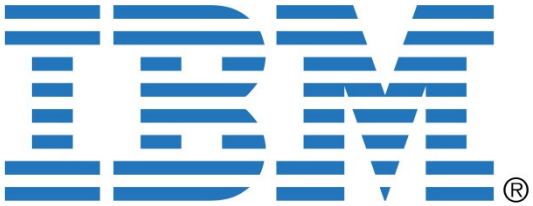
TUTORIAL:
Achieving Portable Stimulus with Graph-Based Verification

25 September 2014



Tutorial Objectives

- Provide overview of technical requirements that is driving us towards a portable stimulus standard

- Describe Graph Based Verification
  - What is a graph?
  - How tests modeled with graphs
  - How graphs enable portable tests
  - Verification reuse from IP to subsystem to full-chip

Today's Agenda and Presenters

• Introductions	Josef Derner	5 mins
• Do we Need it?	Holger Horbach	20 mins
• Portable Stimulus for SW	Frederic Krampac	25 mins
• Portable and Efficient Graph-Based Tests	Staffan Berg	25 mins
• Conclusion	Staffan Berg	5 mins

Introducing Today's Presenters

- IBM
  - Holger Horbach, Verification Engineer
- Breker
  - Frederic Krampac, Sr Applications Engineer
- Mentor Graphics
  - Staffan Berg, European Applications Engineer FV

Portable Tests with Graph-Based Scenario Models

Hopes, Dreams and Aspirations

Frederic Krampac

Breker Verification Systems

accellera SYSTEMS INITIATIVE™

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

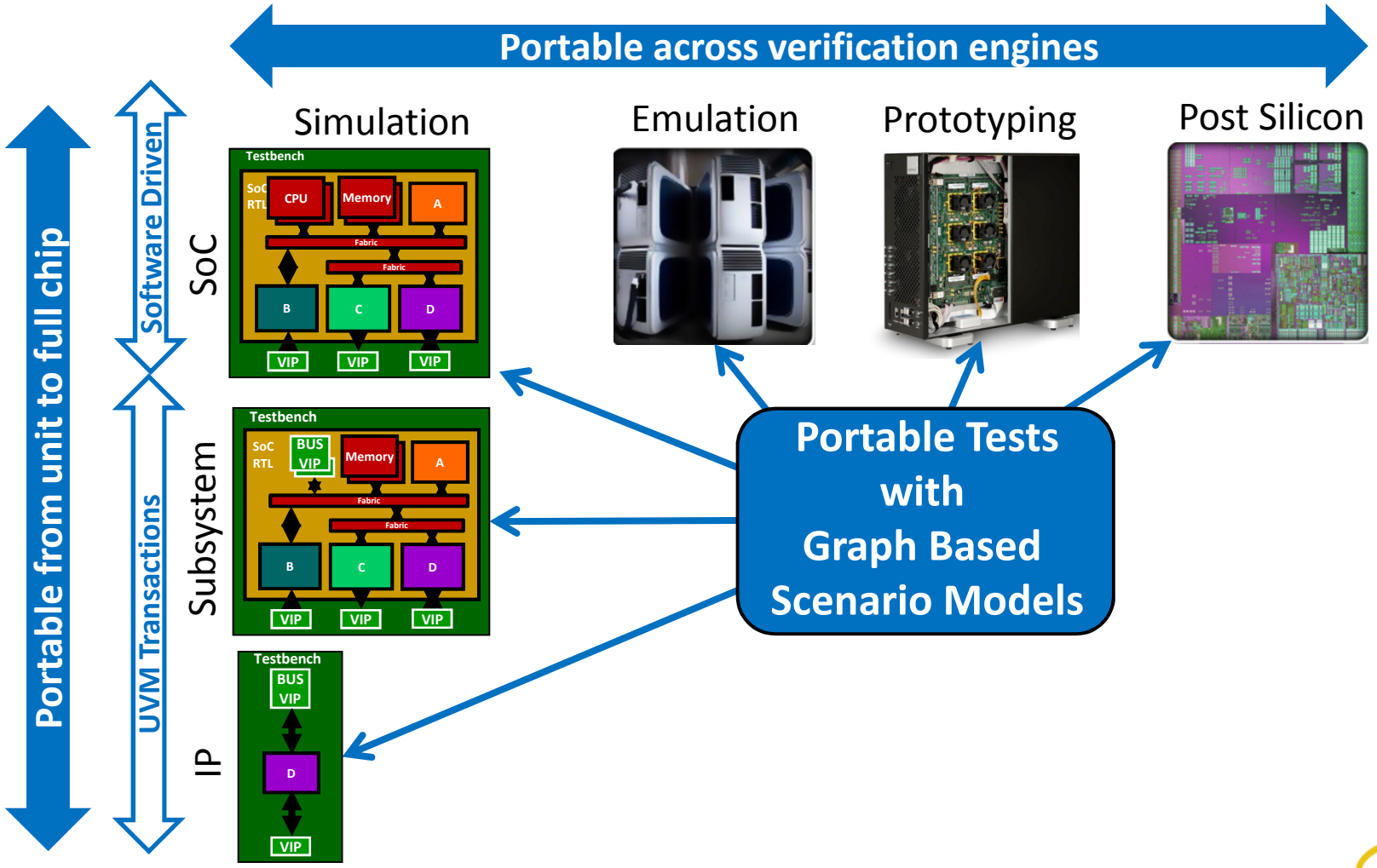
Agenda

- Motivation
- Defining Verification Intent
- Reachability Analysis
- Verification Intent Coverage
- Composability
- UVM transactions vs. Software Driven Tests

Motivation

- Separate verification intent from testbench implementation
- Verification Intent covers both stimulus and checks
- Define verification intent once, use it at each stage of verification

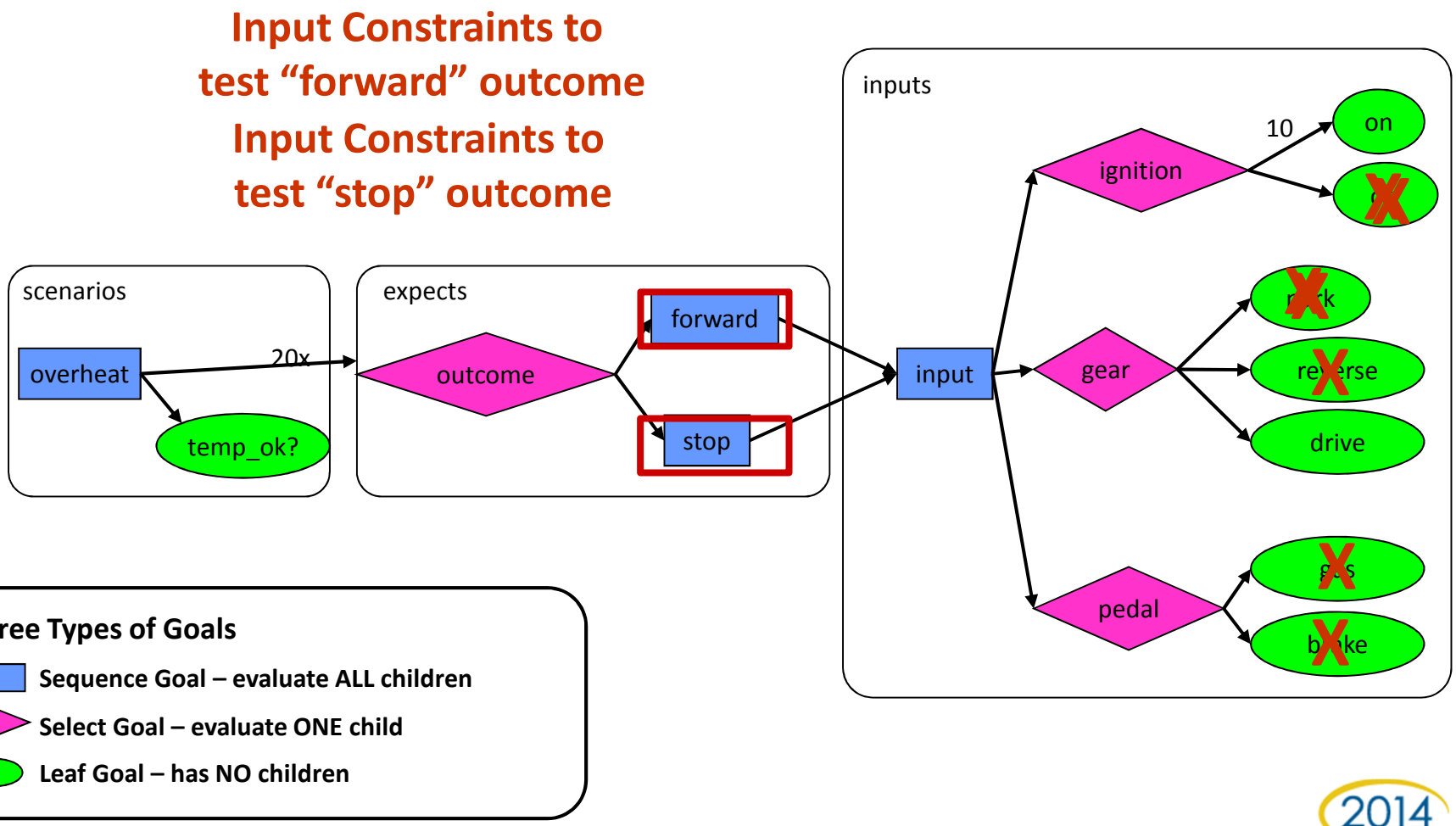
Motivation: Portable Tests



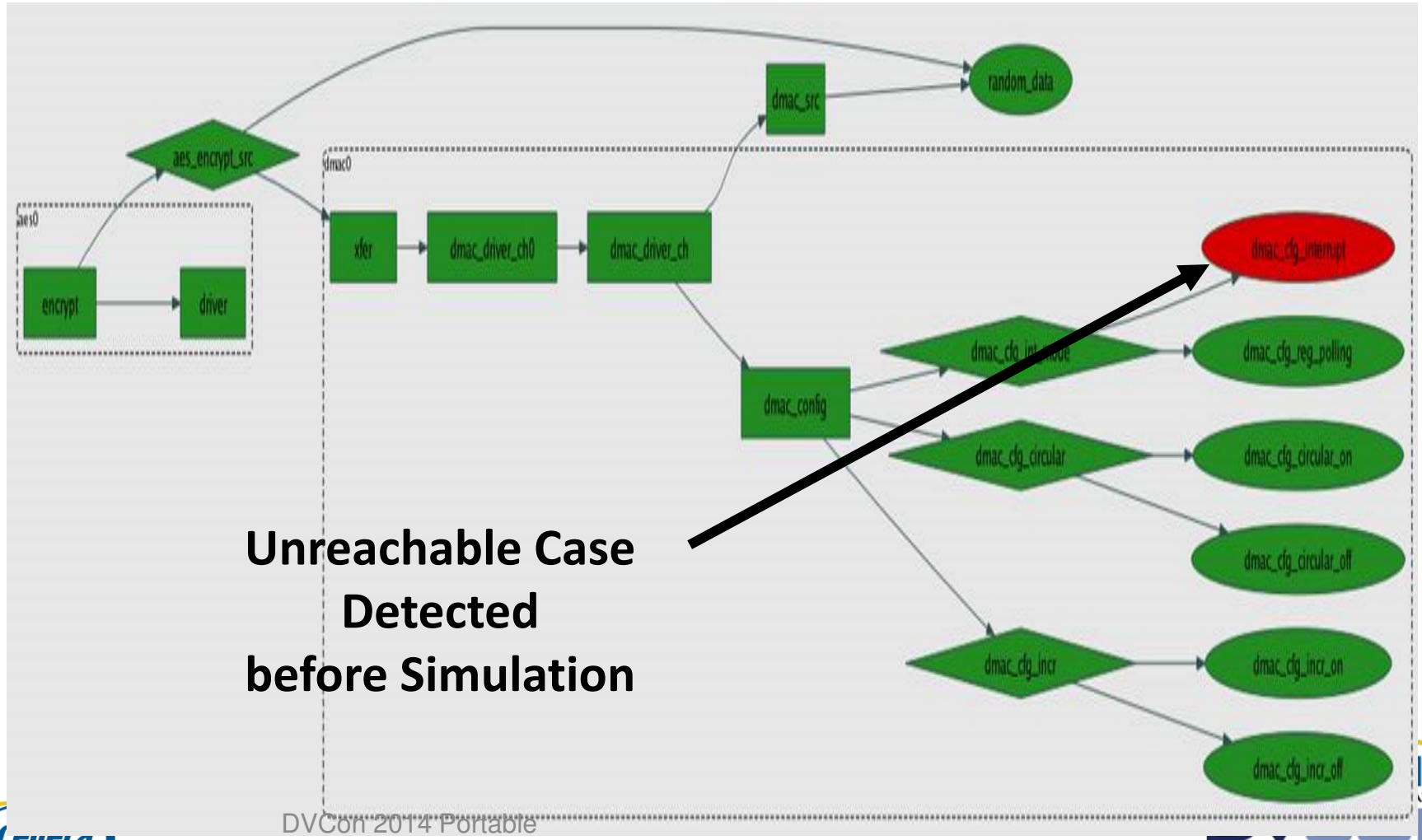
Defining Verification Intent

- Feature – capability from spec
  - Check – what precisely must be checked in TB
    - Stimulus – how to to sensitize check

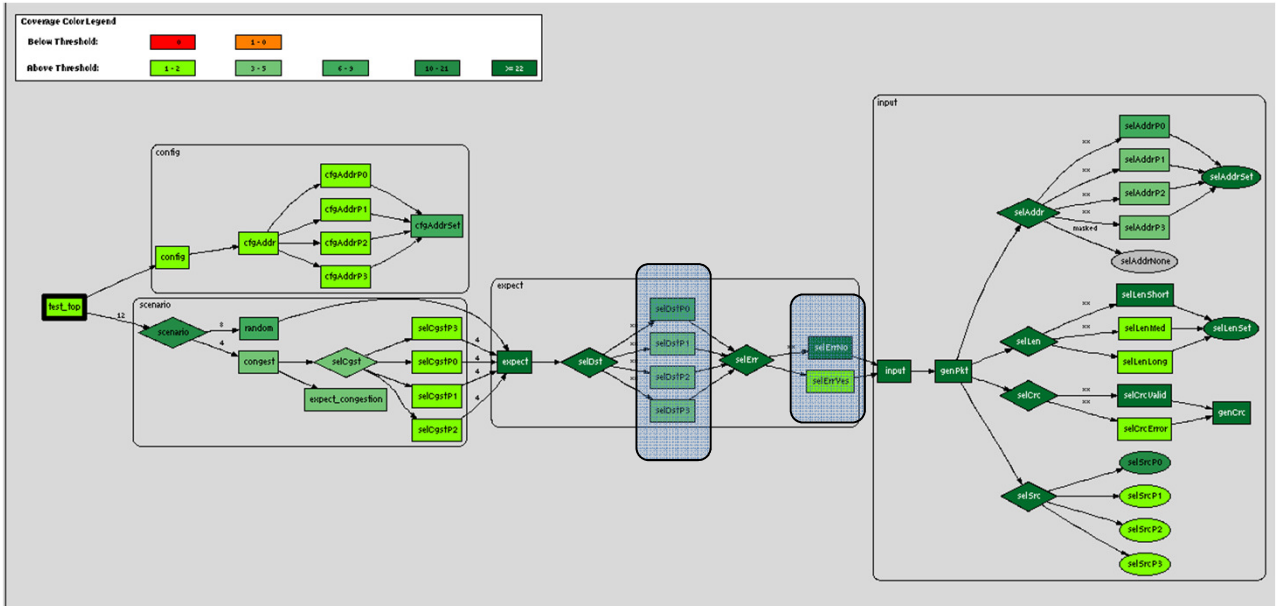
Defining Intent: Verifying a Car



Reachability Analysis



Verification Intent Coverage

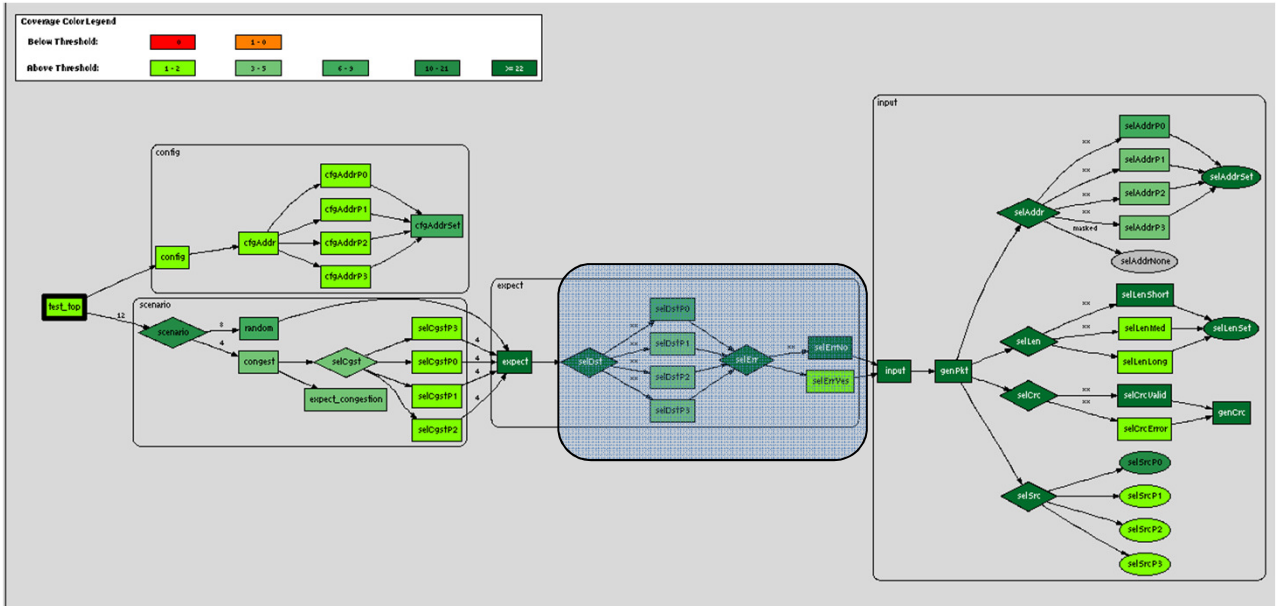


Target A: Hit all input ports

Target B: Hit both error conditions

Target C: Cross A x B (8 total paths)

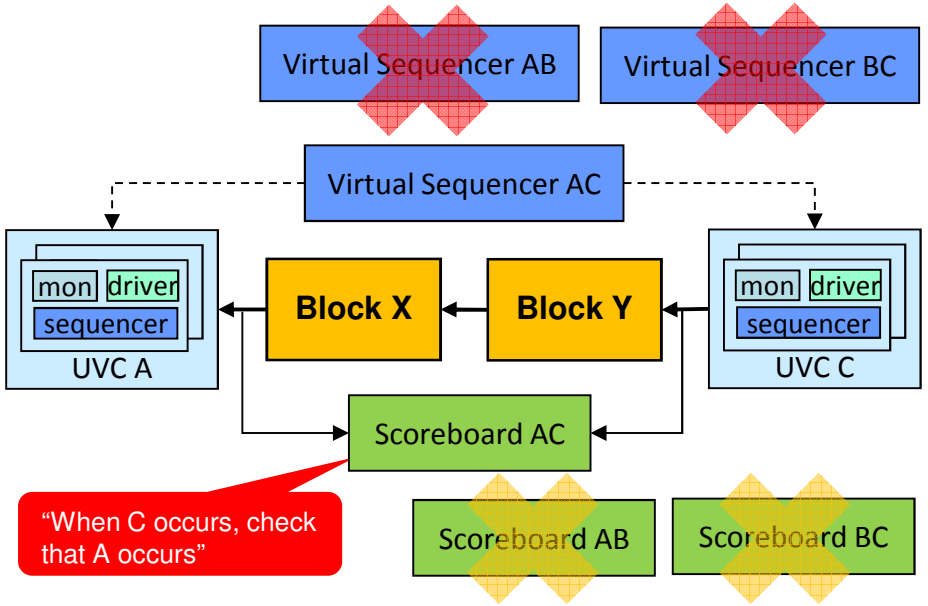
Automatic Coverage Closure

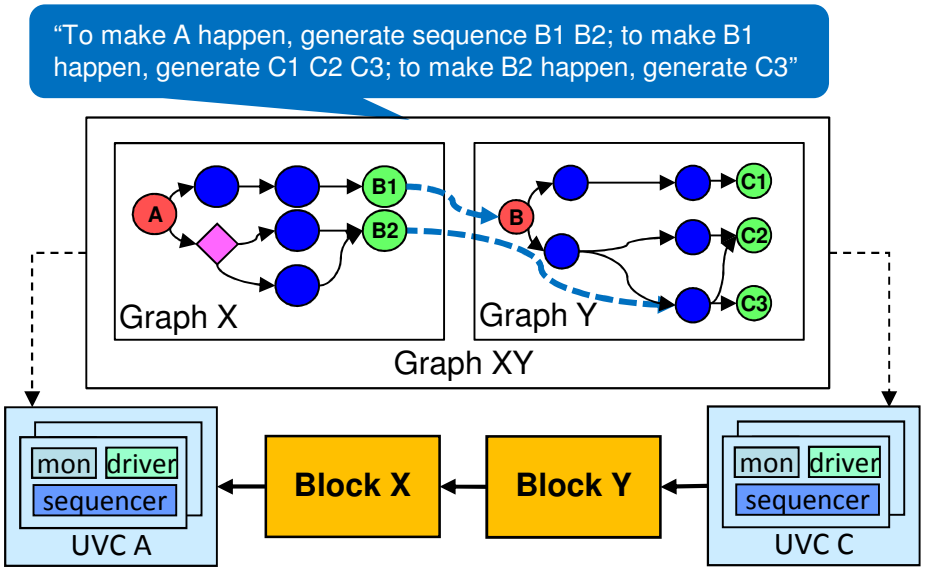


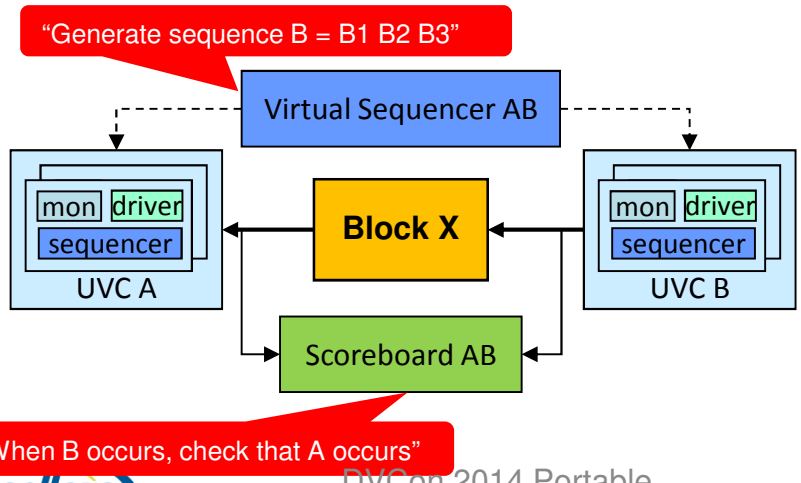
Automatically Close Coverage Targets

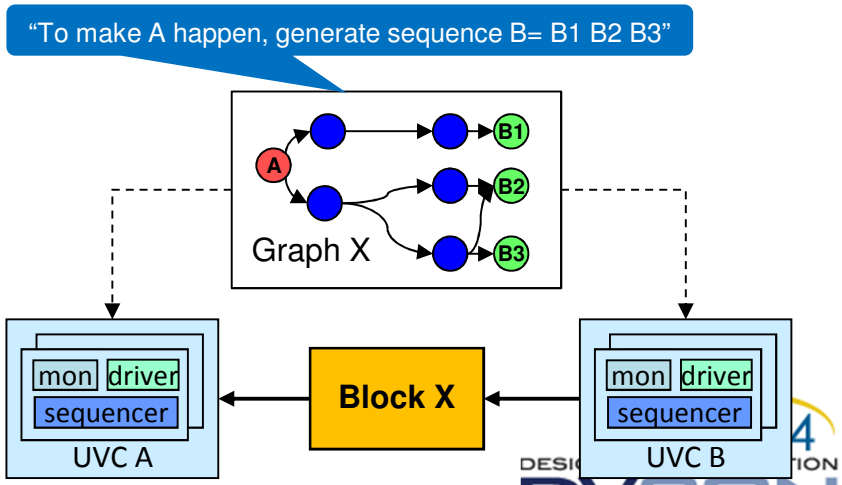
Example: “cross 2 choices and walk all 8 paths”

Composability

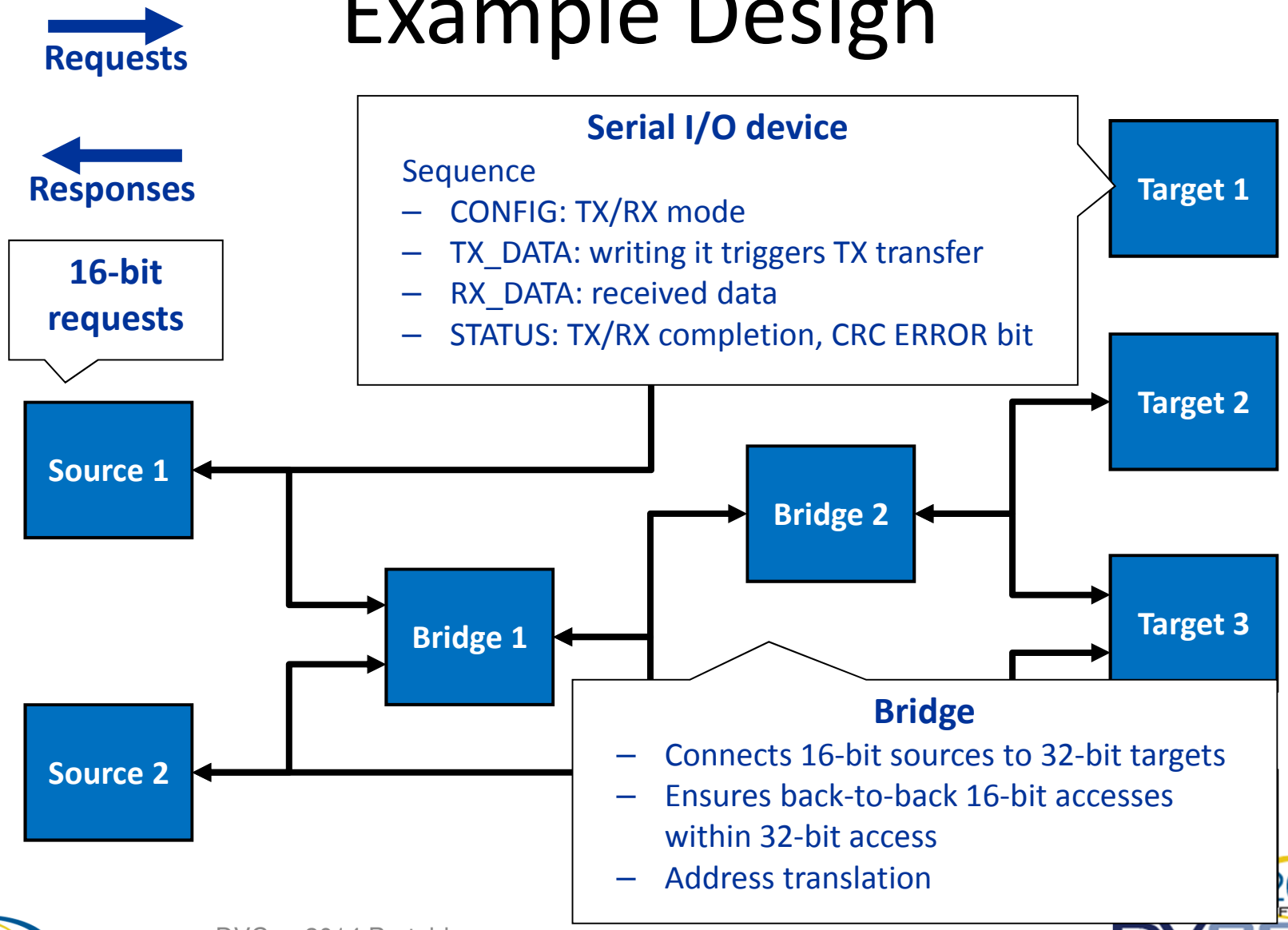




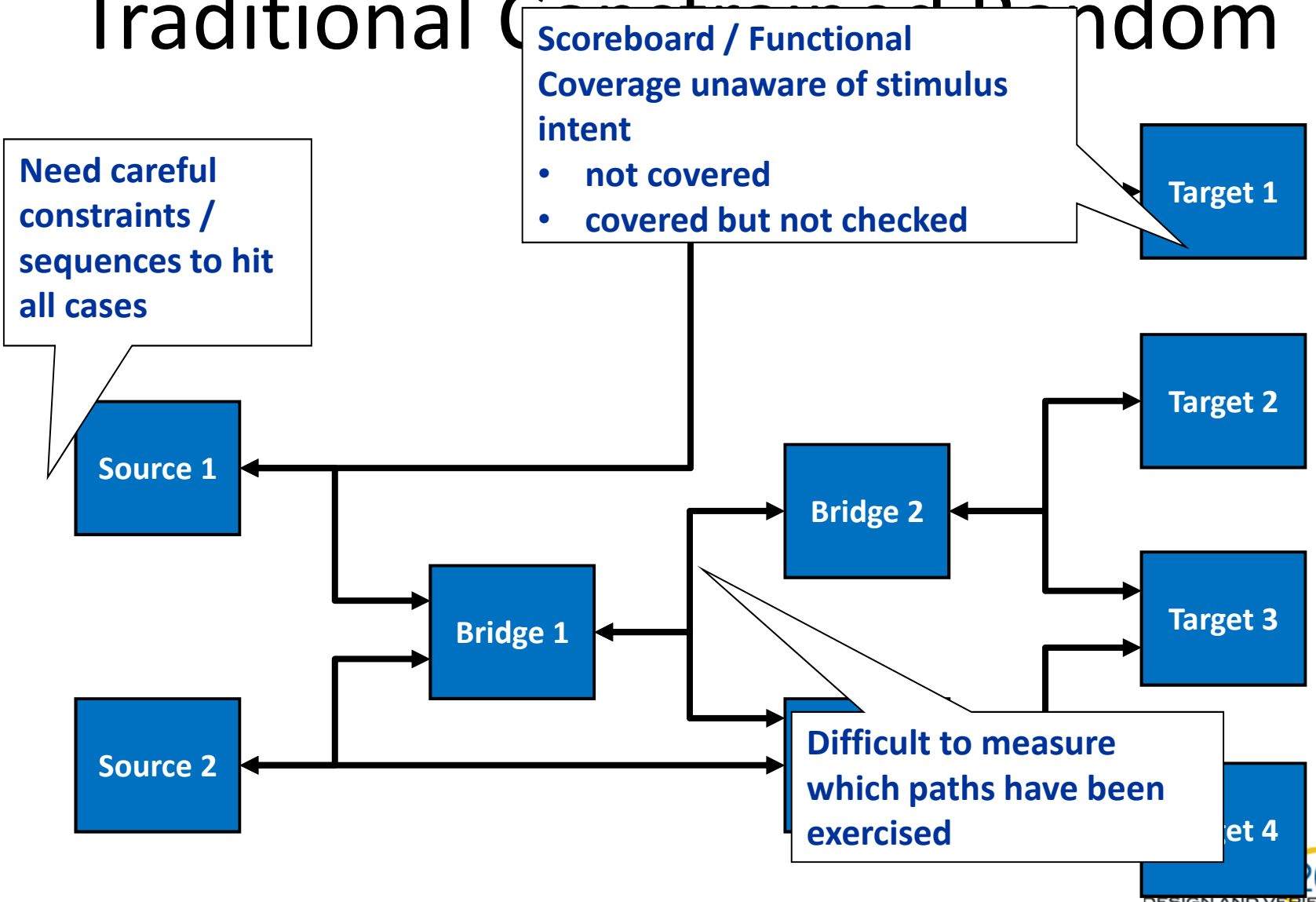




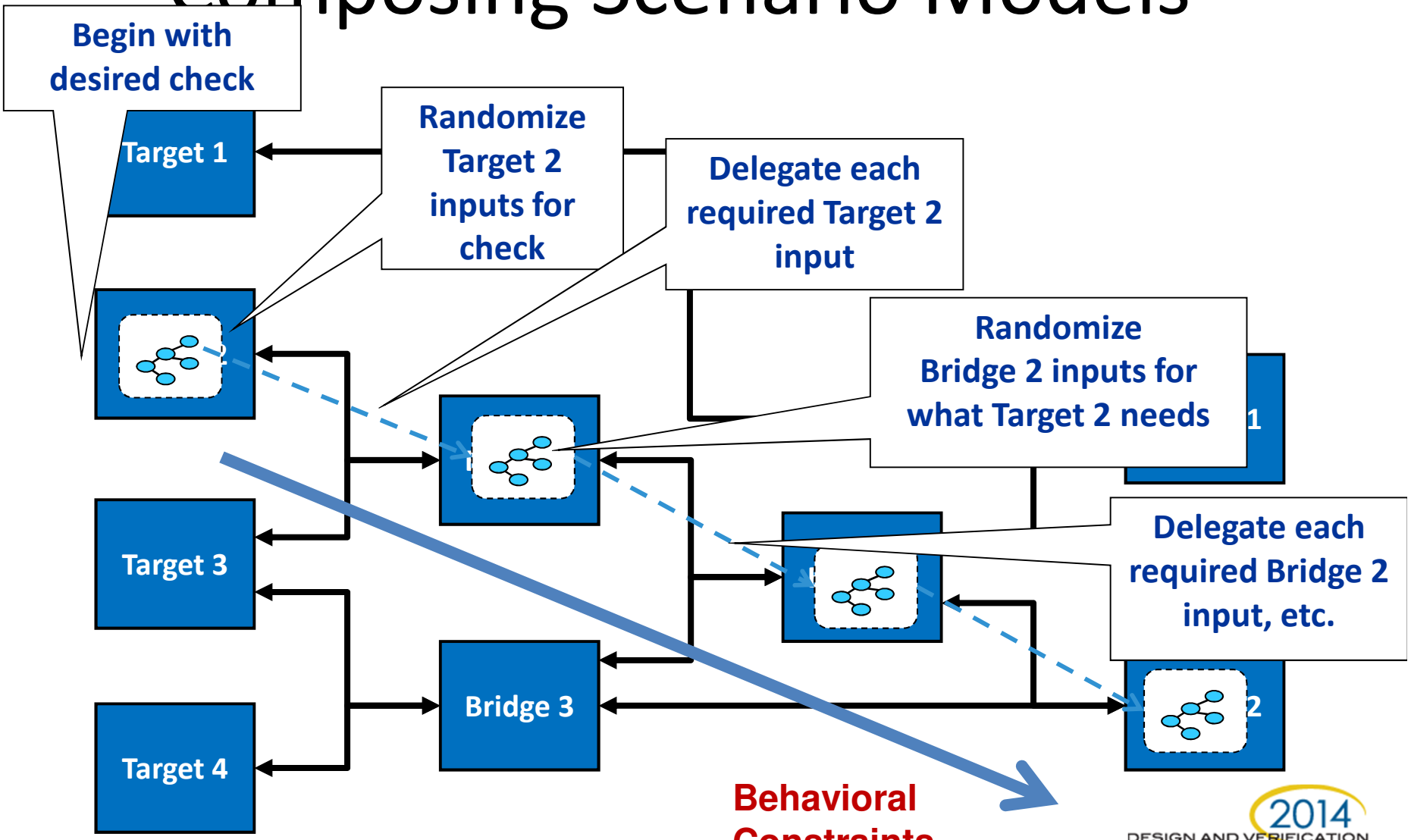
Example Design



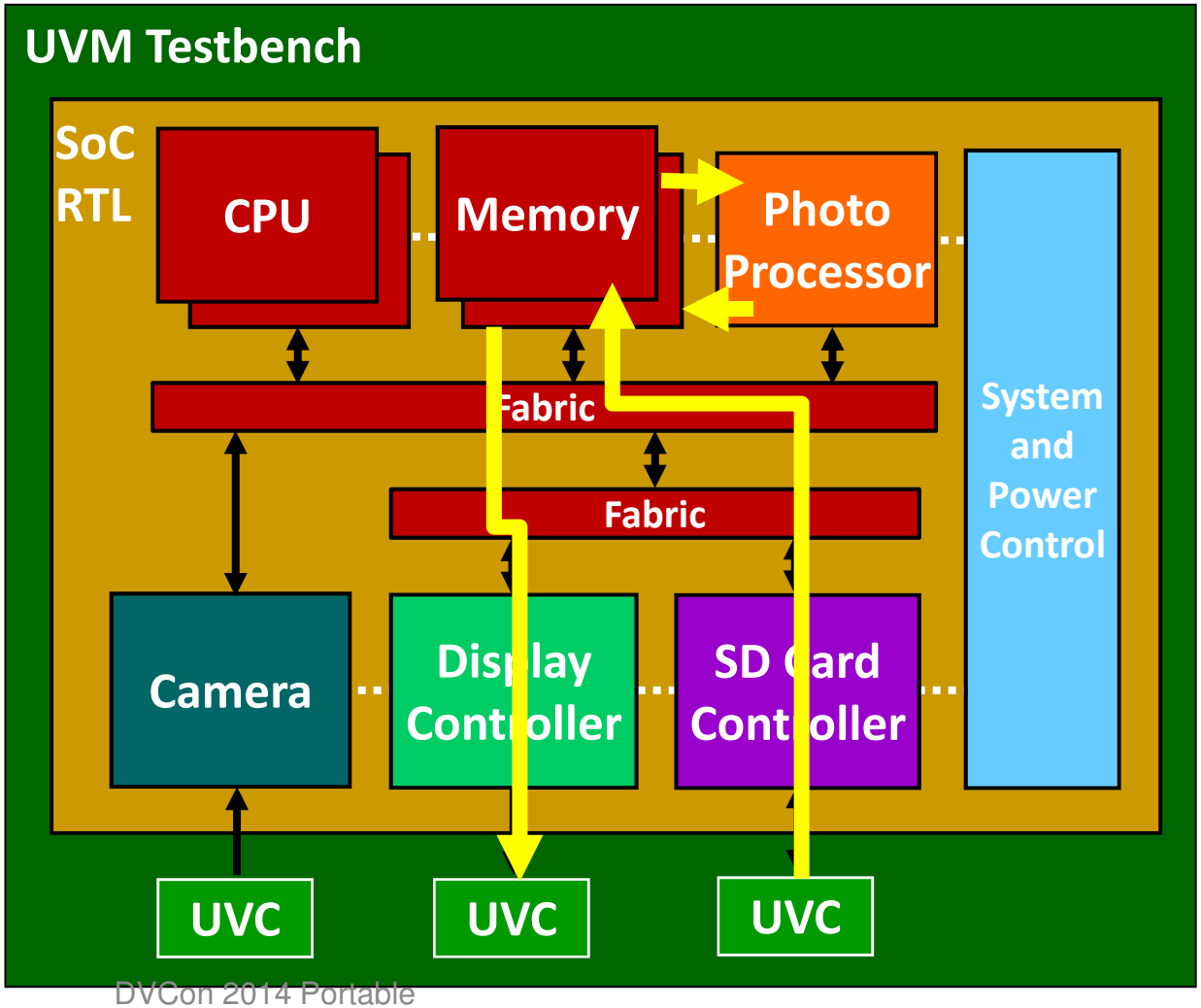
Traditional Constrained Random



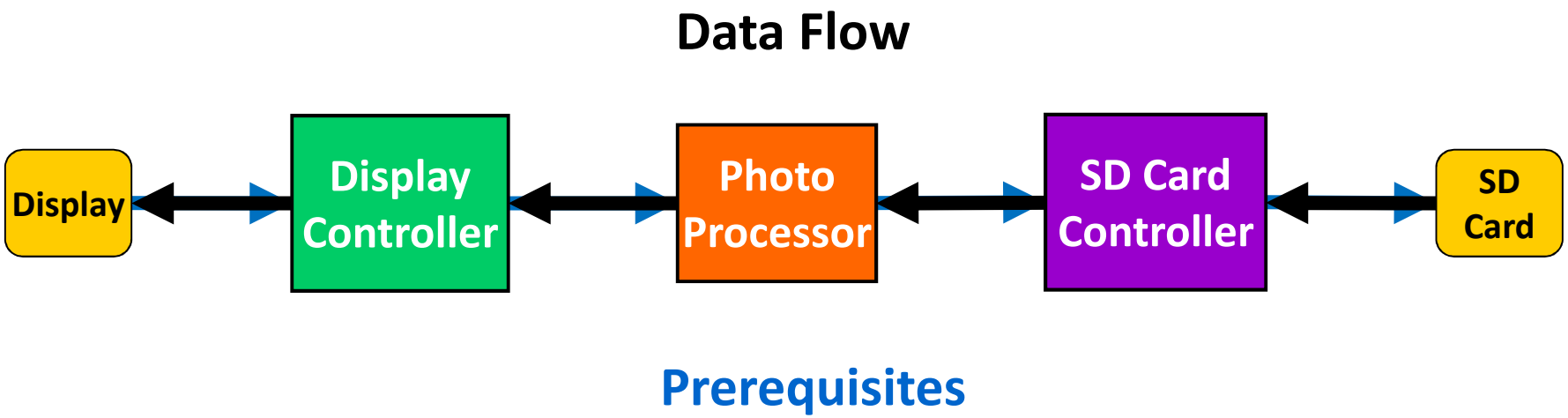
Composing Scenario Models



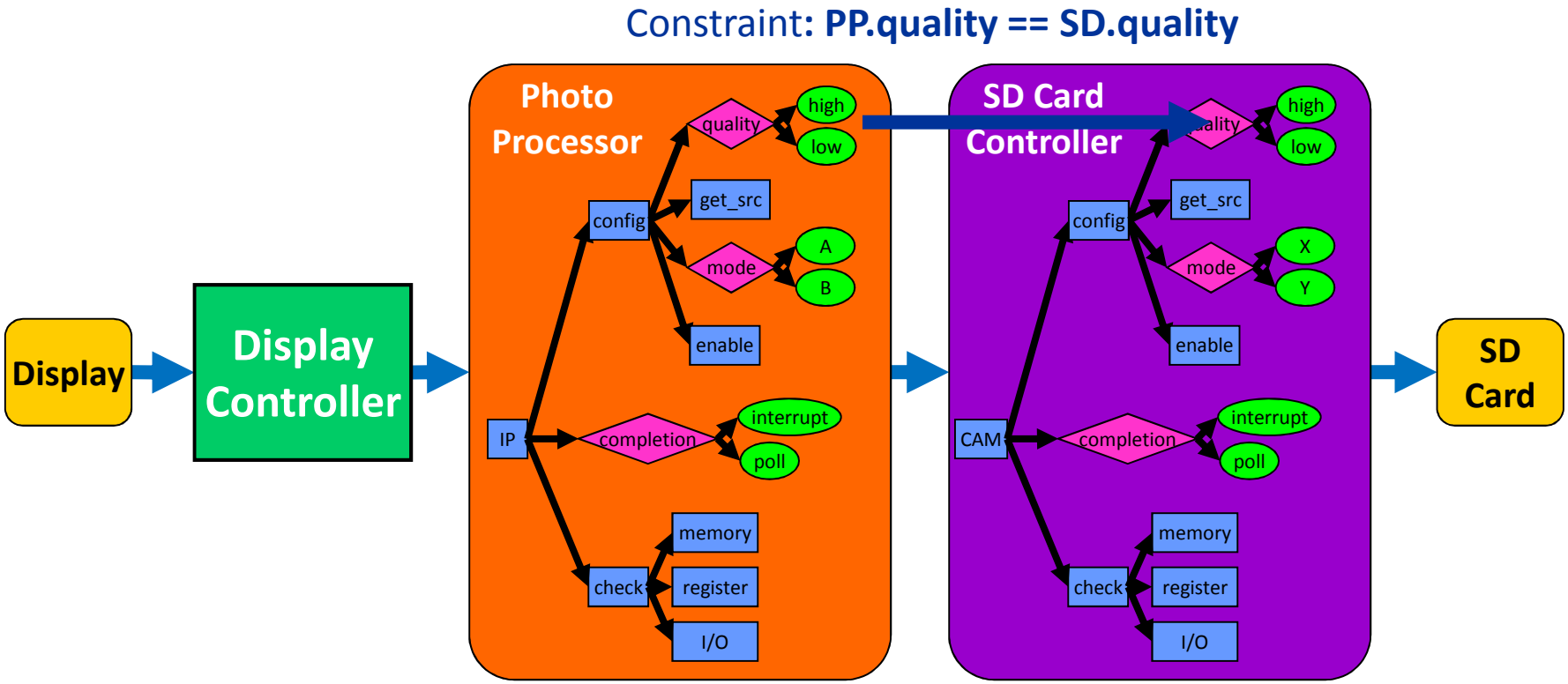
Composing Software Driven Tests



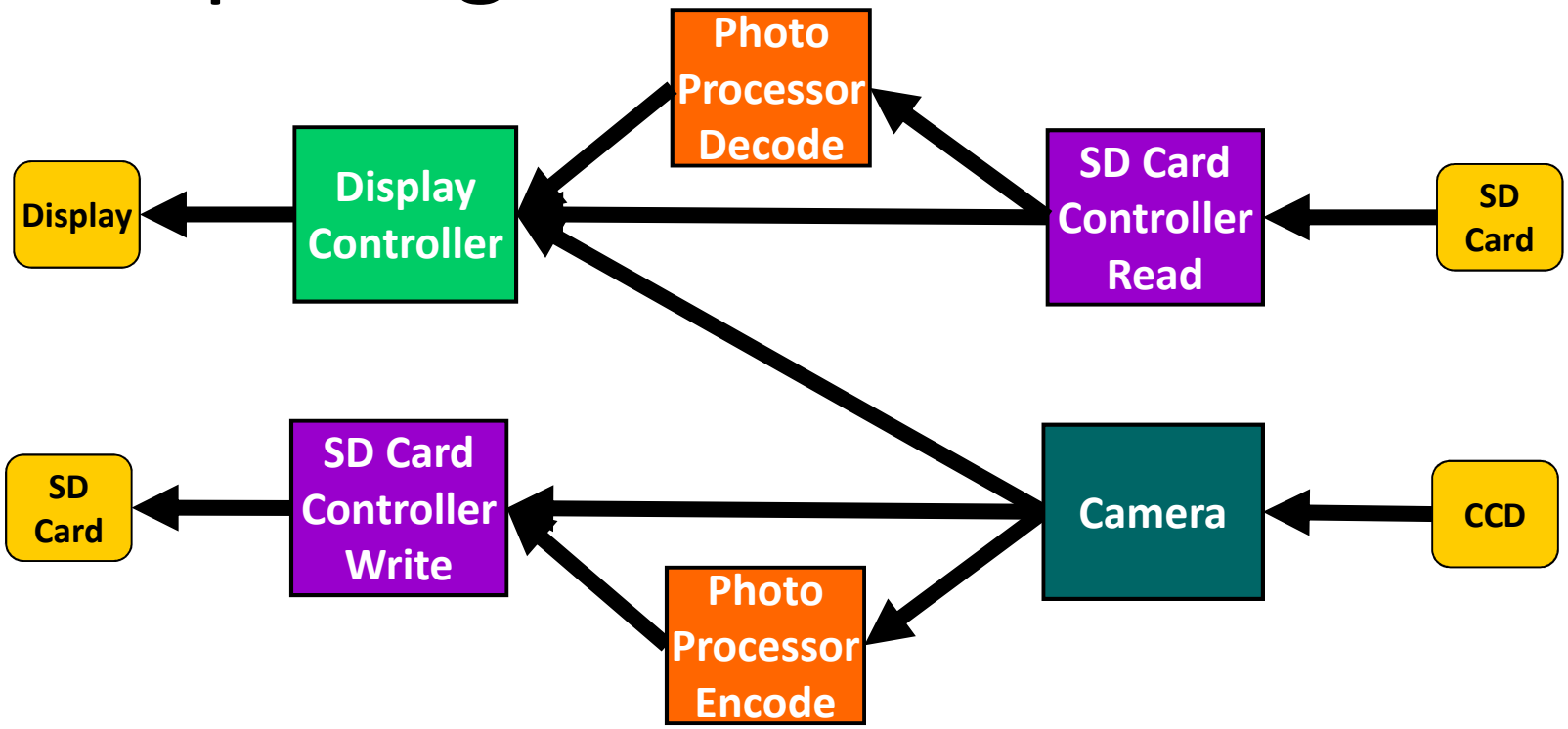
Composing Software Driven Tests

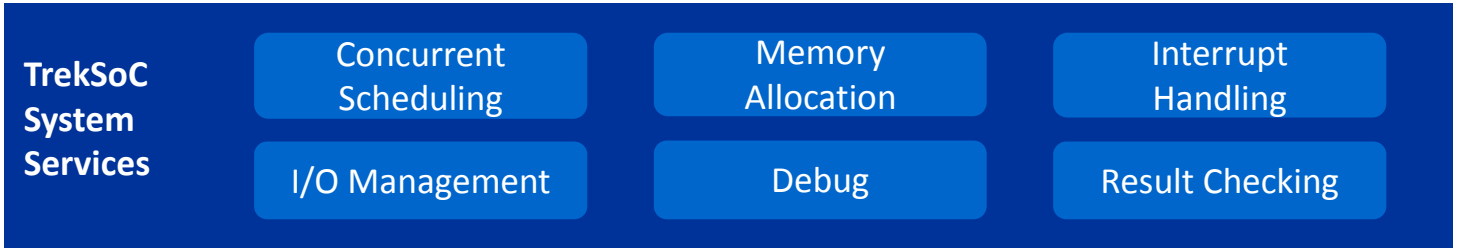


Composing Software Driven Tests

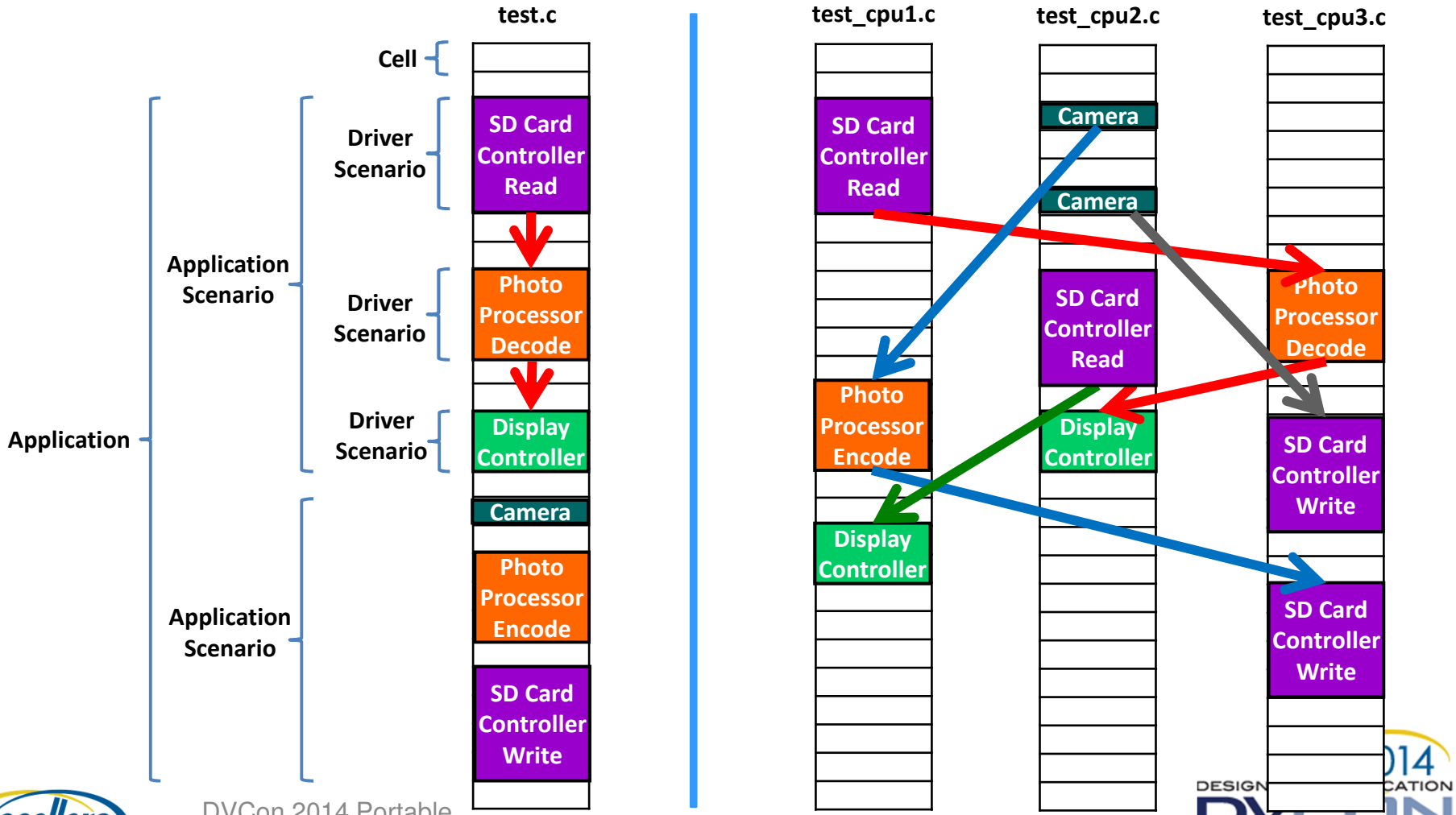


Composing Software Driven Tests

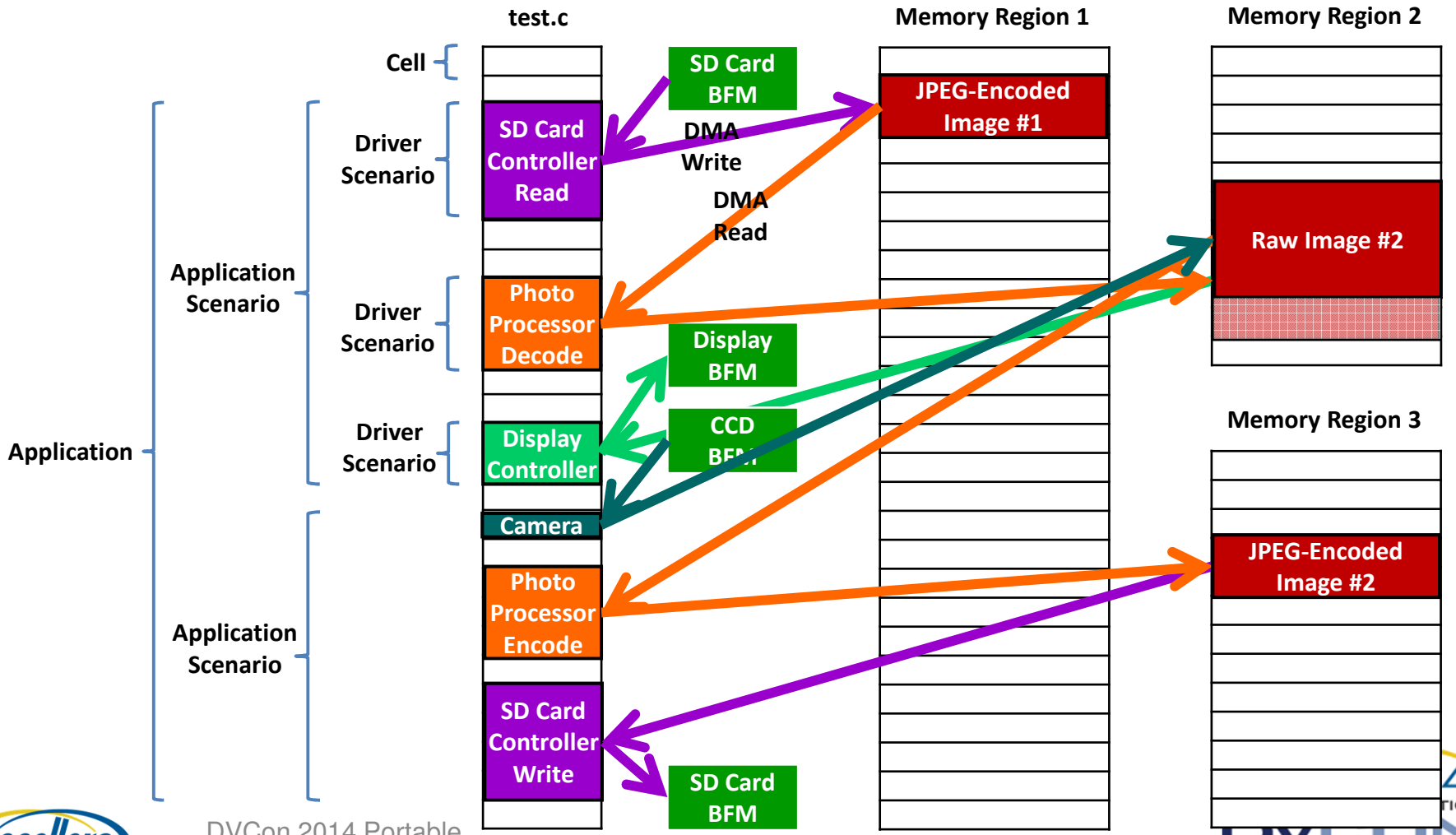




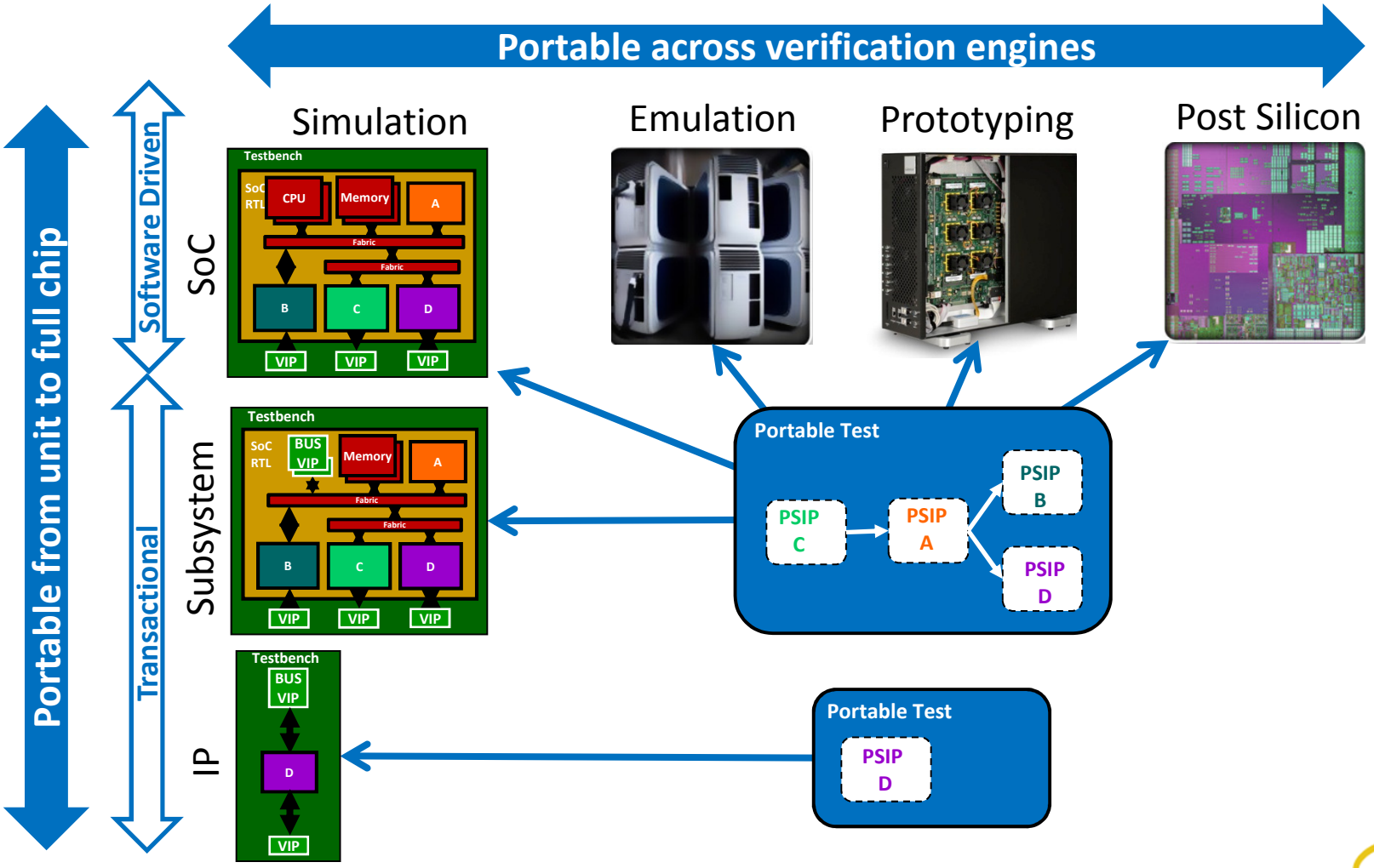
Multi-Processor Scheduling



Memory Scheduling



UVM Transaction vs Software Driven Tests



Summary

- Portable stimulus must be extended to checks and coverage -> portable tests
- Graph-based scenario models provide portable tests
- Efficient coverage closure of stimulus and checks
- Portability
  - From unit to cluster/subsystem to full-chip
  - From simulation to hardware platforms to silicon
  - Both transactional and SoC software-driven tests

Portable and Efficient Graph-Based Tests

Staffan Berg

Mentor Graphics®

accellera SYSTEMS INITIATIVE™

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

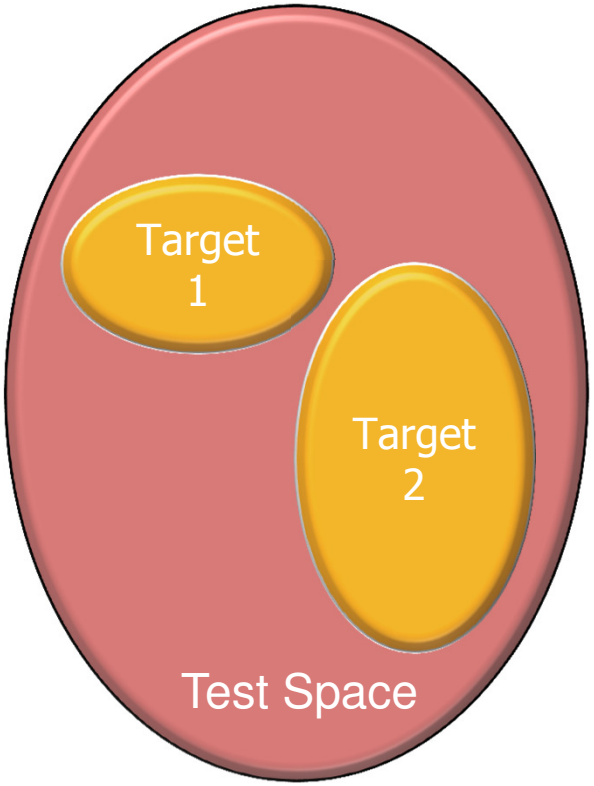
- Modeling Tests with Graphs
- Portable Stimulus with Graphs
- Graph-Based Stimulus Applications

Agenda

- Modeling Tests with Graphs
- Portable Stimulus with Graphs
- Graph-Based Stimulus Applications

Stimulus Specification Fundamentals

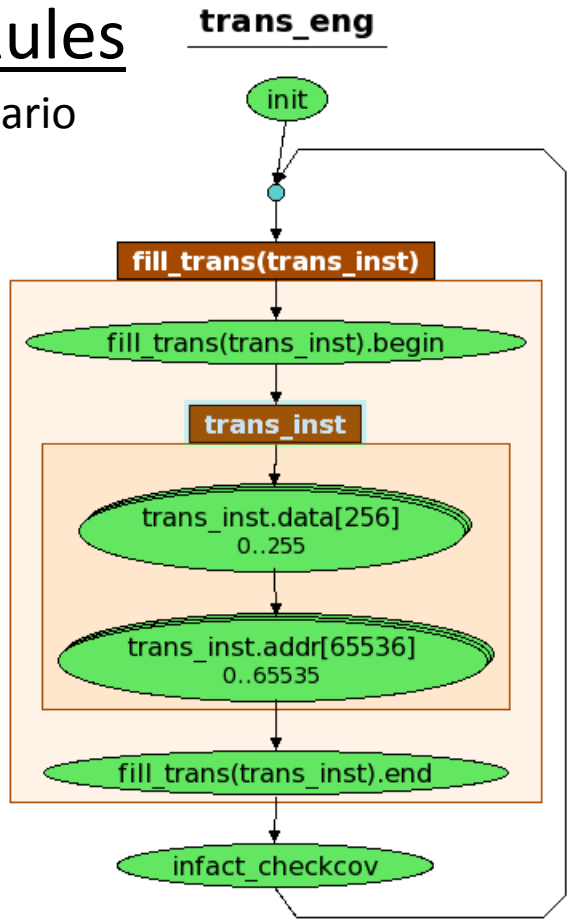
- What is legal
  - Universe of what could happen
  - Captures both data and scenario
  - Enables creation of 'unexpected' cases
- What to target
  - Cases of specific interest
  - What to verify today, during this test



Graph-Based Stimulus Description

- Stimulus scenario described using Rules
  - Captures data and control flow aspects of test scenario
  - Describes legal stimulus scenario space
  - Efficient description mechanism
- Rules are compiled into Graphs
  - Visual representation of the stimulus model
  - Easy to review

```
rule_graph trans_eng {
    action init, infact_checkcov;

    struct trans {
        meta_action data [unsigned 7:0];
        meta_action addr [unsigned 15:0];
        constraint on_addr {addr < 0x8000}
    }

    trans trans_inst;
    interface fill_trans(trans);

    trans_eng = init repeat {
        fill_trans(trans_inst)
        infact_checkcov
    };
}
```



Graph-Based Stimulus Description

Captures data and data relationships

- Scalar types
  - Signed and unsigned integer types
  - Enumerated types
- Composite data structures
  - 'struct', supports type extension
- Aggregate data types
  - Single and multi-dimensional fixed-size arrays
- Variables can be input or output
  - Output variables (default) send values to the environment
  - Input variables bring values in from the environment
- Constraints
  - Algebraic expressions, inside, if/else, foreach, etc

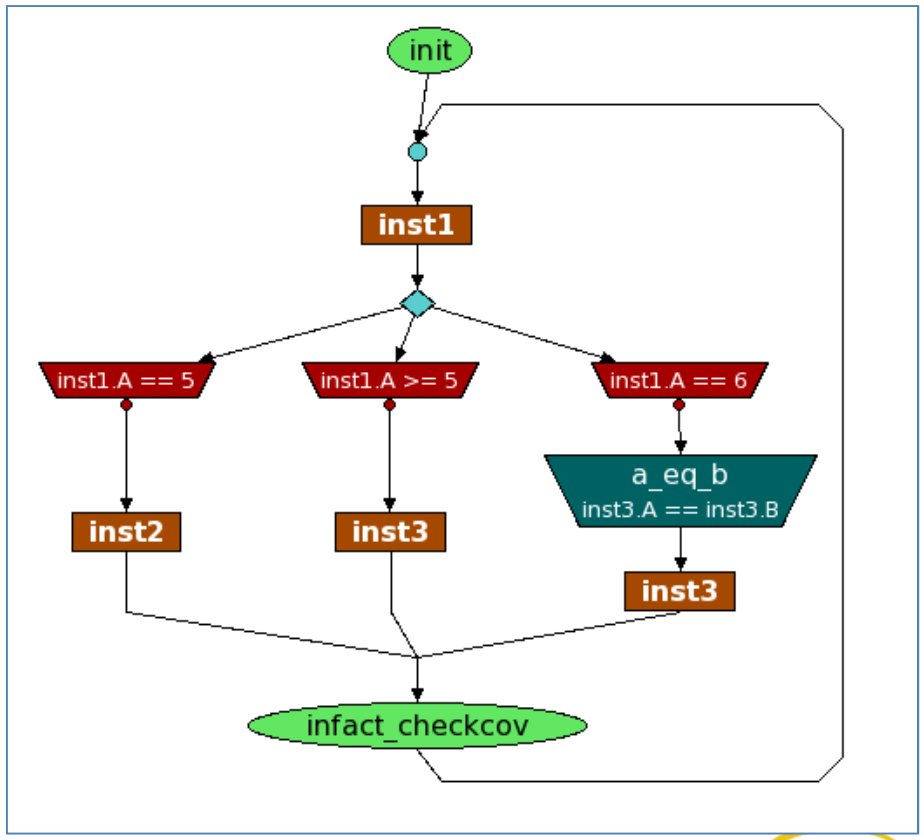
```
struct my_struct1 {
    meta_action     A[unsigned 3:0];
    meta_action     B[unsigned 3:0];
}

struct my_struct2 extends my_struct1 {
    meta_action     C[unsigned 3:0];

    constraint c {
        C < A;
    }
}
```

Graph-Based Stimulus Description

Captures test scenario control flow

- Captures process of stimulus generation
  - Sequences of operations
  - Choices
  - Loops
- Branch-specific constraints
  - Conditional execution
  - Partitions scenario structurally

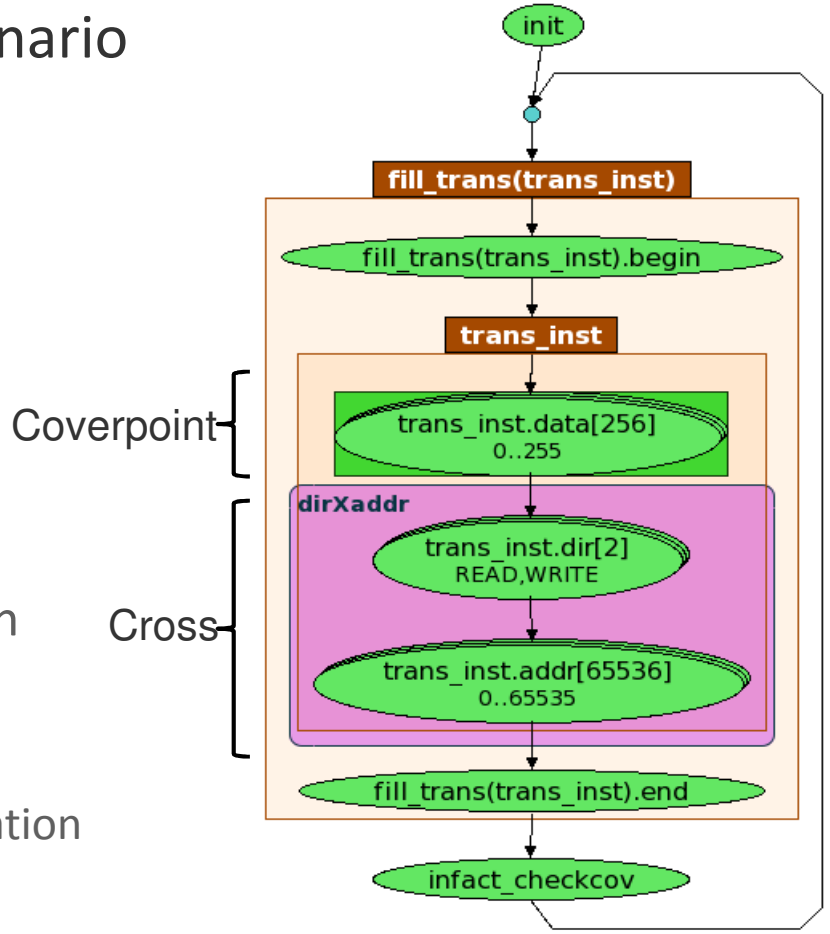
```
constraint a_eq_b dynamic {
    inst3.A == inst3.B
}

my_graph = init repeat {
    inst1
    if {inst1.A == 5} (inst2) |
    if {inst1.A >= 5} (inst3) |
    if {inst1.A == 6} (a_eq_b inst3)
    infact_checkcov
};
```



Test Selection and Prioritization

Coverage Strategy

- Coverage strategy expresses test scenario goals
  - Key stimulus values
  - Key stimulus combinations
  - Key stimulus sequences
- Flexible
  - Prioritize certain goals
  - Combine random/systematic generation
- Reactive
  - Adapts to changes in the DUT or the verification environment state



Test Selection and Prioritization

Target value specification

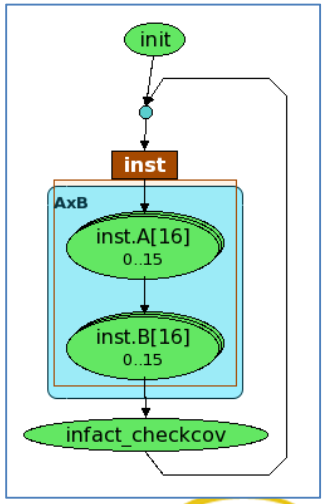
- Variable domains can be divided using 'bins'
  - Split a value range into N bins
  - Split a value range into bins of size N

```
my_struct1 inst;


bin_scheme small_vals {
    inst.A [0..3]:1 [4..15]/4;
    inst.B [0..1]:1 [2..15]/8;
}
```

- Coverage constraints select target space with expressions
  - Only active when targeting the coverage goal
  - Prioritizes specific combinations
  - Full legal space reachable otherwise

- Example: A x B
  - Full legal space is 256 (16 * 16)
  - Constraint A < B selects 120 combinations

```
my_struct1 inst;

constraint a_less_b coverage {
    inst.A < inst.B;
}
```



AXI4 Adjacency Example

Efficient description at transaction level

- *Struct* captures transaction
  - Key transaction fields
  - Constraints

```
rule_segment {

    struct axi4_master_rw_transaction {
        meta_action read_or_write<axi4_rw_e>[enum READ, WRITE];
        meta_action addr[unsigned 31:0];
        meta_action burst<axi4_burst_e>[enum AXI4_FIXED, AXI4_INCR, AXI4_WRAP];
        meta_action burst_length[unsigned 7:0];
        meta_action size<axi4_size_e>[enum AXI4_BYTES_1, AXI4_BYTES_2, AXI4_BYTES_3, AXI4_BYTES_4];

        // Align address for wraps
        constraint addr_align_c {
            if (burst == AXI4_WRAP) {
                (addr % (1 << size) == 0);
            }
        }

        // Limit burst length for wrapping bursts
        constraint burst_len_c {
            burst_length <= 15;
            if (burst == AXI4_WRAP) {
                burst_length inside [1,3,7,15];
            }
        }
    }

}
```

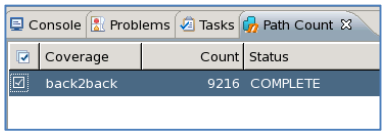
- Reusable in multiple graphs

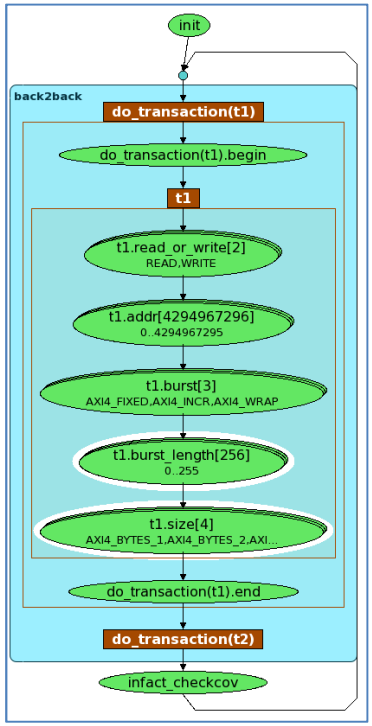
AXI4 Adjacency Example

Efficient scenario specification and selection

- Specify target address ranges
  - Four target devices
  - Target four ranges in each device
- Generate transaction sequences
  - R/W, address, burst sequences
  - T2->T1 adjacencies are random
  - Burst length, size random
- Pre-simulation size analysis
  - 9216 sequences selected
- Efficient and flexible generation

```
rule_graph axi4_back2back_graph {
    import "axi4_master_rw_transaction.rseg";

    action init, infact_checkcov;

    // Focus on specific address ranges for each target
    bins axi4_master_rw_transaction.addr
        [0x00000000..0x0003FFFF]/4 // Target 1
        [0x00040000..0x0007FFFF]/4 // Target 2
        [0x00080000..0x000BFFFF]/4 // Target 3
        [0x000C0000..0x000FFFFF]/4 // Target 4
        ;

    interface do_transaction(axi4_master_rw_transaction);

    axi4_master_rw_transaction t1, t2;

    axi4_back2back_graph = init repeat {
        do_transaction(t1)
        do_transaction(t2)
        infact_checkcov
    };
}
```



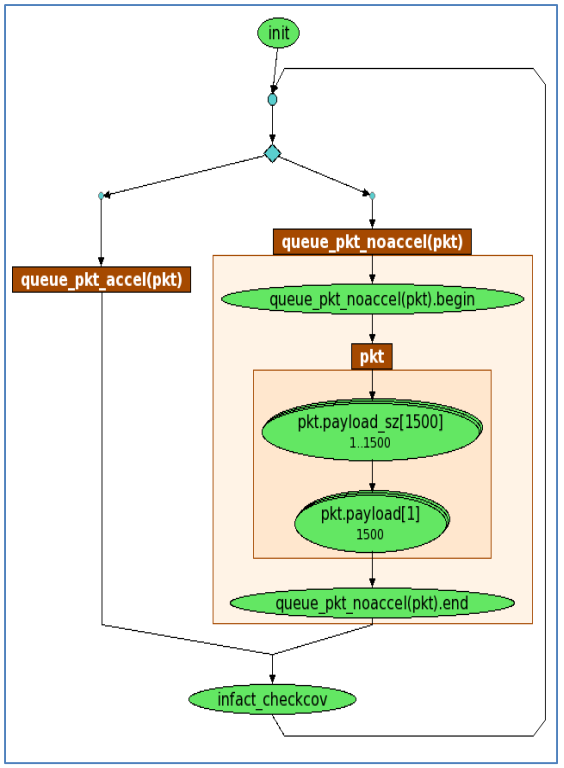


Agenda

- Modeling Tests with Graphs
- Portable Stimulus with Graphs
- Graph-Based Stimulus Applications

Graph Reuse and Portability

- Graphs are language and environment-independent
  - Self-contained
  - Describe data and sequence scenario
- Graphs mapped to specific environments
  - Communicate data to/from environment
  - Synchronize execution
- Can reuse graphs across environments
  - Create with UVM, reuse with embedded sw
  - Create with C model, reuse in UVM



Graphs Enable Reuse

Testbench Import leverages existing SystemVerilog

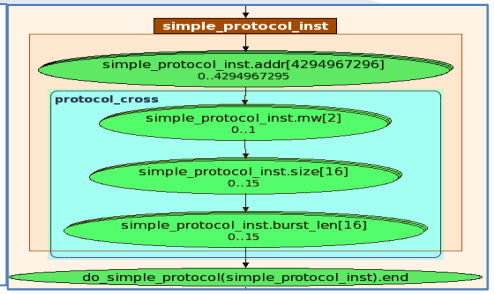
- Imports
  - SV classes
    - Fields, constraints
  - Covergroups
- Creates
  - Rules
  - Coverage strategy
  - Testbench integration
- Leverages existing SV
- Raises abstraction level
  - Import transaction classes
  - Build complex scenarios on top

```
class simple_protocol;
  rand bit[31:0]   addr;
  rand bit         rnw;
  rand bit[3:0]    size;
  rand bit[3:0]    burst_len;

  constraint protocol_c {
    (addr % (1<<size)*burst_len) == 0;

    burst_len inside {[1:4]};
    size inside {1,2,4};
  }
endclass
```

```
covergroup simple_protocol_cg;
  rnw_cp  : coverpoint item.rnw;
  size_cp : coverpoint item.size {
    bins size[] = {1,2,4};
  }

  burst_cp : coverpoint item.burst_len {
    bins burst[] = {[1:4]};
  }

  protocol_cross : cross rnw_cp, size_cp, burst_cp;
endgroup
```

inFact Testbench Import

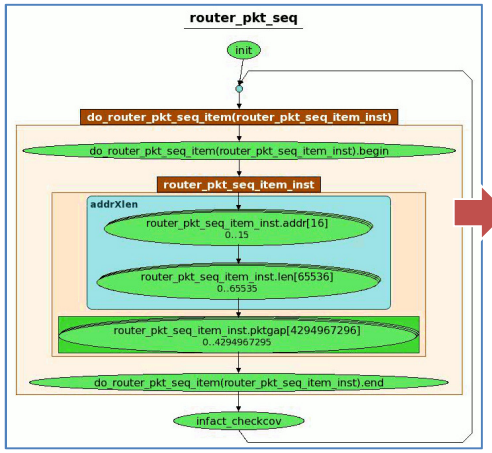
```
rule_segment {

    struct simple_protocol {
        meta_action addr [unsigned 31:0];
        meta_action rnw [unsigned 0:0];
        meta_action size [unsigned 3:0];
        meta_action burst_len [unsigned 3:0];

        constraint protocol_c {
            (addr % (1 << size)*burst_len) == 0;
            burst_len inside [1..4];
            size inside [1,2,4];
        }
    }
}
```



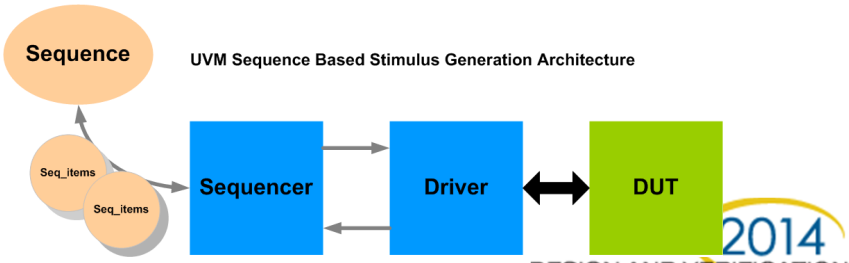
Integration Example

Transaction-level UVM

- Graph runs in a UVM sequence
- Graph nodes set sequence-item fields
- Each graph iteration produces a transaction
- Graph-based sequence is “just another UVM sequence”



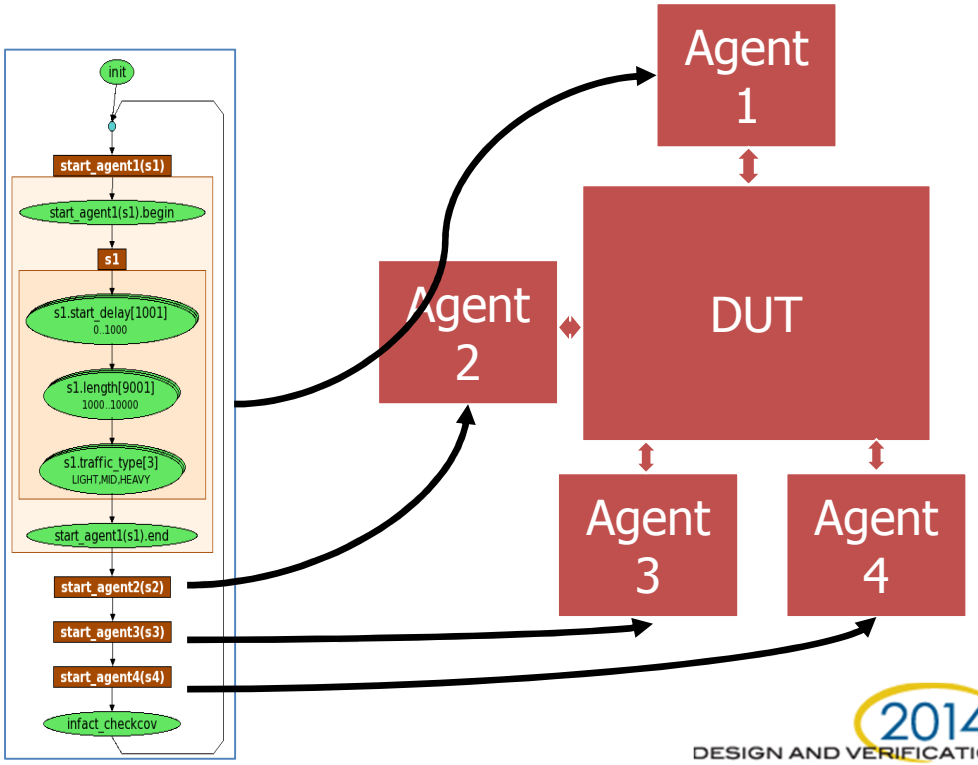

```
class router_pkt_seq_item extends uvm_sequence_item;
  `uvm_object_utils(router_pkt_seq_item)

  rand bit[3:0]                addr;
  rand bit[15:0]               len;
  rand int                     pktgap;

  ...

endclass
```



Integration Example

Virtual Sequence UVM

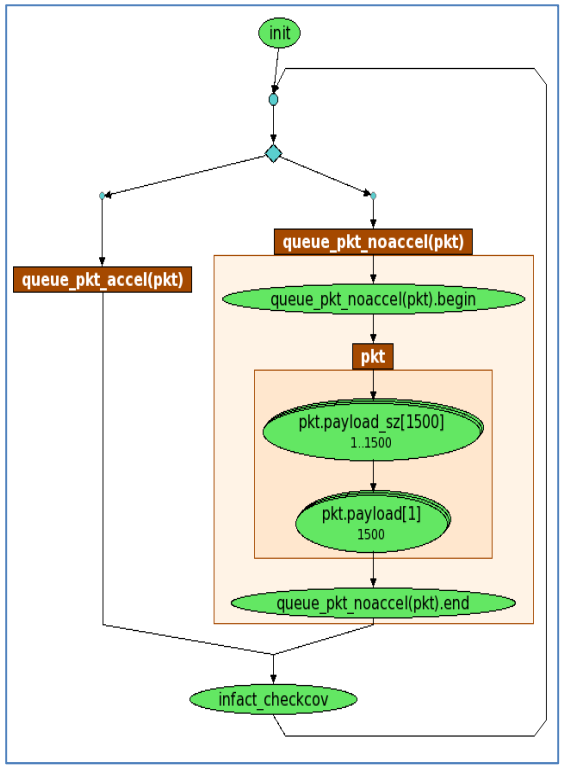
- Graph runs in a UVM virtual sequence
- Graph execution
  - Selects sequence parameters
  - Starts sequences
- Sub-sequences can be
  - Graph-based UVM
  - Random or directed



Integration Example

Embedded software

- Graphs call existing 'C' methods
  - Select parameter values
  - Sequence method calls
- Graphs can run
  - Independently
  - Cooperatively
  - Single/multi-core
  - Single/multi-thread
- Supports
  - Simulation
  - Emulation
  - Silicon

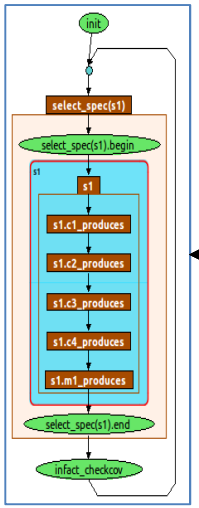


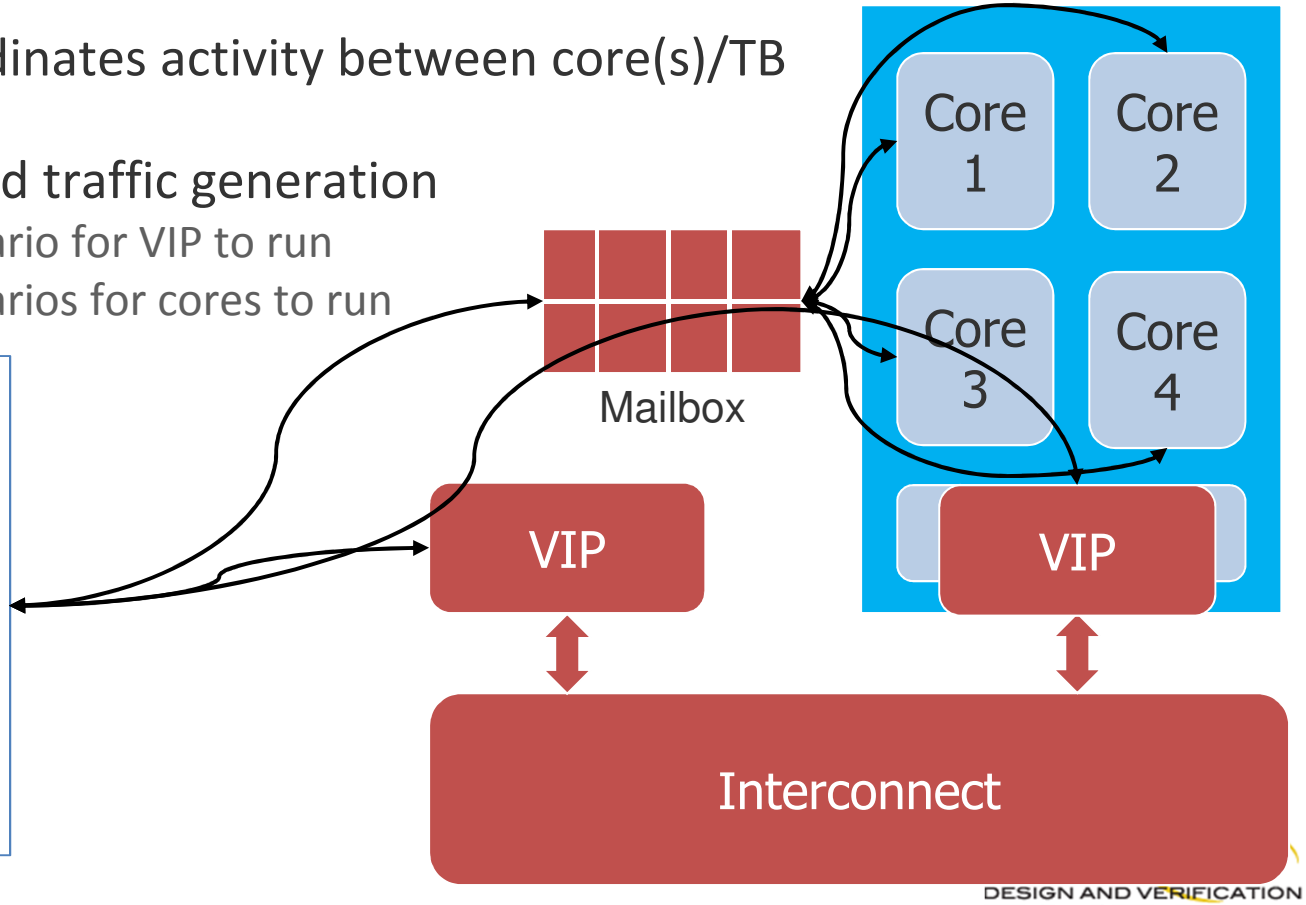

```
typedef struct {
    uint16_t payload_sz;
    uint8_t  payload[1500];
} pkt_t;


// Queue for sw processing
void queue_pkt_accel(pkt_t *pkt);

// Queue for hw-accelerated processing
void queue_pkt_noaccel(pkt_t *pkt);
```

Integration Example

Embedded Software – Coordinated Scenarios

- Hw/Sw scenarios coordinated via a mailbox
  - Mailbox infrastructure provided
- Scenario graph coordinates activity between core(s)/TB
- Example: Coordinated traffic generation
  - Graph selects scenario for VIP to run
  - Graph selects scenarios for cores to run





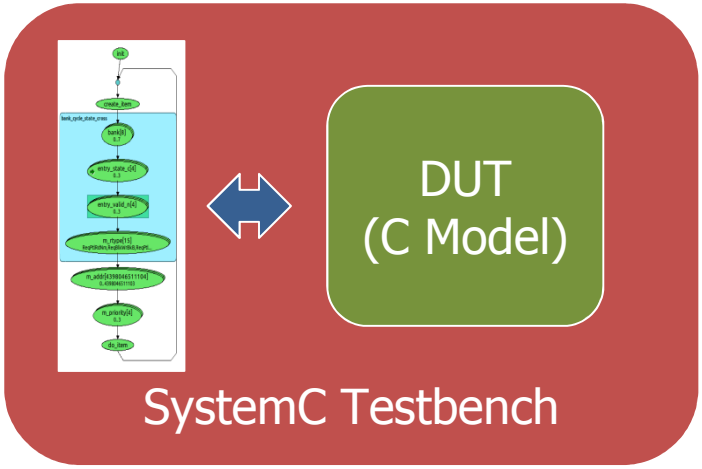
Agenda

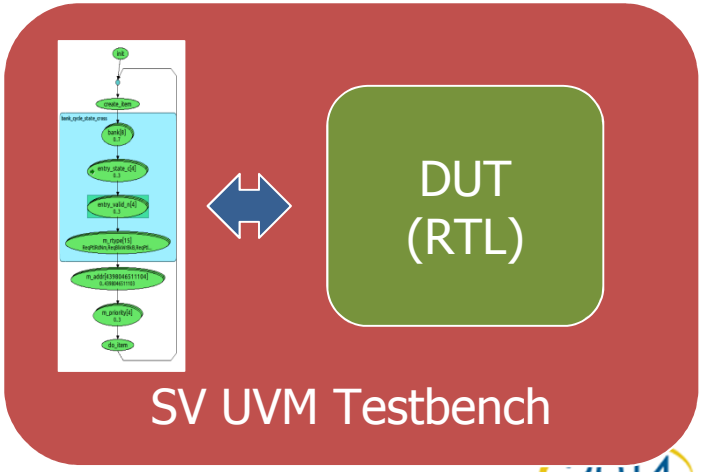
- Modeling Tests with Graphs
- Portable Stimulus with Graphs
- Graph-Based Stimulus Applications

C Model Verification

Portable stimulus across block-Level environments

- Verify C model for high-level synthesis
  - SystemC simulation environment
- Re-run same tests on RTL result of synthesis
  - SystemVerilog simulation environment
  - UVM testbench
- Graph Benefits
  - Same graph used in both environments
  - Highly-productive test creation SystemC
  - Systematic verification

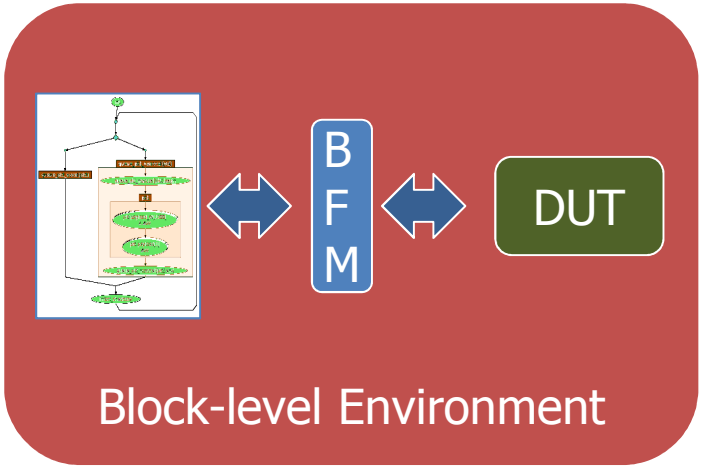


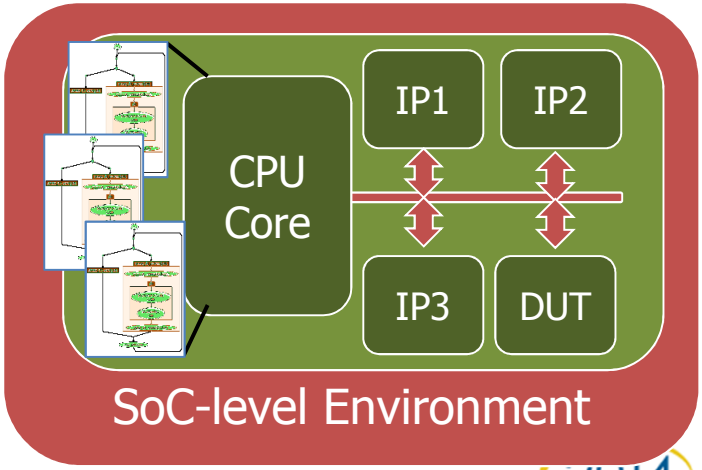


Test Reuse IP to SoC

Portable stimulus enables vertical reuse

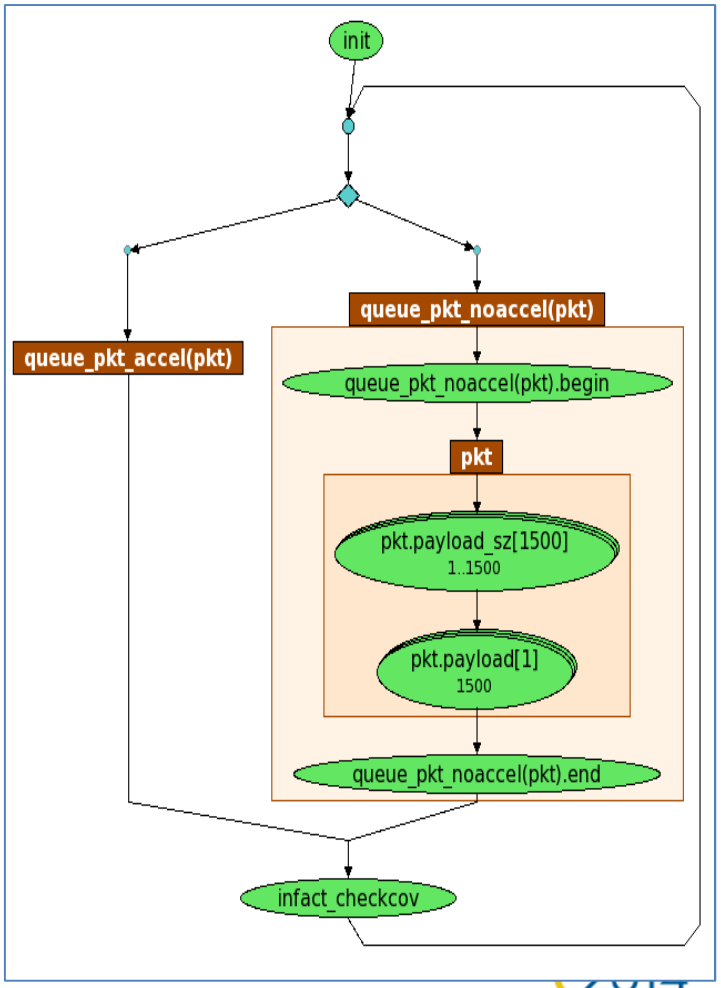
- Run IP-level test in UVM environment
  - Scenario driven via BFM
  - Test targets single IP block
- Re-run test in SoC environment
  - Scenario driven via processor core
  - Run tests for multiple IP blocks in parallel
- Graph Benefits
  - Same test scenario run in both environments
  - More-comprehensive testing at SoC level





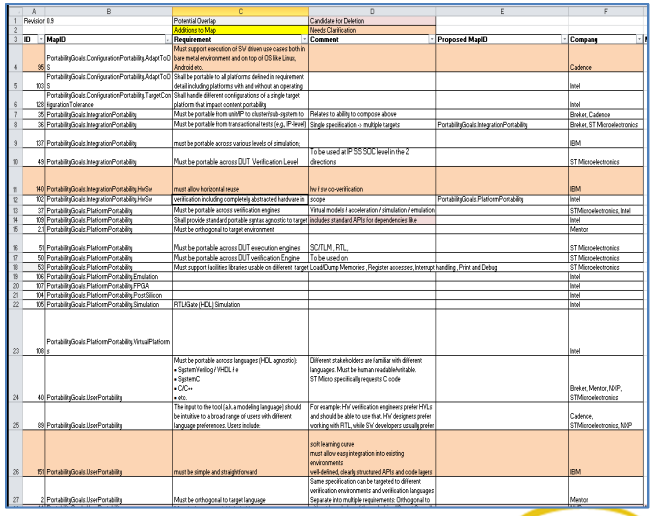
Graph-Based Portable Stimulus

- High-Productivity Input Specification
  - Familiar data constructs and constraints
  - Formally captures control flow
- Efficient and flexible execution
  - 10-100x more efficient than random
  - Scalable to a simulation farm
- Portable
  - Existing HVLs
  - Embedded software
- Highly Automatable
  - Import/export
  - Analysis



Portable Stimulus Standardization

- Portable Stimulus Proposed Working Group
  - Launched in May 2014
- PSPWG Charter
  - Investigate need and requirements for a Portable Stimulus standard
  - Report to the Accellera board
    - Recommendation on formation of a working group
    - Scope of the proposed working group
- PSPWG Status
  - Weekly meetings
  - Collecting and organizing requirements
  - Users and vendors actively participating
  - Report to Accellera board in November



	A	B	C	D	E	F
1	Revision 0.9		Potential Overlap	Candidate for Deletion		
2			Additions to Map	Needs Clarification		
3	ID	MapID	Requirement	Comment	Proposed MapID	Company
4	95	PortabilityGoals.ConfigurationPortability.AdaptToG S	Must support execution of SV driven use cases both in bare metal environment and on top of OS like Linux, Android etc.			Cadence
5	103	PortabilityGoals.ConfigurationPortability.AdaptToG S	Shall be portable to all platforms defined in requirement detail including platforms with and without an operating			Intel
6	128	PortabilityGoals.ConfigurationPortability.TargetCon figurationTolerance	Shall handle different configurations of a single target platform that impact content portability			Intel
7	35	PortabilityGoals.IntegrationPortability	Must be portable from unit/IP to cluster/sub-system to	Relates to ability to compose above		Breker, Cadence
8	36	PortabilityGoals.IntegrationPortability	Must be portable from transactional tests (e.g., IP level	Single specification -> multiple targets	PortabilityGoals.IntegrationPortability	Breker, ST Microelectronics
9	137	PortabilityGoals.IntegrationPortability	must be portable across various levels of simulation,			IBM
10	48	PortabilityGoals.IntegrationPortability	Must be portable across DUT Verification Level	To be used at IP SS SOC level in the 2 directions		ST Microelectronics
11	140	PortabilityGoals.IntegrationPortability.HwSw	must allow horizontal reuse	hw / sw co-verification		IBM
12	102	PortabilityGoals.IntegrationPortability.HwSw	verification including completely abstracted hardware in	scope	PortabilityGoals.PlatformPortability	Intel
13	127	PortabilityGoals.PlatformPortability	Must be portable across verification engines	Virtual models / acceleration / simulation / emulation		STMicroelectronics, Intel
14	106	PortabilityGoals.PlatformPortability	Shall provide standard portable cpp/sv agnostic to target	includes standard APIs for dependencies like		Intel
15	21	PortabilityGoals.PlatformPortability	Must be orthogonal to target environment			Mentor
16	51	PortabilityGoals.PlatformPortability	Must be portable across DUT execution engines	SC/TLM , RTL,		ST Microelectronics
17	50	PortabilityGoals.PlatformPortability	Must be portable across DUT verification Engine	To be used on		ST Microelectronics
18	53	PortabilityGoals.PlatformPortability	Must support facilities libraries usable on different target	Load/Dump Memories , Register accesses, Interrupt handling , Print and Debug		ST Microelectronics
19	105	PortabilityGoals.PlatformPortability.Emulation				Intel
20	107	PortabilityGoals.PlatformPortability.FPGA				Intel
21	104	PortabilityGoals.PlatformPortability.PostSilicon				Intel
22	109	PortabilityGoals.PlatformPortability.Simulation	RTL/Gate (HDL) Simulation			Intel
23	100	PortabilityGoals.PlatformPortability.VirtualPlatform s				Intel
24	49	PortabilityGoals.UserPortability	Must be portable across languages (HDL agnostic): • SystemVerilog / VHDL / e • SystemC • C/C++ • etc.	Different stakeholders are familiar with different languages. Must be human readable/writable. ST Micro specifically requests C code		Breker, Mentor, NXP, STMicroelectronics
25	88	PortabilityGoals.UserPortability	The input to the tool (a.k.a modeling language) should be intuitive to a broad range of users with different language preferences. Users include:	For example HW verification engineers prefer HVLs and should be able to use that, HW designers prefer working with RTL, while SV developers usually prefer		Cadence, STMicroelectronics, NXP
26	151	PortabilityGoals.UserPortability	must be simple and straightforward	soft learning curve must allow easy integration into existing environments well-defined, clearly structured APIs and code layers		IBM
27	2	PortabilityGoals.UserPortability	Must be orthogonal to target language	Same specification can be targeted to different verification environments and verification languages Separate into multiple requirements: Orthogonal to		Mentor

Questions

Finalize slide set with questions slide

Thank You!

www.mentor.com

www.brekersystems.com