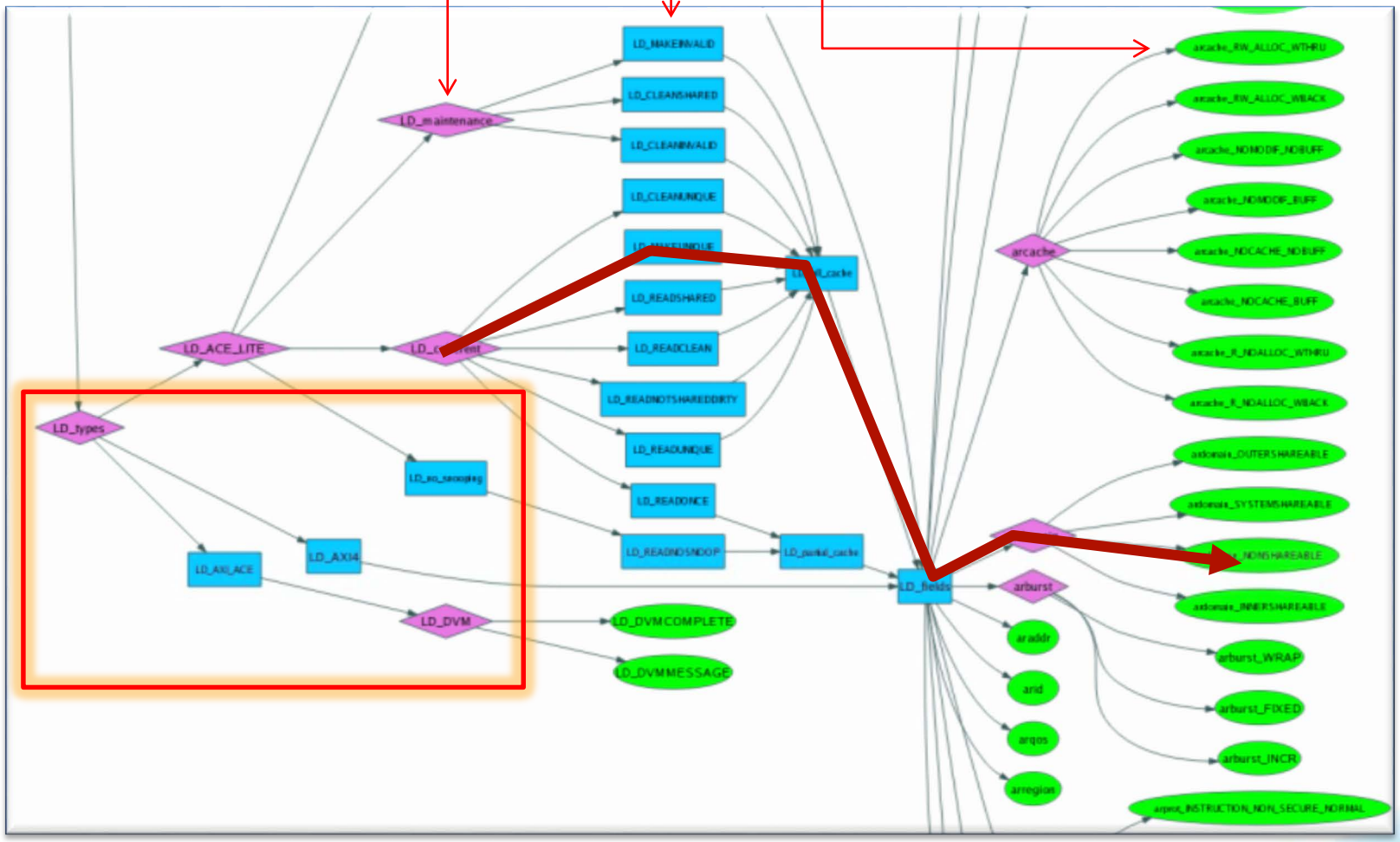
Graph Construction

- Graphs are constructed from a C like language that describes a functional space or protocol
- The language is then used to construct a graph
- The graph can be used to review for accuracy and documentation
- This is an example of an ACE graph



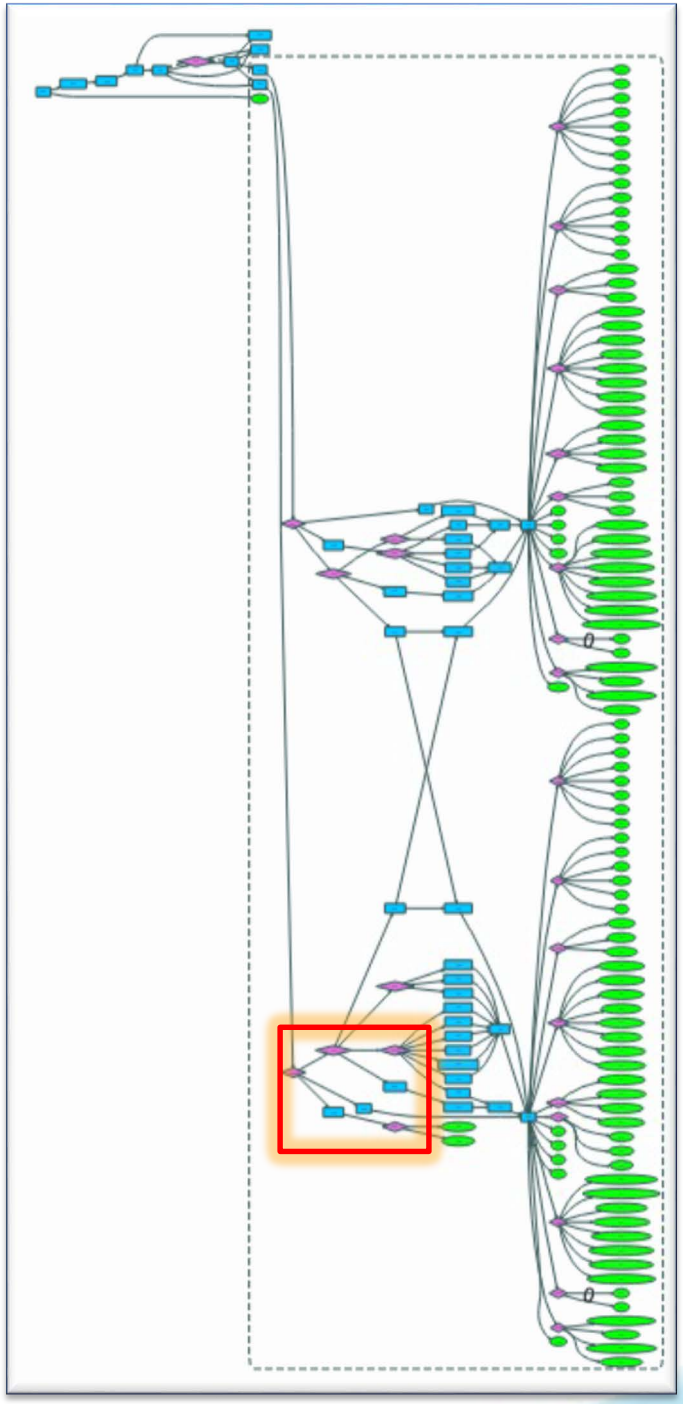
Graph Elements

- Decision points
- Transactions
- Transaction Properties



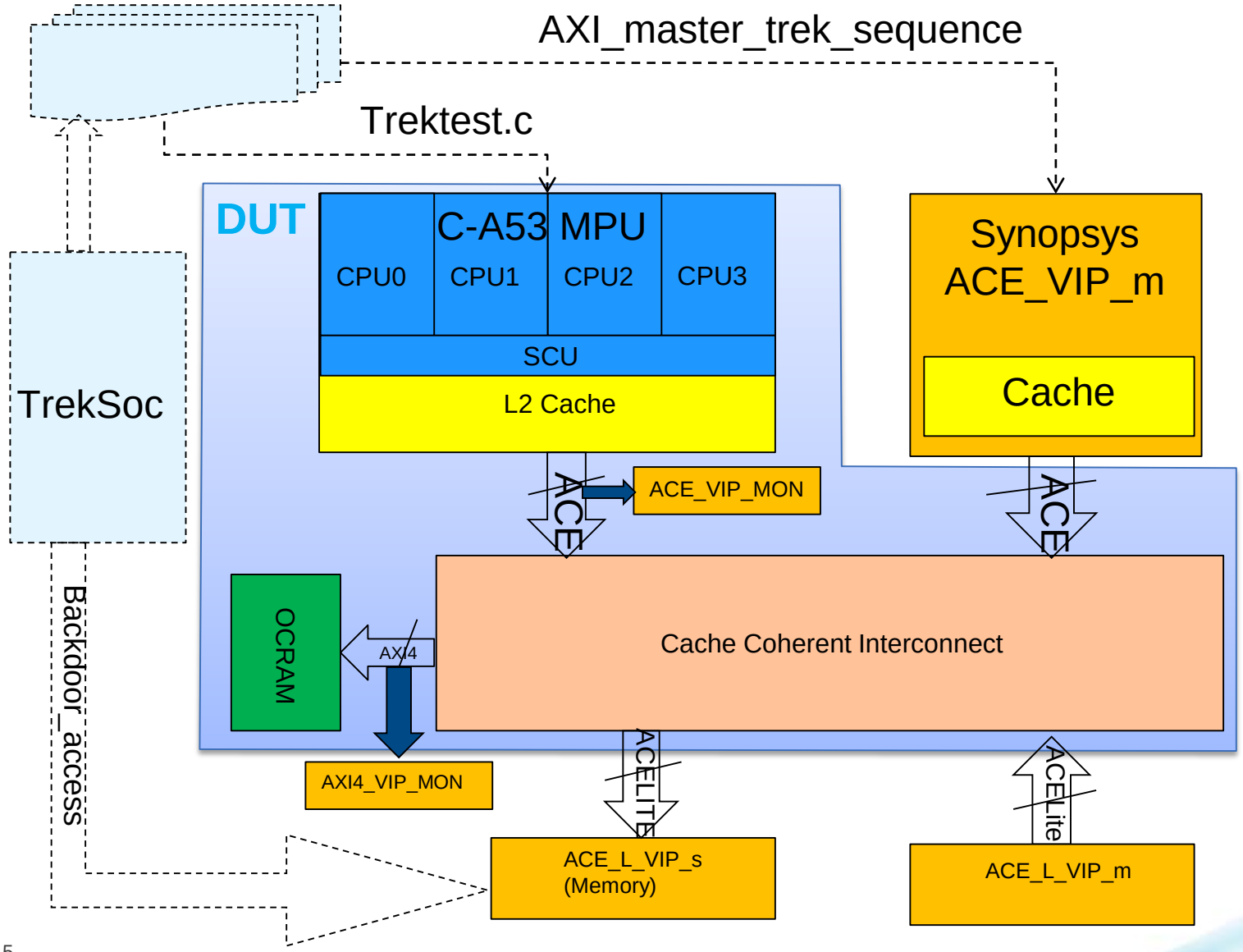
Graph Construction

- A big picture view of the complete ACE graph
- The language used is currently proprietary
- There is an active working group in Accelerra [pswg] working to standardize this type of language

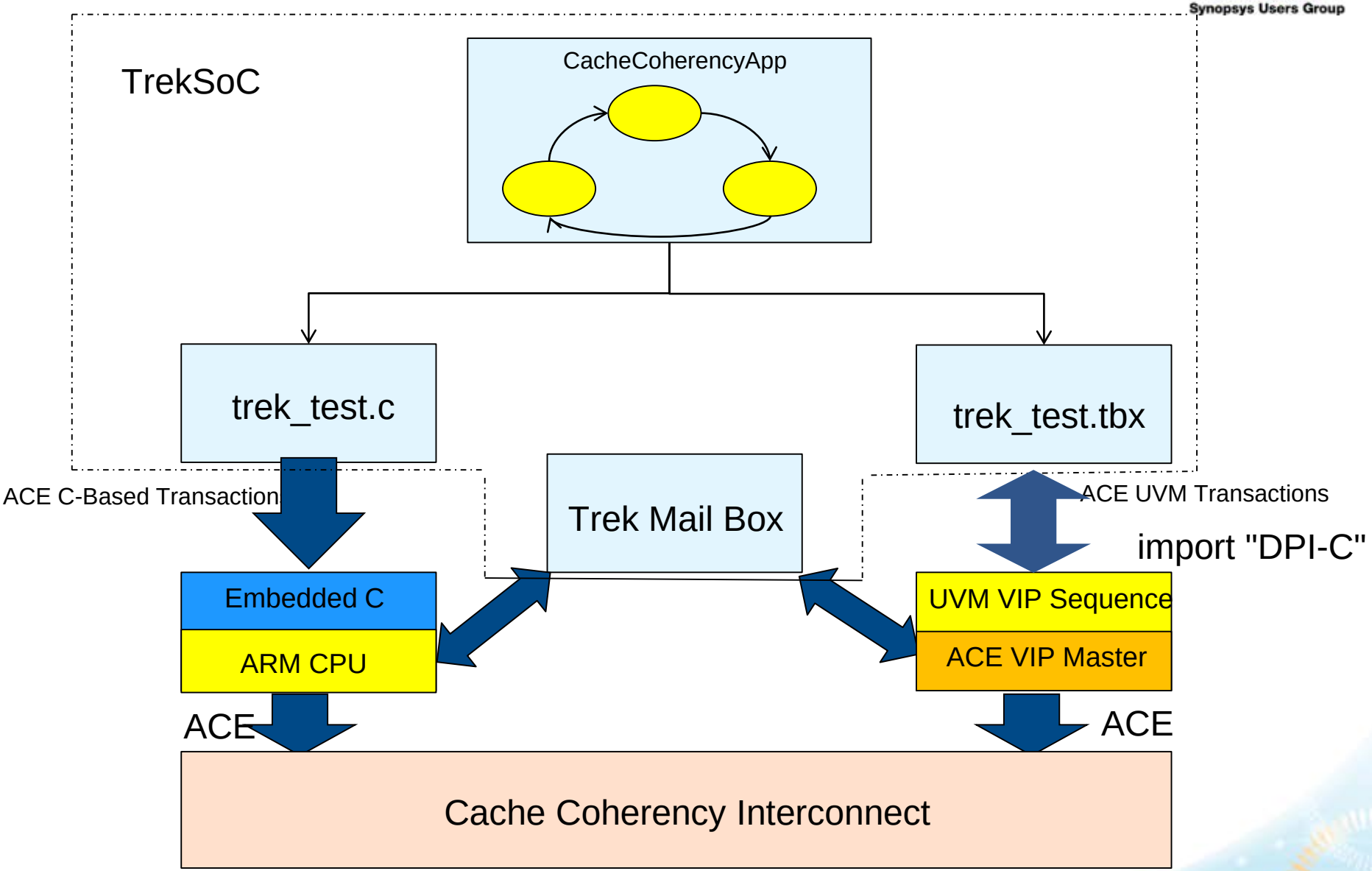


Application to a Cache Coherent Interconnect System

Block Diagram of Test Bench & DUT

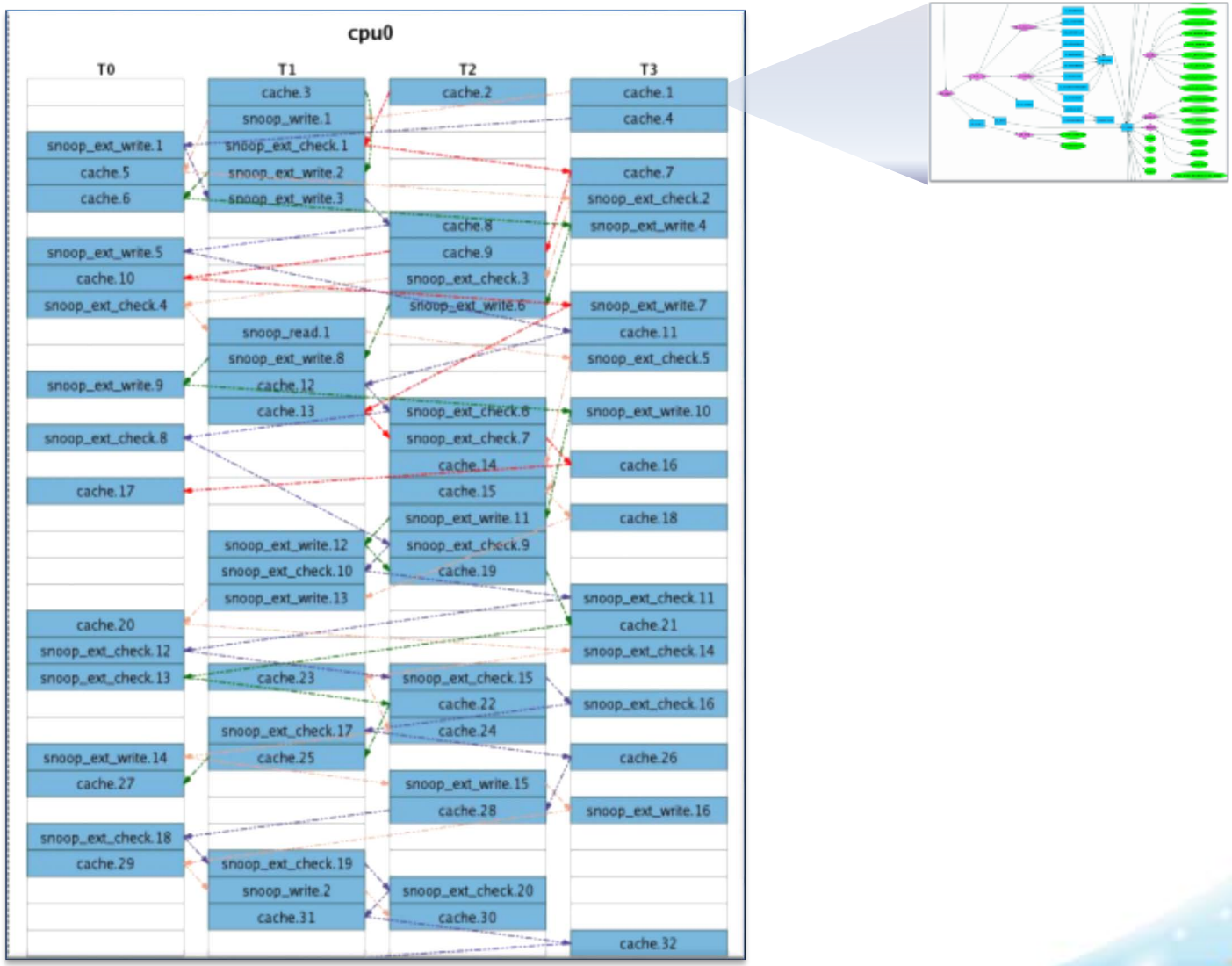


Stimulus Generation & Synchronization

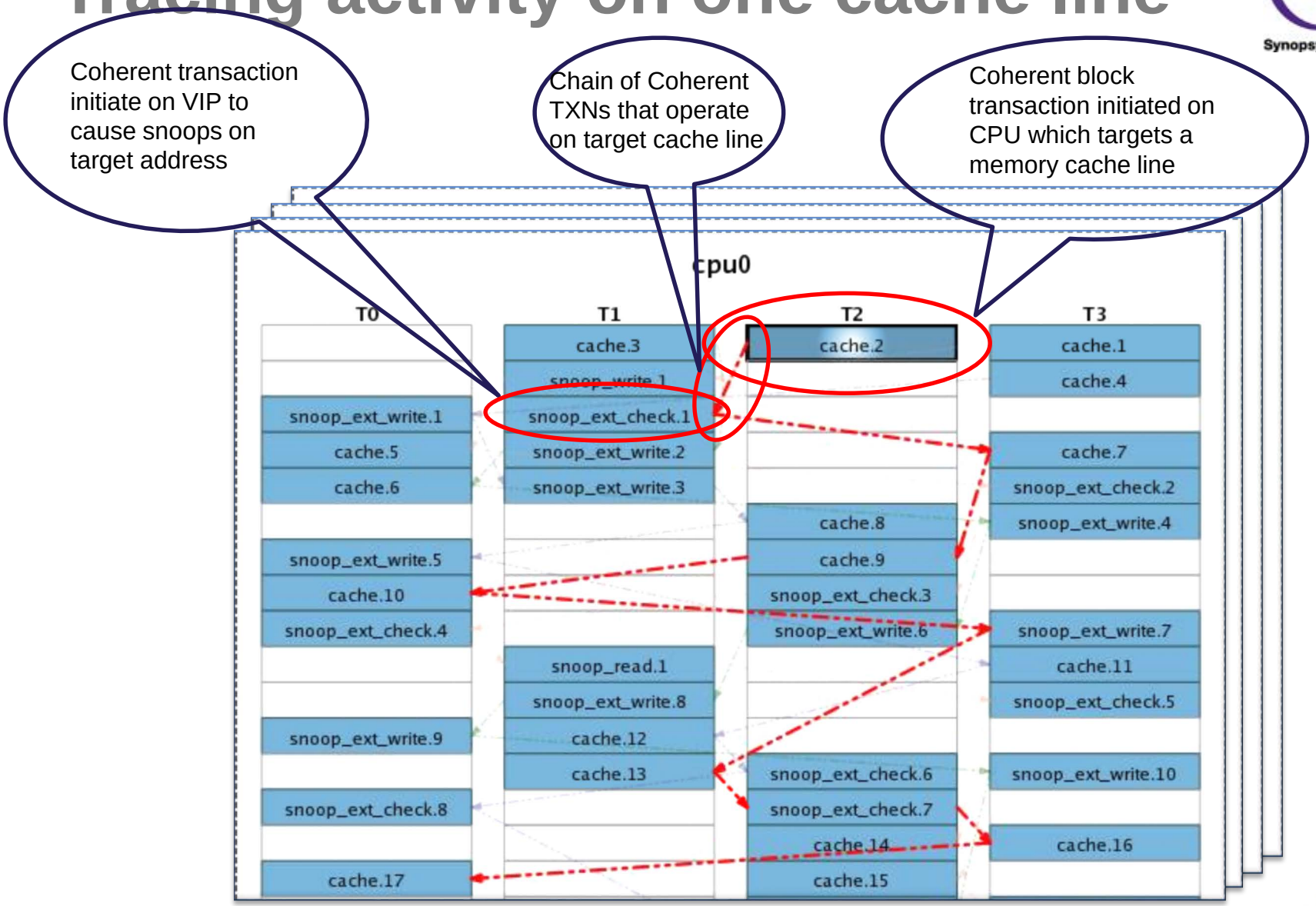


Big picture view.

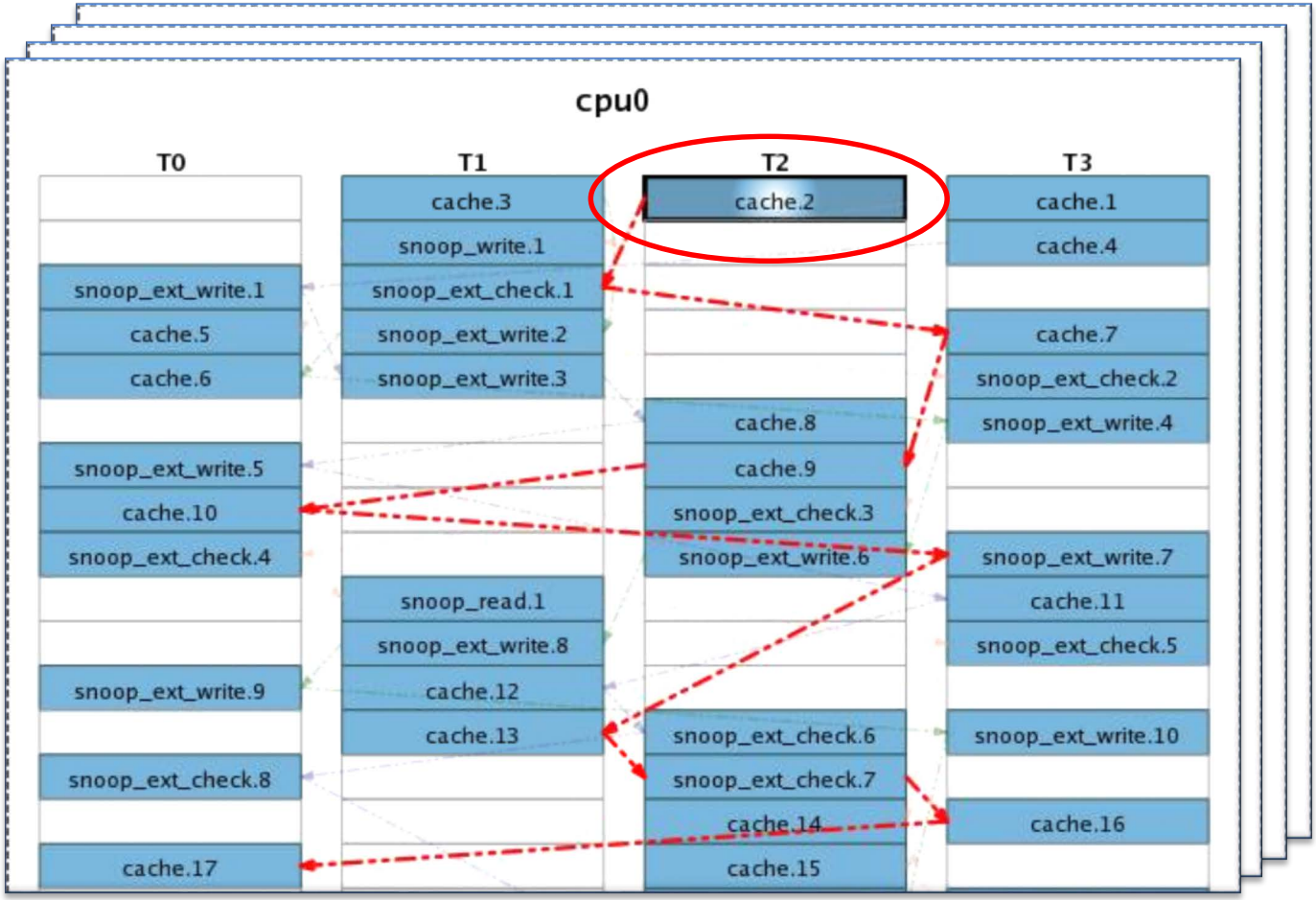
Shows A53 and UVM events



Tracing activity on one cache line



Tracing activity on one cache line

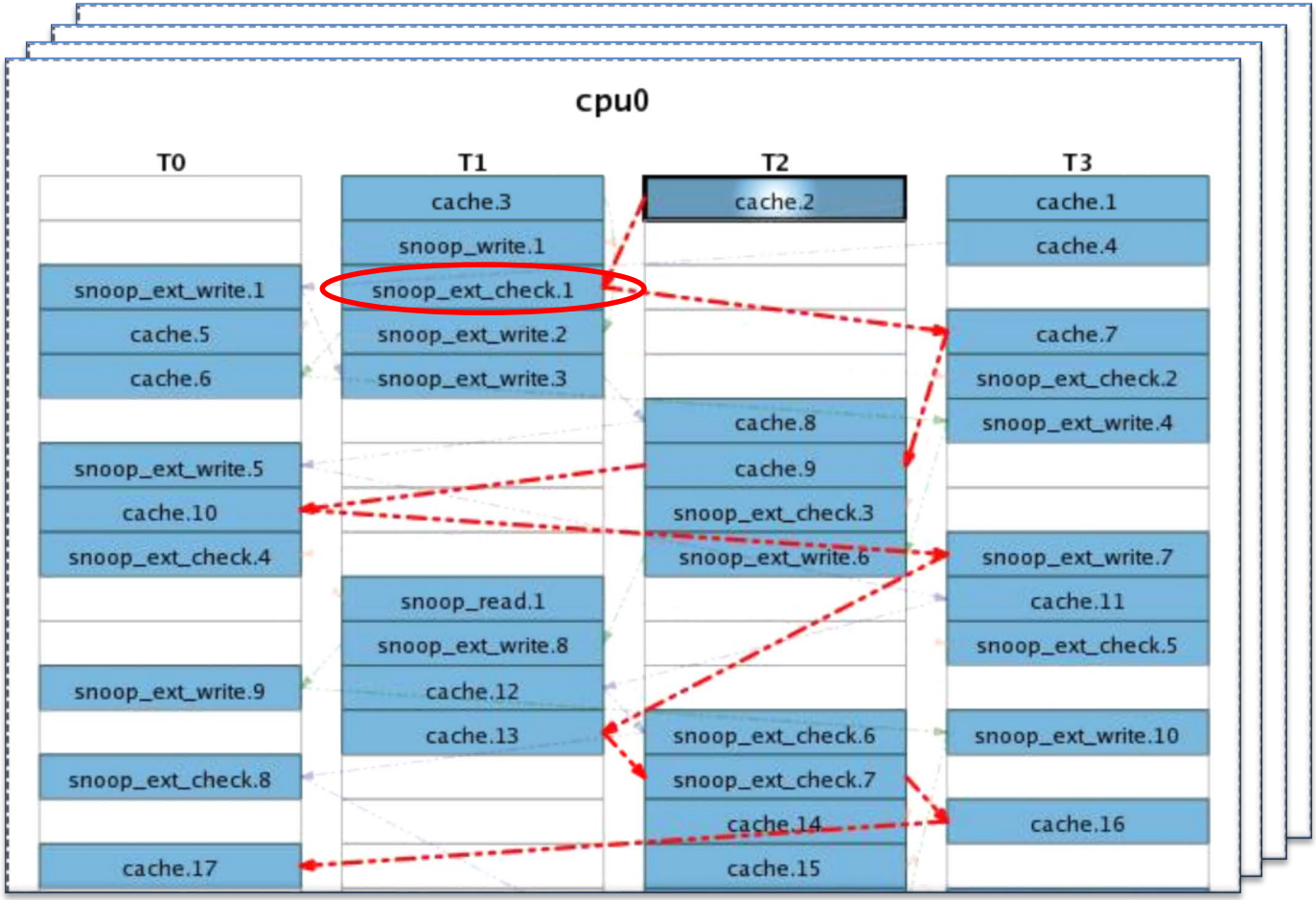


Details about CPU based events

Synopsys Users Group

```
/*****************************************************************************/
// cache.2

    trek_tbx_event( cpu0_mbox, 159 );    // [event:159 cpu:cpu0 thread:T2 instance:cach
        /* tbx: trek_message ("Begin cache.2");  */
    /* initialize data for chain */
    WMEM32 ( trek_mem_APS2MPFE+0x6fffff40, 0x7f1e07db ); // trek_write_memory ()
    /* READ MISS (exclusive) on trek_mem_APS2MPFE+0x6fffff40 */
    trek_nop_delay ( 6 );  // pre-delay
    if ( MEM32 ( trek_mem_APS2MPFE+0x6fffff40 ) != 0x7f1e07db ) {  // trek_check_memo
      trek_tbx_arg ( cpu0_mbox, MEM32 ( trek_mem_APS2MPFE+0x6fffff40 ) );
      trek_tbx_event( cpu0_mbox, 160 ); // [event:160 cpu:cpu0 thread:T2 instance:cach
        /* tbx: trek_check_memory ( trek_mem_APS2MPFE+0x6fffff40, 0x7f1e07db, 2 );  */
    }

    /* READ HIT on trek_mem_APS2MPFE+0x6fffff40 */
    if ( MEM32 ( trek_mem_APS2MPFE+0x6fffff40 ) != 0x7f1e07db ) {  // trek_check_memo
      trek_tbx_arg ( cpu0_mbox, MEM32 ( trek_mem_APS2MPFE+0x6fffff40 ) );
      trek_tbx_event( cpu0_mbox, 161 ); // [event:161 cpu:cpu0 thread:T2 instance:cach
        /* tbx: trek_check_memory ( trek_mem_APS2MPFE+0x6fffff40, 0x7f1e07db, 2 );  */
    }

    /* EXCLUSIVE expected for trek_mem_APS2MPFE+0x6fffff40 */
    trek_tbx_event( cpu0_mbox, 162 );    // [event:162 cpu:cpu0 thread:T2 instance:cach
        /* tbx: trek_message ("End cache.2");  */
    WMEM32_SHARED(trek_cpu0_test_T2_state, 0x2);
    break;
  case (0x2):

/*****************************************************************************/
```

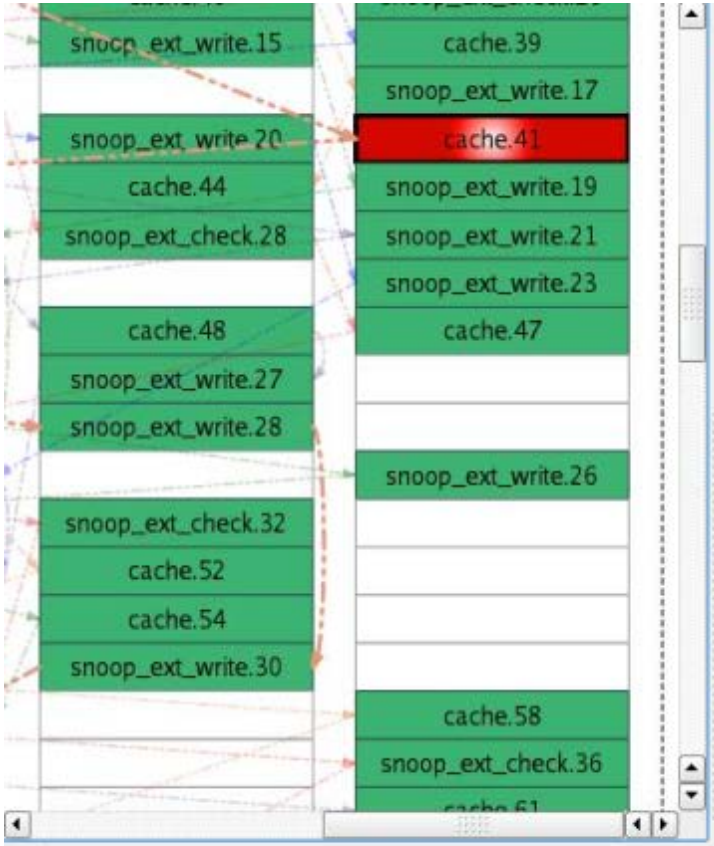
Tracing activity on one cache line



Details about a VIP (UVM) event

```
// snoop_ext_check.1
    WMEM32_SHARED(trek_cpu0_test_T1_state, 0x5);
  case (0x5):     // wait for cache.2
    if (MEM32_SHARED(trek_cpu0_test_T2_state) < 0x2) break;

    trek_tbx_event( cpu0_mbox, 89 );     // [event:89 cpu:cpu0 thread:T1 instance:snoop_ext_check.1]
        /* tbx: trek message ("Begin snoop ext check.1");  */
    /* vip check (trek_mem_APS2MPFE+0x6fffff40, 4, 7f1e07db) */
    /* READONCE axi0.LD_partial_cache ARLOCK=NORMAL ARCACHE=R_NOALLOC_WBACK ARDOMAIN=INNERSHAREABLE
ARPROT=DATA_NON_SECURE_PRIVILEGED  */
    WMEM32_UNCACHED ( trek_mem_APS2MPFE+0x6e88a640, 0 );
    trek_tbx_event( cpu0_mbox, 90 );     // [event:90 cpu:cpu0 thread:T1 instance:snoop_ext_check.1]
        /* tbx: trek_drive_io_txn_and_wait (AA_path, ARADDR, ARBURST, ARCACHE, ARDOMAIN, ARID, ARLEN
ARREGION, ARSIZE, ARUSER, burst_size_in_bytes, coherent_xact_type, direction, port, read_strb, req_a
total_length_in_bytes, valid_length_in_bytes, xact_type);
            AA_path = READONCE axi0.LD_partial_cache ARLOCK=NORMAL ARCACHE=R_NOALLOC_WBACK ARDOMA
ARBURST=INCR ARPROT=DATA_NON_SECURE_PRIVILEGED
            ARADDR = 2147483456
            ARBURST = 1
            ARCACHE = 11
            ARDOMAIN = 1
            ARID = 8
            ARLEN = 1
            ARLOCK = 0
            ARPROT = 3
            ARQOS = 0
            ARREGION = 0
            ARSIZE = 2
            ARUSER = 2
            burst_size_in_bytes = 4
            coherent_xact_type = 1
            direction = LOAD
            port = fpga2aps_ace_master
            read_strb = _ff
            req_address = 2147483456
            req_length_in_bytes = 4
            total_length_in_bytes = 4
            valid_length_in_bytes = 4
            xact_type = 3  */
```

Runtime view of a failed event




```
143543022 trek: info:
trek_check_memory (
trek_mem_APS2MPFE+0x04ad9dc0,
0x482aee54, 2 ); [event:922 cpu:cpu0
thread:T3 instance:cache.41]
143543022 trek: error:     mismatch
expected: 0x482aee54, actual:
0x5995ee54 [event:922 cpu:cpu0
thread:T3 instance:cache.41]
```

Log

C Test Source

```
    trek_tbx_event( cpu0_mbox, 922 );    //
[event:922 cpu:cpu0 thread:T3 instance:cache.41]
        /* tbx: trek_check_memory (
trek_mem_APS2MPFE+0x04ad9dc0, 0x482aee54, 2 );
*/
    /* EXCLUSIVE expected for
trek_mem_APS2MPFE+0x04ad9dc0 */
    /* READ HIT on trek_mem_APS2MPFE+0x04ad9dc0
*/
    trek_tbx_arg ( cpu0_mbox, MEM32 (
trek_mem_APS2MPFE+0x04ad9dc0 ) );
    trek_tbx_event( cpu0_mbox, 923 );    //
[event:923 cpu:cpu0 thread:T3 instance:cache.41]
        /* tbx: trek_check_memory (
trek_mem_APS2MPFE+0x04ad9dc0, 0x482aee54, 2 );
*/
    /* EXCLUSIVE expected for
trek_mem_APS2MPFE+0x04ad9dc0 */
    trek_tbx_event( cpu0_mbox, 924 );    //
[event:924 cpu:cpu0 thread:T3 instance:cache.41]
        /* tbx: trek_message ("End cache.41");
*/
    WMEM32_SHARED(trek_cpu0_test_T3_state,
0x3c);
    break;
  case (0x3c):

/*****************************************
**********************************/
// snoop_ext_write.19
    WMEM32_SHARED(trek_cpu0_test_T3_state,
```

Conclusion

- It only took one week to complete the following
  - Install the Breker Trek tool
  - Integrate the tool into embedded SW (Make) flow
  - Integrate the tool with the Synopsys VIP UVM sequences
  - Develop highly complex coherent system traffic

- We also found an RTL bug that second week

- Since that time we have also deployed both the Breker Trek and Synopsys ACE and DDR VIP to develop complex traffic for our memory sub system

- Currently working to extend this verification methodology of graph based stimulus and UVM components to “non-AMBA” areas of SOC

Thank You