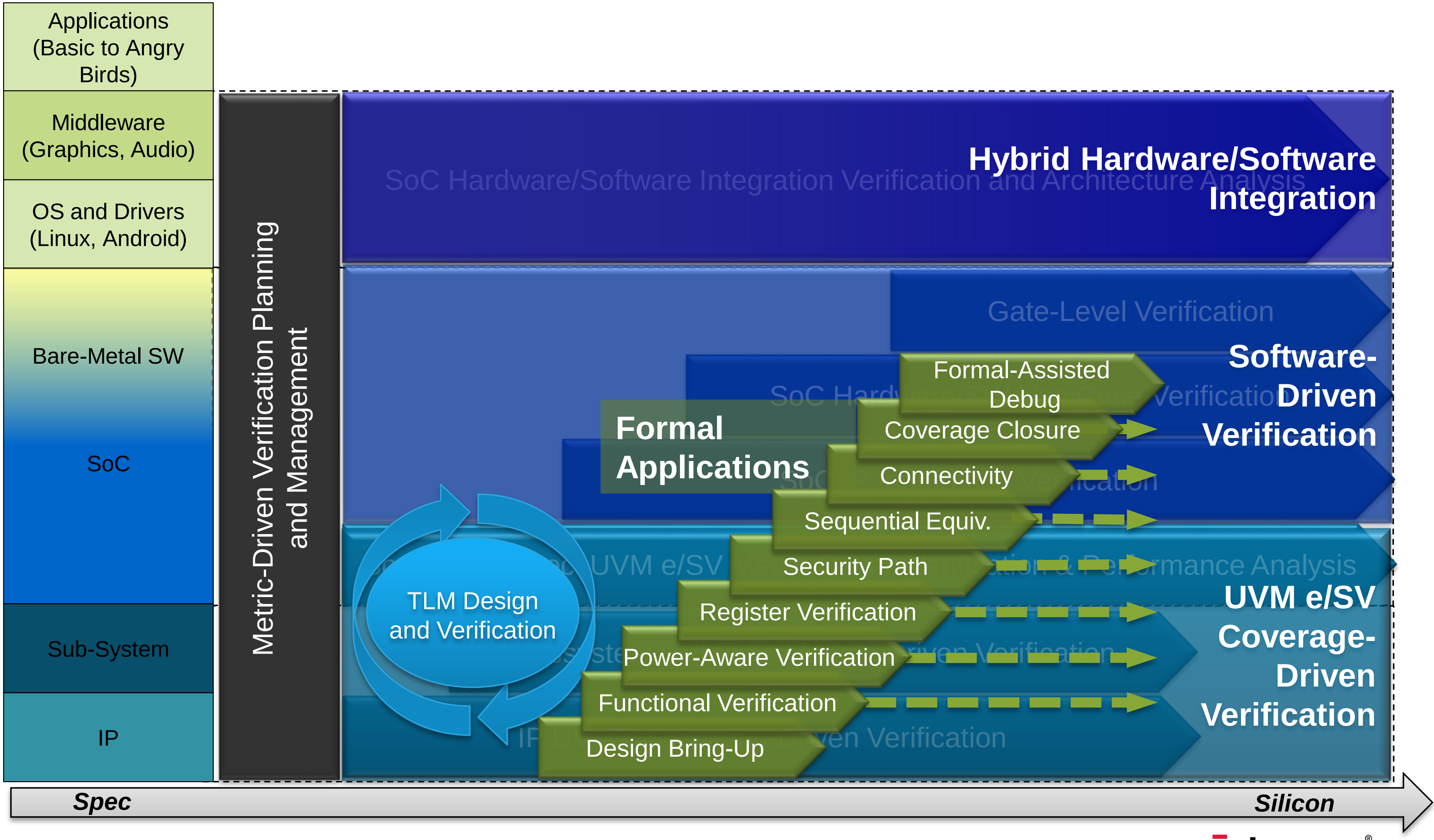
Perspec System Verifier Overview

June 2015

IP to SoC hardware/software integration and verification flows—Cadence methodology and focus



Applications (Basic to Angry Birds)

Middleware (Graphics, Audio)

OS and Drivers (Linux, Android)

Bare-Metal SW

SoC

Sub-System

IP

Metric-Driven Verification Planning and Management

SoC Hardware/Software Integration Verification and Architecture Analysis

Hybrid Hardware/Software Integration

Gate-Level Verification

SoC Hardware/Software Verification

Software-Driven Verification

SoC ... Verification

Formal Applications

Formal-Assisted Debug

Coverage Closure

Connectivity

Sequential Equiv.

Security Path

Register Verification

Power-Aware Verification

Functional Verification

Design Bring-Up

TLM Design and Verification

SoC UVM e/SV ... Integration & Performance Analysis

Sub-System ... Driven Verification

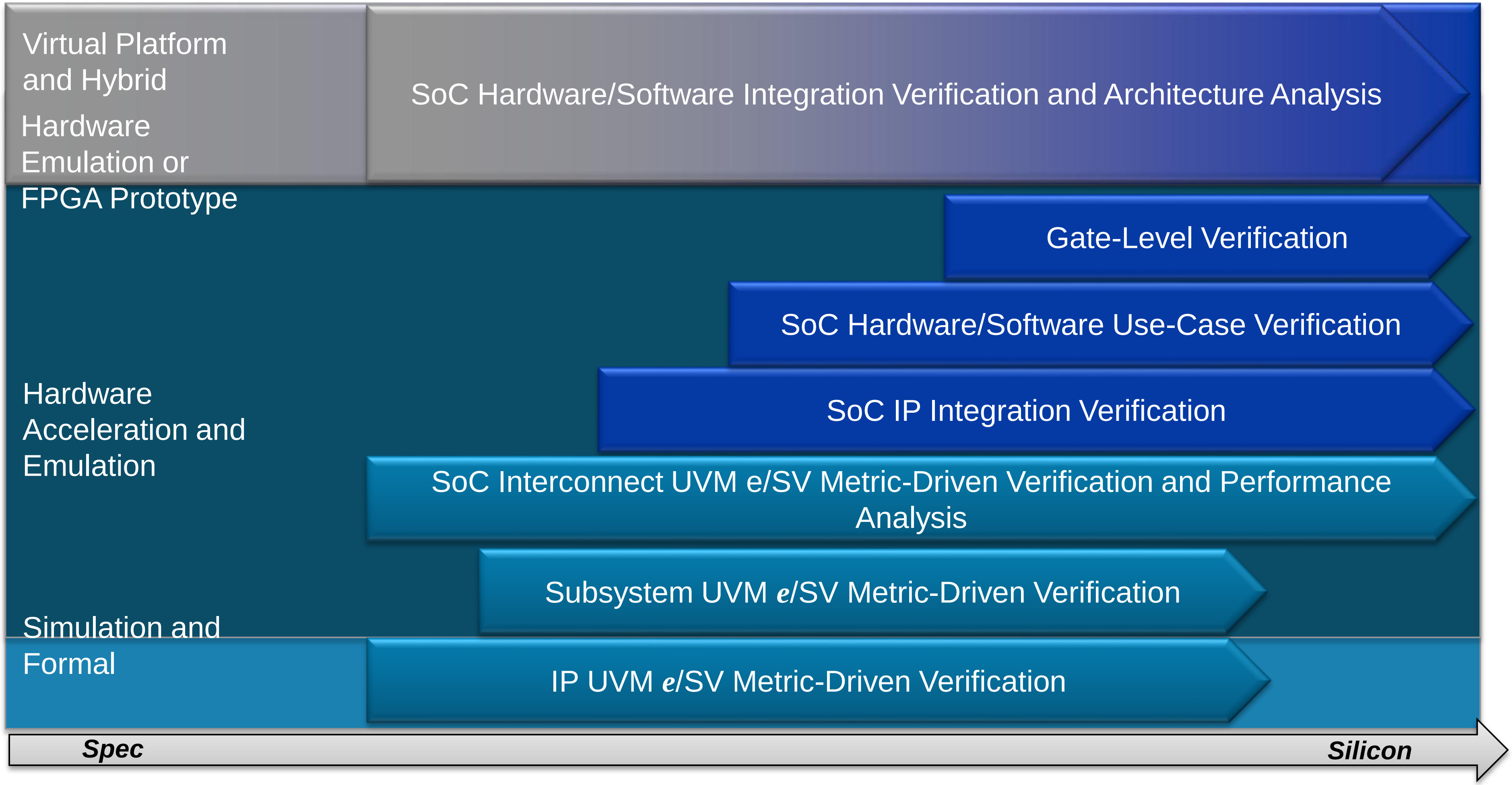
IP UVM ... Driven Verification

UVM e/SV Coverage-Driven Verification

Spec

Silicon

IP to SoC pre-silicon verification platforms



Virtual Platform and Hybrid

Hardware Emulation or FPGA Prototype

SoC Hardware/Software Integration Verification and Architecture Analysis

Gate-Level Verification

SoC Hardware/Software Use-Case Verification

Hardware Acceleration and Emulation

SoC IP Integration Verification

SoC Interconnect UVM e/SV Metric-Driven Verification and Performance Analysis

Subsystem UVM *e*/SV Metric-Driven Verification

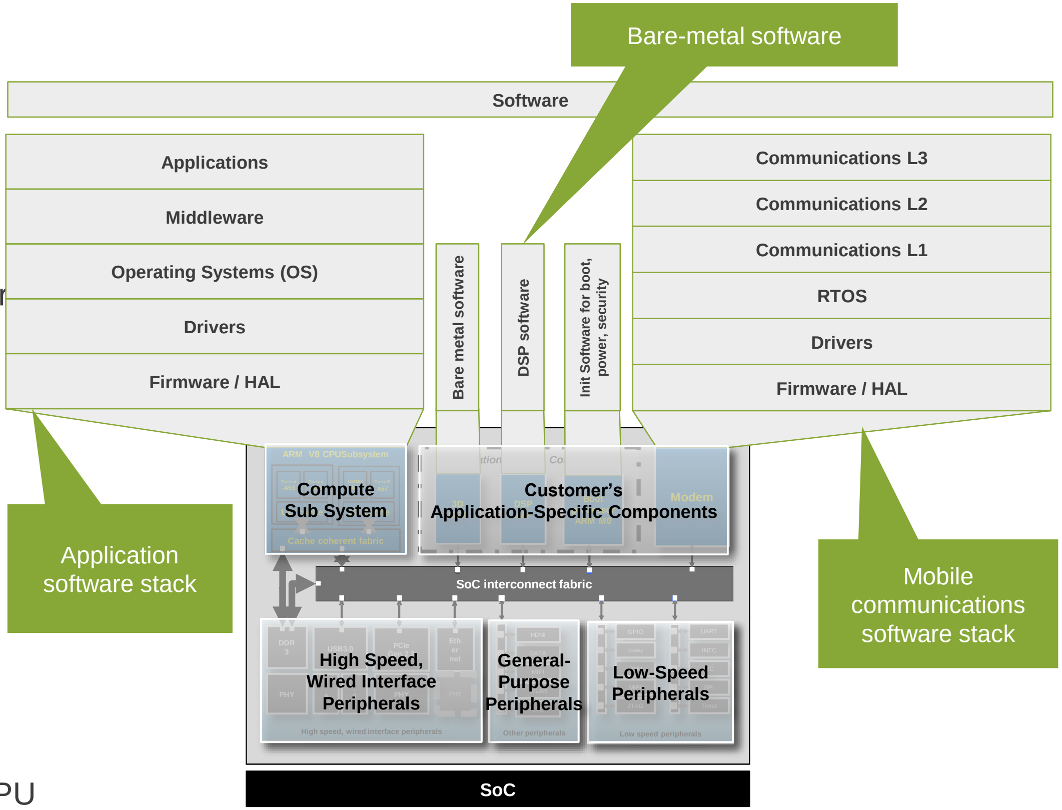
Simulation and Formal

IP UVM *e*/SV Metric-Driven Verification

Perspec System Verifier Overview

A system-centric look at a modern SoC

- Many IPs
  - Standard I/O
    - Wifi, USB, PCI Express® (PCIe®), etc.
  - System infrastructure
    - Interconnect, interrupt control, uart, timer
  - Differentiators
    - custom accelerators, modem…
- Many cores
  - Both symmetric and asymmetric
  - Both homogeneous and heterogeneous
- Lots of software
  - Part of core functionality
    - Communication stack, DSP software, GPU microcode…
  - User application software infrastructure
    - Android, Linux…



Bare-metal software

Software

Applications

Middleware

Operating Systems (OS)

Drivers

Firmware / HAL

Bare metal software

DSP software

Init Software for boot, power, security

Communications L3

Communications L2

Communications L1

RTOS

Drivers

Firmware / HAL

Application software stack

Mobile communications software stack

Compute Sub System

Customer's Application-Specific Components

SoC interconnect fabric

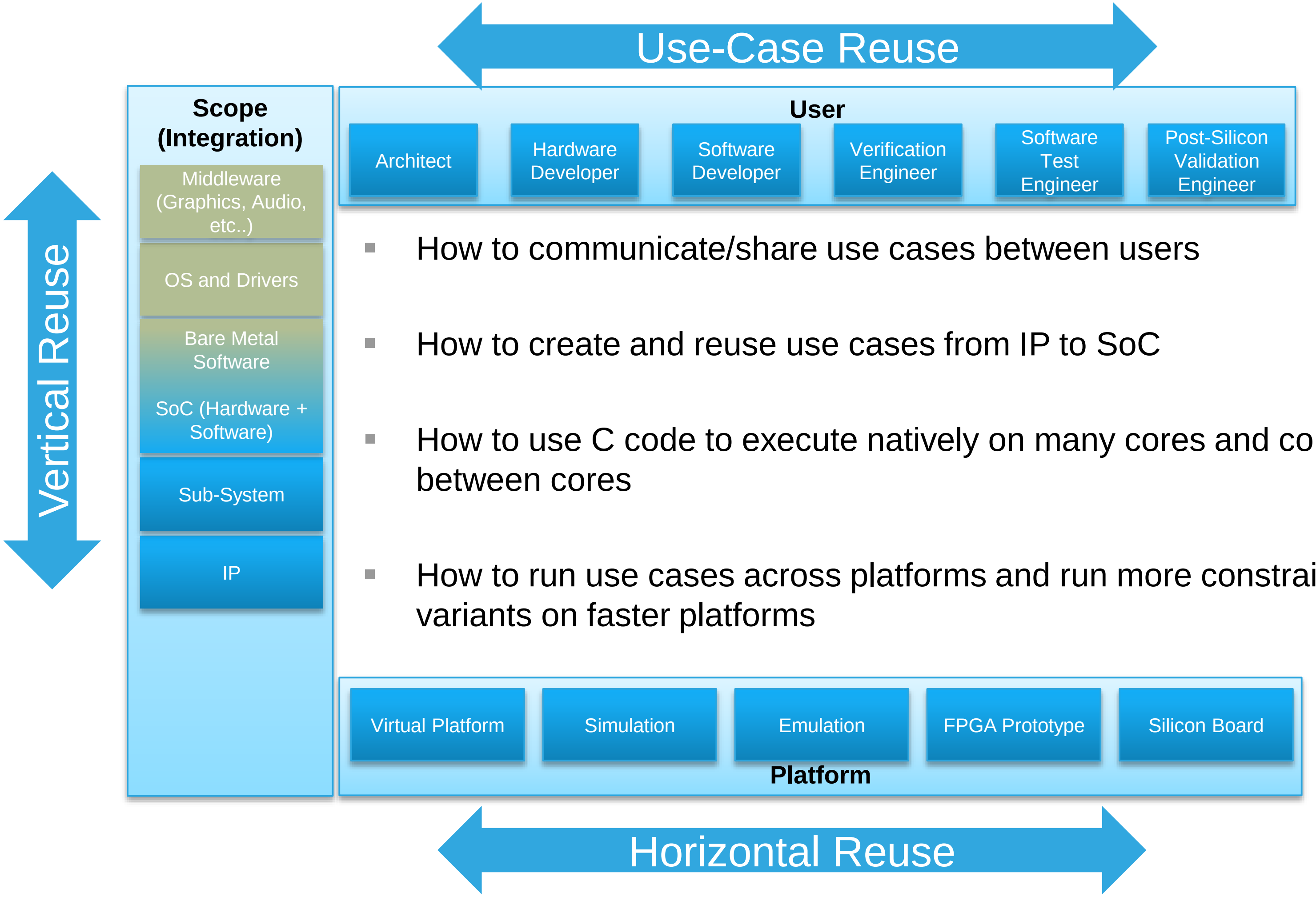
High Speed, Wired Interface Peripherals

General-Purpose Peripherals

Low-Speed Peripherals

SoC

SoC-level verification and validation requirements



Use-Case Reuse

User

Architect | Hardware Developer | Software Developer | Verification Engineer | Software Test Engineer | Post-Silicon Validation Engineer

Scope (Integration)

Middleware (Graphics, Audio, etc..)

OS and Drivers

Bare Metal Software

SoC (Hardware + Software)

Sub-System

IP

Vertical Reuse

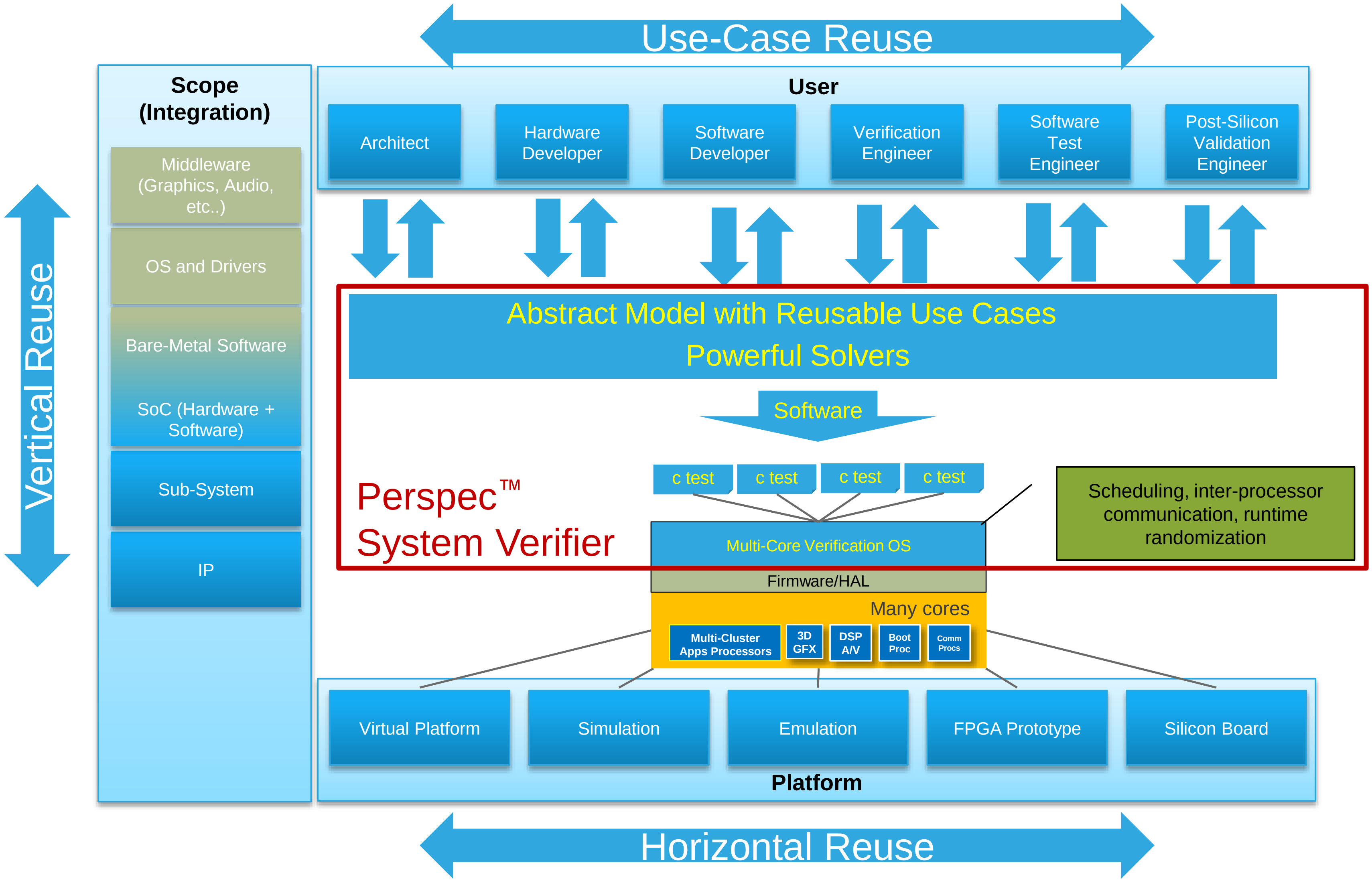
- How to communicate/share use cases between users
- How to create and reuse use cases from IP to SoC
- How to use C code to execute natively on many cores and communicate between cores
- How to run use cases across platforms and run more constrained random variants on faster platforms

Virtual Platform | Simulation | Emulation | FPGA Prototype | Silicon Board

Platform

Horizontal Reuse

The solution: Perspec System Verifier

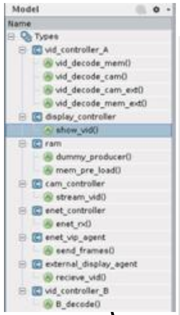


Automated use case verification

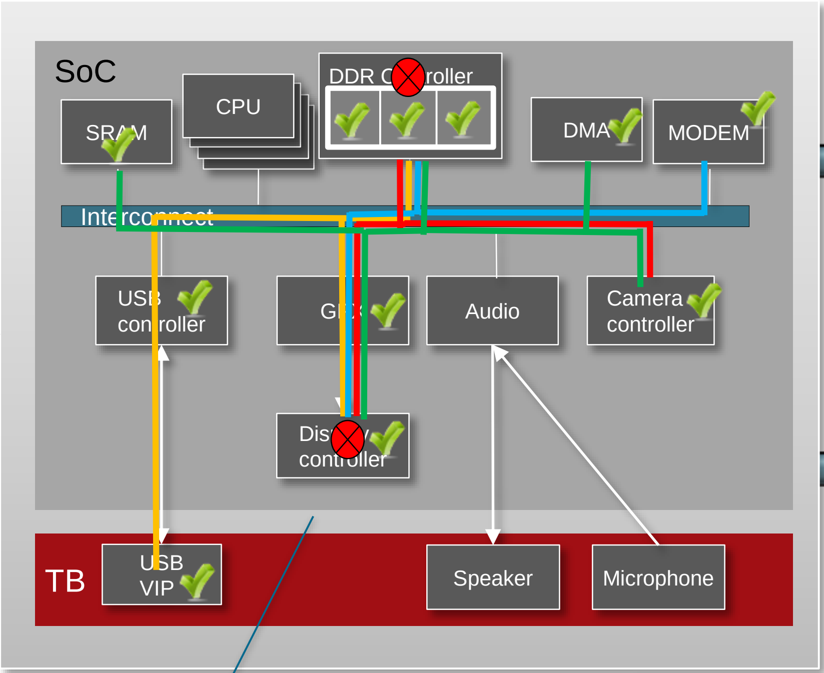
Desired Scenario:

Decode video from the DDR and show on the display

SLN Models

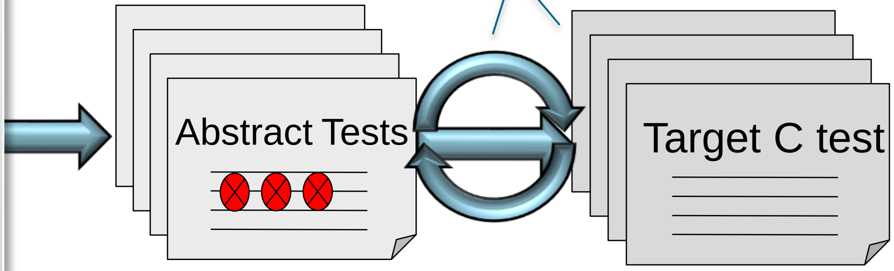


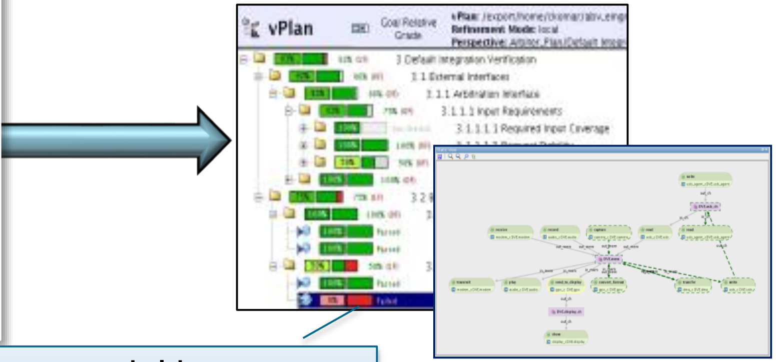
```
Model
Name
⊟ Types
  ⊟ vid_controller_A
      vid_decode_mem()
      vid_decode_cam()
      vid_decode_cam_ext()
      vid_decode_mem_ext()
  ⊟ display_controller
      show_vid()
  ⊟ ram
      dummy_producer()
      mem_pre_load()
  ⊟ cam_controller
      stream_vid()
  ⊟ enet_controller
      enet_rx()
  ⊟ enet_vip_agent
      send_frames()
  ⊟ external_display_agent
      recieve_vid()
  ⊟ vid_controller_B
      B_decode()
```



ge

Re-generate the code for derivatives, spec changes, etc…





Perspec solver checks the feasibility of the goals statically

Coverage model is auto-created and pruned for reachable scenarios

*Perspec™ System Verifier automatically and **exhaustively** completes the goals in to full legal scenarios*

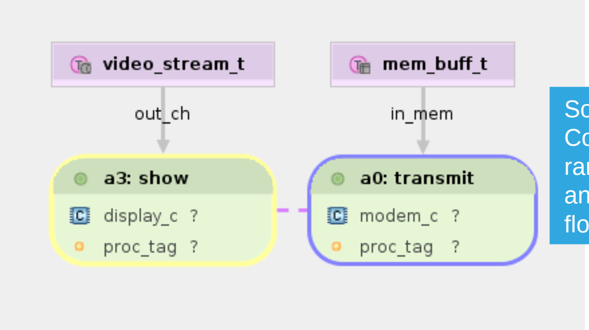
Example use case

Translating end-user use case to system-level bare-metal actions

End-user use case:

Mobile phone requirement:

view a video while uploading it
(6 words)

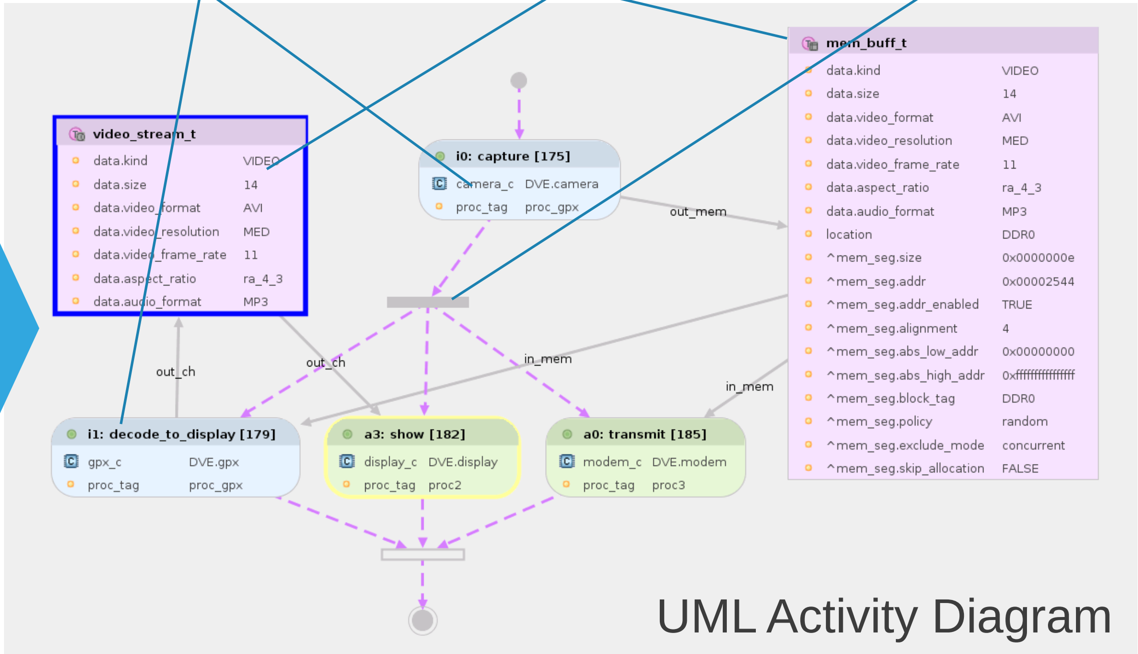


Solver: Constrained random data and control flow

Solver randomly selects legal video source and video stream source

Solver randomly selects legal attributes of video stream and memory buffer

Solver schedules show and transmit in parallel as required by use case and adds decode to display



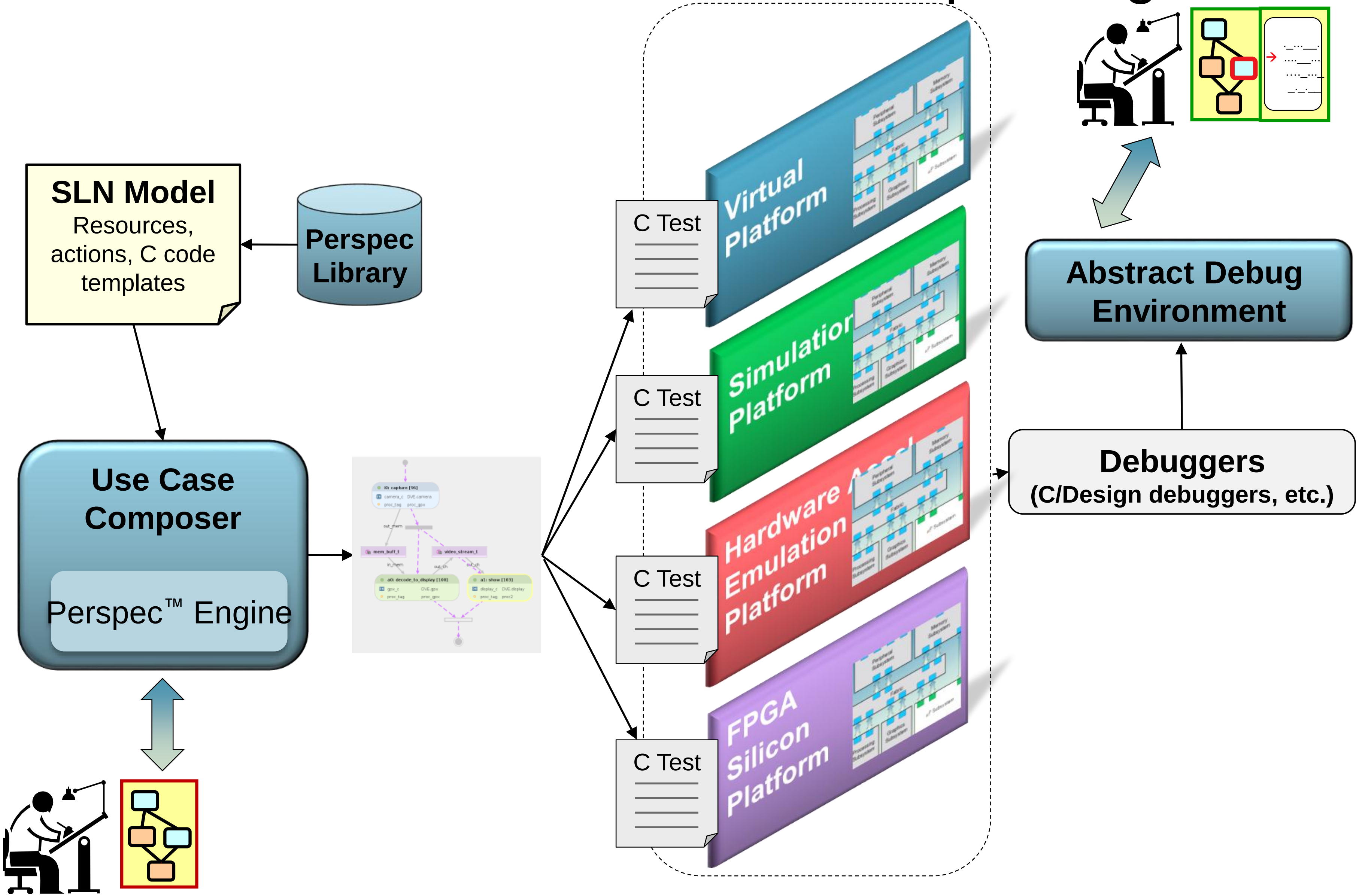
UML Activity Diagram

System-level bare-metal actions:

Capture a video with camera **using graphics processor** and save it to a **memory buffer in DDR0** in **AVI format with medium resolution** and **MPEG3 audio** with **4x3 aspect ratio** then **transmit** the video using the **modem and processor 3** while **processor 2 shows** the video on the built in display being **streamed by the graphics processor** of the video **already saved in DDR0 memory buffer**
(66 words)

Flow

Use-case verification flow with Perspec engine



SLN Model

Resources, actions, C code templates

Perspec Library

Use Case Composer

Perspec™ Engine

C Test

C Test

C Test

C Test

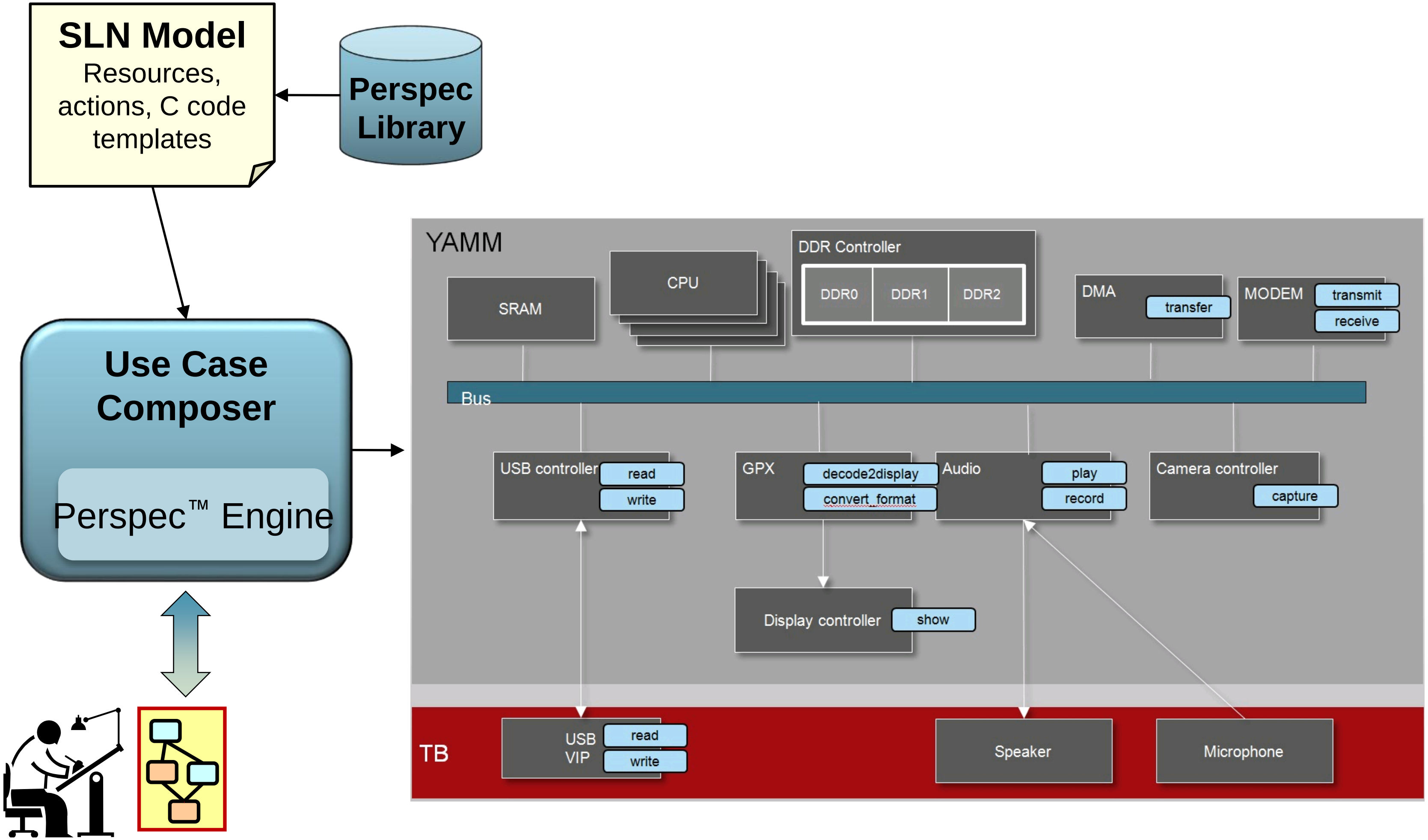
Abstract Debug Environment

Debuggers

(C/Design debuggers, etc.)

cādence®

Step #1: Capture topology and system actions



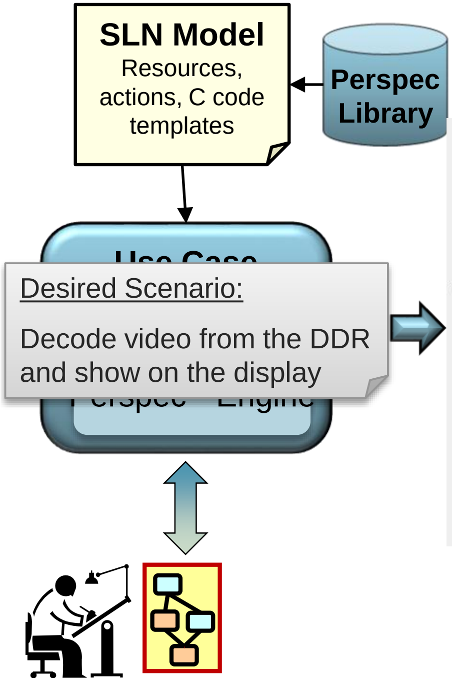
SLN Model
Resources, actions, C code templates

Perspec Library

Use Case Composer

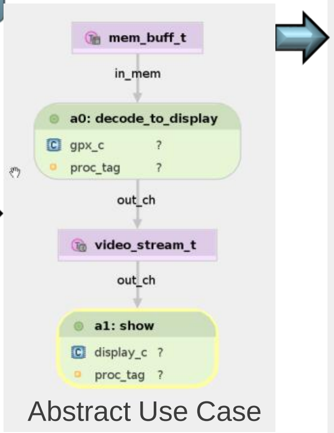
Perspec™ Engine

Step #2: Capture abstract use case >> solve for concrete use case(s) >> analyze gentime coverage

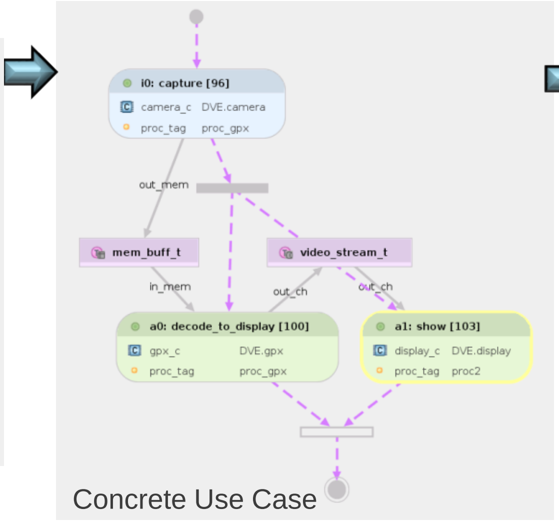


Desired Scenario:

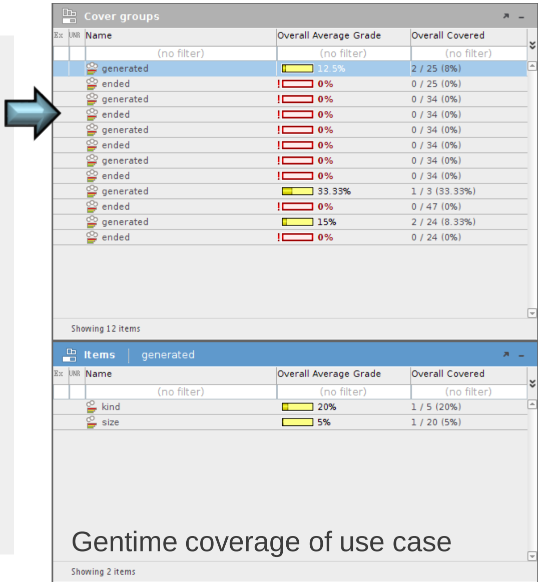
Decode video from the DDR and show on the display



Abstract Use Case



Concrete Use Case



Cover groups

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		generated	12.5%	2 / 25 (8%)
		ended	! 0%	0 / 25 (0%)
		generated	! 0%	0 / 34 (0%)
		ended	! 0%	0 / 34 (0%)
		generated	! 0%	0 / 34 (0%)
		ended	! 0%	0 / 34 (0%)
		generated	! 0%	0 / 34 (0%)
		ended	! 0%	0 / 34 (0%)
		generated	33.33%	1 / 3 (33.33%)
		ended	! 0%	0 / 47 (0%)
		generated	15%	2 / 24 (8.33%)
		ended	! 0%	0 / 24 (0%)

Showing 12 items

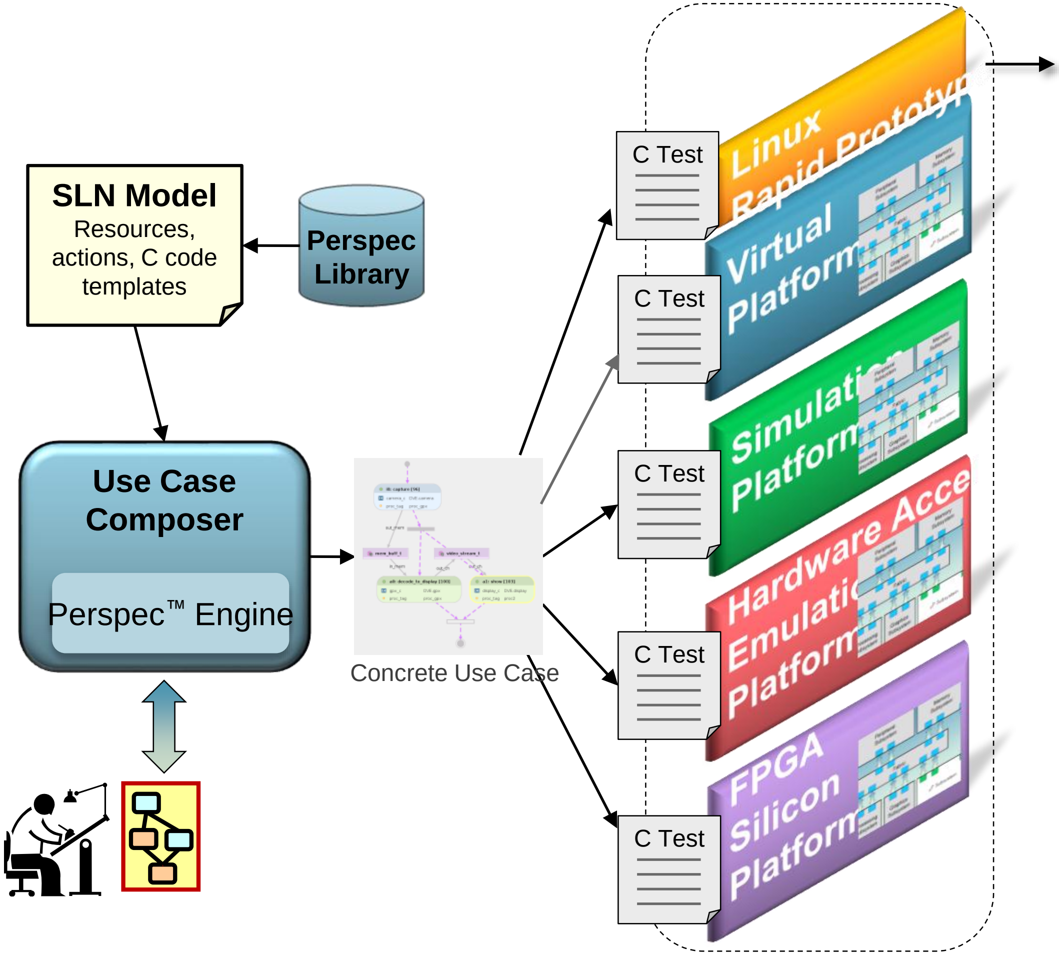
Items | generated

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		kind	20%	1 / 5 (20%)
		size	5%	1 / 20 (5%)

Gentime coverage of use case

Showing 2 items

Step #3: Generate tests for specific platform(s)



SLN Model
Resources,
actions, C code
templates

Perspec Library

Use Case Composer

Perspec™ Engine

Concrete Use Case

C Test

C Test

C Test

C Test

C Test

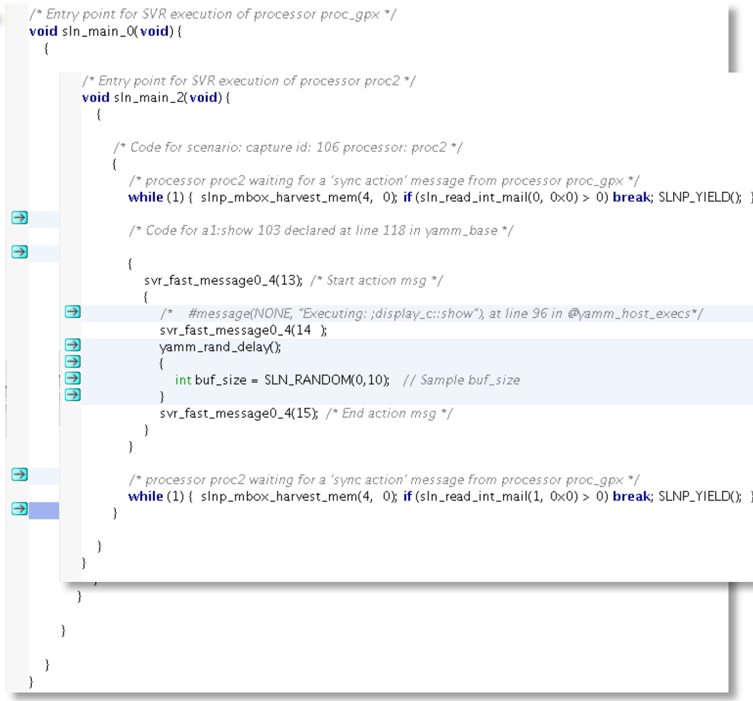
Linux
Rapid Prototyp

Virtual
Platform

Simulati
Platform

Hardware Acce
Emulatio
Platform

FPGA
Silicon
Platform

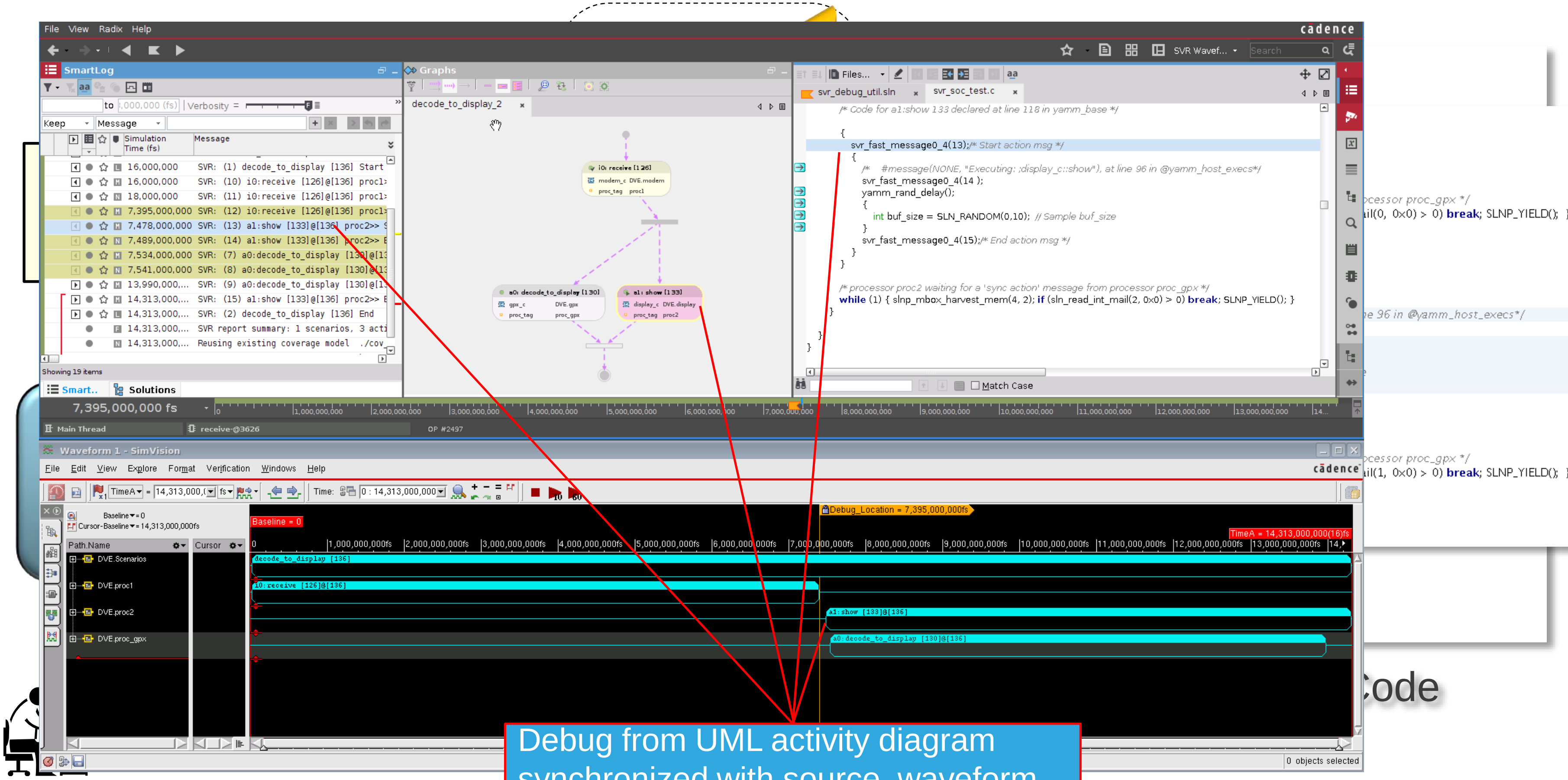


```
/* Entry point for SVR execution of processor proc_gpx */
void sln_main_0(void) {
  {

    /* Entry point for SVR execution of processor proc2 */
    void sln_main_2(void) {
      {

        /* Code for scenario: capture id: 106 processor: proc2 */
        {
          /* processor proc2 waiting for a 'sync action' message from processor proc_gpx */
          while (1) {  slnp_mbox_harvest_mem(4,  0); if (sln_read_int_mail(0,  0x0) > 0) break; SLNP_YIELD();  }

          /* Code for a1:show 103 declared at line 118 in yamm_base */

          {
            svr_fast_message0_4(13);  /* Start action msg */
            {
              /*    #message(NONE, "Executing: ;display_c::show"), at line 96 in @yamm_host_execs*/
              svr_fast_message0_4(14  );
              yamm_rand_delay();
              {
                int buf_size = SLN_RANDOM(0,10);   // Sample buf_size
              }
              svr_fast_message0_4(15);  /* End action msg */
            }
          }

          /* processor proc2 waiting for a 'sync action' message from processor proc_gpx */
          while (1) {  slnp_mbox_harvest_mem(4,  0); if (sln_read_int_mail(1,  0x0) > 0) break; SLNP_YIELD();  }
        }

      }
    }
        }
      }

    }

  }
}
```

Generated C Code

cādence®

Step #4: Run tests and debug



```
Simulation Time (fs)  Message
16,000,000        SVR: (1) decode_to_display [136] Start
16,000,000        SVR: (10) i0:receive [126]@[136] proc1>
18,000,000        SVR: (11) i0:receive [126]@[136] proc1>
7,395,000,000     SVR: (12) i0:receive [126]@[136] proc1>
7,478,000,000     SVR: (13) a1:show [133]@[136] proc2>> S
7,489,000,000     SVR: (14) a1:show [133]@[136] proc2>> E
7,534,000,000     SVR: (7) a0:decode_to_display [130]@[13
7,541,000,000     SVR: (8) a0:decode_to_display [130]@[13
13,990,000,...    SVR: (9) a0:decode_to_display [130]@[13
14,313,000,...    SVR: (15) a1:show [133]@[136] proc2>> E
14,313,000,...    SVR: (2) decode_to_display [136] End
14,313,000,...    SVR report summary: 1 scenarios, 3 acti
14,313,000,...    Reusing existing coverage model  ./cov_
```

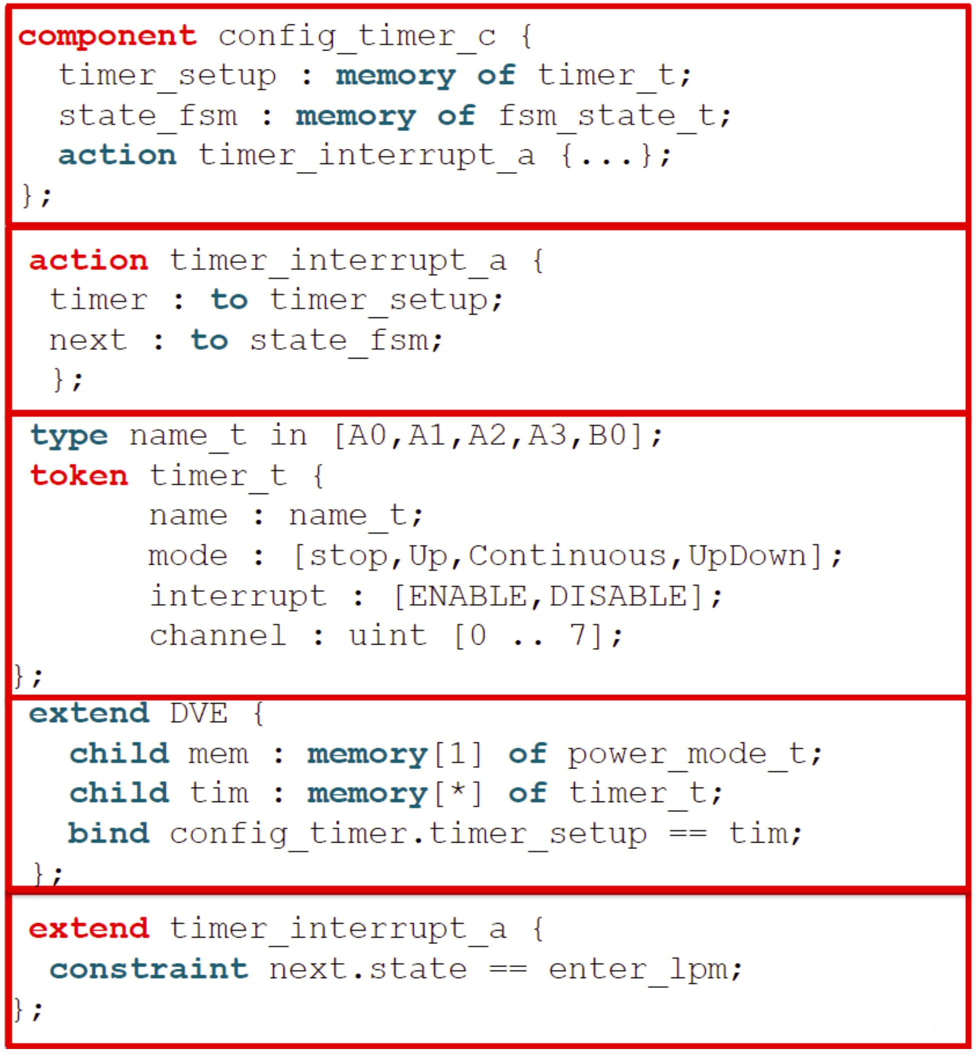
```c
/* Code for a1:show 133 declared at line 118 in yamm_base */

{
  svr_fast_message0_4(13);/* Start action msg */
  {
    /*    #message(NONE, "Executing: ;display_c::show"), at line 96 in @yamm_host_execs*/
    svr_fast_message0_4(14 );
    yamm_rand_delay();
    {
      int buf_size = SLN_RANDOM(0,10);  // Sample buf_size
    }
    svr_fast_message0_4(15);/* End action msg */
  }
}

/* processor proc2 waiting for a 'sync action' message from processor proc_gpx */
while (1) { slnp_mbox_harvest_mem(4, 2); if (sln_read_int_mail(2, 0x0) > 0) break; SLNP_YIELD(); }
}

  }
}
```

Debug from UML activity diagram synchronized with source, waveform and log messages

Perspec Modeling

Modeling elements

- Component: Functional unit groups actions and resources
- Action: Abstract operation of function
- Token: Include information for pre-conditions and outcomes
- Place: Defines interaction of tokens and actions (memory, channel, lock)
- Extend: Extending functionality of actions, components, and tokens



```
component config_timer_c {
  timer_setup : memory of timer_t;
  state_fsm : memory of fsm_state_t;
  action timer_interrupt_a {...};
};
```

```
action timer_interrupt_a {
 timer : to timer_setup;
 next : to state_fsm;
 };
```

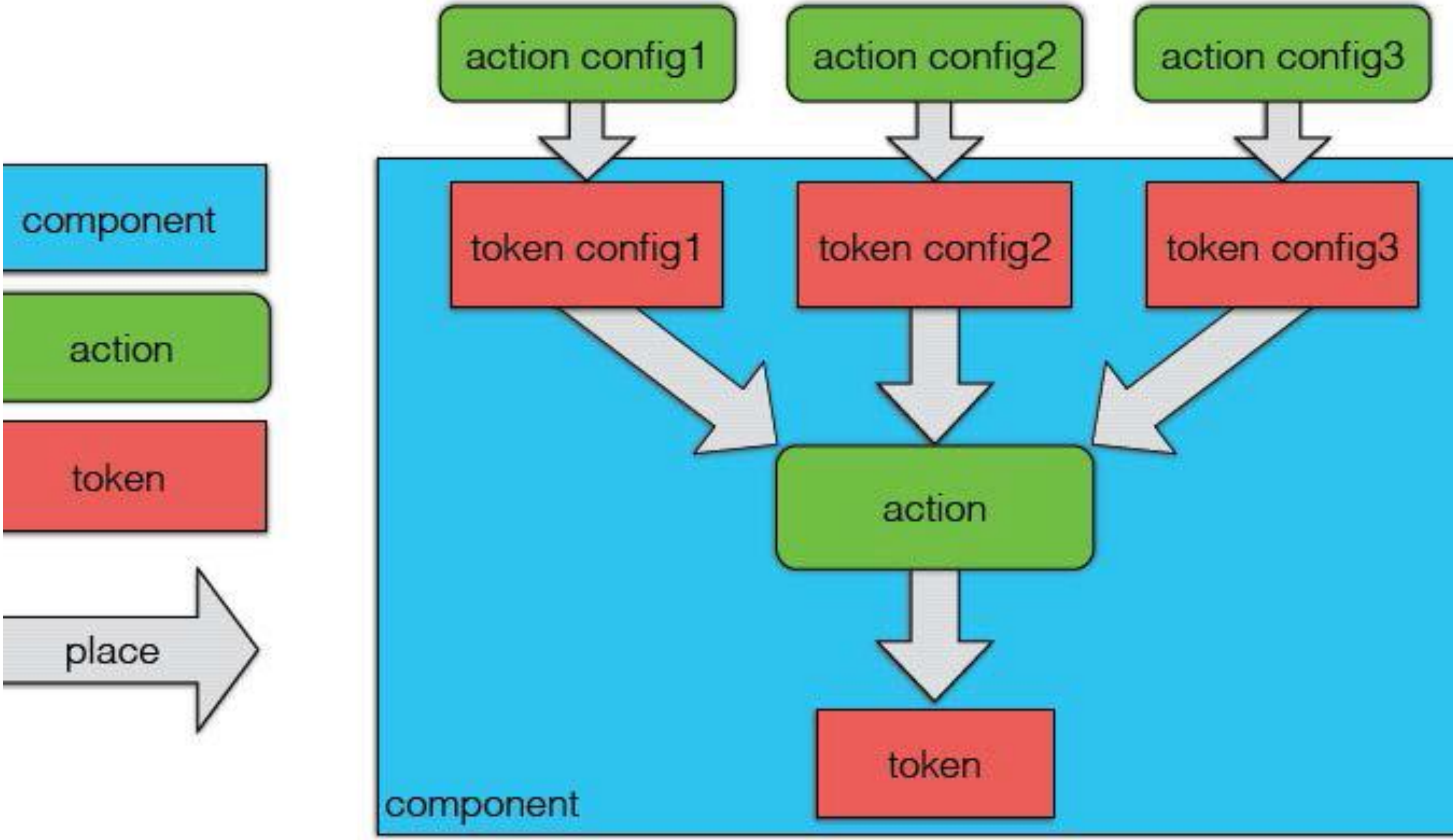
```
type name_t in [A0,A1,A2,A3,B0];
token timer_t {
      name : name_t;
      mode : [stop,Up,Continuous,UpDown];
      interrupt : [ENABLE,DISABLE];
      channel : uint [0 .. 7];
};
```

```
extend DVE {
  child mem : memory[1] of power_mode_t;
  child tim : memory[*] of timer_t;
  bind config_timer.timer_setup == tim;
 };
```

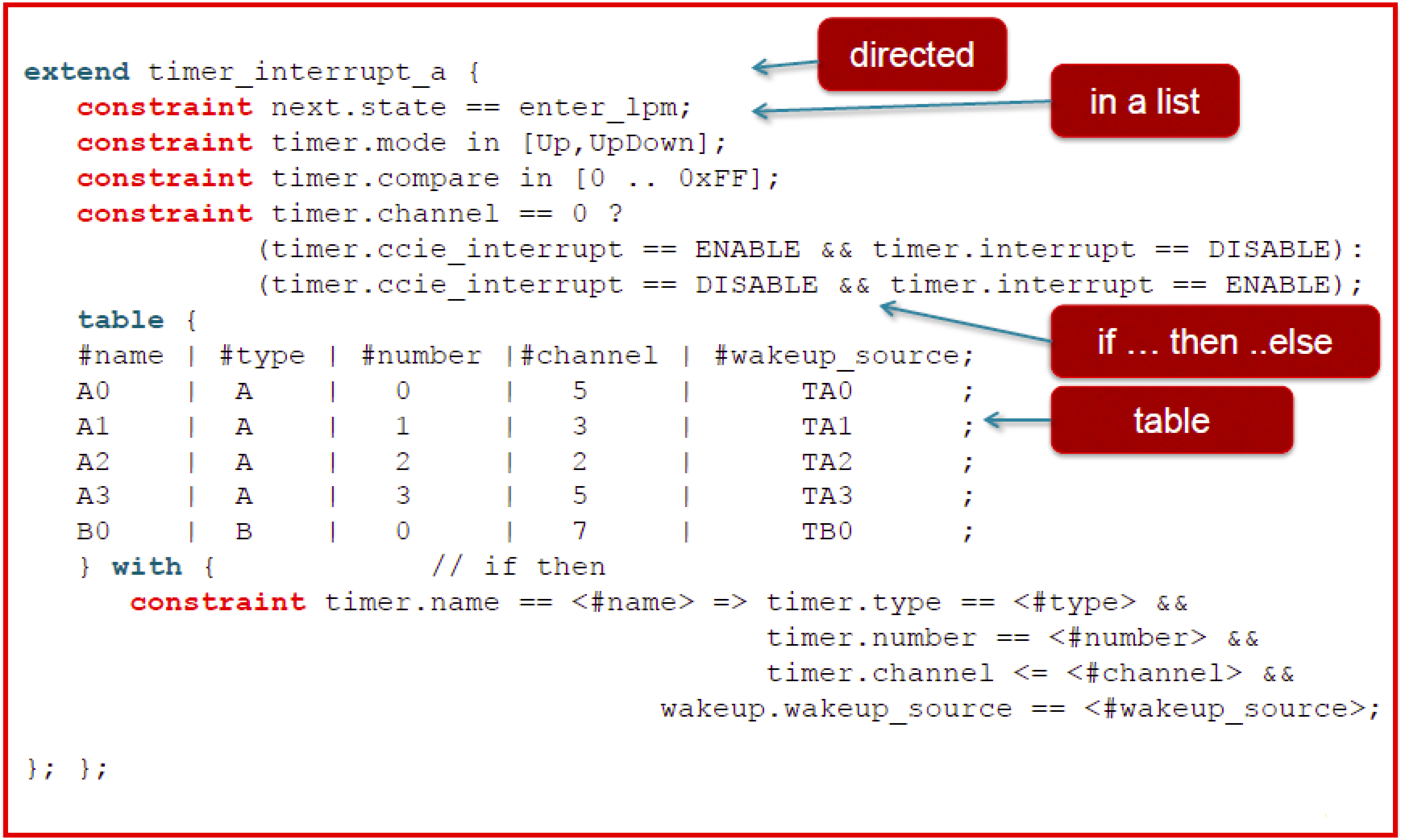
```
extend timer_interrupt_a {
 constraint next.state == enter_lpm;
};
```

Modeling with Perspec System Verifier

System language notation (SLN)



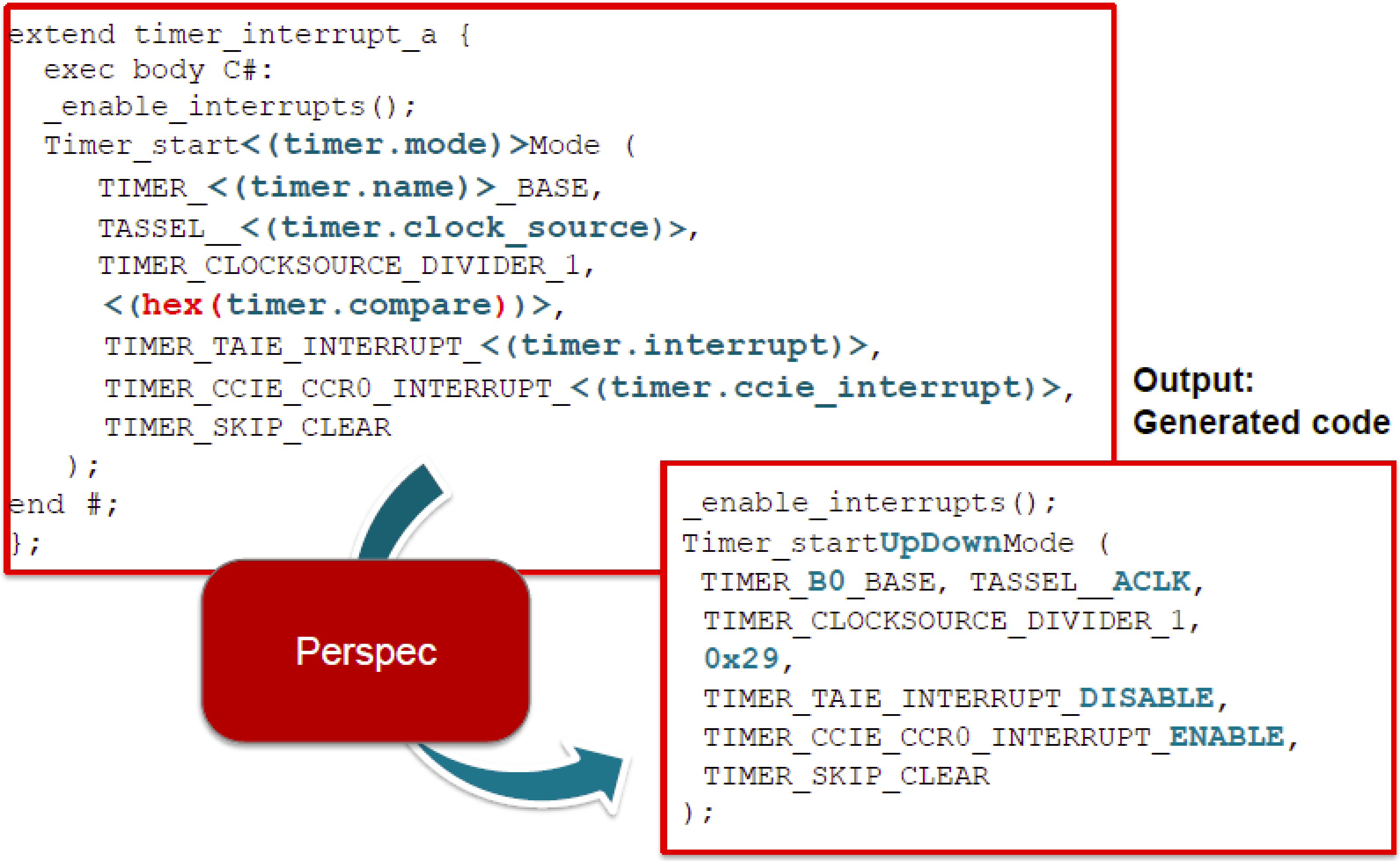
Constraints



```
extend timer_interrupt_a {
   constraint next.state == enter_lpm;
   constraint timer.mode in [Up,UpDown];
   constraint timer.compare in [0 .. 0xFF];
   constraint timer.channel == 0 ?
             (timer.ccie_interrupt == ENABLE && timer.interrupt == DISABLE):
             (timer.ccie_interrupt == DISABLE && timer.interrupt == ENABLE);
   table {
   #name | #type | #number |#channel | #wakeup_source;
   A0    |  A    |   0     |   5     |      TA0      ;
   A1    |  A    |   1     |   3     |      TA1      ;
   A2    |  A    |   2     |   2     |      TA2      ;
   A3    |  A    |   3     |   5     |      TA3      ;
   B0    |  B    |   0     |   7     |      TB0      ;
   } with {            // if then
      constraint timer.name == <#name> => timer.type == <#type> &&
                                          timer.number == <#number> &&
                                          timer.channel <= <#channel> &&
                                    wakeup.wakeup_source == <#wakeup_source>;

}; };
```

C-code generation

Input: a code template is always connected to an action.

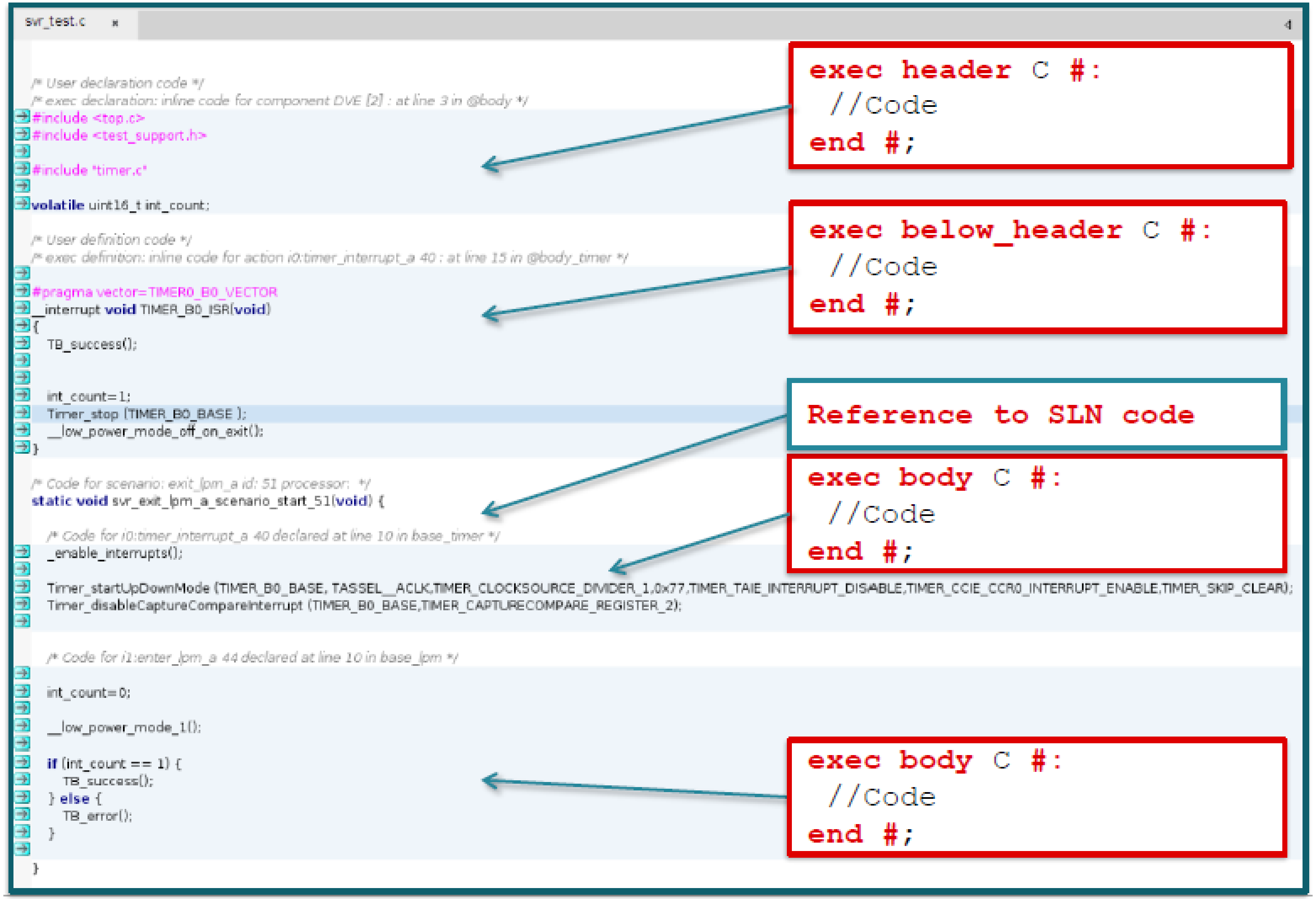


```
extend timer_interrupt_a {
  exec body C#:
  _enable_interrupts();
  Timer_start<(timer.mode)>Mode (
     TIMER_<(timer.name)>_BASE,
     TASSEL__<(timer.clock_source)>,
     TIMER_CLOCKSOURCE_DIVIDER_1,
     <(hex(timer.compare))>,
     TIMER_TAIE_INTERRUPT_<(timer.interrupt)>,
     TIMER_CCIE_CCR0_INTERRUPT_<(timer.ccie_interrupt)>,
     TIMER_SKIP_CLEAR
   );
end #;
};
```

Output:
Generated code

```
_enable_interrupts();
Timer_startUpDownMode (
 TIMER_B0_BASE, TASSEL__ACLK,
 TIMER_CLOCKSOURCE_DIVIDER_1,
 0x29,
 TIMER_TAIE_INTERRUPT_DISABLE,
 TIMER_CCIE_CCR0_INTERRUPT_ENABLE,
 TIMER_SKIP_CLEAR
);
```

Code generation



```
/* User declaration code */
/* exec declaration: inline code for component DVE [2] : at line 3 in @body */
#include <top.c>
#include <test_support.h>

#include "timer.c"

volatile uint16_t int_count;

/* User definition code */
/* exec definition: inline code for action i0:timer_interrupt_a 40 : at line 15 in @body_timer */

#pragma vector=TIMER0_B0_VECTOR
__interrupt void TIMER_B0_ISR(void)
{
   TB_success();


   int_count=1;
   Timer_stop (TIMER_B0_BASE );
   __low_power_mode_off_on_exit();
}

/* Code for scenario: exit_lpm_a id: 51 processor:  */
static void svr_exit_lpm_a_scenario_start_51(void) {

   /* Code for i0:timer_interrupt_a 40 declared at line 10 in base_timer */
   _enable_interrupts();

   Timer_startUpDownMode (TIMER_B0_BASE, TASSEL__ACLK,TIMER_CLOCKSOURCE_DIVIDER_1,0x77,TIMER_TAIE_INTERRUPT_DISABLE,TIMER_CCIE_CCR0_INTERRUPT_ENABLE,TIMER_SKIP_CLEAR);
   Timer_disableCaptureCompareInterrupt (TIMER_B0_BASE,TIMER_CAPTURECOMPARE_REGISTER_2);


   /* Code for i1:enter_lpm_a 44 declared at line 10 in base_lpm */

   int_count=0;

   __low_power_mode_1();

   if (int_count == 1) {
      TB_success();
   } else {
      TB_error();
   }

}
```

```
exec header C #:
 //Code
end #;
```

```
exec below_header C #:
 //Code
end #;
```

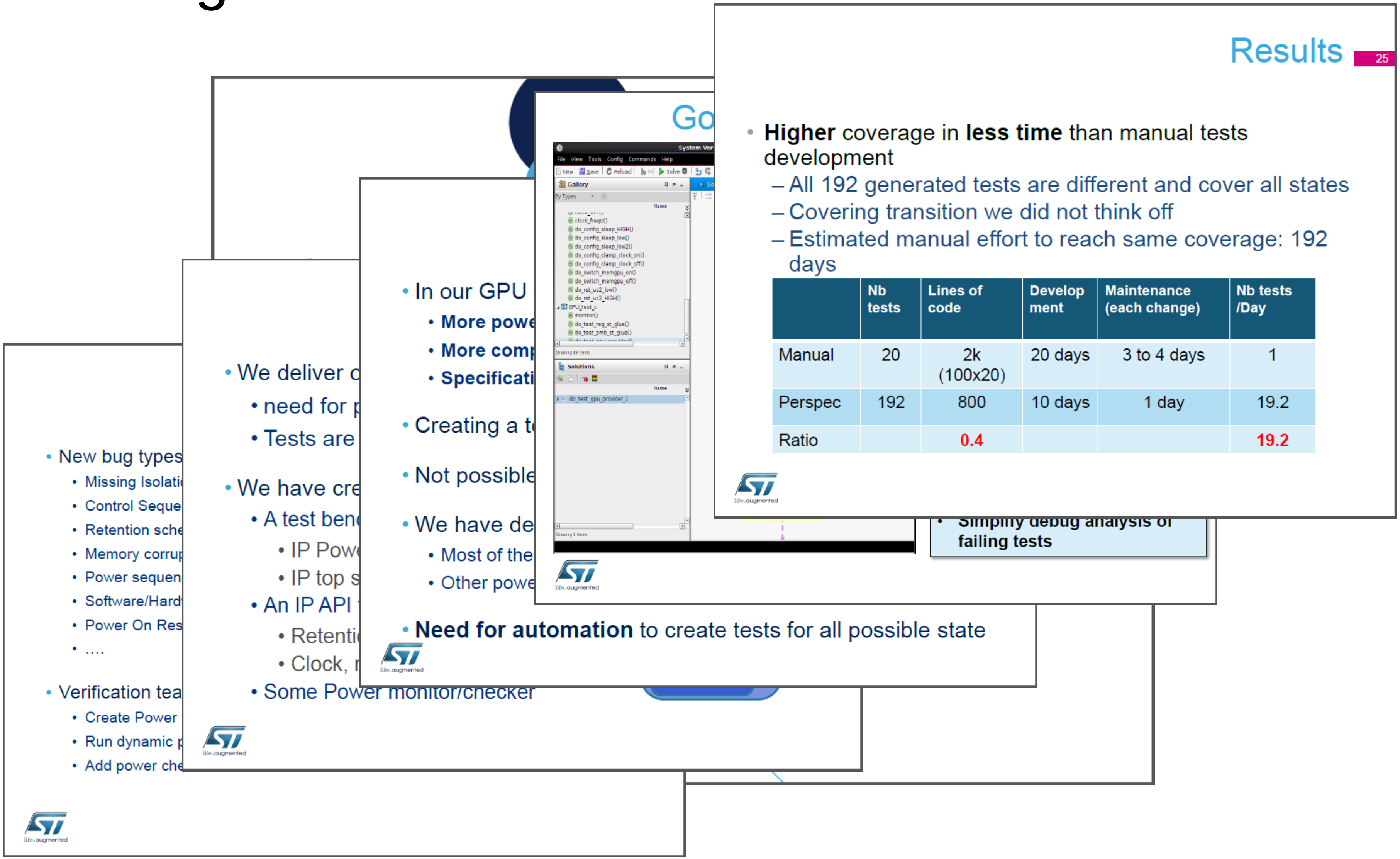
Reference to SLN code

```
exec body C #:
 //Code
end #;
```

```
exec body C #:
 //Code
end #;
```

Public Success Stories

ST TRD: Verify GPU modified for SoC power management



- New bug types
  - Missing Isolati
  - Control Seque
  - Retention sche
  - Memory corrup
  - Power sequen
  - Software/Hard
  - Power On Res
  - ….
- Verification tea
  - Create Power
  - Run dynamic p
  - Add power che

- We deliver d
  - need for p
  - Tests are
- We have cre
  - A test benc
    - IP Pow
    - IP top s
  - An IP API f
    - Retenti
    - Clock, r
  - Some Power monitor/checker

- In our GPU
  - **More powe**
  - **More comp**
  - **Specificati**
- Creating a te
- Not possible
- We have de
  - Most of the
  - Other powe
- **Need for automation** to create tests for all possible state

- Simplify debug analysis of failing tests

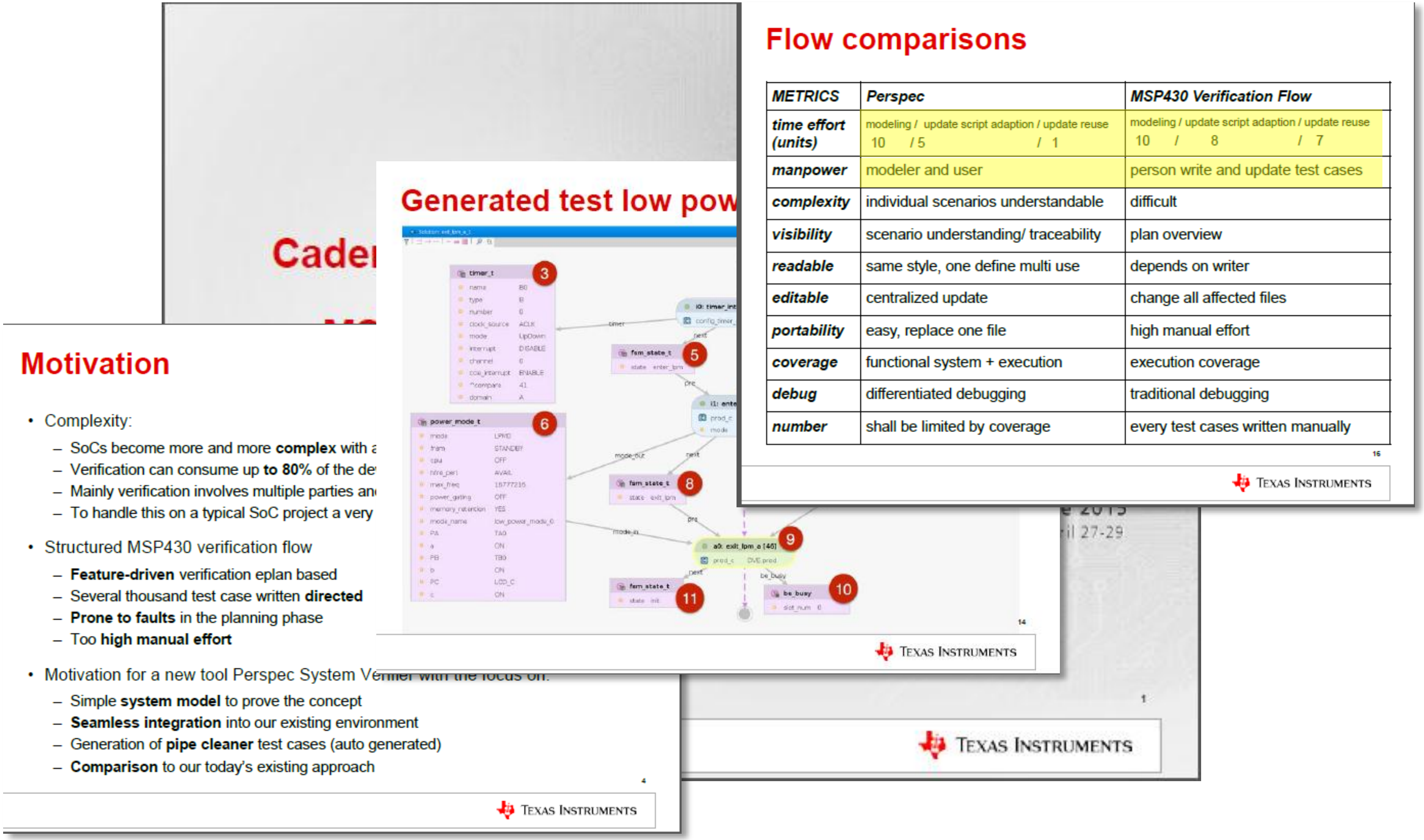
Results

25

- **Higher** coverage in **less time** than manual tests development
  - All 192 generated tests are different and cover all states
  - Covering transition we did not think off
  - Estimated manual effort to reach same coverage: 192 days

	Nb tests	Lines of code	Develop ment	Maintenance (each change)	Nb tests /Day
Manual	20	2k (100x20)	20 days	3 to 4 days	1
Perspec	192	800	10 days	1 day	19.2
Ratio		0.4			19.2

Texas Instruments: SoC verification flow



Motivation

- Complexity:
  - SoCs become more and more **complex** with a
  - Verification can consume up **to 80%** of the de
  - Mainly verification involves multiple parties an
  - To handle this on a typical SoC project a very
- Structured MSP430 verification flow
  - **Feature-driven** verification eplan based
  - Several thousand test case written **directed**
  - **Prone to faults** in the planning phase
  - Too **high manual effort**
- Motivation for a new tool Perspec System Verifier with the focus on:
  - Simple **system model** to prove the concept
  - **Seamless integration** into our existing environment
  - Generation of **pipe cleaner** test cases (auto generated)
  - **Comparison** to our today's existing approach

4

Texas Instruments

Generated test low pow

14

Texas Instruments

Flow comparisons

METRICS	Perspec	MSP430 Verification Flow
time effort (units)	modeling / update script adaption / update reuse 10 / 5 / 1	modeling / update script adaption / update reuse 10 / 8 / 7
manpower	modeler and user	person write and update test cases
complexity	individual scenarios understandable	difficult
visibility	scenario understanding/ traceability	plan overview
readable	same style, one define multi use	depends on writer
editable	centralized update	change all affected files
portability	easy, replace one file	high manual effort
coverage	functional system + execution	execution coverage
debug	differentiated debugging	traditional debugging
number	shall be limited by coverage	every test cases written manually

16

Texas Instruments

Cader

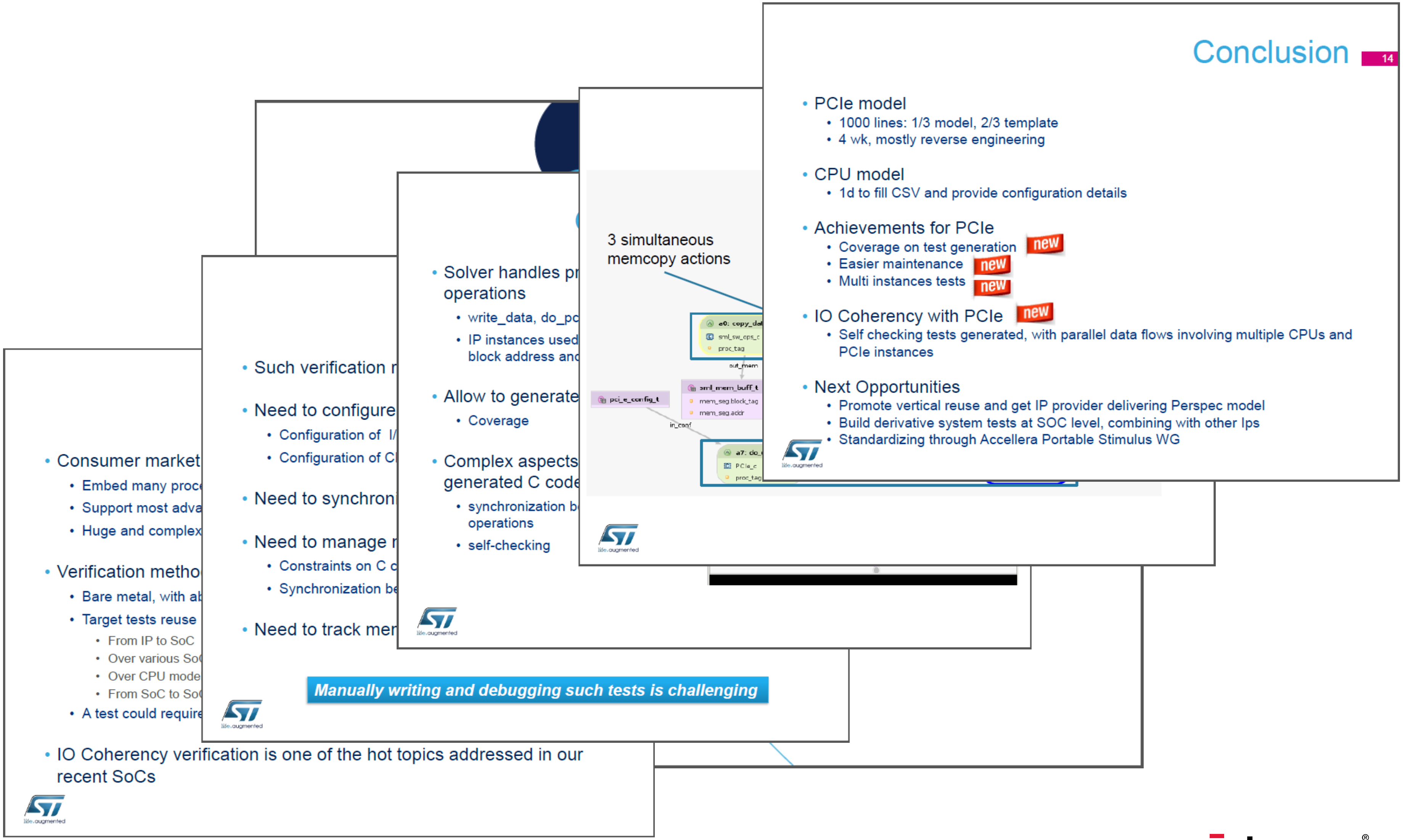
e 2015

il 27-29

1

Texas Instruments

ST CPD: Complex SoC scenarios



- Consumer market
  - Embed many proc
  - Support most adva
  - Huge and complex
- Verification metho
  - Bare metal, with ab
  - Target tests reuse
    - From IP to SoC
    - Over various So
    - Over CPU mode
    - From SoC to So
  - A test could require
- IO Coherency verification is one of the hot topics addressed in our recent SoCs

life.augmented

- Such verification r
- Need to configure
  - Configuration of I/
  - Configuration of C
- Need to synchroni
- Need to manage r
  - Constraints on C c
  - Synchronization be
- Need to track mem

Manually writing and debugging such tests is challenging

life.augmented

- Solver handles pr operations
  - write_data, do_pc
  - IP instances used block address and
- Allow to generate
  - Coverage
- Complex aspects generated C code
  - synchronization b operations
  - self-checking

life.augmented

3 simultaneous memcopy actions

life.augmented

Conclusion

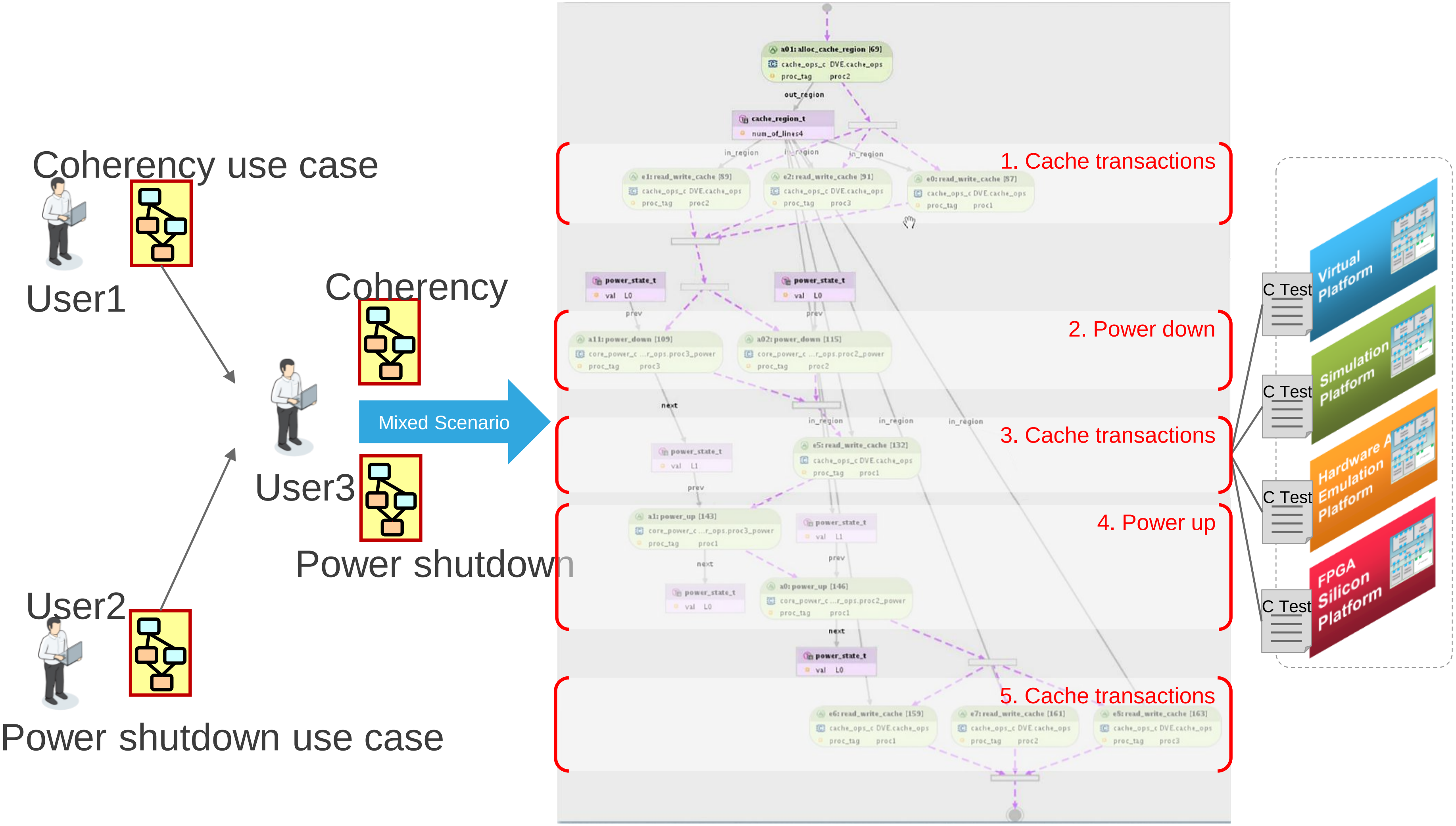
14

- PCIe model
  - 1000 lines: 1/3 model, 2/3 template
  - 4 wk, mostly reverse engineering
- CPU model
  - 1d to fill CSV and provide configuration details
- Achievements for PCIe
  - Coverage on test generation new
  - Easier maintenance new
  - Multi instances tests new
- IO Coherency with PCIe new
  - Self checking tests generated, with parallel data flows involving multiple CPUs and PCIe instances
- Next Opportunities
  - Promote vertical reuse and get IP provider delivering Perspec model
  - Build derivative system tests at SOC level, combining with other Ips
  - Standardizing through Accellera Portable Stimulus WG

life.augmented

Summary

Connecting it together



Coherency use case

User1

Coherency

User3

Mixed Scenario

Power shutdown

User2

Power shutdown use case

1. Cache transactions

2. Power down

3. Cache transactions

4. Power up

5. Cache transactions

C Test

C Test

C Test

C Test

Virtual Platform

Simulation Platform

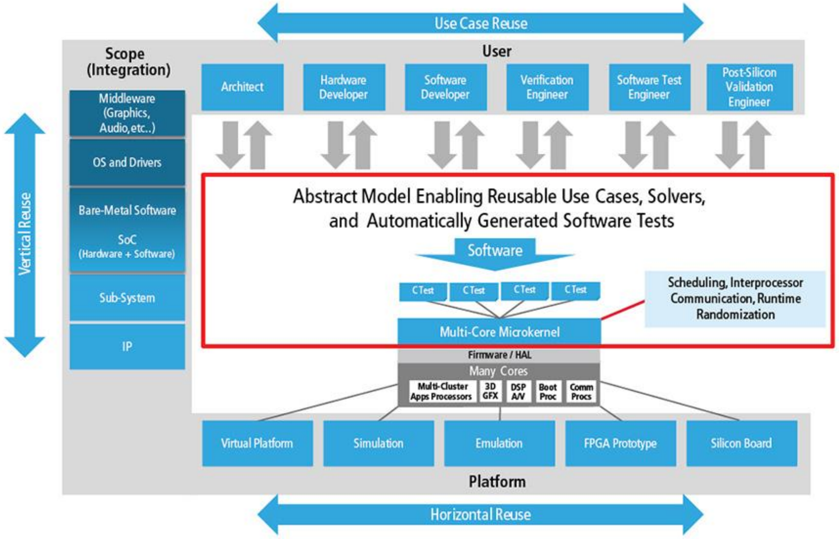
Hardware Acceleration Emulation Platform

FPGA Silicon Platform

a01: alloc_cache_region [69]
cache_ops_c DVE.cache_ops
proc_tag proc2
out_region
cache_region_t
num_of_lines4
in_region
in_region
in_region
e1: read_write_cache [89]
cache_ops_c DVE.cache_ops
proc_tag proc2
e2: read_write_cache [91]
cache_ops_c DVE.cache_ops
proc_tag proc3
e0: read_write_cache [87]
cache_ops_c DVE.cache_ops
proc_tag proc1
power_state_t
val L0
power_state_t
val L0
prev
prev
a11: power_down [109]
core_power_c ...r_ops.proc3_power
proc_tag proc3
a02: power_down [115]
core_power_c ...r_ops.proc2_power
proc_tag proc2
next
in_region
in_region
in_region
power_state_t
val L1
e5: read_write_cache [132]
cache_ops_c DVE.cache_ops
proc_tag proc1
prev
a1: power_up [143]
core_power_c ...r_ops.proc3_power
proc_tag proc1
power_state_t
val L1
next
prev
power_state_t
val L0
a0: power_up [146]
core_power_c ...r_ops.proc2_power
proc_tag proc1
next
power_state_t
val L0
e6: read_write_cache [159]
cache_ops_c DVE.cache_ops
proc_tag proc1
e7: read_write_cache [161]
cache_ops_c DVE.cache_ops
proc_tag proc2
e8: read_write_cache [163]
cache_ops_c DVE.cache_ops
proc_tag proc3

Perspec System Verifier

- **Productivity**
  10X improvement for complex SoC test creation
- **Abstraction**
  UML-style use-case diagrams
- **Automation**
  System use-case test generation
- **Portability**
  Reuse across all execution platforms
- **Measurement**
  SoC-level hardware/software coverage metrics



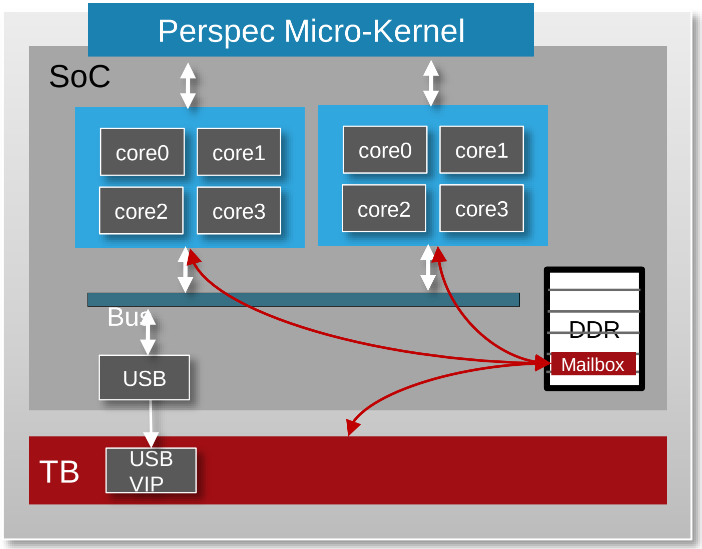
Perspec System Verifier Technology

cadence®

Backup

Micro-kernel runtime environment

- Perspec™ System Verifier has ability to manage resources, parallel actions, and test scheduling
  - Before C code is created, automatic planning of the scenario takes place
  - In the C code, sync points are added between cores and testbenches VIP
  - Resources availability is also managed in runtime
- Perspec sync is done via an abstract mailbox
  - Modeled in layers to support multiple communication schemes, e.g., memory, sockets, GPIO, etc…
  - Thread safe to enable multiple cores and same-time communication
  - Small in size and efficient
- Some of the applications of this infrastructure
  - Sync of any activity across languages and platforms
  - Unified analysis and debug
  - Runtime coverage and checking
  - Control external VIP/component
  - Print messages from the embedded cores



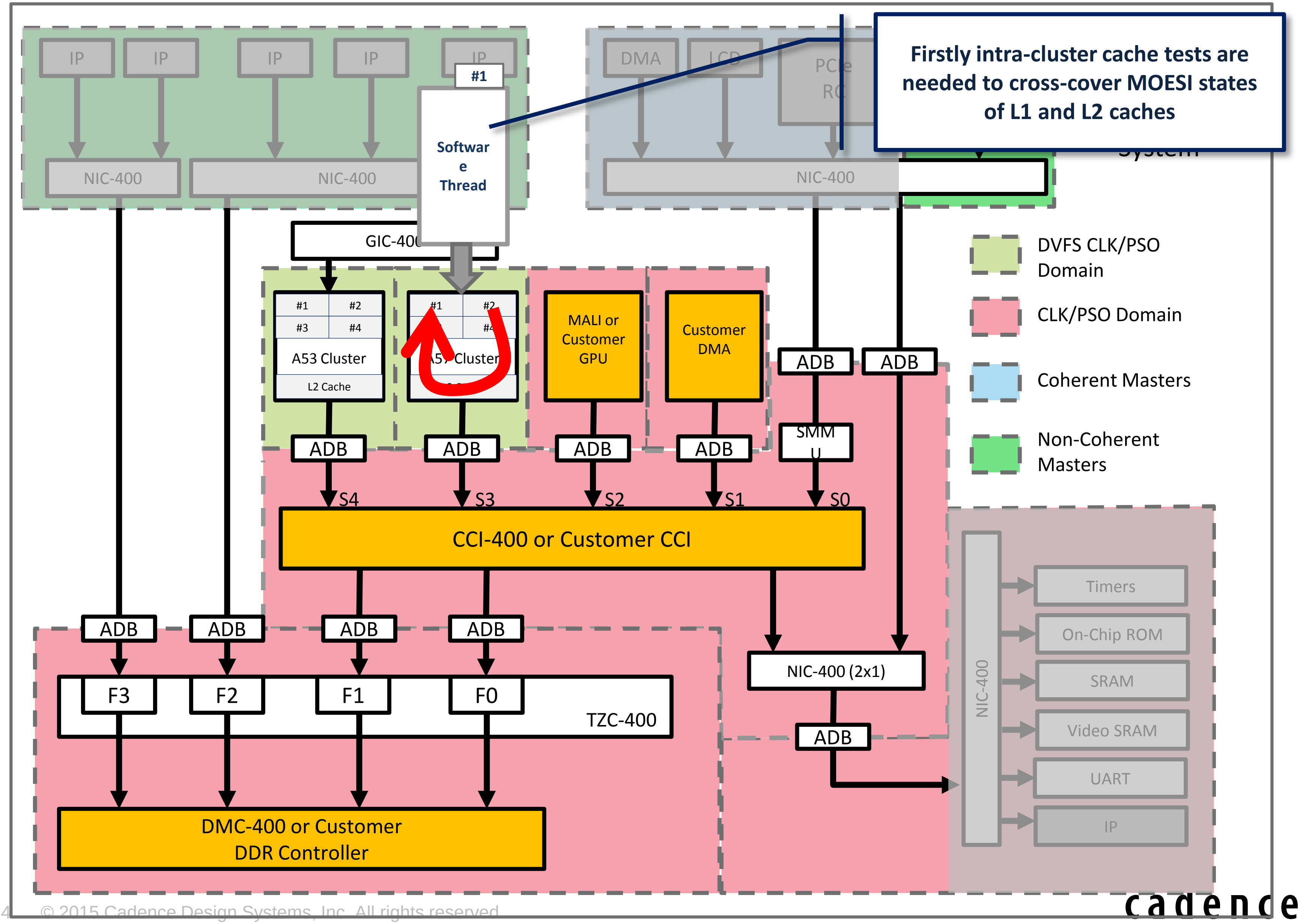
Perspec Libraries

Perspec libraries

- Many SoCs have many common characteristics
  - Typically have CPUs, caches, memories, low-power features, etc…
  - Enables capturing a general set of SoC model building blocks
- Cadence provides pre-built libraries for Perspec™
  - Reduces modeling effort and time for customer
  - Models follow good coding style, built for reuse
- Perspec System Methodology Library (SML)
  - Captures system modeling including memories, processors, etc…
  - Customer uses spreadsheet template to configure for specific SoC
- Example: Perspec library for ARM® Architecture
  - Captures configurable cache, MMU, and low-power models

Coherency verification challenges

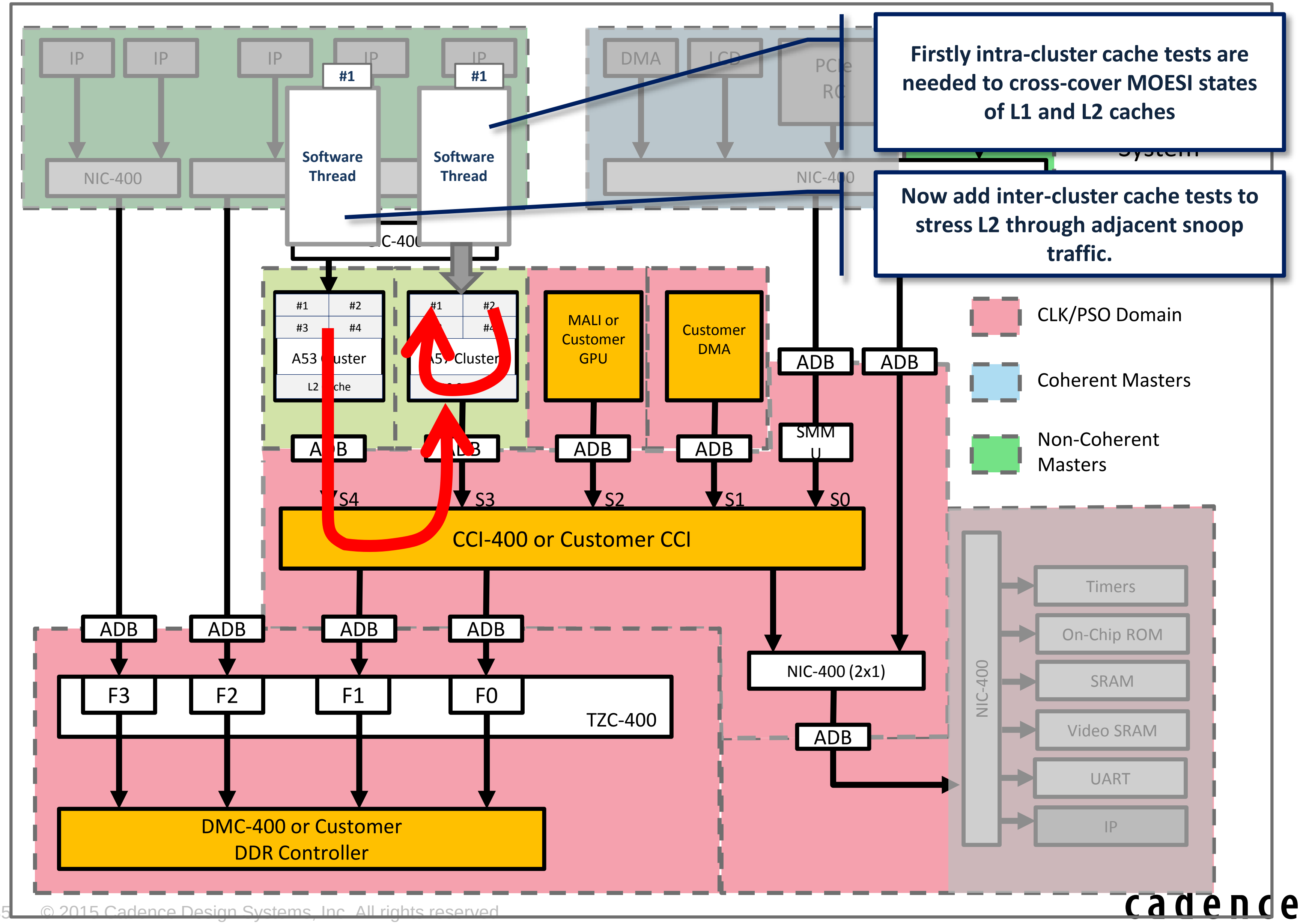
Intra-cluster



Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches

Coherency verification challenges

Intra-cluster

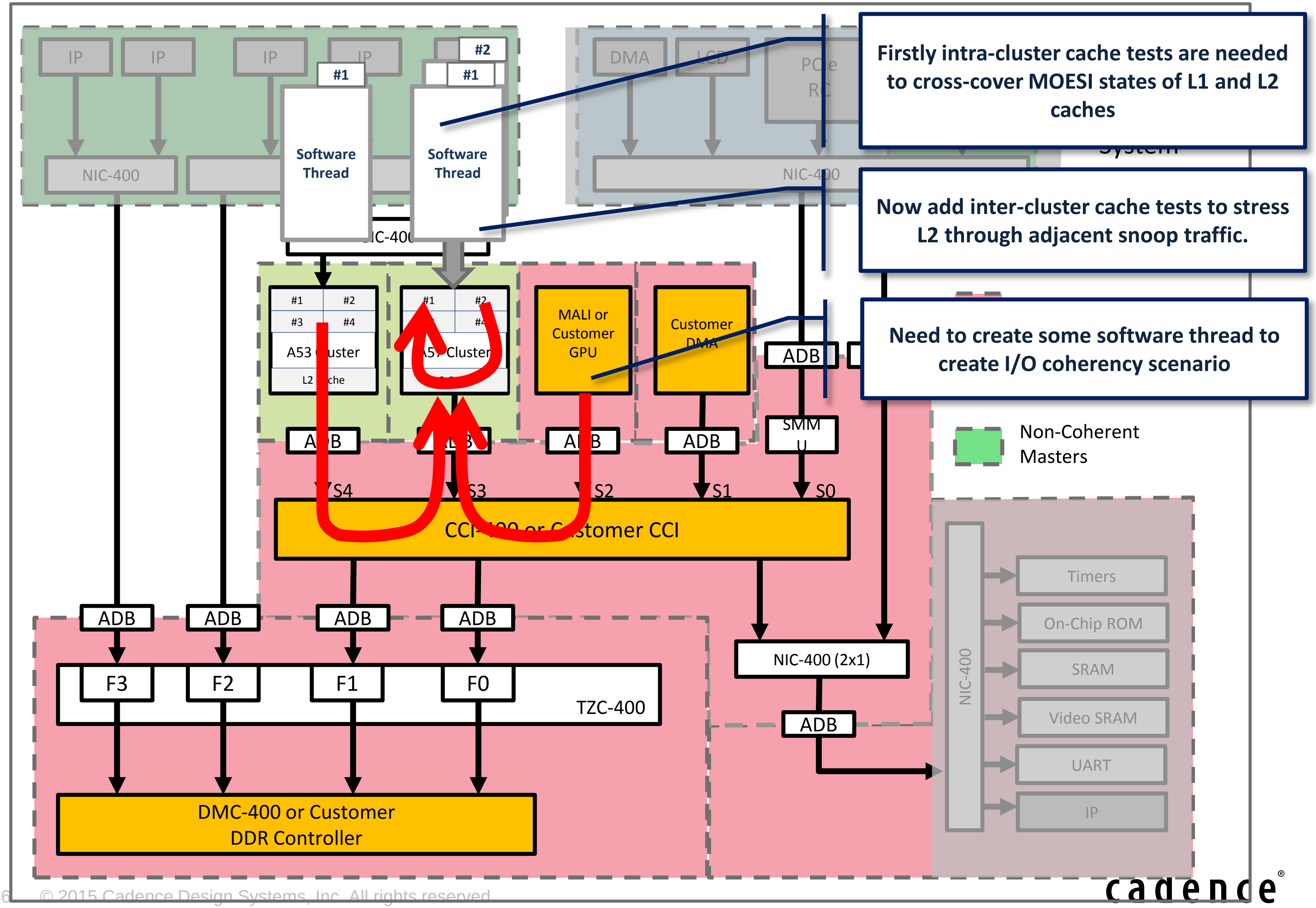


Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.

Coherency verification challenges

Critical coherent I/O

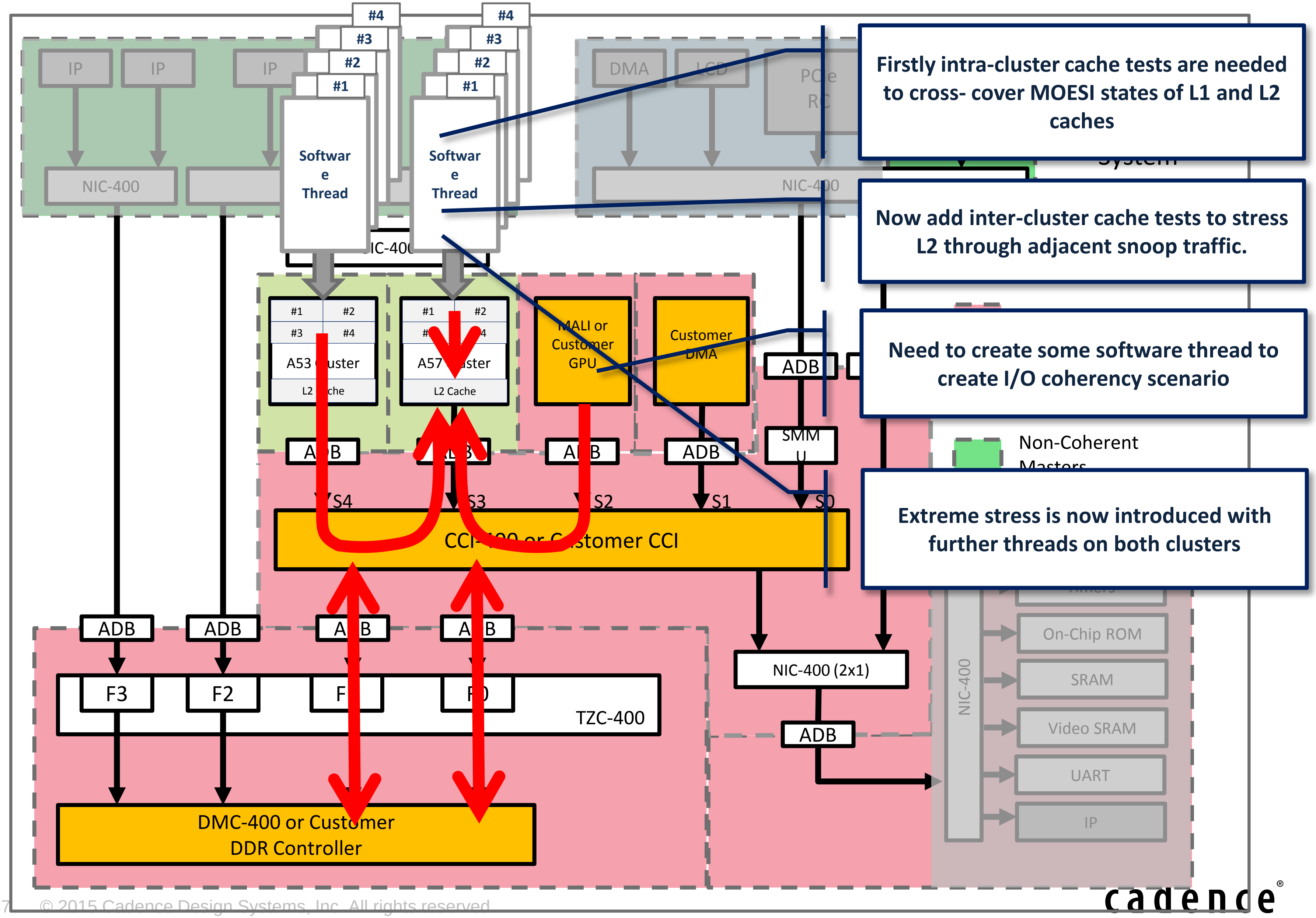


Firstly intra-cluster cache tests are needed to cross-cover MOESI states of L1 and L2 caches

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.

Need to create some software thread to create I/O coherency scenario

Coherency verification challenges



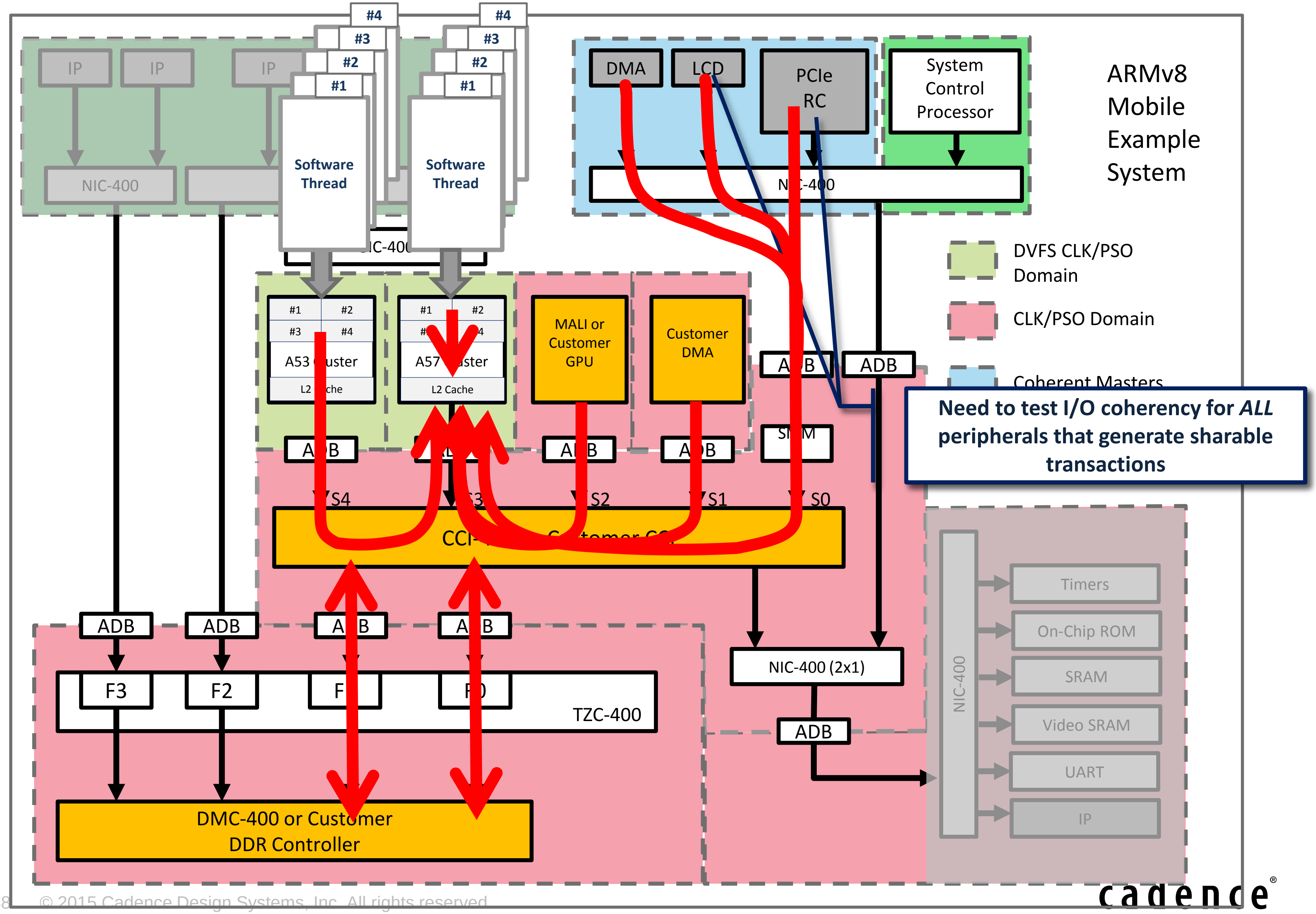
Firstly intra-cluster cache tests are needed to cross- cover MOESI states of L1 and L2 caches

Now add inter-cluster cache tests to stress L2 through adjacent snoop traffic.

Need to create some software thread to create I/O coherency scenario

Extreme stress is now introduced with further threads on both clusters

I/O coherency verification challenges

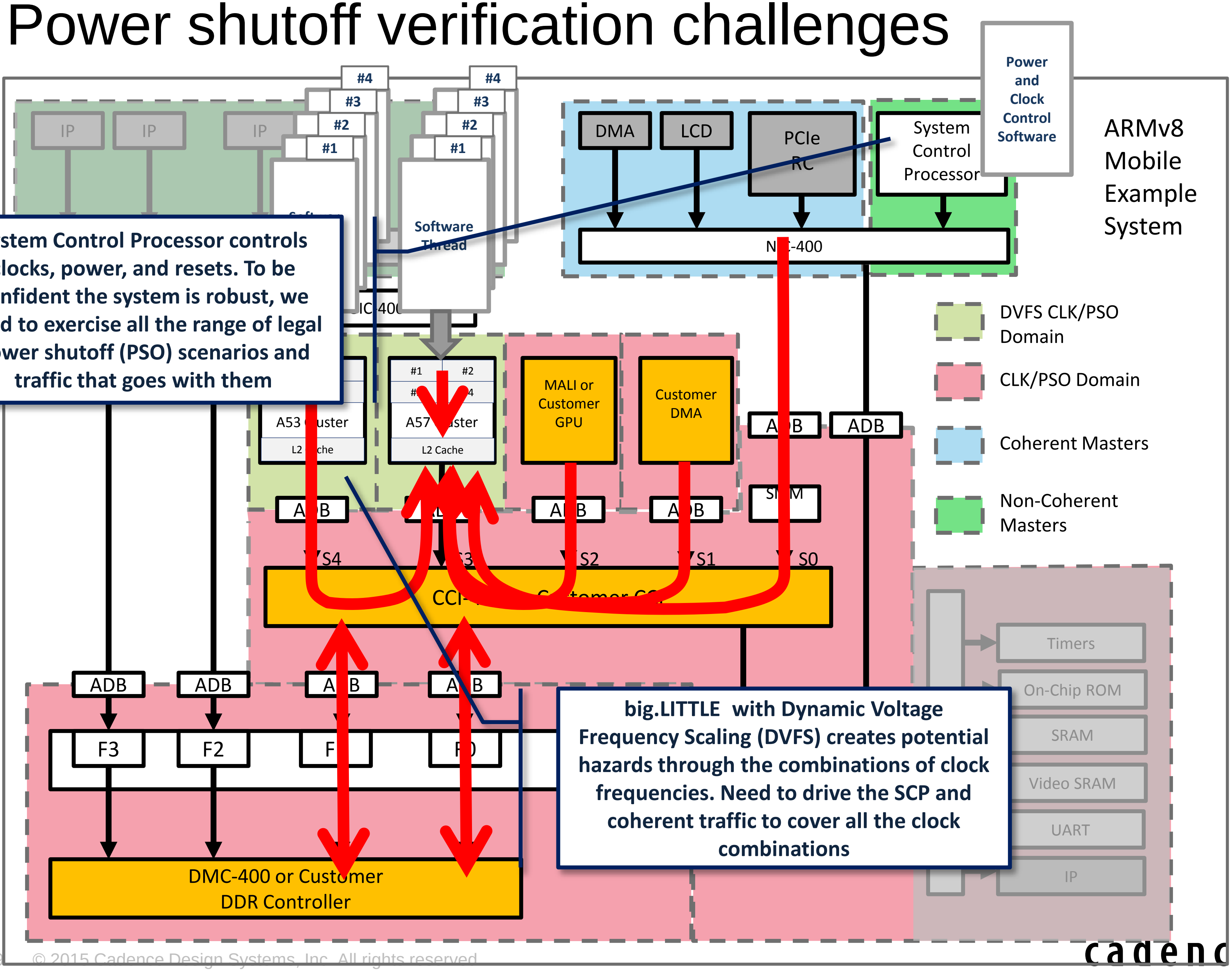


ARMv8
Mobile
Example
System

- 🟩 DVFS CLK/PSO Domain
- 🟥 CLK/PSO Domain
- 🟦 Coherent Masters

Need to test I/O coherency for *ALL* peripherals that generate sharable transactions

Power shutoff verification challenges



System Control Processor controls clocks, power, and resets. To be confident the system is robust, we need to exercise all the range of legal power shutoff (PSO) scenarios and traffic that goes with them

big.LITTLE with Dynamic Voltage Frequency Scaling (DVFS) creates potential hazards through the combinations of clock frequencies. Need to drive the SCP and coherent traffic to cover all the clock combinations

cadence®

Example: Library for ARM Architecture

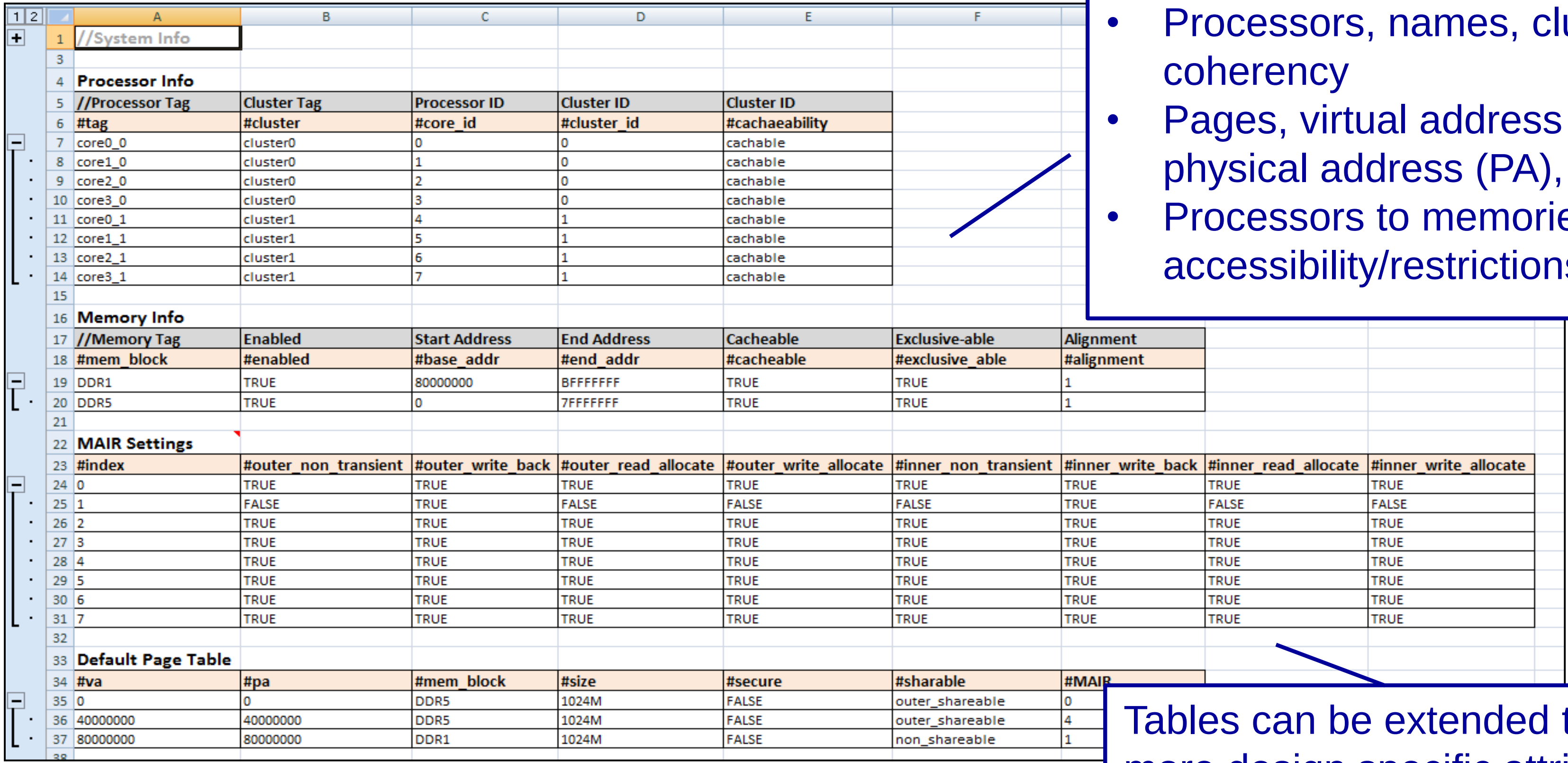
Perspec library for ARM Architecture

- Set of operations to manage the ARM compute cluster
  - Memory management
    - Page table handling, virtual address
  - Predefined actions that user can use in their program
    - Write Data, Read data, Copy data
  - Caching operation
    - True Sharing
    - False Sharing
    - I/O Coherency
  - Low power

- Takes a description of the system in a spreadsheet and creates system scenarios

Perspec library processor configuration tables

- Zero modeling is required since CPU and memory sub-system models are automatically generated from library by reading system configuration tables (shown below)



//System Info

Processor Info

//Processor Tag	Cluster Tag	Processor ID	Cluster ID	Cluster ID
#tag	#cluster	#core_id	#cluster_id	#cachaeability
core0_0	cluster0	0	0	cachable
core1_0	cluster0	1	0	cachable
core2_0	cluster0	2	0	cachable
core3_0	cluster0	3	0	cachable
core0_1	cluster1	4	1	cachable
core1_1	cluster1	5	1	cachable
core2_1	cluster1	6	1	cachable
core3_1	cluster1	7	1	cachable

Memory Info

//Memory Tag	Enabled	Start Address	End Address	Cacheable	Exclusive-able	Alignment
#mem_block	#enabled	#base_addr	#end_addr	#cacheable	#exclusive_able	#alignment
DDR1	TRUE	80000000	BFFFFFFF	TRUE	TRUE	1
DDR5	TRUE	0	7FFFFFFF	TRUE	TRUE	1

MAIR Settings

#index	#outer_non_transient	#outer_write_back	#outer_read_allocate	#outer_write_allocate	#inner_non_transient	#inner_write_back	#inner_read_allocate	#inner_write_allocate
0	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
1	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
2	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
3	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
4	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
5	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
6	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
7	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

Default Page Table

#va	#pa	#mem_block	#size	#secure	#sharable	#MAIR
0	0	DDR5	1024M	FALSE	outer_shareable	0
40000000	40000000	DDR5	1024M	FALSE	outer_shareable	4
80000000	80000000	DDR1	1024M	FALSE	non_shareable	1

- Memory blocks
- Processors, names, clusters, coherency
- Pages, virtual address (VA), physical address (PA), size
- Processors to memories accessibility/restrictions

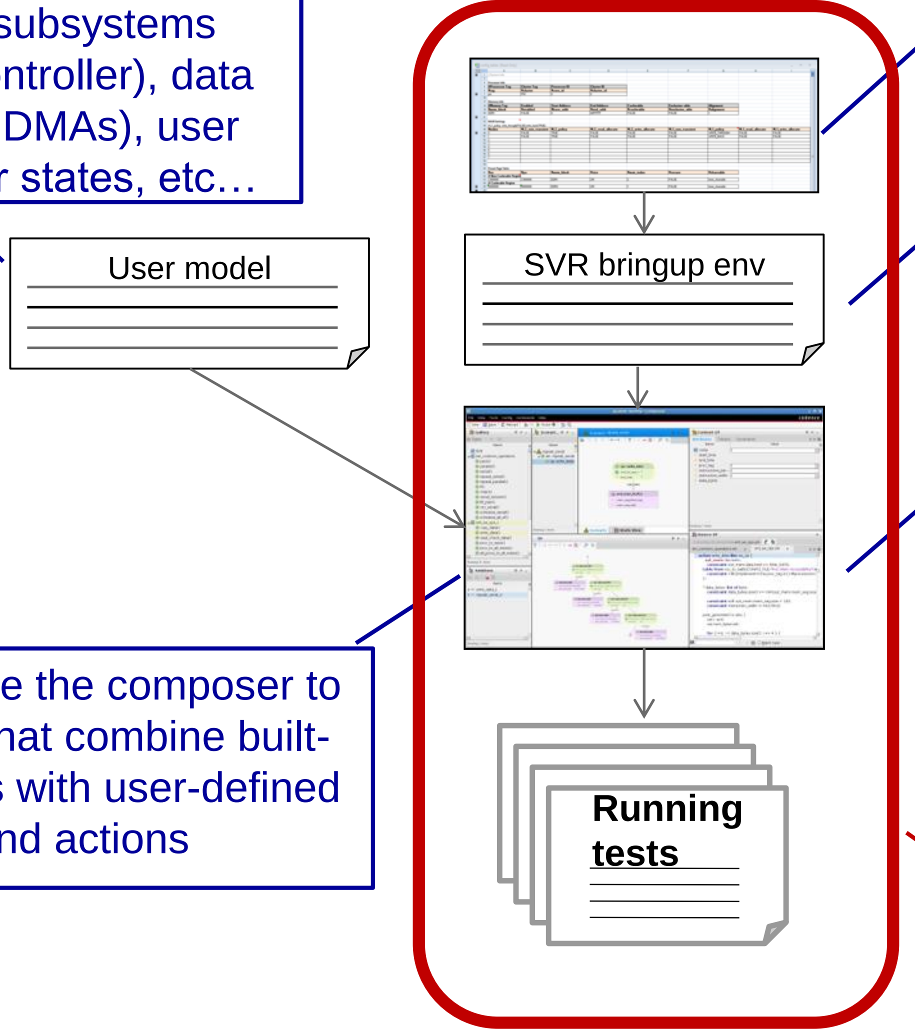
Tables can be extended to add more design specific attributes

Perspec library use model

Step #4: As needed model SoC-specific subsystems (e.g., PCIe controller), data movers (e.g., DMAs), user defined power states, etc…

Step #1: Fill in the standard SML and Coherency tables

Step #2: Load the SML bring-up environment + user boot code



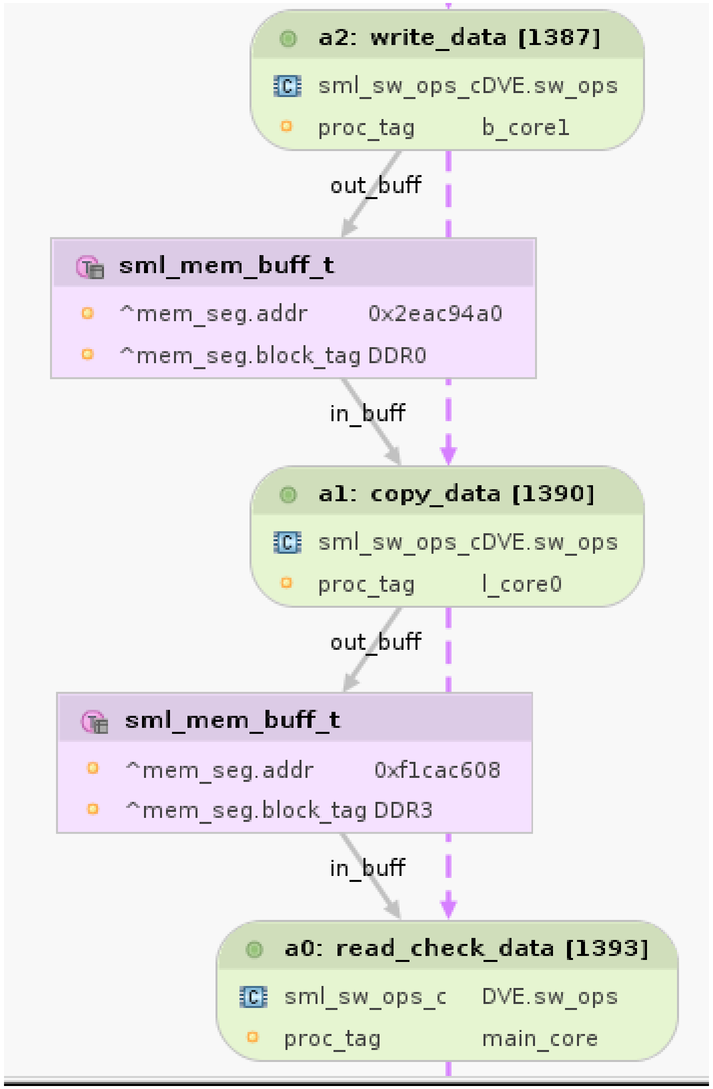
Step #3: Use the composer to write scenarios and use-cases: Memory, coherency, power, and DVM, tune the built-in coverage as needed

Step #5: Use the composer to write tests that combine built-in scenarios with user-defined scenarios and actions

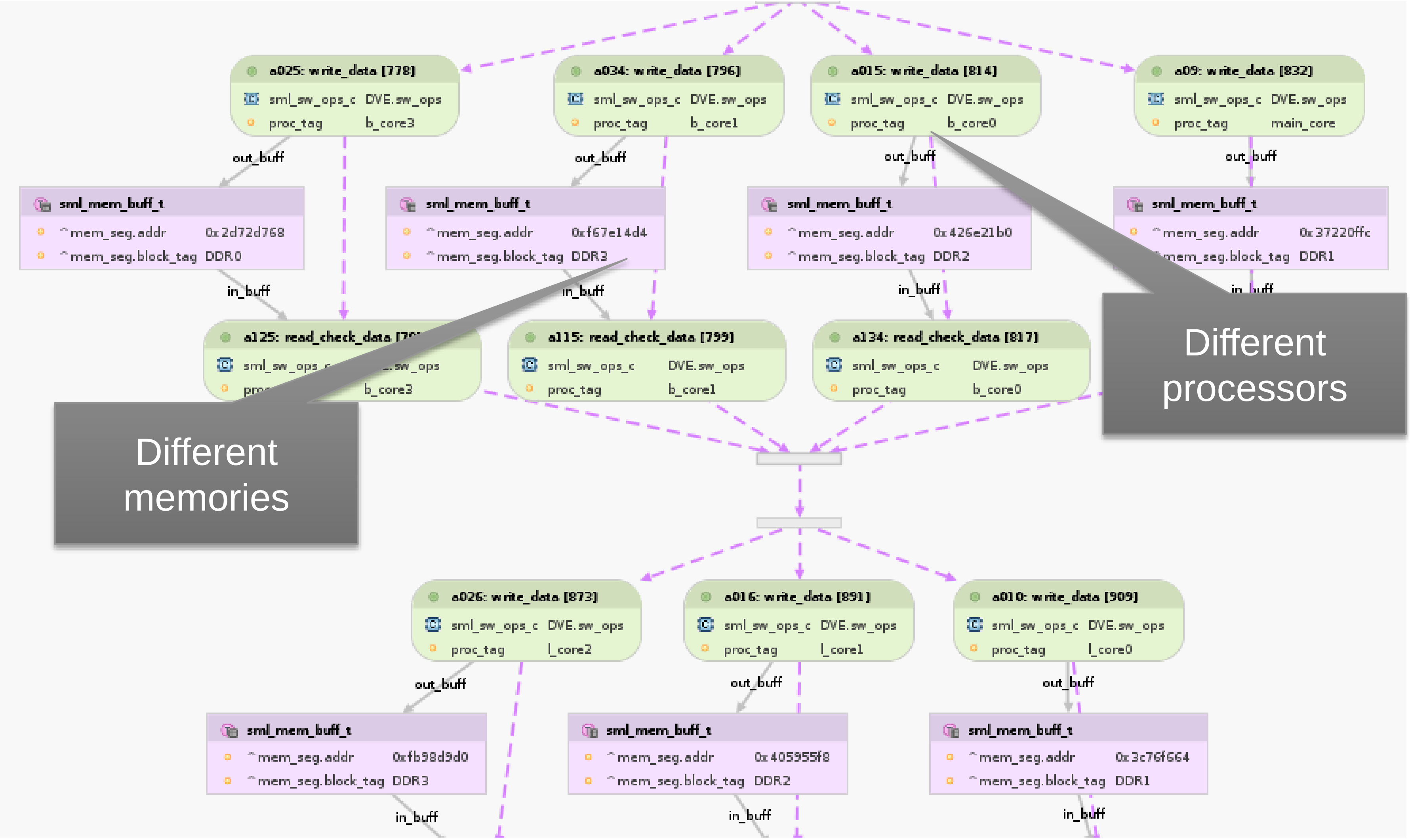
No modeling effort

Pre-defined basic software operations

- Basic software operations
  - Write: *write_data*
    - Generates data and writes it into the memory
  - Copy: *copy_data*
    - Copy data from one area to another
  - Read: *read_check_data*
    - Read data from previously written area
    - Checks against the reference model
- Main control knobs
  - Alignment
  - Data size
  - Memory block/address



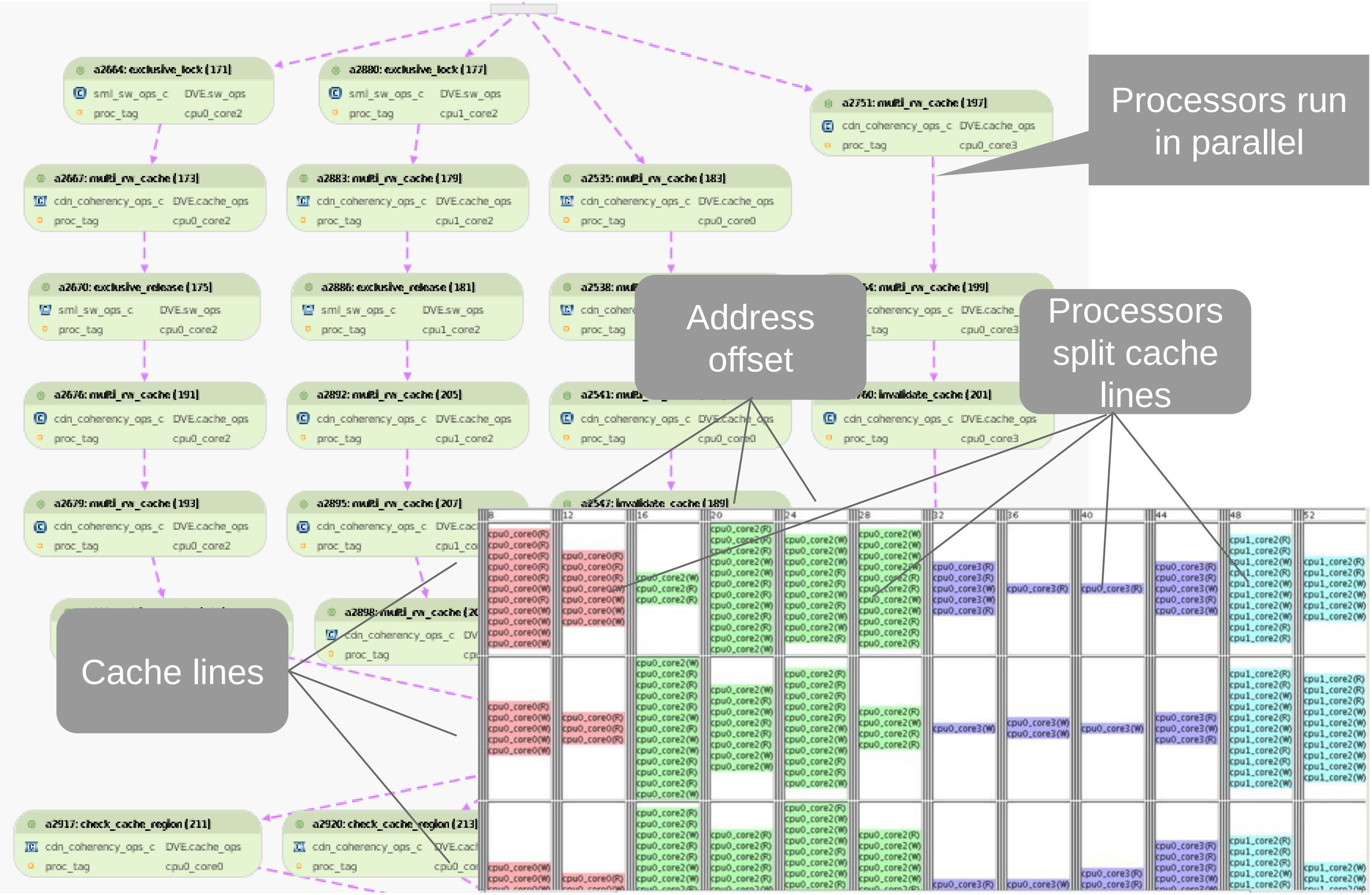
Advanced software operation: All processors to all memories



Different memories

Different processors

Coherency—False Sharing


a2664: exclusive_lock [171]
sml_sw_ops_c DVE.sw_ops
proc_tag cpu0_core2
a2880: exclusive_lock [177]
sml_sw_ops_c DVE.sw_ops
proc_tag cpu1_core2
a2751: multi_rw_cache [197]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core3
a2667: multi_rw_cache [173]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core2
a2883: multi_rw_cache [179]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu1_core2
a2535: multi_rw_cache [183]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core0
a2670: exclusive_release [175]
sml_sw_ops_c DVE.sw_ops
proc_tag cpu0_core2
a2886: exclusive_release [181]
sml_sw_ops_c DVE.sw_ops
proc_tag cpu1_core2
a2538: mul
cdn_coher
proc_tag
4: multi_rw_cache [199]
coherency_ops_c DVE.cache_
tag cpu0_core3
a2676: multi_rw_cache [191]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core2
a2892: multi_rw_cache [205]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu1_core2
a2541: multi
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core0
760: invalidate_cache [201]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core3
a2679: multi_rw_cache [193]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core2
a2895: multi_rw_cache [207]
cdn_coherency_ops_c DVE.cac
proc_tag cpu1_co
a2547: invalidate_cache [189]
a2898: multi_rw_cache [20
cdn_coherency_ops_c DV
proc_tag cp
a2917: check_cache_region [211]
cdn_coherency_ops_c DVE.cache_ops
proc_tag cpu0_core0
a2920: check_cache_region [213]
cdn_coherency_ops_c DVE.cac
proc_tag cpu0_co
8
12
16
20
24
28
32
36
40
44
48
52
cpu0_core0(R)
cpu0_core0(W)
cpu0_core2(R)
cpu0_core2(W)
cpu0_core3(R)
cpu0_core3(W)
cpu1_core2(R)
cpu1_core2(W)

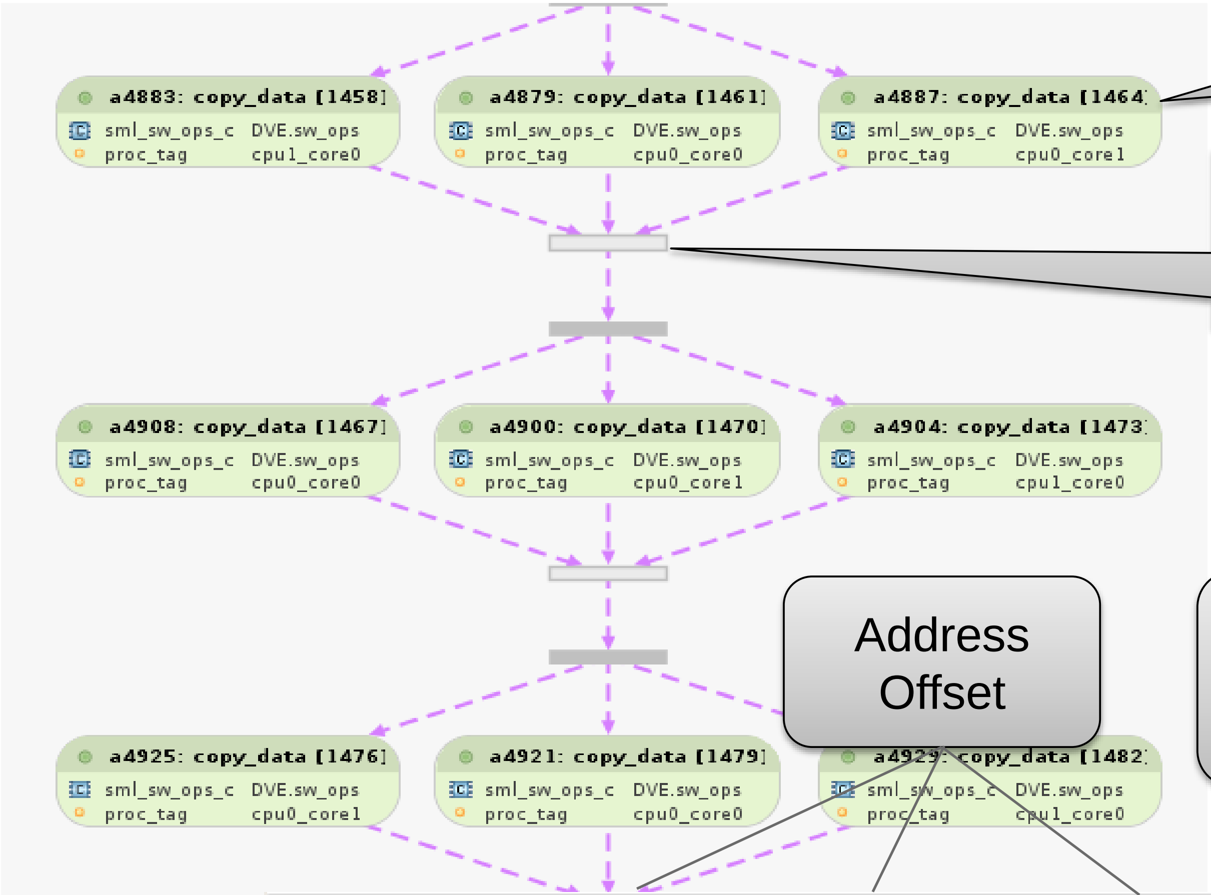
Processors run in parallel

Address offset

Processors split cache lines

Cache lines

Coherency – True Sharing Scenarios



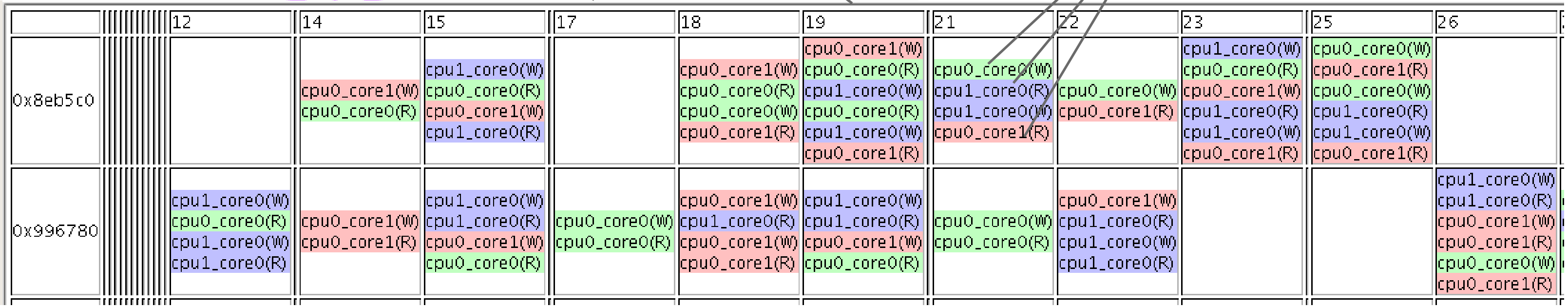
Processors run in parallel

And synchronize

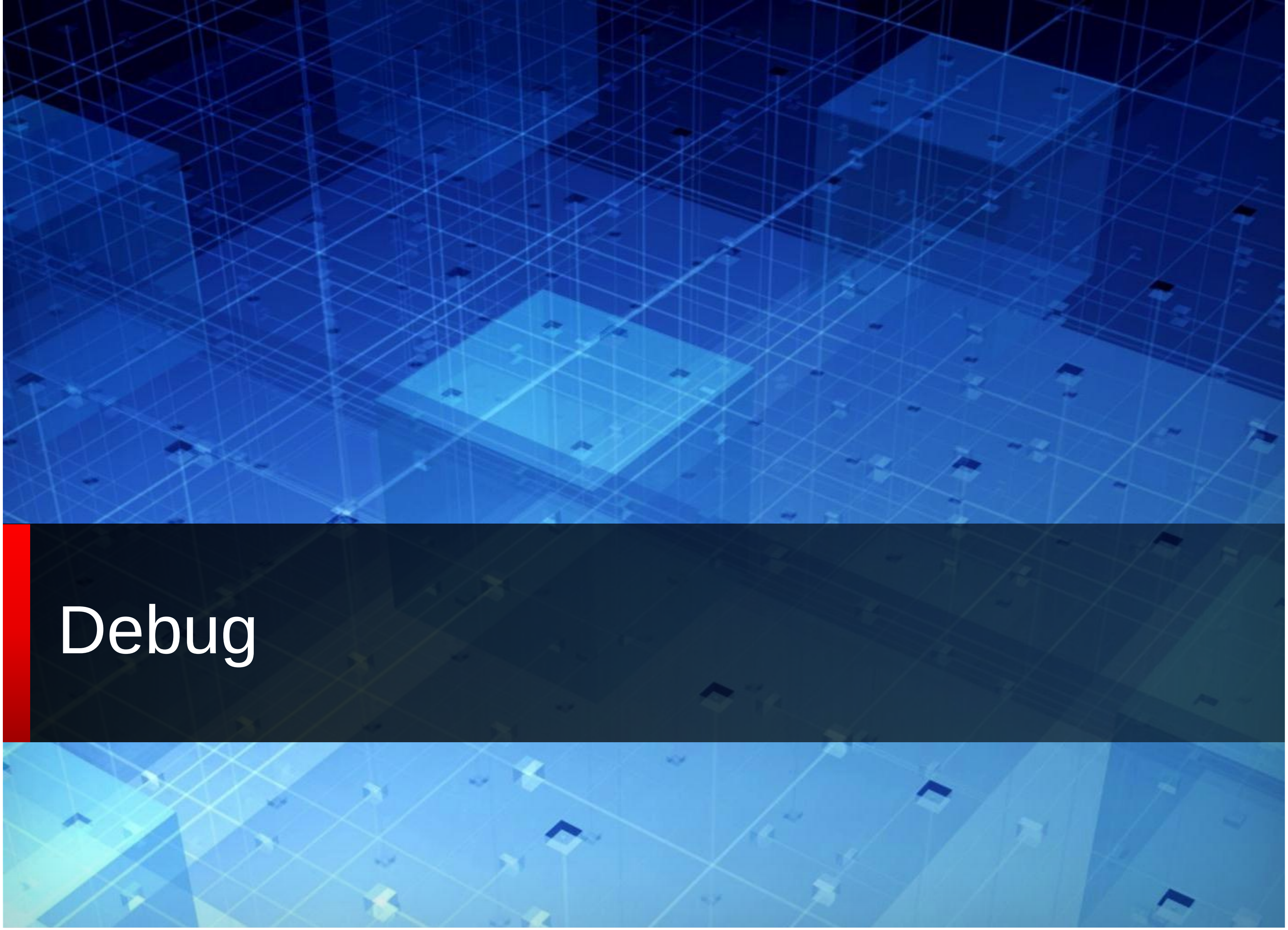
Address Offset

Processors access same address at different times

Cache Lines



		12	14	15	17	18	19	21	22	23	25	26
0x8eb5c0			cpu0_core1(W) cpu0_core0(R)	cpu1_core0(W) cpu0_core0(R) cpu0_core1(W) cpu1_core0(R)		cpu0_core1(W) cpu0_core0(R) cpu0_core0(W) cpu0_core1(R)	cpu0_core1(W) cpu0_core0(R) cpu1_core0(W) cpu0_core0(R) cpu1_core0(W) cpu0_core1(R)	cpu0_core0(W) cpu1_core0(R) cpu1_core0(W) cpu0_core1(R)	cpu0_core0(W) cpu0_core1(R)	cpu1_core0(W) cpu0_core0(R) cpu0_core1(W) cpu1_core0(R) cpu1_core0(W) cpu0_core1(R)	cpu0_core0(W) cpu0_core1(R) cpu0_core0(W) cpu1_core0(R) cpu1_core0(W) cpu0_core1(R)	
0x996780		cpu1_core0(W) cpu0_core0(R) cpu1_core0(W) cpu1_core0(R)	cpu0_core1(W) cpu0_core1(R)	cpu1_core0(W) cpu1_core0(R) cpu0_core1(W) cpu0_core0(R)	cpu0_core0(W) cpu0_core0(R)	cpu0_core1(W) cpu1_core0(R) cpu0_core1(W) cpu0_core1(R)	cpu1_core0(W) cpu1_core0(R) cpu0_core1(W) cpu0_core0(R)	cpu0_core0(W) cpu0_core0(R)	cpu0_core1(W) cpu1_core0(R) cpu1_core0(W) cpu1_core0(R)			cpu1_core0(W) cpu1_core0(R) cpu0_core1(W) cpu0_core1(R) cpu0_core0(W) cpu0_core1(R)

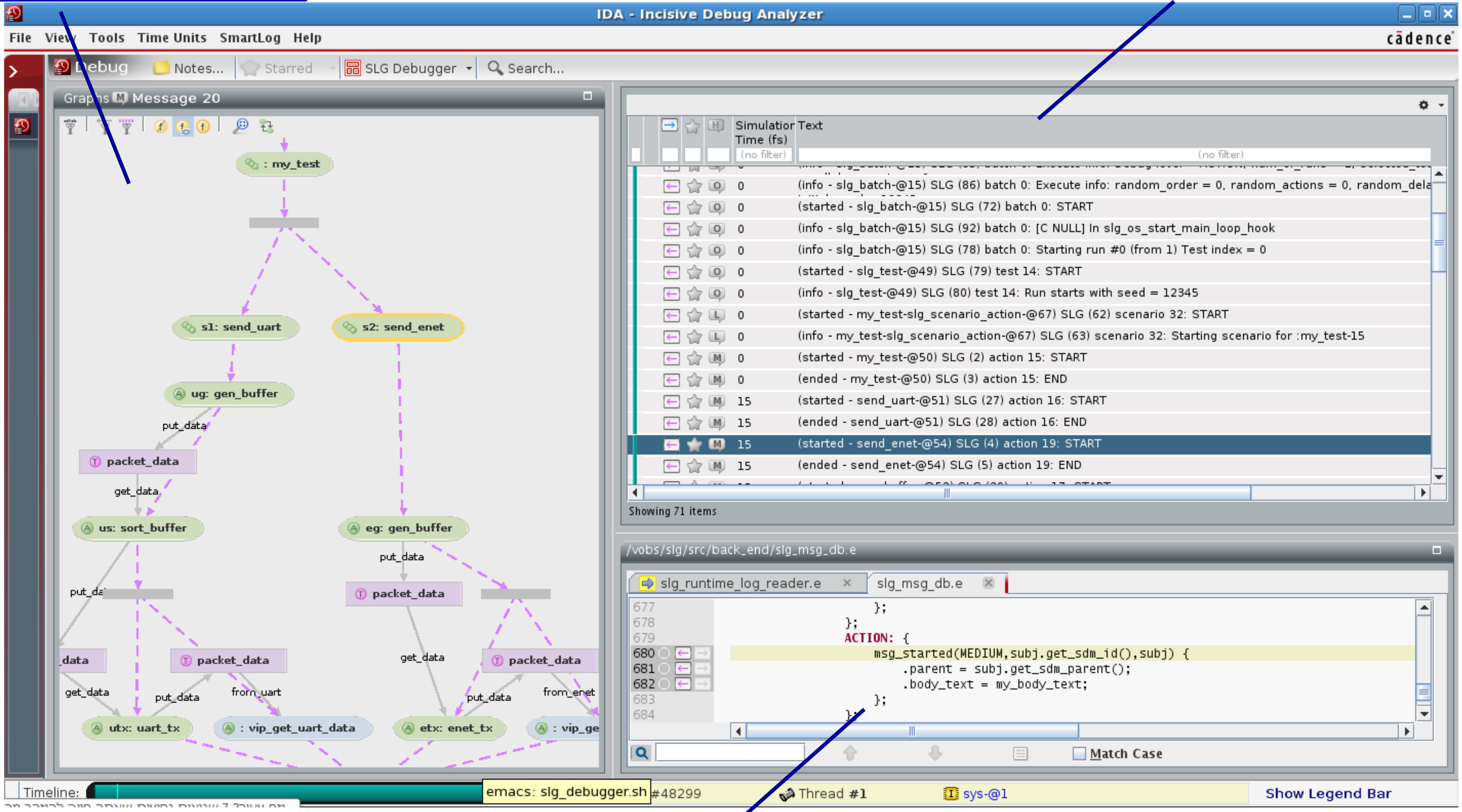


Debug

Perspec debug capabilities

Abstract debug using UML activity diagram

Smart log filtering, stepping, and searching



```
0    (info - slg_batch-@15) SLG (86) batch 0: Execute info: random_order = 0, random_actions = 0, random_dela
0    (started - slg_batch-@15) SLG (72) batch 0: START
0    (info - slg_batch-@15) SLG (92) batch 0: [C NULL] In slg_os_start_main_loop_hook
0    (info - slg_batch-@15) SLG (78) batch 0: Starting run #0 (from 1) Test index = 0
0    (started - slg_test-@49) SLG (79) test 14: START
0    (info - slg_test-@49) SLG (80) test 14: Run starts with seed = 12345
0    (started - my_test-slg_scenario_action-@67) SLG (62) scenario 32: START
0    (info - my_test-slg_scenario_action-@67) SLG (63) scenario 32: Starting scenario for :my_test-15
0    (started - my_test-@50) SLG (2) action 15: START
0    (ended - my_test-@50) SLG (3) action 15: END
15   (started - send_uart-@51) SLG (27) action 16: START
15   (ended - send_uart-@51) SLG (28) action 16: END
15   (started - send_enet-@54) SLG (4) action 19: START
15   (ended - send_enet-@54) SLG (5) action 19: END
```

```
            };
        };
        ACTION: {
            msg_started(MEDIUM,subj.get_sdm_id(),subj) {
                .parent = subj.get_sdm_parent();
                .body_text = my_body_text;
            };
        };
```

Lock-step execution of activity diagram, log, C code, and waveform