

BREKER™ THE LEADER IN PORTABLE STIMULUS

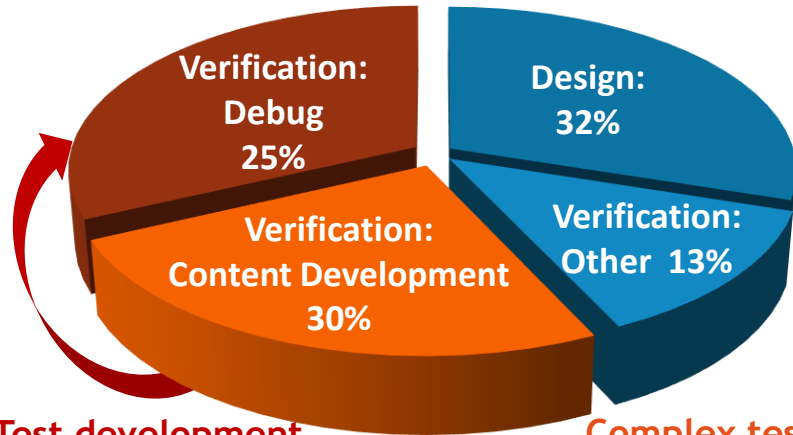
Top-Down, UVM-Style Testbenches With PSS

DVClub May 2022

Dave Kelf

The State of Existing Testbenches

Project Resource Deployment



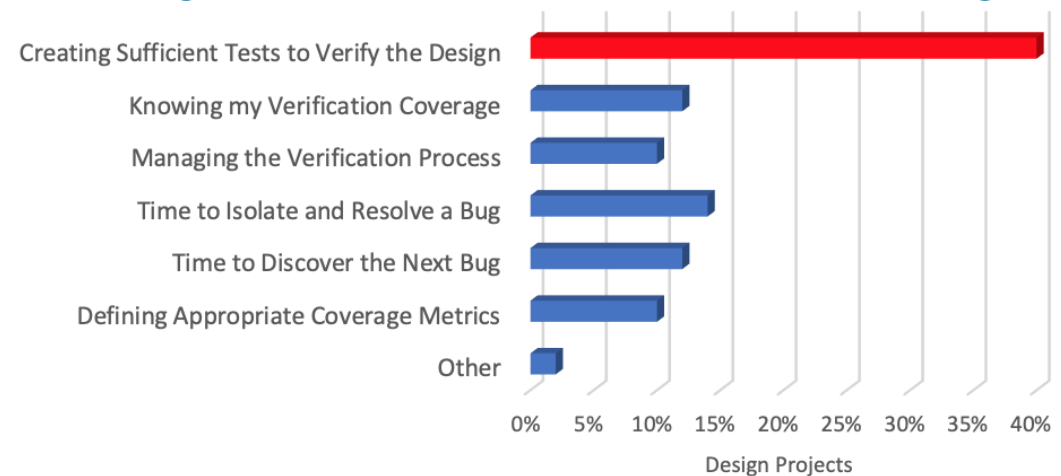
Test development
drives debug

Complex tests
hard to get right

Source: Wilson Research 2020

- We've all seen these kind of charts
- Testbenches have come a long way
- But we still spend a lot of time on them

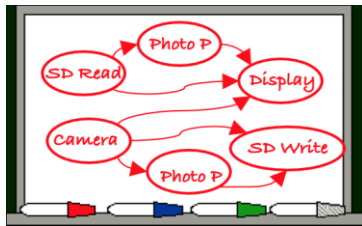
Largest Functional Verification Challenge



Source: Wilson Research 2020

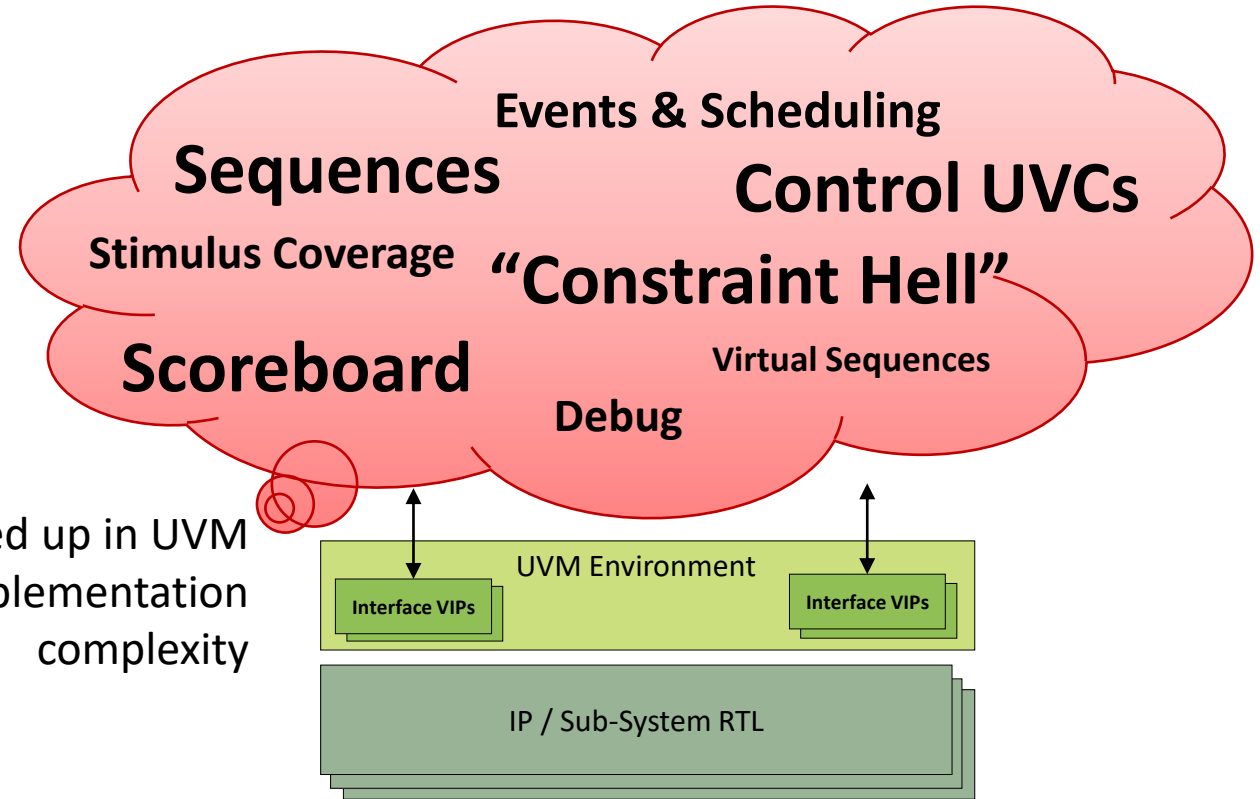
Scaling UVM

UVM is a strong standard but has trouble scaling to complex IPs and sub-systems
UVM testbench & sequence development overshadows verification work



High value verification knowledge...

... locked up in UVM implementation complexity



The SoC & Beyond Verification Requirement

*Block Functionality
UVM simulation*



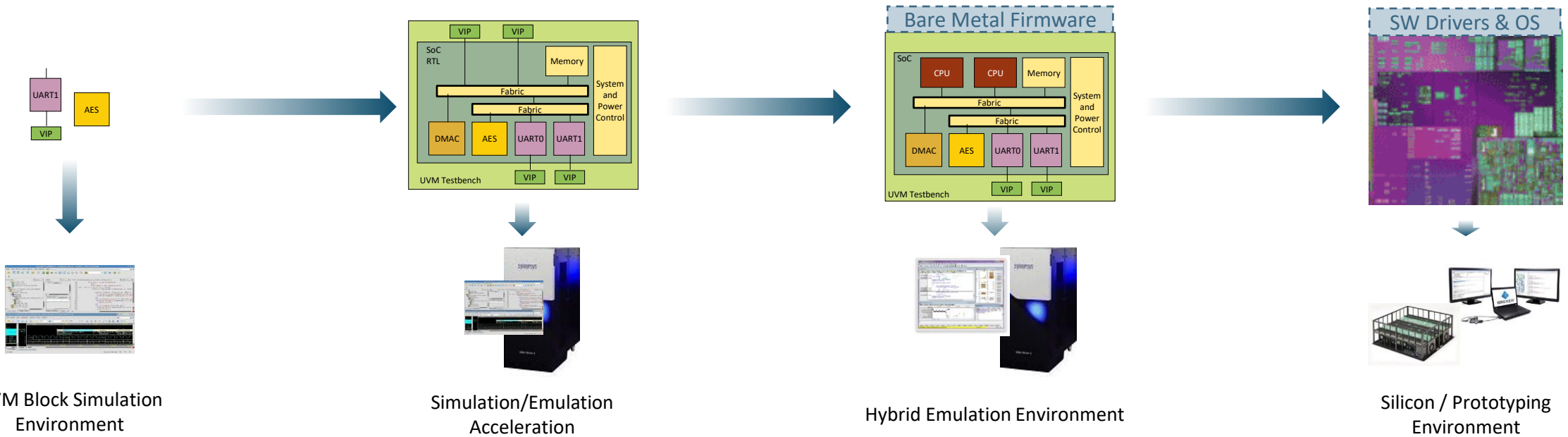
*SoC Integration "Gap"
Ad hoc test content*



*System Validation
Real-workloads on HW*

flexibility

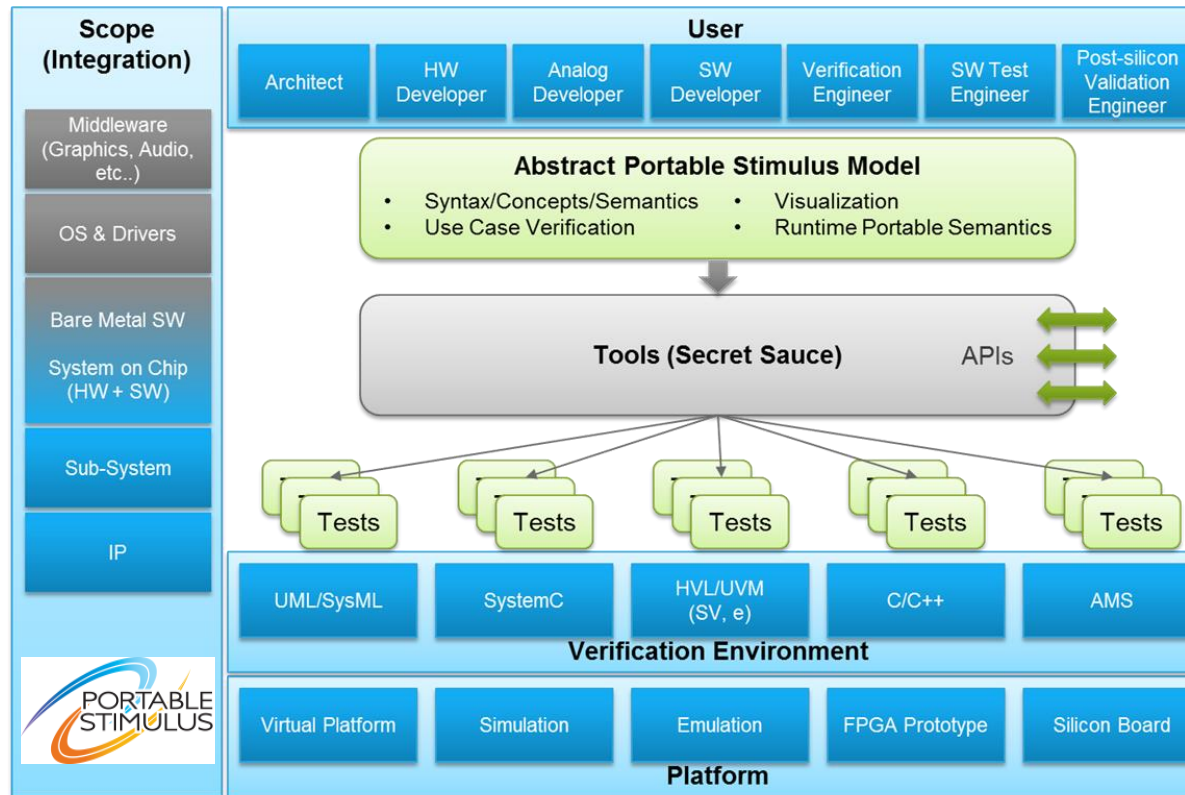
performance



Accellera PSS Created to Address Portability and Scaling

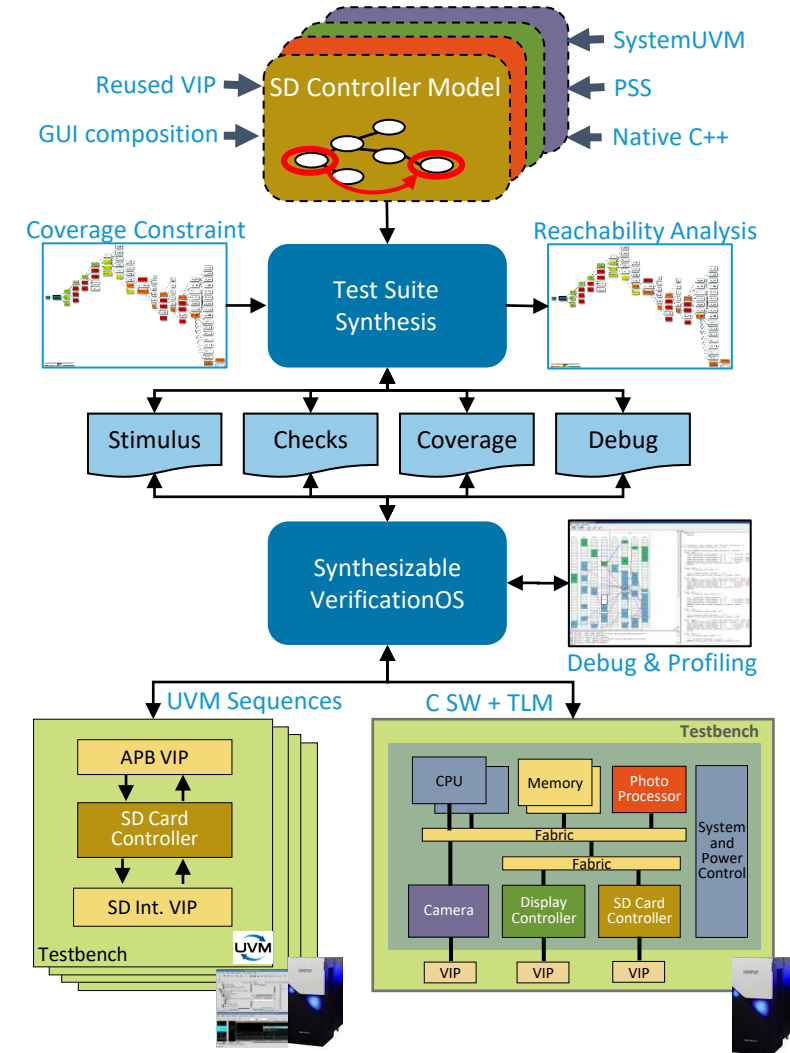
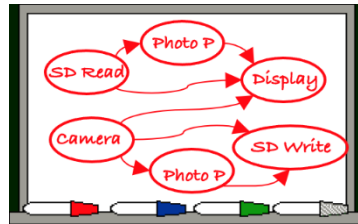
PSS convenient to describe a specification as a graph or flow

Accellera Sponsored Portable Stimulus Working Group



Proposed Portable Stimulus Specification (Courtesy: Accellera Systems Initiative)

Breker Uses PSS/C++ in a Test Content Synthesis Flow



- Easy, familiar model content generation

- Self checking – no scoreboard
- Easy-to-use, abstract constraints
- Automated virtual sequencing

- Pre-randomized, coverage-driven, “no bug standing” tests

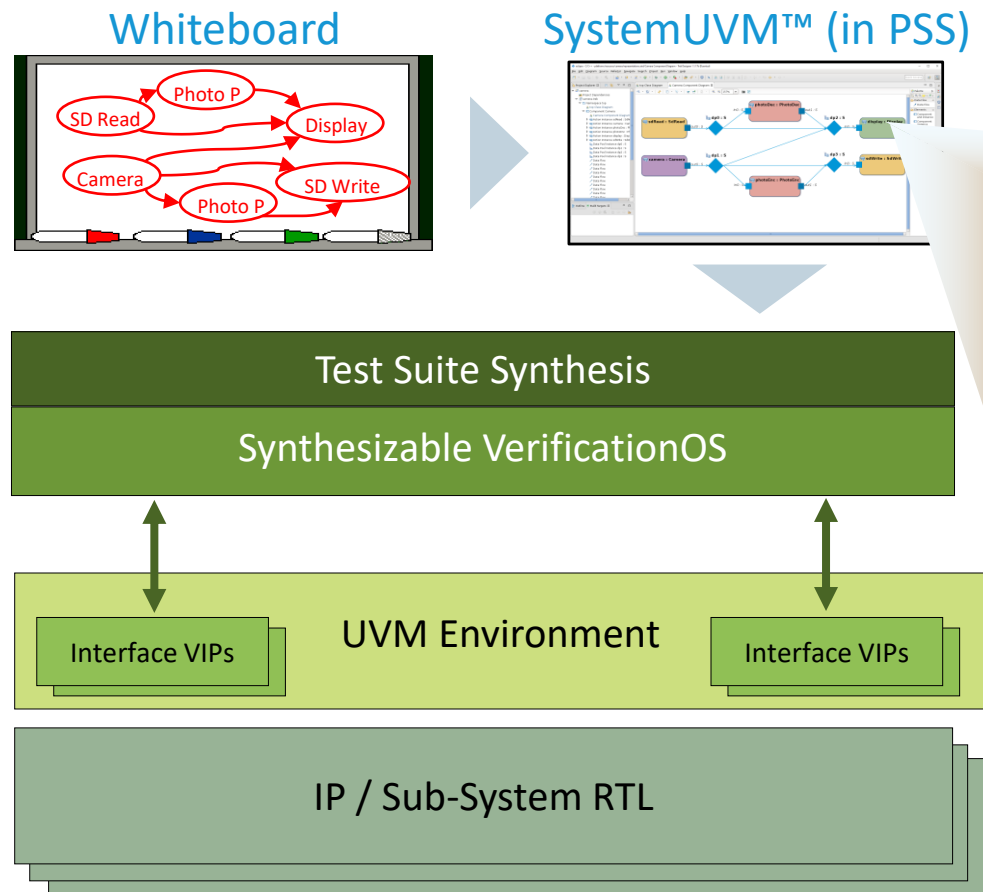
- Deep sequential, AI Planning Algorithm optimized generation
- Pre-RTL coverage, no separate model
- Coverage-driven: eliminate respins

- Easily integrated, portable (re)use model

- Works with existing UVM testbench
- Modular reuse across projects & SoC
- Pre-randomized emulation acceleration

Can we combine PSS and UVM? SystemUVM™ Concept

UVM-Compliant Classes and Utilities built into PSS
Similar Idea to UVM Class Library for SystemVerilog



SystemUVM™ Characteristics

- Seamless interoperability between PSS & UVM
- UVM compliant class libraires in PSS
 - TLM ports
 - UVM compliant register model
 - UVM compliant messaging
- Synthesizable VerificationOS for tight integration into existing UVM testbenches
- Abstract constraints to reduce complexity

SystemUVM PSS Struct definition

```
struct uart_tx {  
    bit[8] payload;  
    bit[1] stop_bits;  
    bit[16] transmit_delay;  
};
```

generate

Generated uvm transaction definitions

```
class uart_tx extends uvm_sequence_item;  
    bit [7:0] payload;  
    bit [1:0] stop_bits;  
    int unsigned transmit_delay;  
    ...  
    task get( input string tb_path, output bit end_of_test );  
        ...  
        if ( ! end_of_test ) begin  
            payload      = trek_get_int_unsigned(tb_path, "payload");  
            stop_bits    = trek_get_int_unsigned(tb_path, "stop_bits");  
            transmit_delay = trek_get_int_unsigned(tb_path, "transmit_delay");  
        end  
    endtask  
endclass: uart_tx
```

```
// Modeled after UVM TLM ports  
put_port<uart_tx> tx_port;
```

```
uart_tx tx;  
tx.payload = 5;  
tx_port.put(tx);
```

Input ipxact register definitions

```
<spirit:register>
  <spirit:name> UART_LCR </spirit:name>
  <spirit:addressOffset> 0x00000003 </spirit:addressOffset>
  <spirit:size> 8 </spirit:size>
  <spirit:access> read-write </spirit:access>
  <spirit:description> Line Control Register </spirit:description>
  <spirit:reset>
    <spirit:value> 0x03 </spirit:value>
  </spirit:reset>

  <spirit:field>
    <spirit:name> CHAR_SIZE </spirit:name>
    <spirit:bitOffset> 0 </spirit:bitOffset>
    <spirit:bitWidth> 2 </spirit:bitWidth>
    <spirit:access> read-write </spirit:access>
    <spirit:description> Number of bits in each char </spirit:description>
    <spirit:values>
```

generate

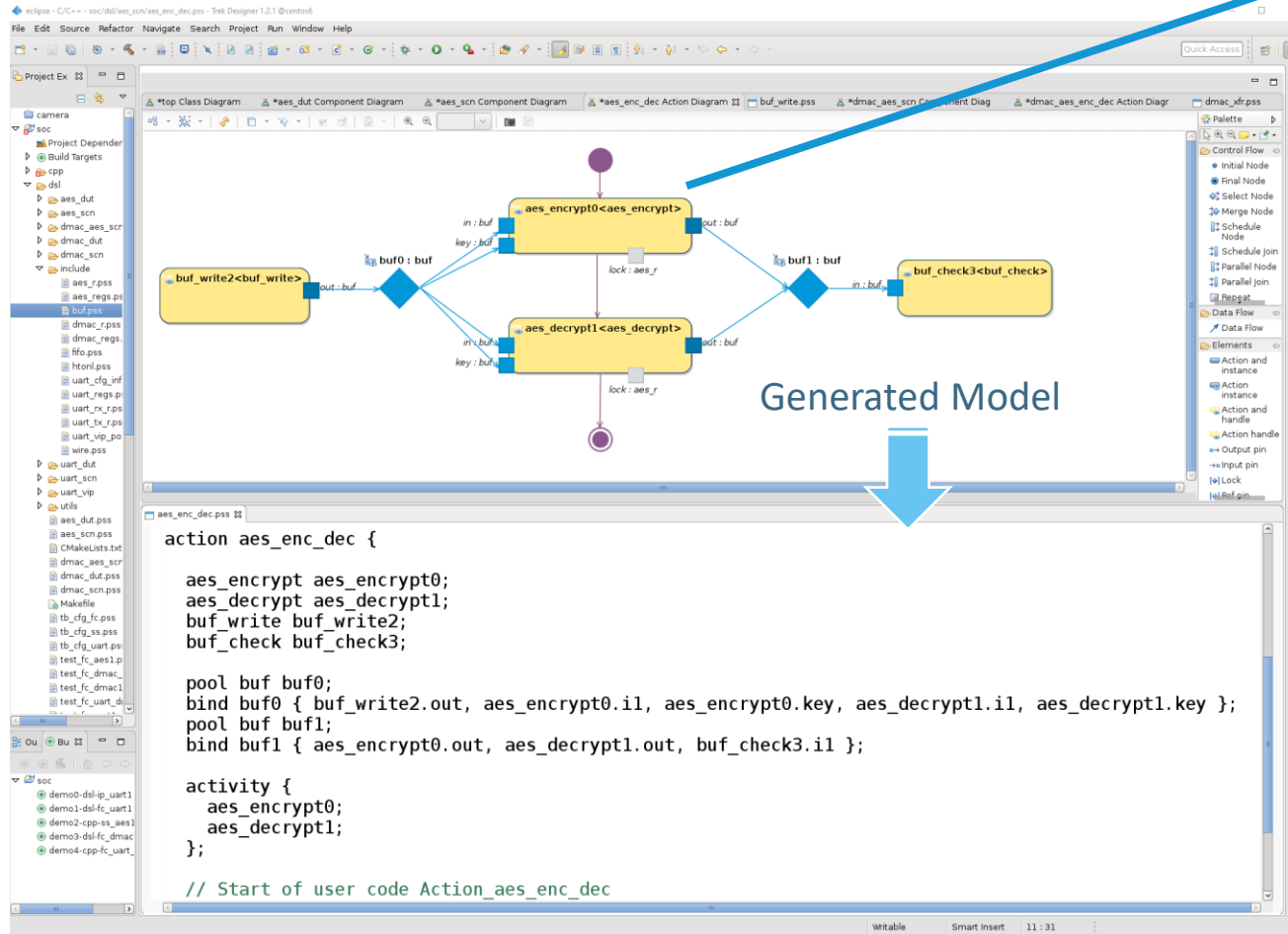
Generated SystemUVM PSS register definitions

```
class reg_ALL_REGISTERS_MMAP__UART_LCR : public reg {
public:
  reg_ALL_REGISTERS_MMAP__UART_LCR(const scope& s) : reg(this,8) {};
  reg_field CHAR_SIZE { "CHAR_SIZE", 2, 0, "RW", 0, 0, 1, 0, 1 };
  reg_field STOP_BITS { "STOP_BITS", 1, 2, "RW", 0, 0, 1, 0, 1 };
  reg_field PARITY_ENABLE { "PARITY_ENABLE", 1, 3, "RW", 0, 0, 1, 0, 1 };
  reg_field PARITY_EVEN { "PARITY_EVEN", 1, 4, "RW", 0, 0, 1, 0, 1 };
  reg_field PARITY_STICK { "PARITY_STICK", 1, 5, "RW", 0, 0, 1, 0, 1 };
  reg_field BREAK_CONTROL { "BREAK_CONTROL", 1, 6, "RW", 0, 0, 1, 0, 1 };
  reg_field DIVISOR_ACCESS { "DIVISOR_ACCESS", 1, 7, "RW", 0, 0, 1, 0, 1 };
};

class ALL_REGISTERS_MMAP_reg_block : public reg_block {
public:
  ALL_REGISTERS_MMAP_reg_block(... ) {
    ...
    ALL_REGISTERS_MMAP.add_reg ( _UART_LCR, 0x00000003 );
  }
};
```

```
// Modeled after UVM registers
regs.regA.fieldF.set(3);
regs.regA.write();
```

Writing SystemUVM IP Models



```
action aes_encrypt {  
    input buf in;  
    input buf key;  
    output buf out;  
    lock aes_r lock;
```

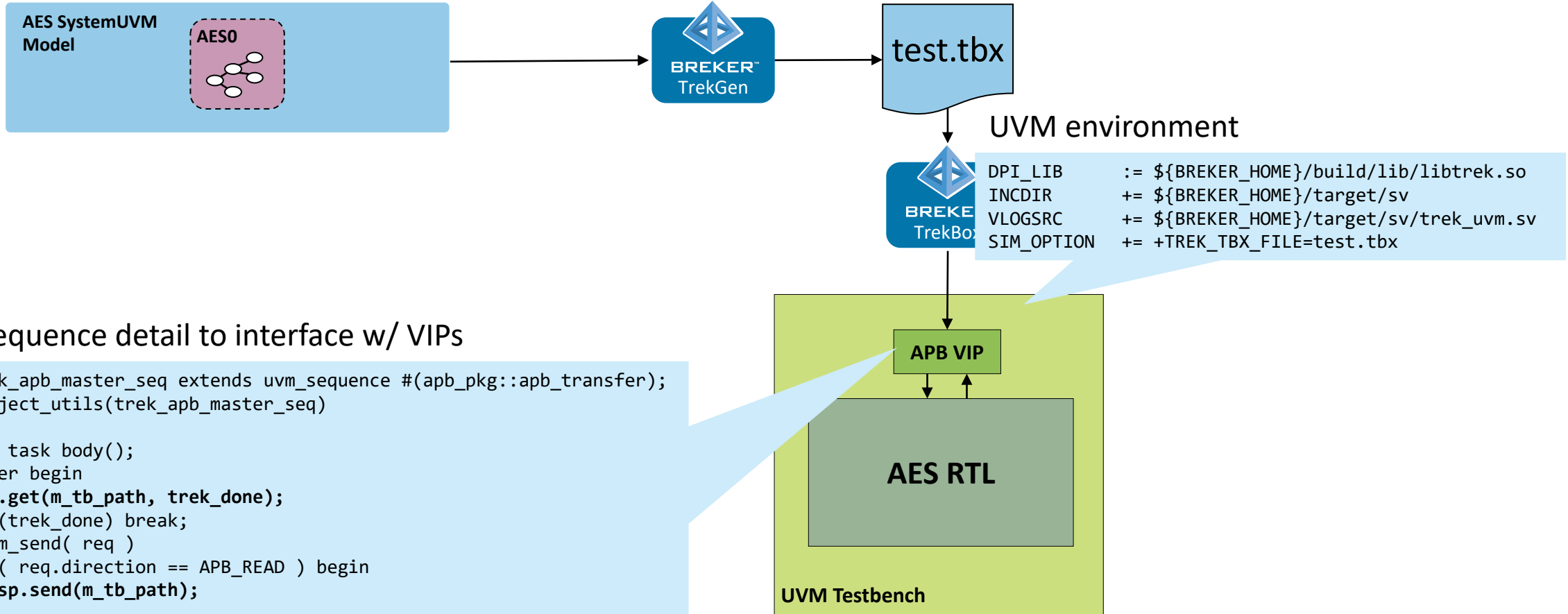
```
// Start of user code Action_aes_encrypt
```

```
constraint in.len == 16;  
constraint key.len == 16;  
constraint out.len == 16;  
ref aes_regs regs;
```

```
void post_solve() {  
    in.addr = regs.AES_INPUT0.get_address();  
    key.addr = regs.AES_KEY0.get_address();  
    out.addr = regs.AES_OUTPUT0.get_address();  
}
```

```
void body() {  
    pss_info( name(), "aes_encrypt", pss::target );  
    regs.AES_CTRL.START.set(1);  
    regs.AES_CTRL.MODE.set(0); // 0 for Encrypt, 1 for Decrypt  
    regs.AES_CTRL.write();  
  
    regs.AES_CTRL.DONE.poll(1); // wait for completion  
  
    // call reference model to predict results  
    encrypt_aes ( in.expect, key.expect);  
    // forward expect to output  
    out.expect = in.expect;  
}  
// End of user code  
};
```

Fitting SystemUVM content into an existing UVM IP testbench



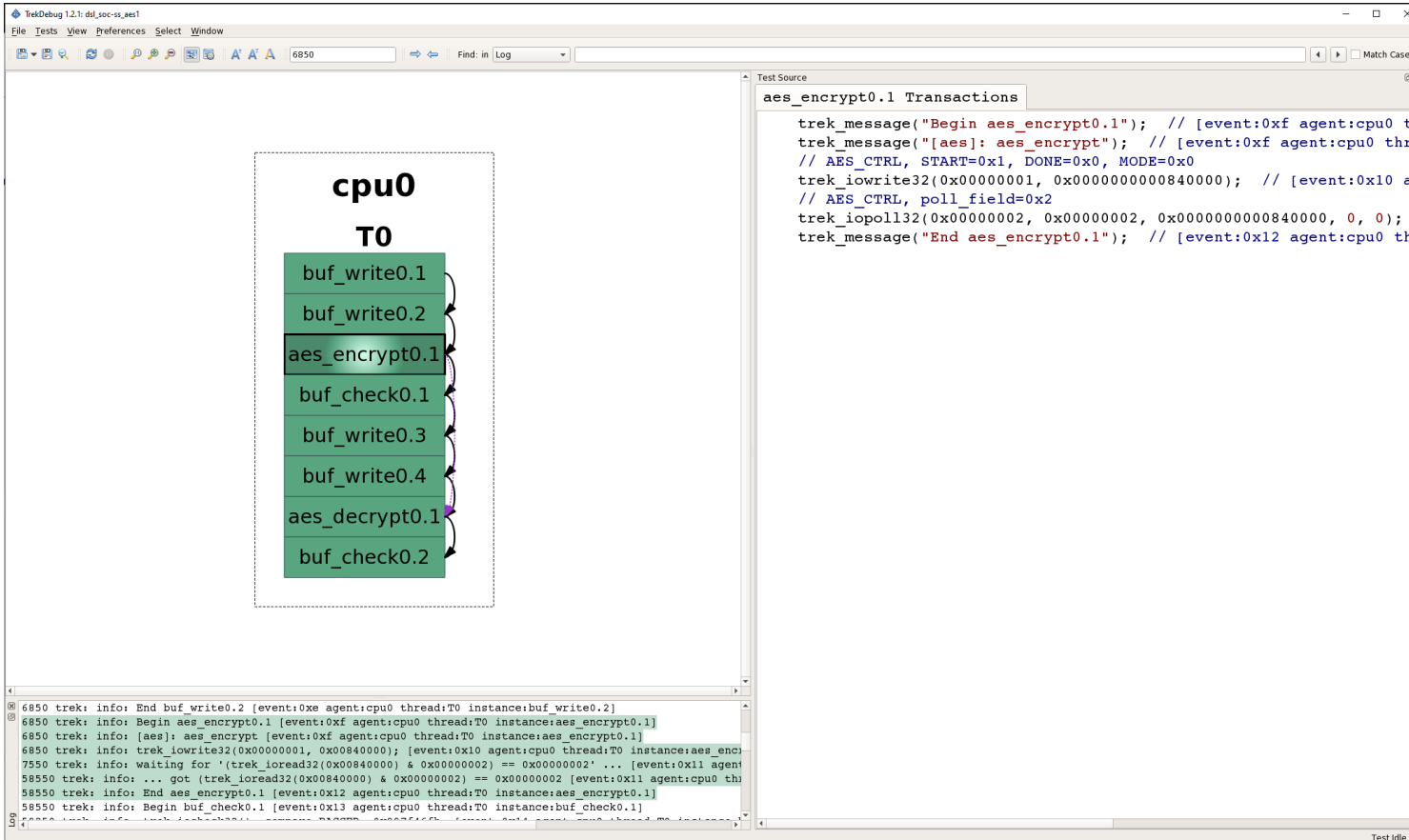
UVM sequence detail to interface w/ VIPs

```

class trek_apb_master_seq extends uvm_sequence #(apb_pkg::apb_transfer);
    `uvm_object_utils(trek_apb_master_seq)

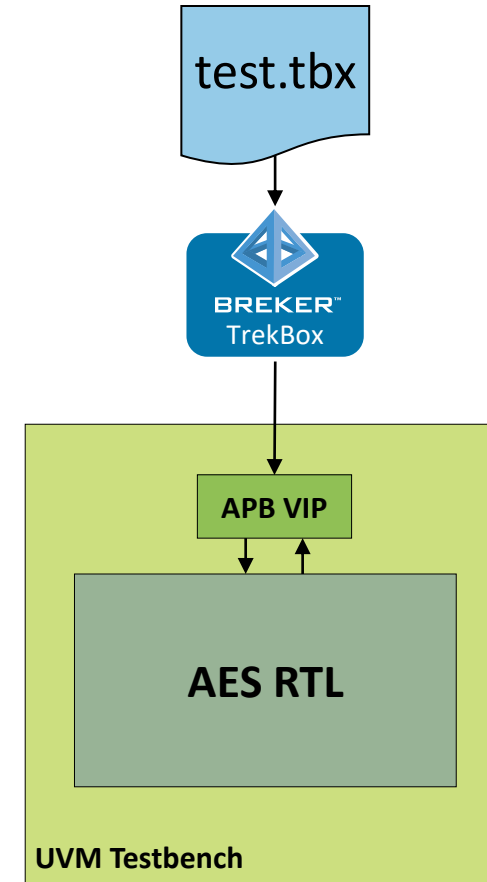
    virtual task body();
        forever begin
            req.get(m_tb_path, trek_done);
            if (trek_done) break;
            `uvm_send( req )
            if ( req.direction == APB_READ ) begin
                rsp.send(m_tb_path);
            end
            req.item_done( m_tb_path );
        end
    endtask
endclass
    
```

Running a single IP test

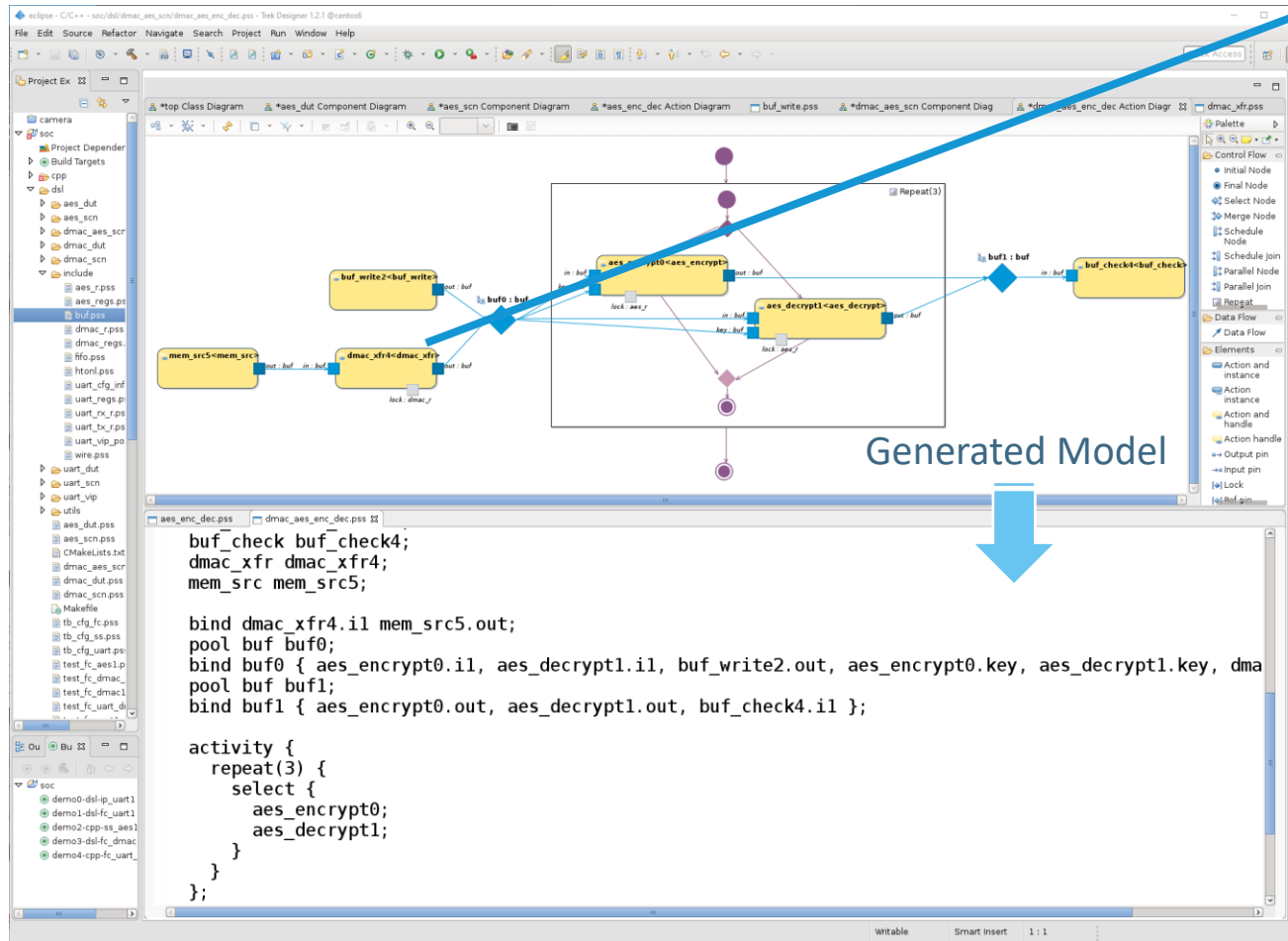


The screenshot shows the TrekDebug 1.2.1 interface. On the left, a diagram shows a **cpu0** thread **T0** executing a sequence of tasks: **buf_write0.1**, **buf_write0.2**, **aes_encrypt0.1**, **buf_check0.1**, **buf_write0.3**, **buf_write0.4**, **aes_decrypt0.1**, and **buf_check0.2**. On the right, the **Test Source** window displays the code for the **aes_encrypt0.1** transaction, including `trek_message`, `trek_iowrite32`, `trek_iopoll32`, and another `trek_message` call. The bottom window shows the **Log** output, with the following entries:

```
6850 trek: info: End buf_write0.2 [event:0xe agent:cpu0 thread:T0 instance:buf_write0.2]
6850 trek: info: Begin aes_encrypt0.1 [event:0xf agent:cpu0 thread:T0 instance:aes_encrypt0.1]
6850 trek: info: [aes]: aes encrypt [event:0xf agent:cpu0 thread:T0 instance:aes_encrypt0.1]
6850 trek: info: trek_iowrite32(0x00000001, 0x00840000); [event:0x10 agent:cpu0 thread:T0 instance:aes_encrypt0.1]
7550 trek: info: waiting for '(trek_ioread32(0x00840000) & 0x00000002) == 0x00000002' ... [event:0x11 agent:cpu0 thread:T0 instance:aes_encrypt0.1]
58550 trek: info: ... got (trek_ioread32(0x00840000) & 0x00000002) == 0x00000002 [event:0x11 agent:cpu0 thread:T0 instance:aes_encrypt0.1]
58550 trek: info: End aes_encrypt0.1 [event:0x12 agent:cpu0 thread:T0 instance:aes_encrypt0.1]
58550 trek: info: Begin buf_check0.1 [event:0x13 agent:cpu0 thread:T0 instance:buf_check0.1]
```



Composing SystemUVM Sub-system Models

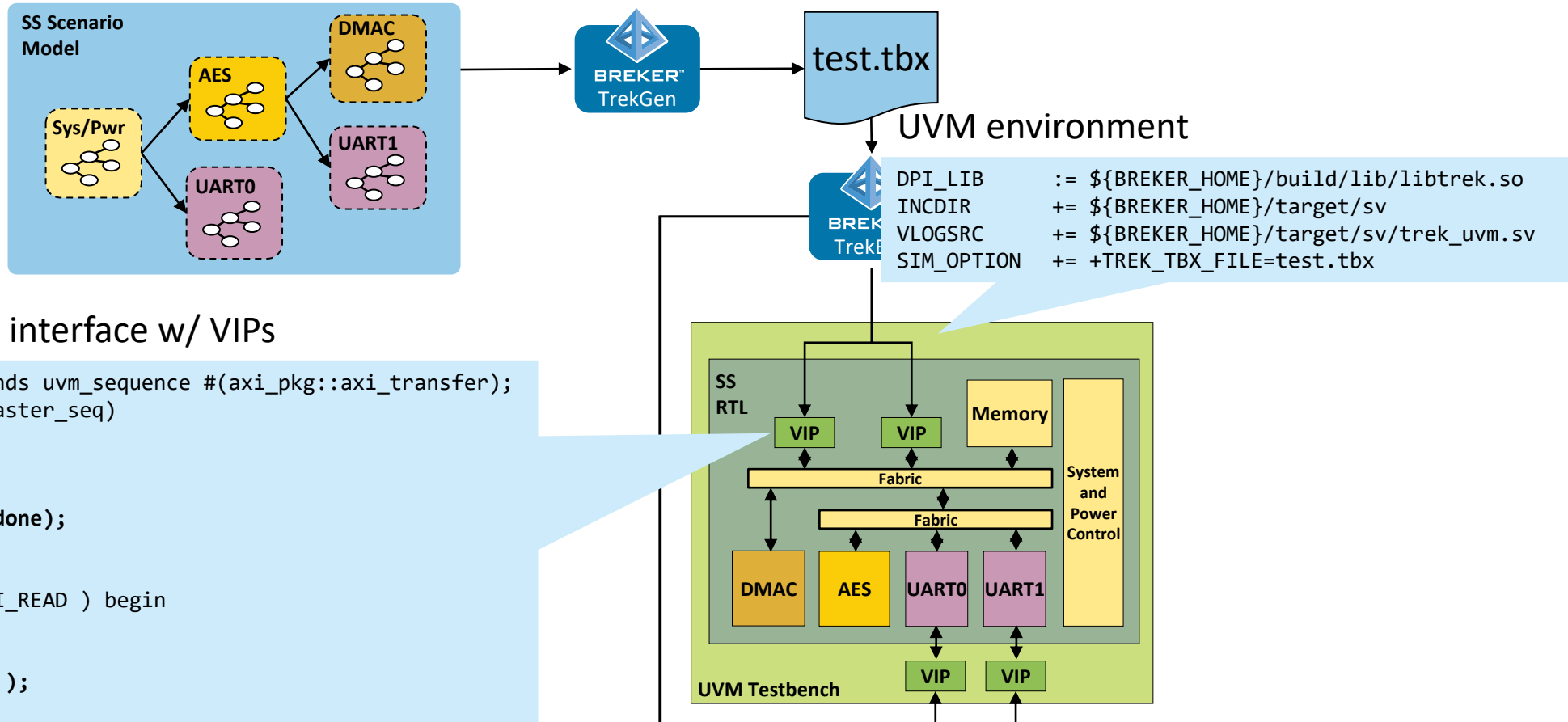


```
action dmac_xfr {
  input buf in;
  output buf out;
  lock dmac_r lock;

  // Start of user code Action_dmac_xfr
  constraint in.len == out.len ;
  ref dmac_regs regs;

  void body() {
    int chan = lock.instance_id;
    pss_info (name(), "dma_xfr", pss::target);
    // configure target and source addr
    regs.dma[chan].DMA_TADDR.ADDRESS.set(out.addr);
    regs.dma[chan].DMA_TADDR.write();
    regs.dma[chan].DMA_SADDR.ADDRESS.set(in.addr);
    regs.dma[chan].DMA_SADDR.write();
    regs.dma[chan].DMA_BUFF.SRC_INCR.set(1);
    regs.dma[chan].DMA_BUFF.DST_INCR.set(1);
    regs.dma[chan].DMA_BUFF.write();
    // start transfer
    regs.dma[chan].DMA_TRANS.SIZE.set(in.len);
    regs.dma[chan].DMA_TRANS.START.set(1);
    regs.dma[chan].DMA_TRANS.write();
    // wait for completion
    regs.dma[chan].DMA_INT_STATUS.COMPLETED.poll(1);
    // forward expect data
    out.expect = in.expect;
  }
  // End of user code
};
```

Fitting SystemUVM into existing UVM Sub-system testbench



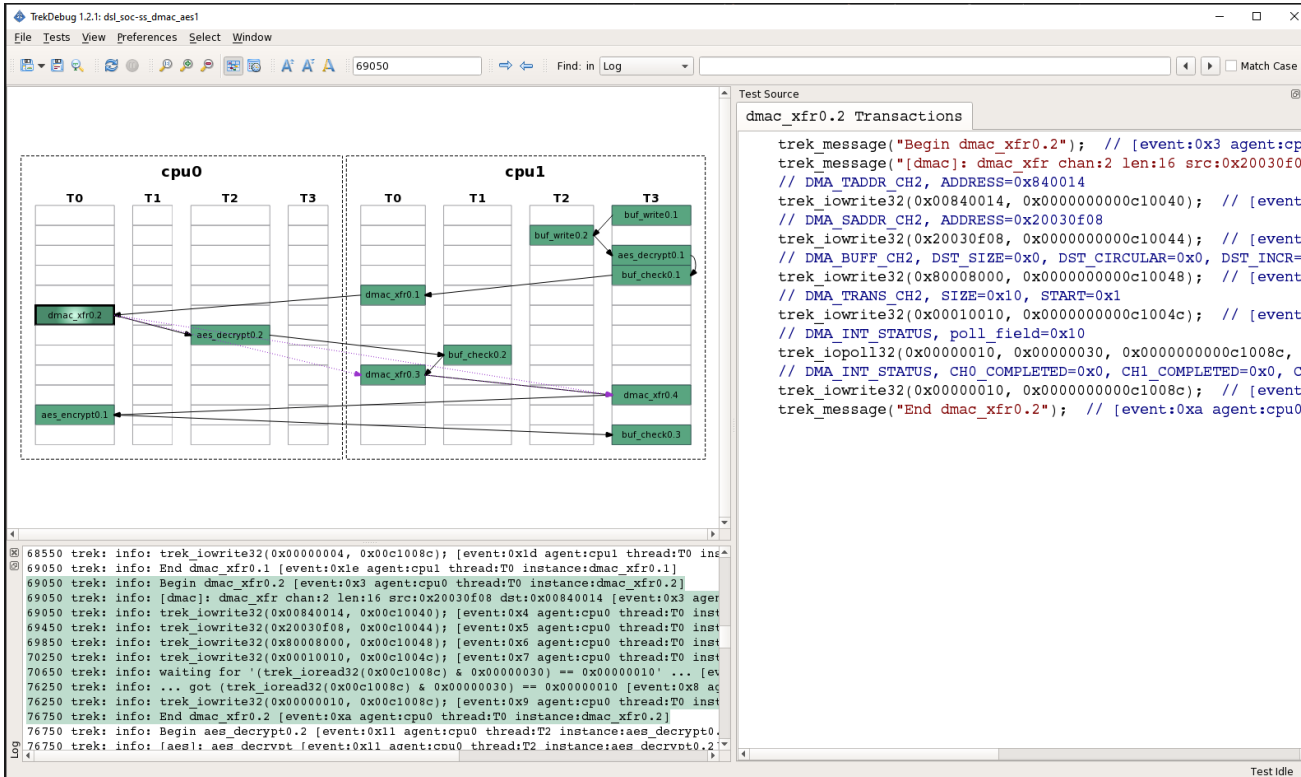
UVM sequence detail to interface w/ VIPs

```

class trek_axi_master_seq extends uvm_sequence #(axi_pkg::axi_transfer);
  `uvm_object_utils(trek_axi_master_seq)

  virtual task body();
    forever begin
      req.get(m_tb_path, trek_done);
      if (trek_done) break;
      `uvm_send( req )
      if ( req.direction == AXI_READ ) begin
        rsp.send(m_tb_path);
      end
      req.item_done( m_tb_path );
    end
  endtask
endclass
    
```

Running multi-IP sub-system test



Test Source

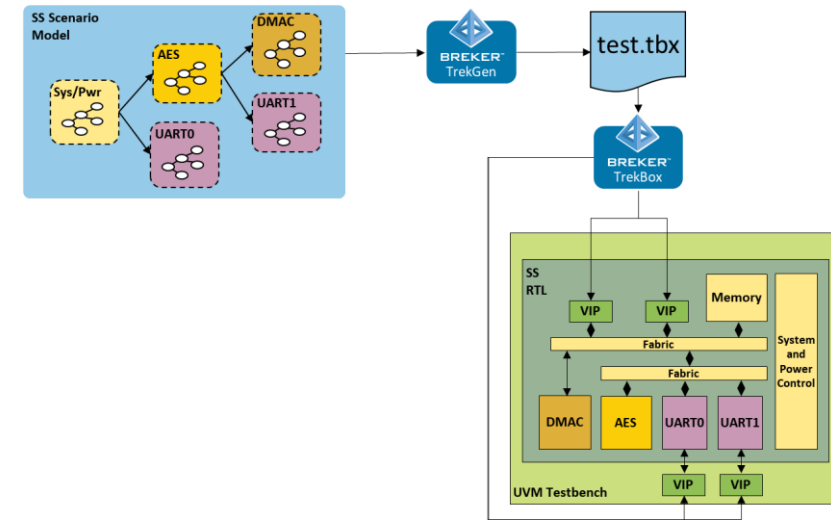
```
dmac_xfr0.2 Transactions
trek_message("Begin dmac_xfr0.2"); // [event:0x3 agent:cpu
trek_message("[dmac]: dmac_xfr chan:2 len:16 src:0x20030f08
// DMA_TADDR_CH2, ADDRESS=0x840014
trek_iowrite32(0x00840014, 0x0000000000c10040); // [event:
// DMA_SADDR_CH2, ADDRESS=0x20030f08
trek_iowrite32(0x20030f08, 0x0000000000c10044); // [event:
// DMA_BUFF_CH2, DST_SIZE=0x0, DST_CIRCULAR=0x0, DST_INCR=0
trek_iowrite32(0x80008000, 0x0000000000c10048); // [event:
// DMA_TRANS_CH2, SIZE=0x10, START=0x1
trek_iowrite32(0x00010010, 0x0000000000c1004c); // [event:
// DMA_INT_STATUS, poll_field=0x10
trek_iopoll32(0x00000010, 0x00000030, 0x0000000000c1008c, 0
// DMA_INT_STATUS, CH0_COMPLETED=0x0, CH1_COMPLETED=0x0, CH
trek_iowrite32(0x00000010, 0x0000000000c1008c); // [event:
trek_message("End dmac_xfr0.2"); // [event:0xa agent:cpu0
```

Log

```
69550 trek: info: trek_iowrite32(0x00000004, 0x00c1008c); [event:0x1d agent:cpu1 thread:T0 inst:
69050 trek: info: End dmac_xfr0.1 [event:0x1e agent:cpu1 thread:T0 instance:dmac_xfr0.1]
69050 trek: info: Begin dmac_xfr0.2 [event:0x3 agent:cpu0 thread:T0 instance:dmac_xfr0.2]
69050 trek: info: [dmac]: dmac_xfr chan:2 len:16 src:0x20030f08 dst:0x00840014 [event:0x3 ager
69050 trek: info: trek_iowrite32(0x00840014, 0x00c10040); [event:0x4 agent:cpu0 thread:T0 inst:
69450 trek: info: trek_iowrite32(0x20030f08, 0x00c10044); [event:0x5 agent:cpu0 thread:T0 inst:
69850 trek: info: trek_iowrite32(0x80008000, 0x00c10048); [event:0x6 agent:cpu0 thread:T0 inst:
70250 trek: info: trek_iowrite32(0x00010010, 0x00c1004c); [event:0x7 agent:cpu0 thread:T0 inst:
70650 trek: info: waiting for '(trek_ioread32(0x00c1008c) & 0x00000030) == 0x00000010' ... [ev
76250 trek: info: ... got (trek_ioread32(0x00c1008c) & 0x00000030) == 0x00000010 [event:0x8 ag
76250 trek: info: trek_iowrite32(0x00000010, 0x00c1008c); [event:0x9 agent:cpu0 thread:T0 inst:
76750 trek: info: End dmac_xfr0.2 [event:0xa agent:cpu0 thread:T0 instance:dmac_xfr0.2]
76750 trek: info: Begin aes_decrypt0.2 [event:0x11 agent:cpu0 thread:T2 instance:aes_decrypt0.
76750 trek: info: [aes]: aes_decrypt [event:0x11 agent:cpu0 thread:T2 instance:aes_decrypt0.2]
```

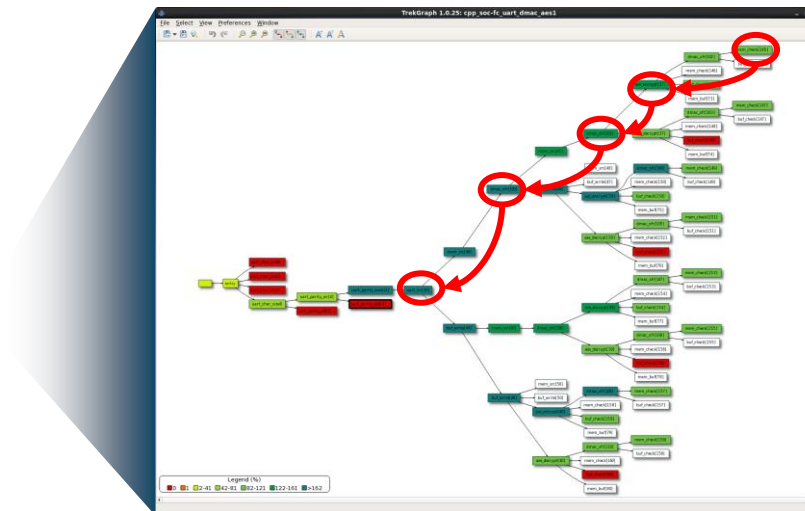
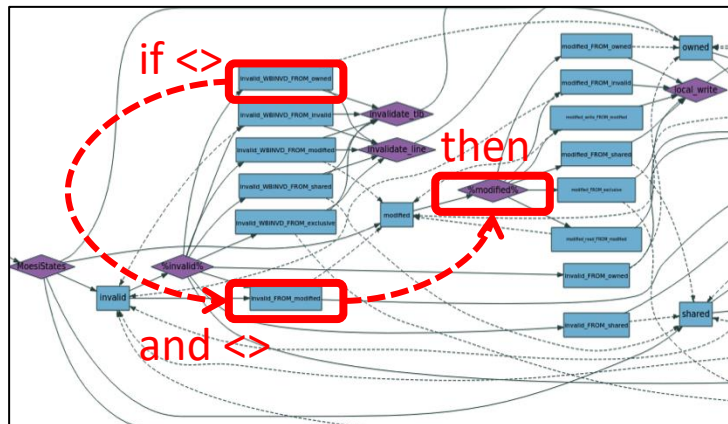
Transaction Diagram

The diagram shows two CPUs, CPU0 and CPU1, with transactions T0, T1, T2, and T3. CPU0 transactions include dmac_xfr0.2, aes_decrypt0.2, and aes_encrypt0.1. CPU1 transactions include dmac_xfr0.1, buf_write0.2, aes_decrypt0.1, buf_check0.1, dmac_xfr0.3, buf_check0.2, dmac_xfr0.4, and buf_check0.3. Arrows indicate data flow between these transactions across the two CPUs.



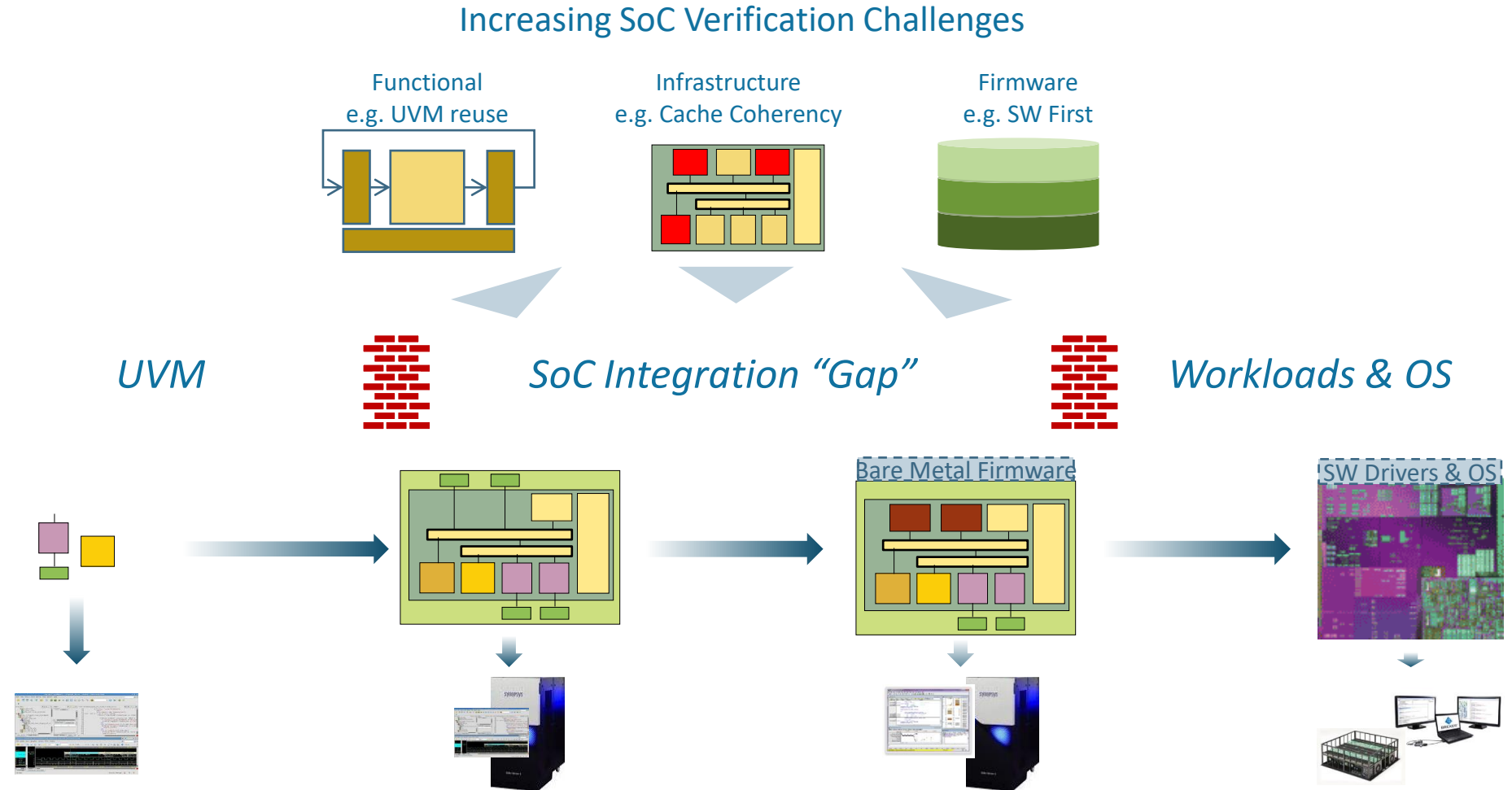
Path Constraints: Tackling the “Over-Constraining” Issue

- Node-based constraints can lead to complexity: hard to understand, manage, and maintain
- Abstraction allows spatial, sequential constraints to be placed across the specification
- Also allows for coverage constraints to be declared in coverage-driven synthesis

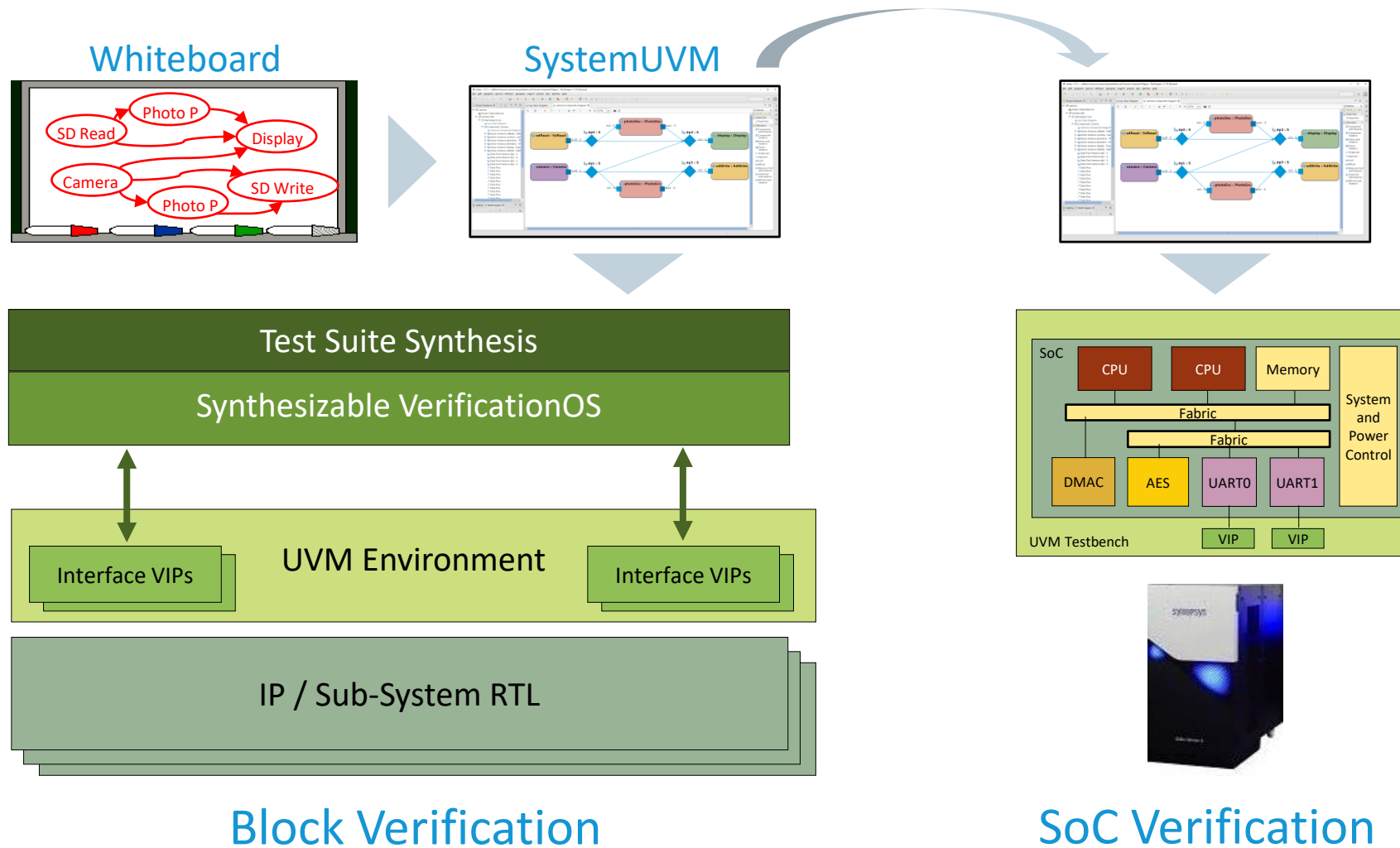


The SoC Verification Gap

SoC infrastructure verification cannot be satisfied by UVM or real workloads



SystemUVM: Easy Port to SoC



Thanks for Listening!
Any Questions?
