2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Practical Applications of the Portable Testing and Stimulus Standard (PSS)

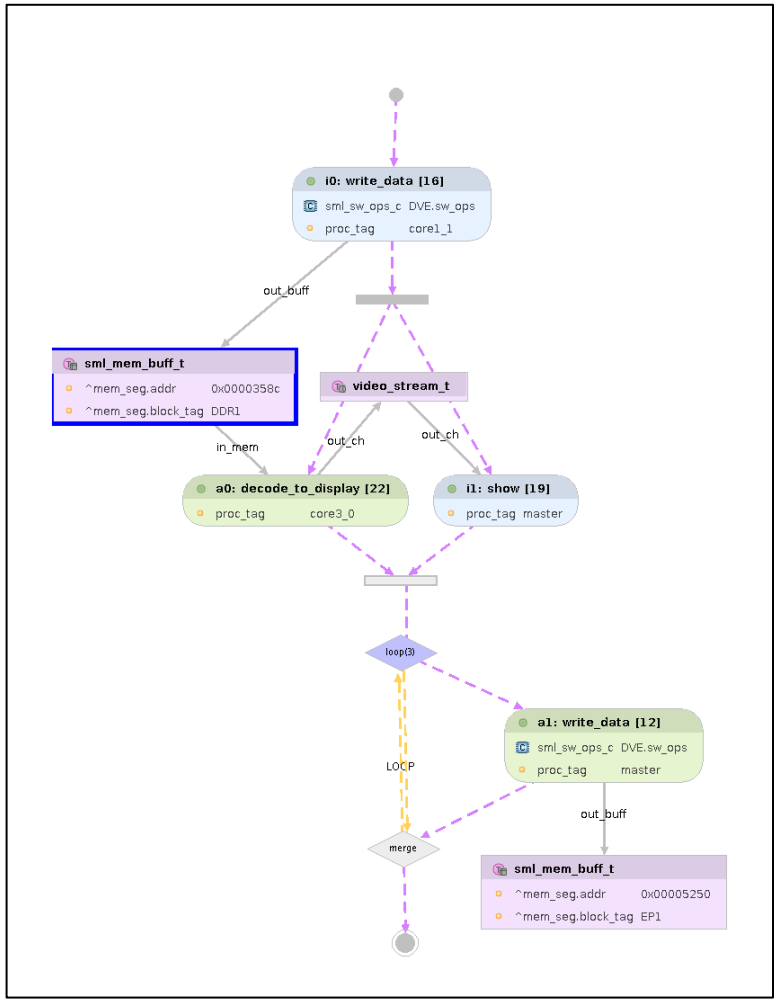
Sharon Rosenberg

cadence®

accellera SYSTEMS INITIATIVE

# What is Portable Testing and Stimulus Standard (PSS)?

- Behavioral standard language to express scenarios
  - Control flow with loops, conditionals
  - Parallelism and sequential execution (similar to fork and join)
- Powerful built-in verification-specific semantics for
  - Resource availability and distribution
  - Configuration, and operation modes
  - Data flow requirements
- Codified in two equally powerful input formats:
  - PSS C++ library – appeals to C++ users
  - PSS Domain Specific Language (DSL) – easier to read and better error messages
- Defined by PSWG in Accellera



# What You Need to Know About PSS

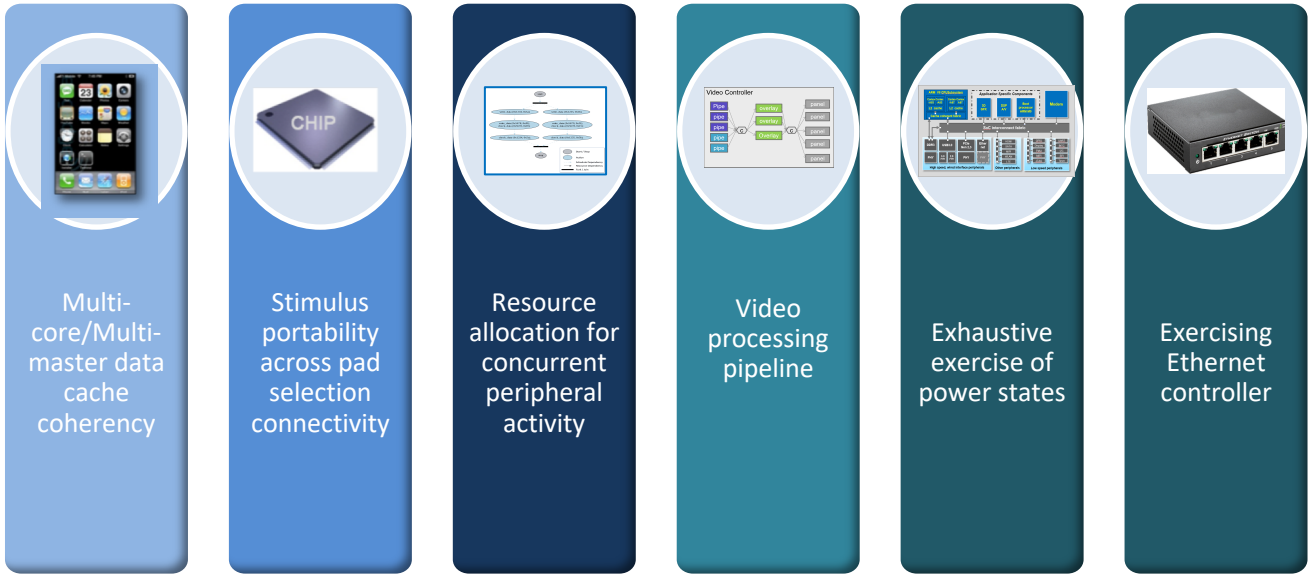
- Motivation and value
  - May vary between teams and verification objectives
  - Concepts, mindset, and syntax
- Modeling patterns and methodology
  - The same concepts can be applied differently to solve various problems
- PSS technology
  - Vendors can provide additional value on top of the standard semantics

*We are going to cover all the above in the context of specific applications*

# Accellera User-Contributed Usage-Examples

## Motivation:

- Define the committee's scope  and drive requirements
- Achieve a regression suite and thus a viable consistent solution across vendors




**Tip!**

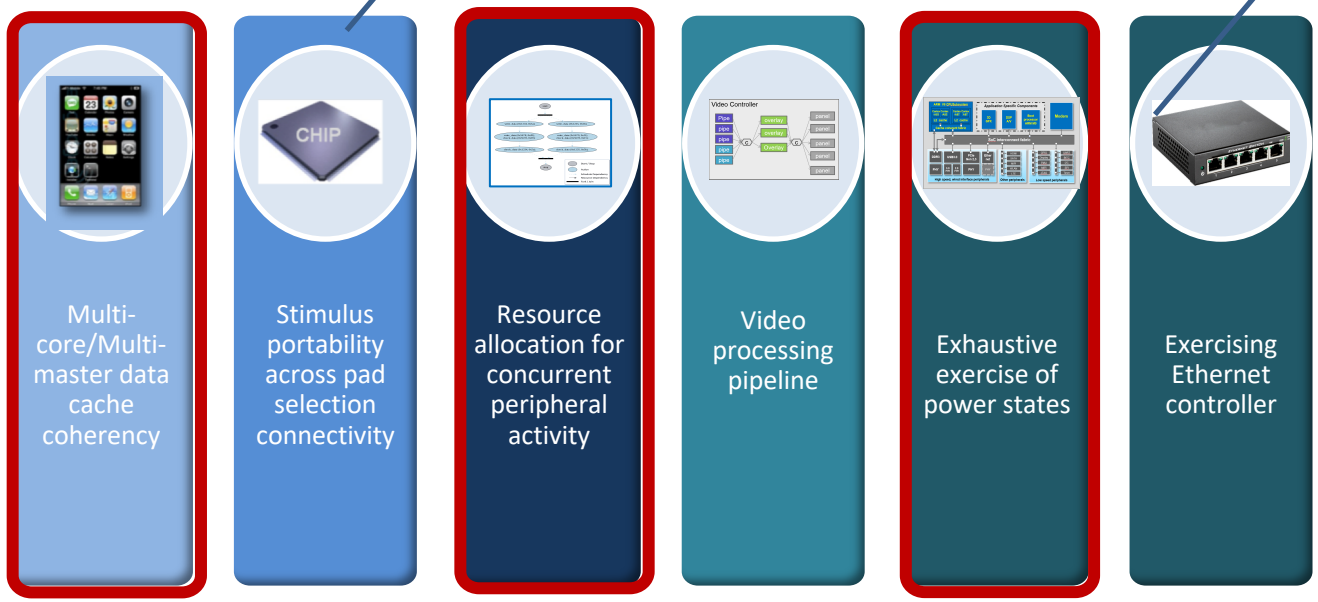
If and when you decide to adopt a PSS solution, ask your vendor to show that it can run the usage examples

# Accellera User-Contributed Usage-Examples

In this workshop we selected a couple of usage examples representing different requirements

We present two additional challenges at the end of the workshop that are demonstrated at the Cadence Booth throughout the week:

1. Multi-core/multi-master data cache coherency
2. Applying PSS to UVM




PSS Standard Scope

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Seminar Table of Content

- Introduction to PSS via the user-contributed usage example
  - DMA allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction to other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

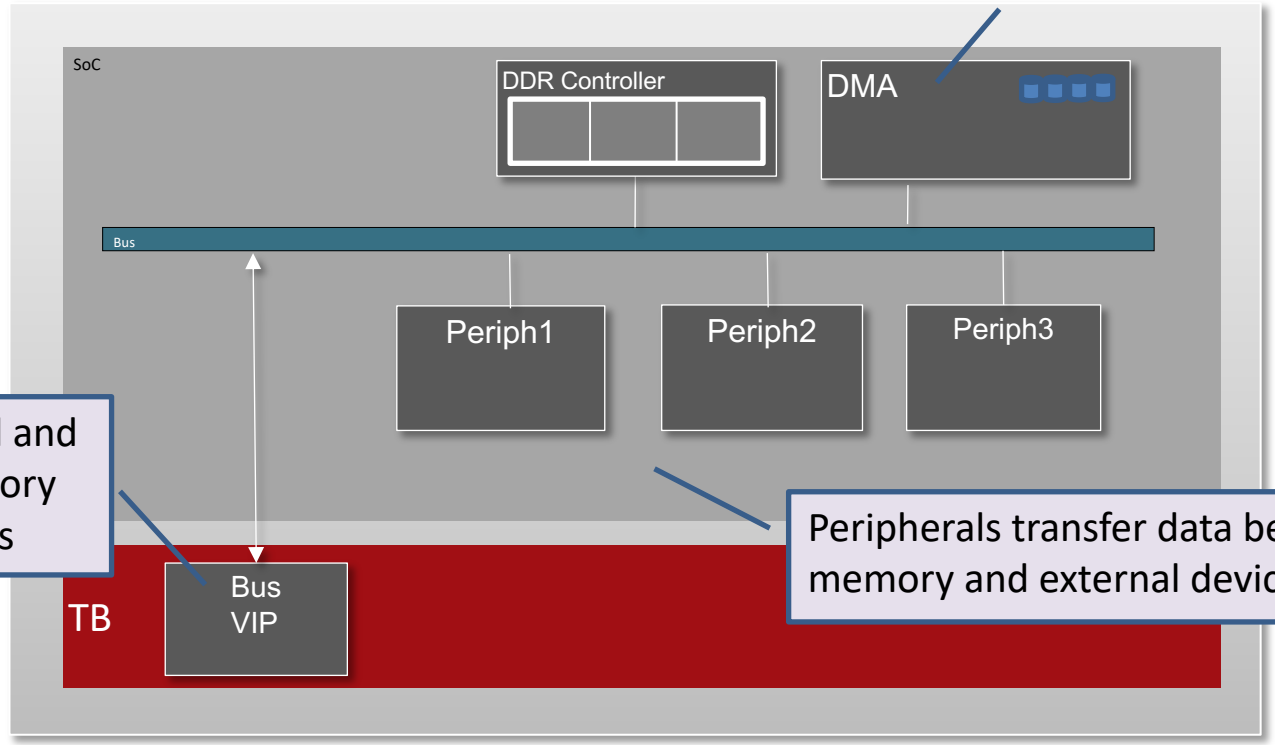
accellera SYSTEMS INITIATIVE

# DMA Allocation for Peripherals Challenge

**Tip!**

Before starting the modelling task, review the test API calls (firmware, sequences, etc')

Four-channel DMA engine moves data between three peripheral devices and memory




Testbench can load and check data in memory via backdoor access

Peripherals transfer data between memory and external device

# Let's do Verification Together!
# Assume we are experienced validation engineer…

Memory related rules:
- Don't use the same memory addresses for concurrent activities.
- Initialize the memory in areas that require read and check
- Not all cores may access all memories

System resource rules:
- There are limited number of DMA channels, CPU cores

Data dependencies rules:
- Don't write before the Peripheral buffer was initialized

my_firmware.h

```
// TB memory initialization
void tb_initial_mem(void *addr, char
*data);

// DMA programming
void dma_program(int chan_num,
    void* src_buff,
    void* dst_buff,
    int size);
void dma_start(int chan_num);
void dma_wait_for_done(int chan_num);

// Peripheral programming
void init_periph(int periph_num);
void periph_write(int periph_num);
Void periph_read(int periph_num);
```

Code can be executed on multiple CPU cores to activate system engines and devices, with a main function for each core, following the system rules

Power related rules:
- Don't use the DMA if it is powered-down

# Without PSS: Let's do Verification Together!

my_firmware.h

```
// TB memory initialization
void tb_initial_mem(void *addr, char
*data);

// DMA programming
void dma_program(int chan_num,
    void* src_buff,
    void* dst_buff,
    int size);
void dma_start(int chan_num);
void dma_wait_for_done(int chan_num);

// Peripheral programming
void periph_write(int periph_num);
Void periph_read(int periph_num);
```

my_first_test.c

```
// my first test
int main_core1()
{
    tb_initial_mem(0x5000,my_data);
    init_periph(2);
    dma_program(1, 0x5000, PERIPH2, 20);
    dma_start(1);
    dma_wait_for_done(1);
    signal_core(2);
    done(1);
}

int main_core2()
{
    wait_for_core(1);
    periph_write(2);
```

Is this a good system test?

No! Need to signal core2 that initialization is done

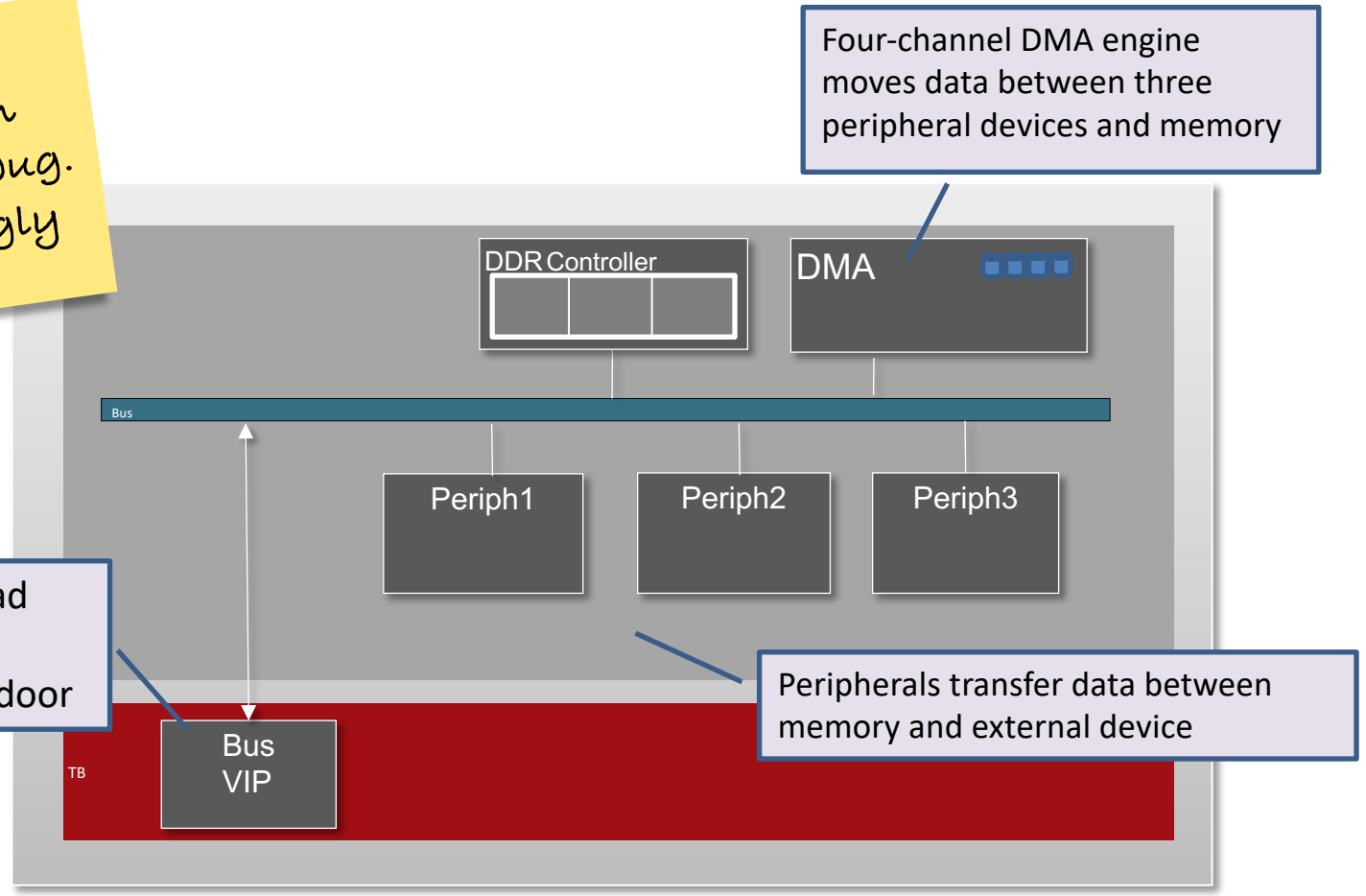
No! Must wait for peripheral initialization to complete

Consider activating 8 cores in parallel for long scenarios, data needs to be shared, cores and devices are powering on or off, operation modes can impact activities – Long iterative process per test!

# DMA Allocation for Peripherals Challenge

**Tip!**

Typically different people own modeling, test creation, and debug. Organize your team accordingly



# PSS Behavioral Modeling Motivation

```
component periph_c {
  action write_out {…};
  action read_in {…};
};
```

Actions are “pieces of behavior”

periph PSS

```
component dma_c {
    action m2q_xfer {…};
    action q2m_xfer {…};
    action m2m_xfer {…};
};
```

DMA PSS


Desired scenarios are captured in compound action activity block

```
action my_scenario {
    activity {
        do m2m_xfer;
        repeat (5) {
            parallel {
                do write_out;
                do read_in;
            };
        };
    };
};
```

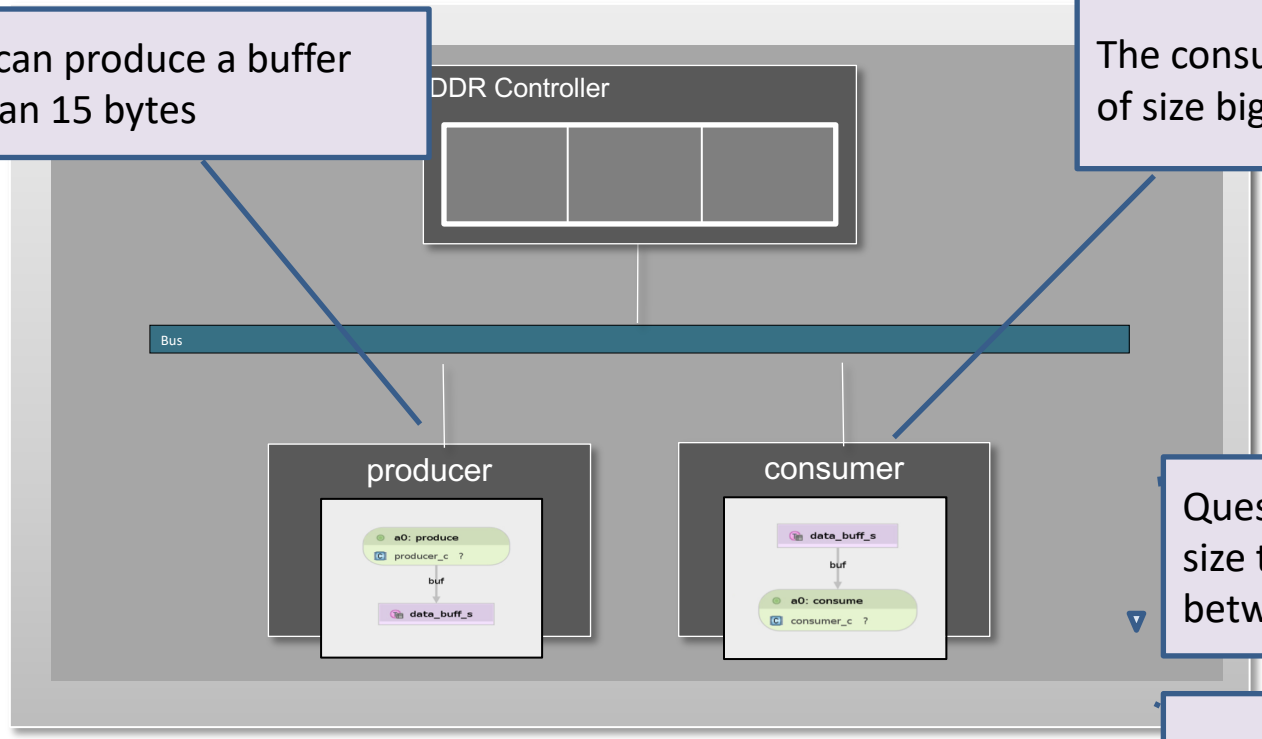
Legal test considerations:

- We should not read non-initialized data
- How do we capture constraints between behaviors?
- Do we have enough resources to this parallel activity? What is a legal resource distribution?
- Is there a configuration that supports this same time activity?
- How do we translate this loop to run within embedded cores, C++ host, SV code?

# About Input and Output Flow-Objects

## Consider producer and consumer IPs connected to system memory

The producer can produce a buffer size smaller than 15 bytes




The consumer can consume a buffer of size bigger than 10 bytes

Question: What will be a proper data size to enable communication between them?

Answer: 11..14 bytes

System assumptions – IPs can be programmed to communicate if they are connected to the same memory, and no restrictions prevent that communication (e.g. data size or data kind mismatch)

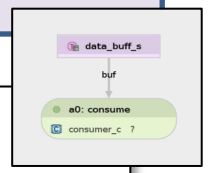
# About Input and Output Flow-Objects (Cont')

## Consider producer and consumer IPs connected to system memory

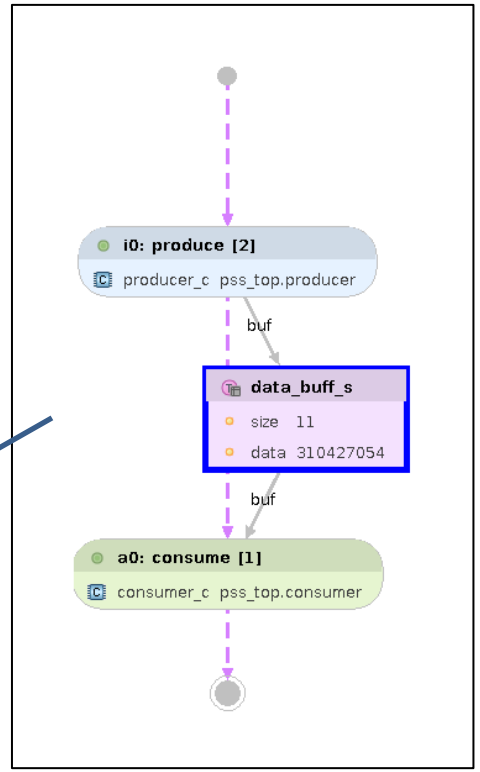
- A **buffer** for modeling memory – once written, the data persists and can be read many times.
- A **stream** is for same-time communication. Producing and consuming must take place simultaneously.

```
stream data_buff_s {
  rand int in [1..20] size;
  rand int data;
};
```

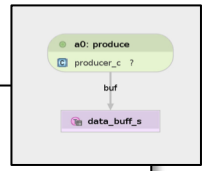
PSS allows capturing the dependencies in a special flow-object struct




```
component consumer_c {
    action consume {
        input data_buff_s buf;
        constraint buf.size > 10;
    };
};
```




A constraint solver solved the scenarios to achieve a legal programming sequence.



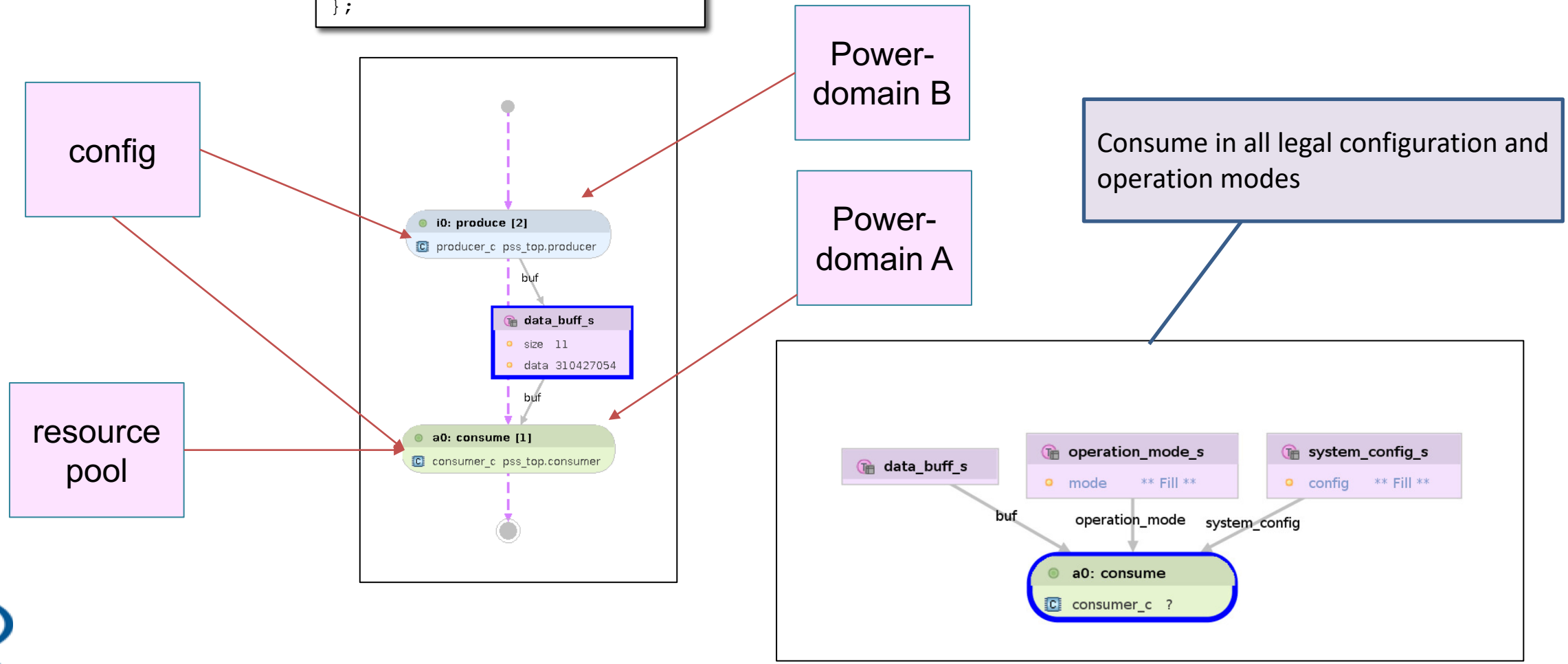

```
component producer_c {
    action produce {
        output data_buff_s buf;
        constraint buf.size < 15;
    };
};
```

Each sub-system model captures its own dependencies according to its specifications.

# About Input and Output Flow-Objects (Cont')

## Consider producer and consumer IPs connected to system memory

```
stream data_buff_s {
  rand int in [1..20] size;
  rand int data;
};
```

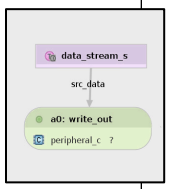


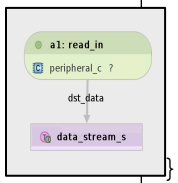
# PSS Behavioral Modeling with

Can choose PSS C++ or PSS (DSL)

periph PSS C++

```
class periph_c : public component {
  attr<int> periph_id { ”periph_id” };
    PSS_CTOR(periph_c,component);

  class write_out : public action {
    PSS_CTOR(write_out,action);
    input<data_stream_s> in{"in"};
  };
  type_decl<write_out> write_out;

  class read_in : public action {
    PSS_CTOR(read_in,action);ß

    output<data_stream_s> out{"out"};
    constraint c1 { … };
  };
  type_decl<read_in> read_in;
};
```

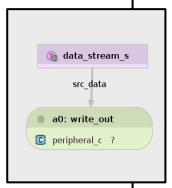


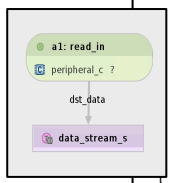



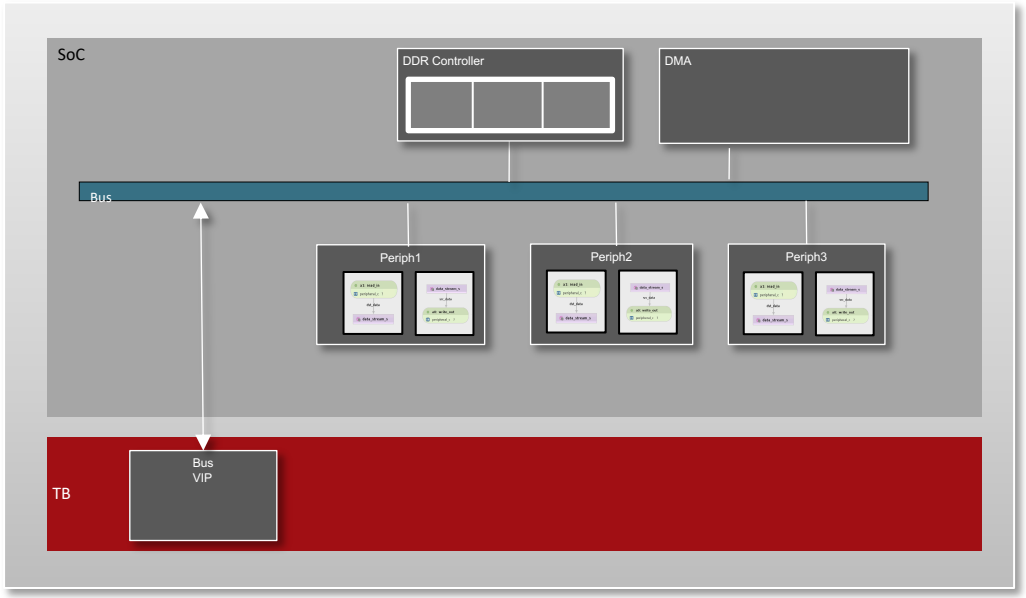

All the peripheral dependencies are captured in PSS

periph PSS

```
component periph_c {
  action write_out {
    input data_stream_s src_data;
    …
  };

  action read_in {
    output data_stream_s dst_data;
    …
  };
};
```



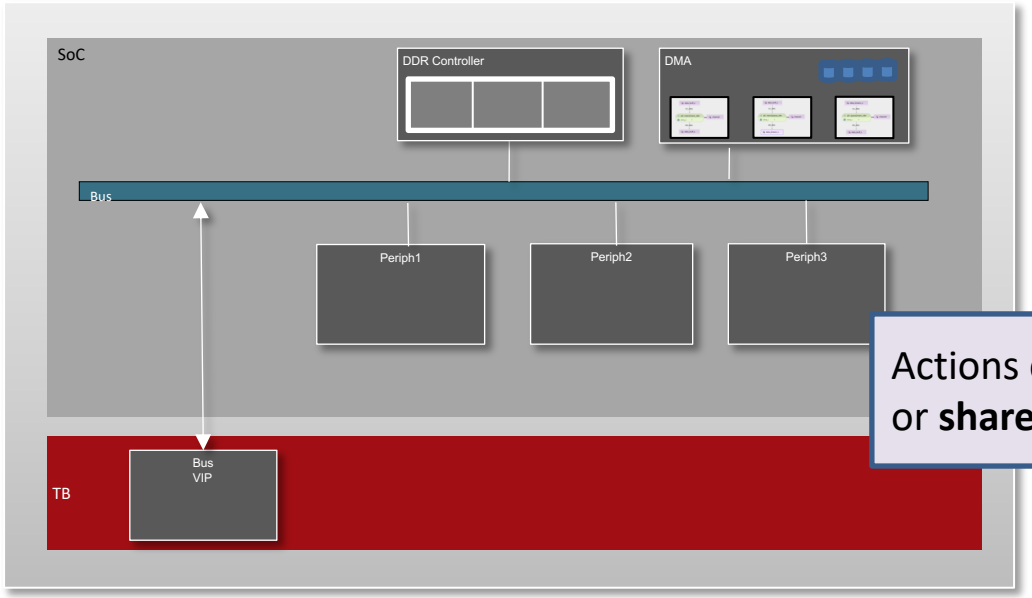



# PSS Behavioral Modeling

## Model Writer Defines Actions and Rules

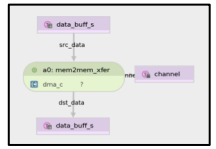
Code is:
- Concise Intuitive translation of the functional Spec
- Well-encapsulated and reusable

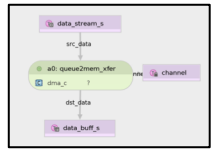
Define a **pool** of four channels




Actions can **lock** or **share** resources

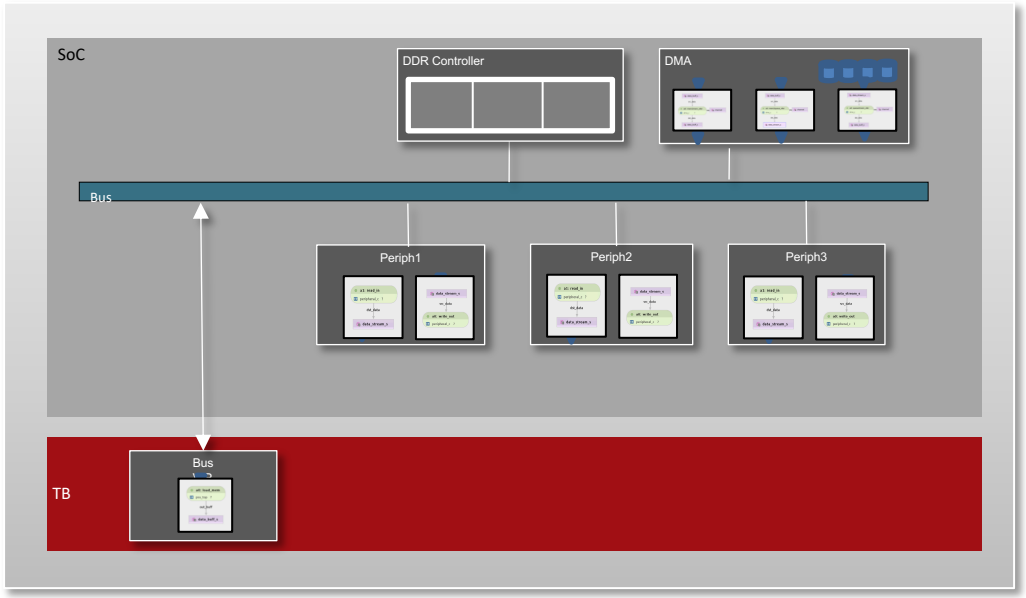
```
component dmac_c {
    resource dma_channel_r {};
    pool [4] dma_channel_r chan_p;
    bind chan_p *;

    action m2m_xfer {
        input data_buff_b src;
        output data_buff_b dst;
        lock dma_channel_r channel;
        constraint src.size == dst.size;
    };

     action q2m_xfer {
        input data_stream_s src;
        output data_buff_b dst;
        lock dma_channel_r channel;
        constraint src.size == dst.size;
    };

    action m2q_xfer {…};
};
```



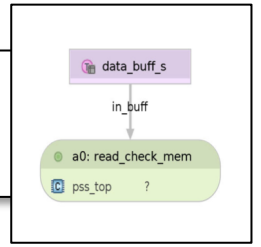


# DMA Allocation for Peripherals

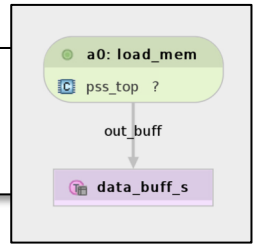
## Model-Writer Defines Actions and Rules




```
action read_check_mem {
    input data_buff_s data_buff;
};
```




```
action load_mem {
    output data_buff_s data_buff;
};
```



# Action Implementation Layer

- Exec 'body' block specifies implementation
  - Call init_periph(), upon starting
  - Call periph_write() at the right time in the test

```
function void i
function void p

extend action p
    exec run_st
        init_pe
    }
    exec body {
        periph_w
    }
}
```

```
extend component dma_c {
    function void dma_program(int channel_id, sml_addr_t src_addr,
sml_addr_t dst_addr, int size);
    function void dma_start(int channel_id);
    function bool dma_is_done(int channel_id);
}
extend action dma_c::transfer {
  exec body {
      comp.dma_program(channel.instance_id,…,in_buff.mem_seg.size);
      comp.dma_start(channel.instance_id);
      while (!comp.dma_is_done(channel.instance_id)) {
          yield();
}}}};
```

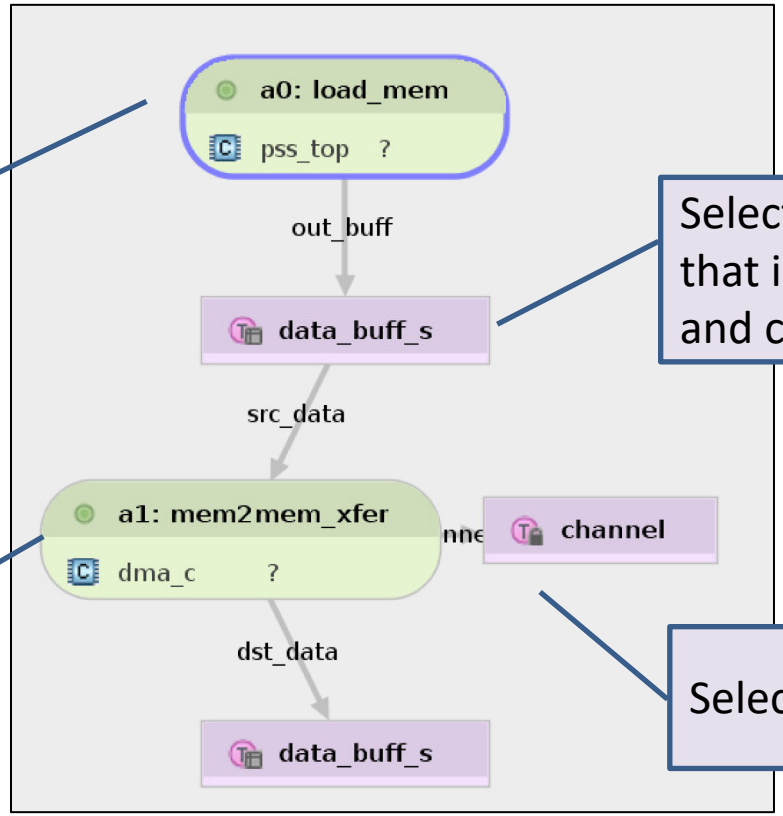
- PSS supports OOP and Aspect Oriented Programming (AOP)
- Allows separation of abstract model to multiple concrete implementations

Can be implemented in any language or testbench

Perspec generates the described control flow in the target platform language

# My First Test Code

My first test: load the memory with data and use the DMA to copy it to a different location




Which CPU core will call the load_mem?

Select legal memory location that is reachable by producer and consumer

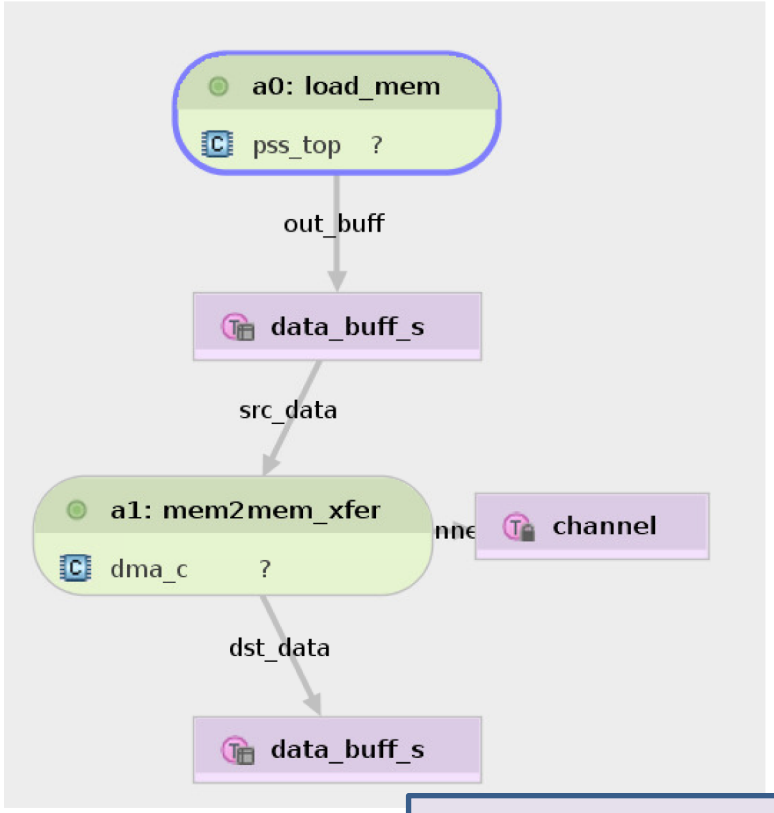
Which CPU core will configure the DMA? Add sync if needed

Select a DMA channel

# My First Test Code (Cont')

My first test: load the memory with data and use the DMA to copy it to a different location

This might be a UVM virtual sequence creating the same test




```
// my first test
int main_core3()
{
    tb_initial_mem(0x5000,my_data);
    signal_core(1);
    done(1);
}

int main_core1()
{
    wait_for_core(1);
    dma_program(2, 0x5000, 0x700, 20);
    dma_start(2);
    dma_wait_for_done(2);
    done(1);
```

User firmware code

- Tool generated code
- Synchronizations, loops, fork and joins, all are done by the PSS tool

# Model Top Instantiation

```
component pss_top {
   peripheral_c periph_a;
   peripheral_c periph_b;
   peripheral_c periph_c;

   dma_c dma;
   // Bind data_buff_s and data_stream_s
   // pools to periph_* and dma components

   pool data_buff_s sysmem;
   bind sysmem *;
   pool data_stream_s sysfabric;
   bind sysfabric *;
};
```

The entire component hierarchy is instantiated directly or indirectly with a singleton pre-defined component called **pss_top**.

Instantiation of flow-objects and binding them to the sub-components

The system memory is instantiated by the integrator following the system configuration

*This simple semantics of actions, inputs and outputs, and resources can be analyzed by tools and enable an efficient automated use-case creation*

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# DMA Allocation Demo


accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Seminar Table of Content

- Introduction of PSS via the user-contributed usage example
  - DMA Allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction of other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

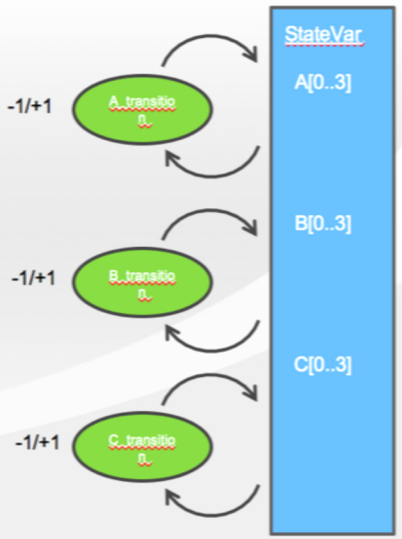
accellera SYSTEMS INITIATIVE

# PSS Support for State-Machines and Operation Modes

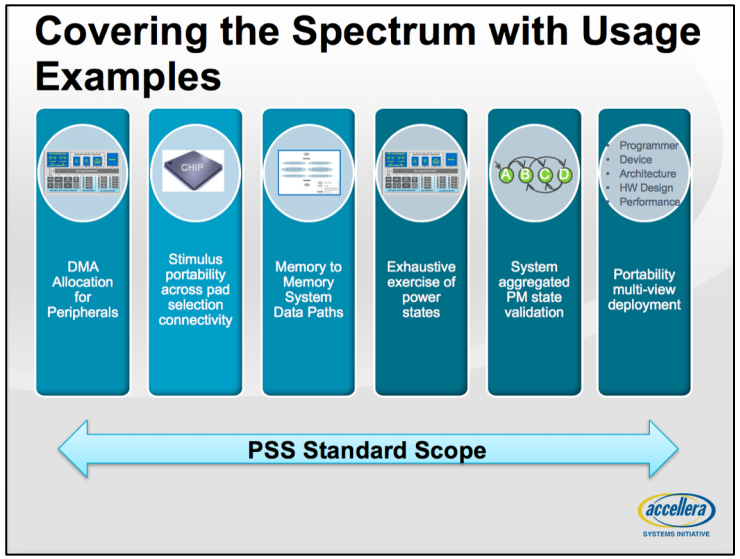
## Exhaustive Exercise of Power States

**Key Requirements and challenges**

- **Product under test:**
  - Three power domains A,B,C
  - Each has four power levels
    - 0: off, 1-3: functional states
  - Domain dependencies
    - level(A) >= level(B)
    - Domain C can be functional iff domain B is off
- **Transitions must be in consecutive steps**
- **Challenge:**
  - Automate complex legal state walks
  - Use coverage to define a subset of the desired walks (per team's role, amount of cycles etc')




accellera SYSTEMS INITIATIVE




Low-power related use-cases are one of the most asked-for user use-cases

# Capturing the State Space

## Model Writer Defines Actions and Rules

```
state power_state_s {
    rand int in [0..3] domain_A,
                       domain_B,
                       domain_C;

    // Level of domain A must be greater or equal
    // to that of B
    constraint domain_A >= domain_B;

    // Domain C can be in a functional state only
    // if B is off
    constraint (domain_C != 0) -> domain_B == 0;

    constraint (initial) -> {
      domain_A == 0 ;
      domain_B == 0 ;
      domain_C == 0;
    }
};
```

- "Each of the domains – A, B, and C – can be in any one of 4 power levels, level 0 (off), and level 1 to 3 (functional states)"
- "The level for switch A must be greater or equal to that of B"
- "Domain C can be in a functional state only if B is off "

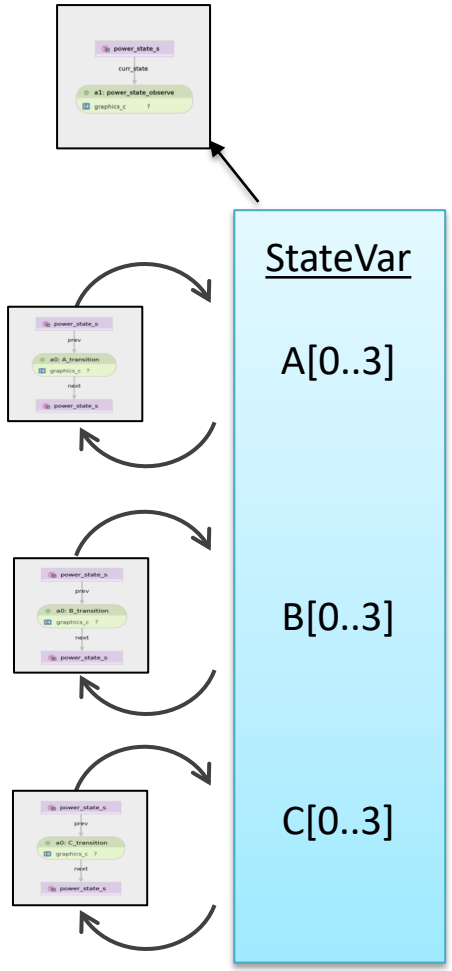
| StateVar |
| --- |
| A[0..3] |
| B[0..3] |
| C[0..3] |

# Transition Actions with Inputs and Outputs

## Model Writer Defines Actions and Rules

```
abstract action power_transition {
      rand int in [-1,1] step;
      input power_state_s prev;
      output power_state_s next;
      constraint A {next.domain_A == prev.domain_A; };
      constraint B {next.domain_B == prev.domain_B; };
      constraint C {next.domain_C == prev.domain_C; };
};
```

```
action A_Transition : power_transition {
     constraint A {next.domain_A == prev.domain_A + step; }
};
action B_Transition : power_transition {
     constraint B {next.domain_B == prev.domain_B + step; }
};
action C_Transition : power_transition {
    constraint C {next.domain_C == prev.domain_C + step; }
};
action power_state_observe {
    input power_state_s curr_state;
};
```



# Exhaustive Exercise of Power States

## Test-Writer Scenario Specification

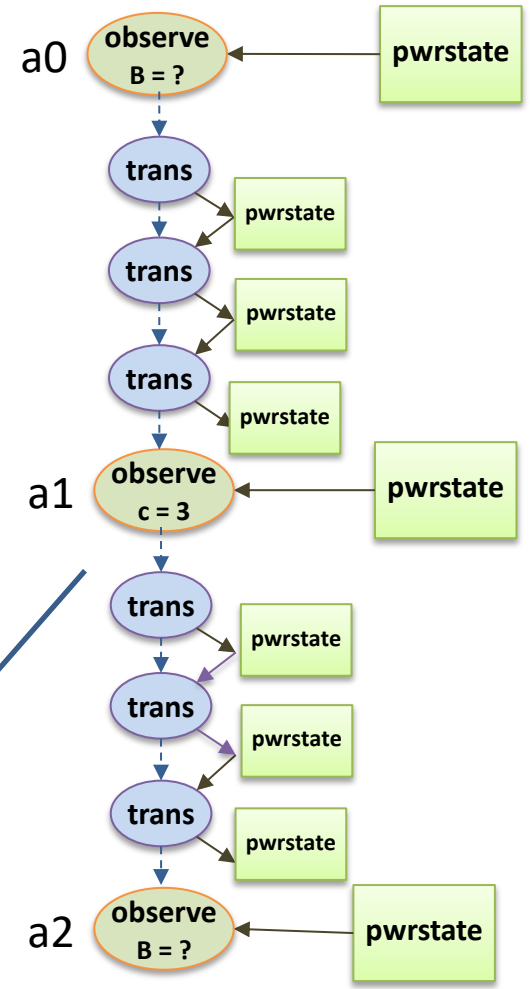
Test requirement:
Get domain_B to be active, followed by domain_C to be in power-level 3, followed by doman_B to be active.

Scenario specification

```
action full_C_between_functional_Bs_a {
   rand int[1..3] first_B_state;
   rand int[1..3] second_B_state;
   activity {
       do power_state_observe_a a0 with {
         curr_state.domain_B == first_B_state;
       };
       do power_state_observe_a a1 with {
            curr_state.domain_C == 3;
       };
       do power_state_observe_a a2 with {
           curr_state.domain_B == second_B_state;
       };
    };
};
```

With this approach, a tool can auto-complete a legal chain per user request.

Scenario instance




**Tips!**

Use the model to check that all power states and all transitions are legal

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Exhaustive Exercise of Power State Demo


accellera SYSTEMS INITIATIVE

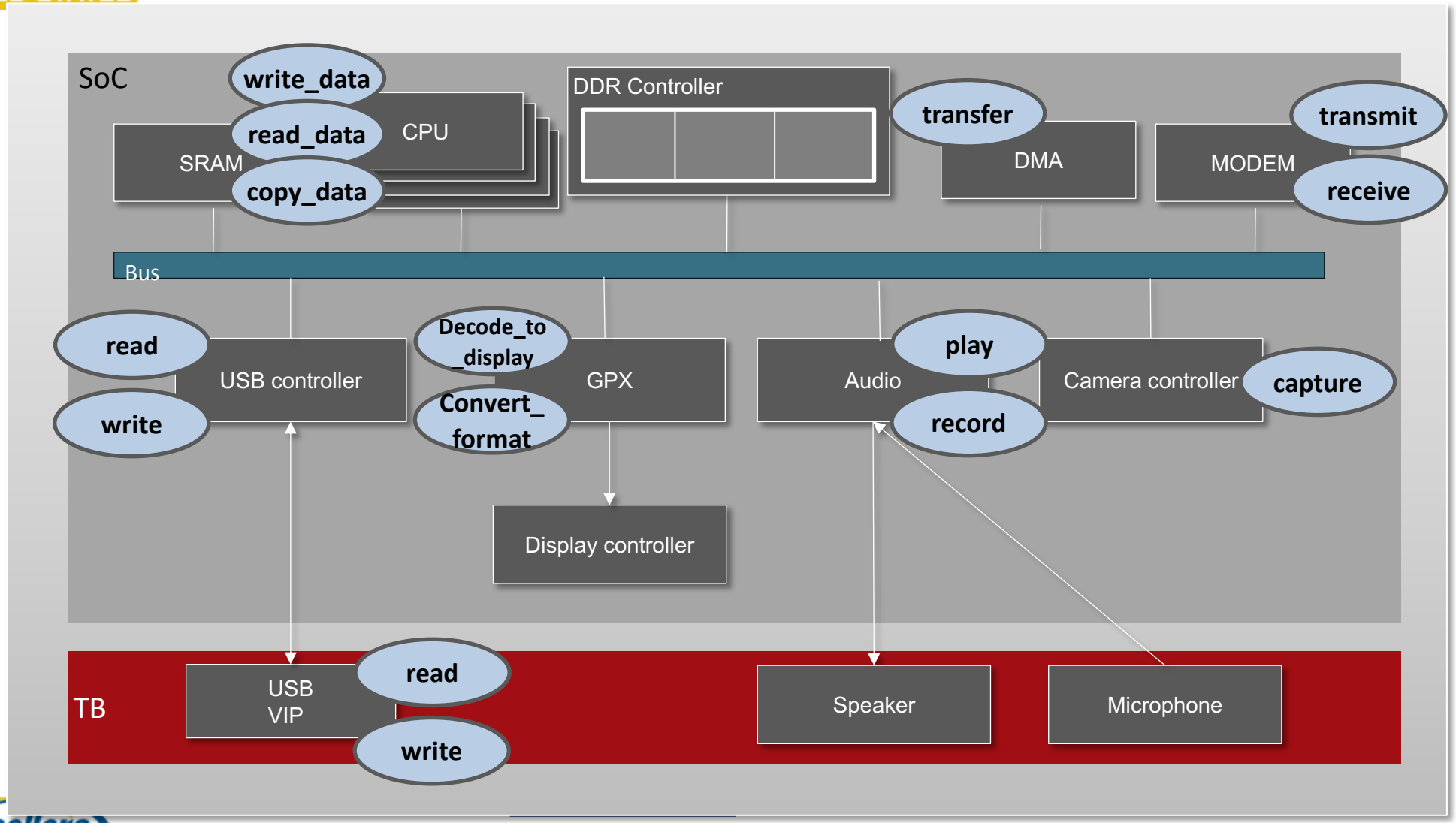
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Seminar Table of Content

- Introduction of PSS via the user-contributed usage example
  - DMA Allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction of other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

accellera SYSTEMS INITIATIVE

# Modeling Abstract System Behavior

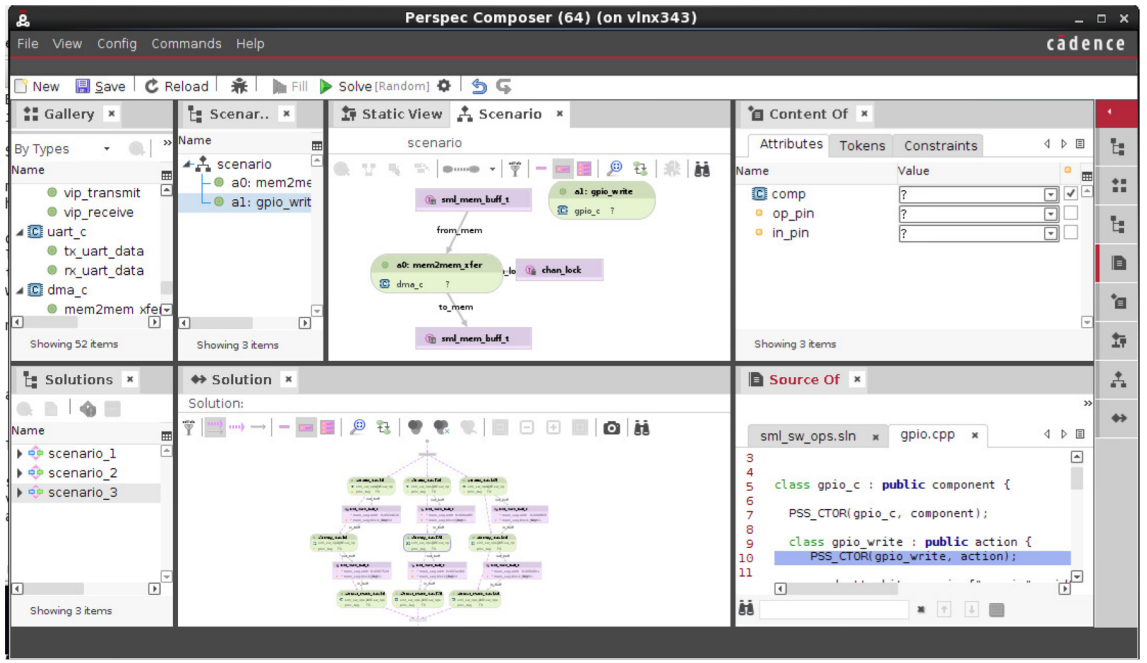



- camera_c
  - capture
- dma_c
  - transfer
- modem_c
  - receive
  - transmit
- gpx_c
  - decode_to_display
  - convert_format
- usb_c
  - read
  - write
- usb_agent_c
  - read
  - write
- display_c
  - show
- audio_c
  - play
  - record

# Goals: Measure Your Potential as a PSS User

Scenario creation

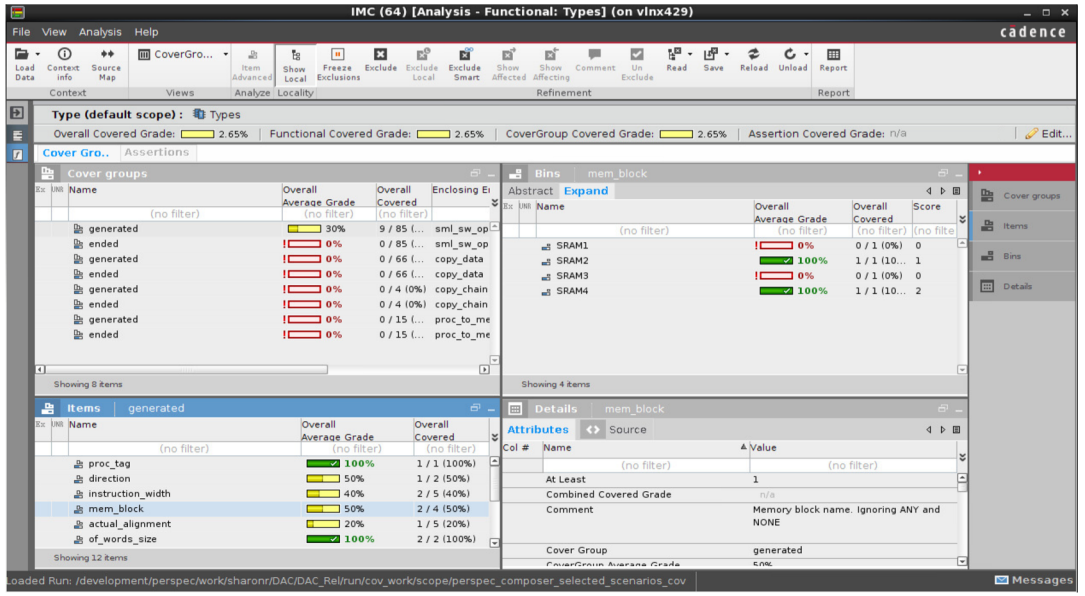
- Task1: prescribed scenario
  - Capture a video from the camera
  - Copy the data three time by selecting either
    - The CPU_core: copy_data
    - The DMA device: transfer
  - Check the result using read_check_data
  - Tip: use repeat, select
- Task2: Use PSS resource-aware random scheduling
  - Generate two random tests with
    - 2 CPU write data
    - 5 DMA transfers
  - Tip: use schedule



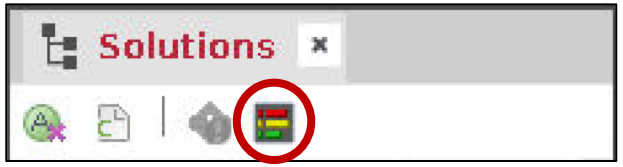

UML based GUI - composer

Password: CdnDVCon2019
%cd yamm_sml_pss/yamm/run
% ./.run.sh

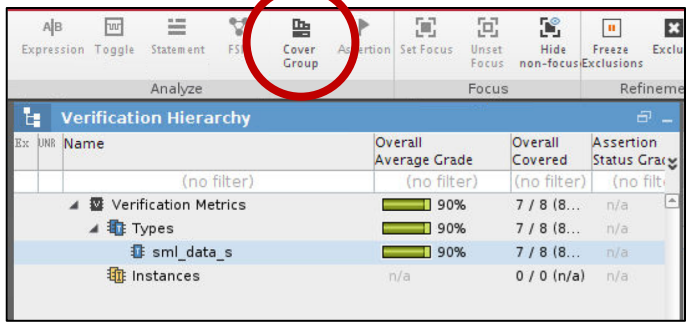
# Goals: Measure Your Potential as a PSS User




- PSS functional coverage analysis
  - Select all the generated solutions from the solution pane (use the shift button)
  - Click on the coverage button to open the PSS coverage viewer (IMC)




- Observe the coverage on the memory blocks
  - Select types->sml_data_s
  - Click on the cover group button
  - What sizes of data will be exercise?




*Good Luck!*

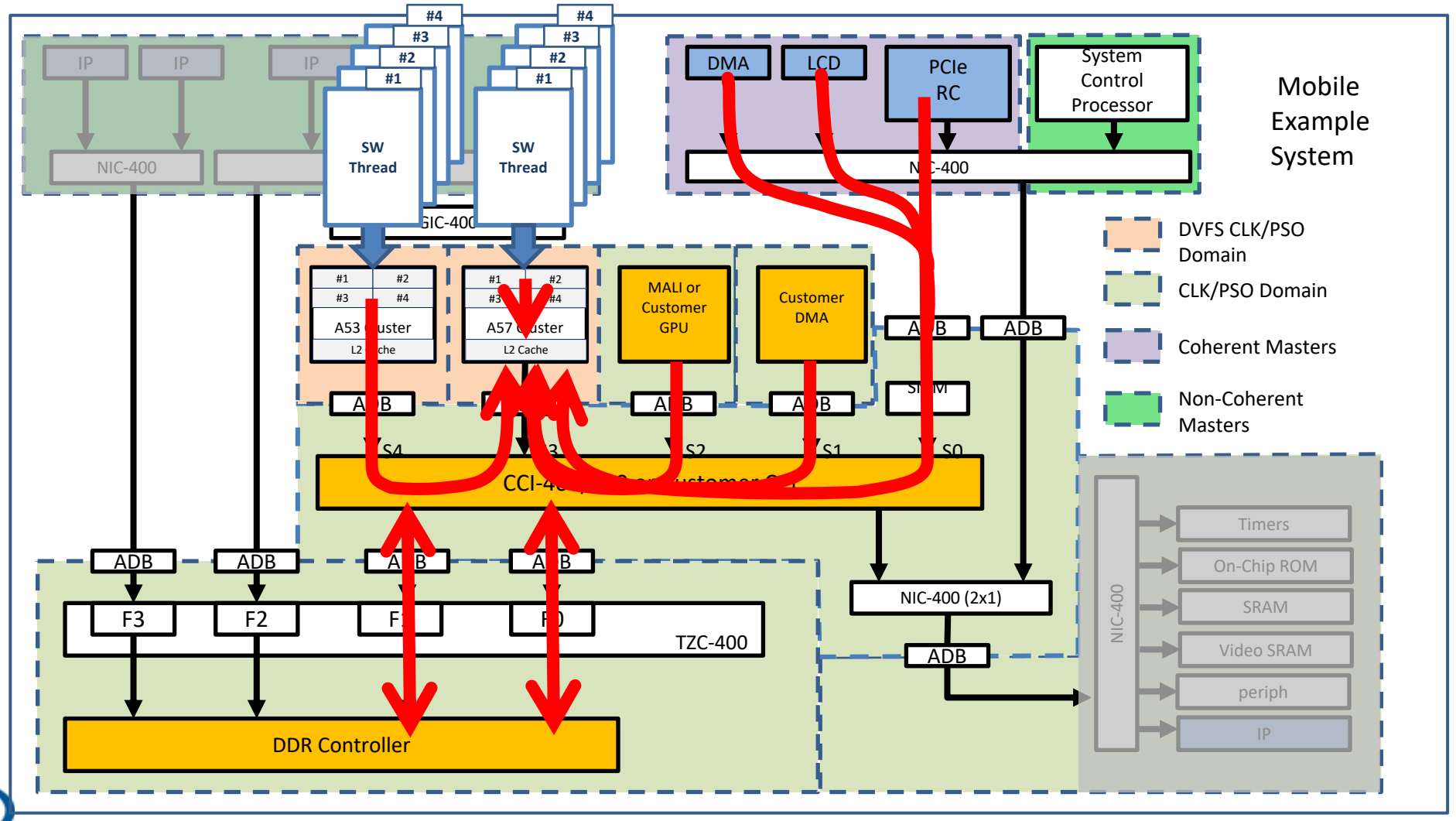
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

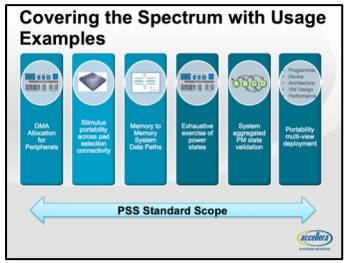
# Seminar Table of Content

- Introduction of PSS via the user-contributed usage example
  - DMA Allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction of other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

accellera SYSTEMS INITIATIVE

# PSS Support for Coherency and Low-power



# Productivity with **Built-in Content**

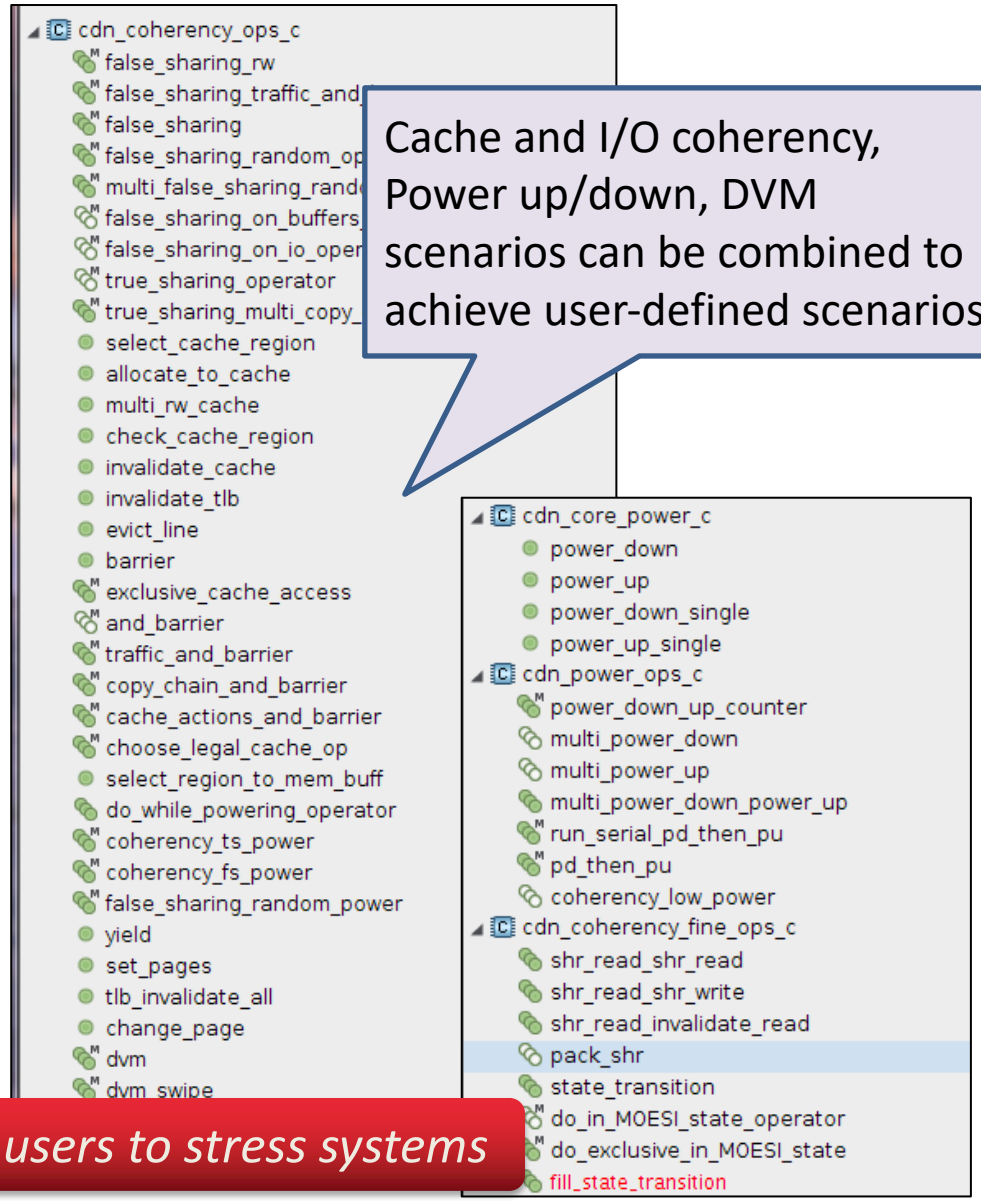
- Requirements/opportunities:
  - Much of the SoC logic is common
  - Libraries can be built for many aspects to provide readability, reuse and out of the box-content
- Cadence PSS reusable action libraries

**System Methodology Library (SML)**

- Captures system modeling including memories, processors, and more
- User provides Excel configuration tables

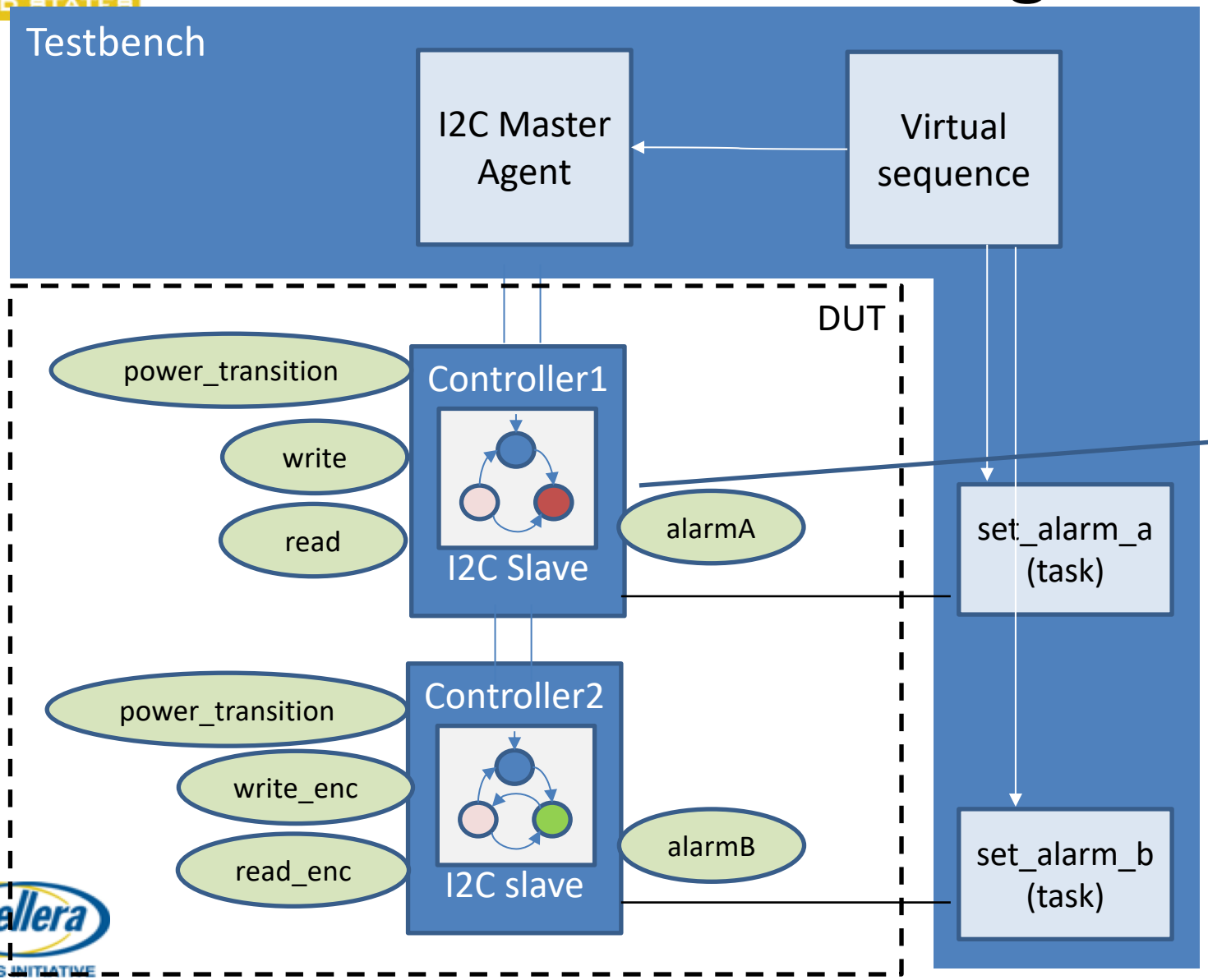
**Coherency Library**

- Provides built-in flexible content for system verification
- e.g coherency, DVM, and low-power scenarios




*Out-of-the-box scenarios and building blocks co-developed with users to stress systems*

# Consider the Following UVM Challenge:



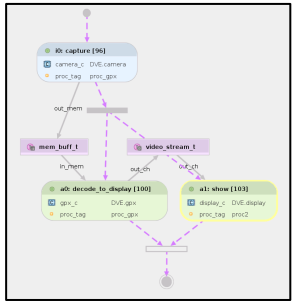

The challenge:

- Visit all power_states
- Send different traffic kinds and mix alarms in all states

The controller has a dozen state-machines that require complex virtual sequences.
HW Verification manager: “I do not know where to start!?”

# Perspec Value for UVM Users

- Automating UVM virtual sequence logic
  - Smart, quality tests reduce manual effort while improving regression quality and thoroughness
  - Lightweight solution to complement and further leverage the existing UVM assets
- Systematic coverage and verification goals filling (coverage maximization)
  - Better aiming at the hard to achieve remaining coverage goals
  - Optimized solution with controlled repetitions
- Portability
  - Allows using the same scenarios on VIP and AVIP
  - Core-to-coreless reuse
  - Portable programming sequences (not part of PSS yet)
- Performance
  - Reduces the randomization time by legally mixing pre-generated traffic
- Simplified test creation via UML GUI



2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Seminar Table of Content

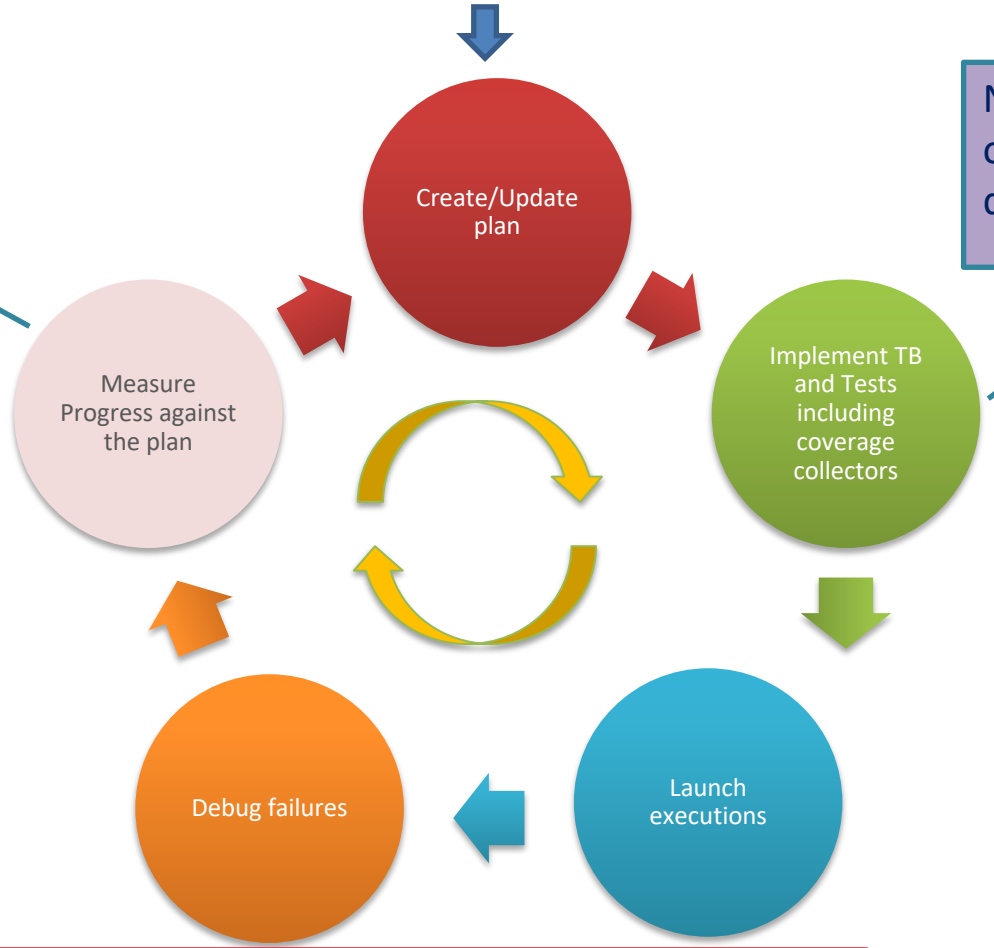
- Introduction of PSS via the user-contributed usage example
  - DMA Allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction of other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Traditional MDV Flow

- Coverage is measured after simulation is done
- Manual work to achieve 100% coverage or for declaring a value to be unreachable (ignored)




Need to re-do the work in case of testbench changes or a different simulator release

*Perspec modifies the traditional MDV flow by optimizing and further automating each step*

accellera SYSTEMS INITIATIVE

*Join our data driven verification tutorial or visit us at the Cadence booth*

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

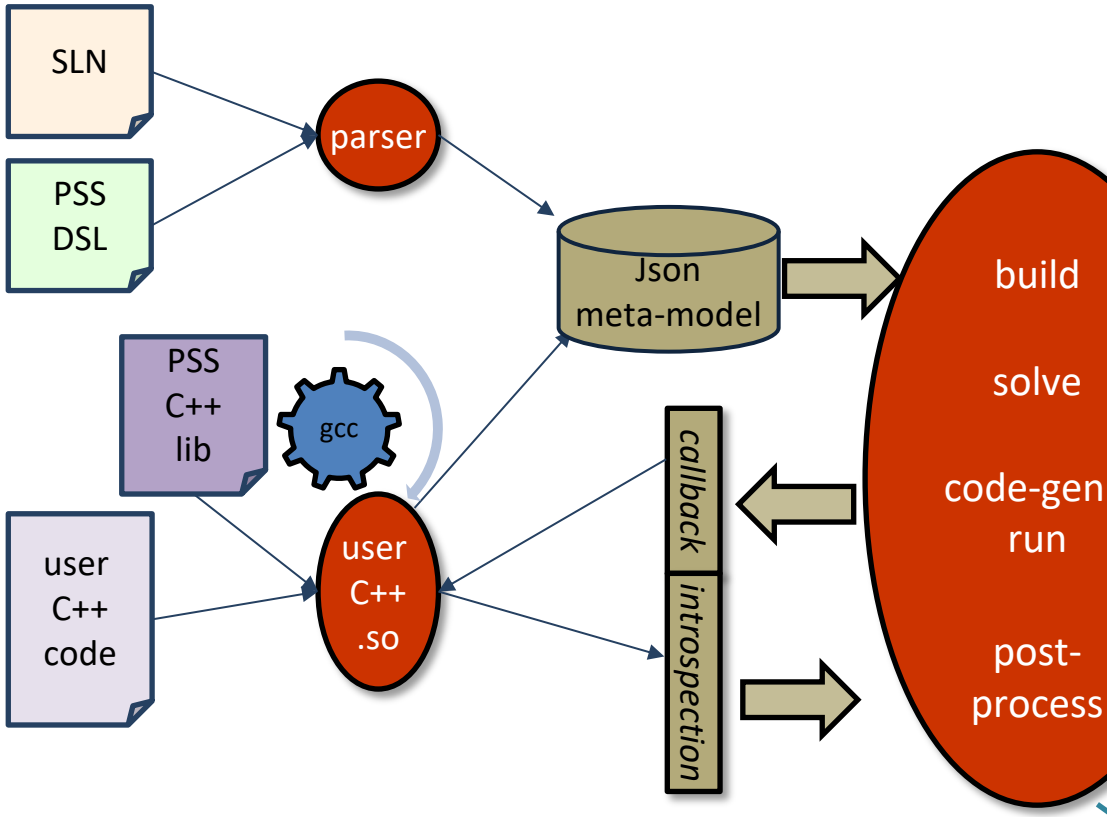
# Seminar Table of Content

- Introduction of PSS via the user-contributed usage example
  - DMA Allocation for peripherals + demo
  - Exhaustive exercise of power-states + demo
- PSS hands-on Exercise
- Introduction of other PSS use-cases
  - Multi-core/master data-cache and coherency
  - PSS UVM example
- PSS and Metric Driven Verification
- Summary

accellera SYSTEMS INITIATIVE

# Perspec Multi-Front-End Architecture

## Proven Support for PSS concepts since 2010




Perspec innovated much of the PSS concepts including, **actions**, **activities**, **resources**, **AOP**, **data-flow** with **inputs/outputs**, **state-machines**, **components**, **exec blocks**, **UML** activity diagrams and more

Methodology and open-source library (SML) were developed

World-wide experienced AEs in PSS practical applications

Optimized to support sub-systems all the way to the largest systems in the industry today

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# If Any of This Is Interesting …

- We will be happy to asses the relevancy of PSS to your specific needs
  - Typically requires two hours of white-board discussion

- We have two more interesting demos at the Cadence booth
  - UVM automation and coverage maximization
  - Multi-core/master data-cache and coherency

- Talk to your Cadence local deployment team email us at pss_info@cadence.com

Thanks!

accellera SYSTEMS INITIATIVE