2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Dealing with Programmable IP:

Where the Rubber Meets the PSS Deployment Road

Karthick Gururaj, Vayavya Labs Pvt. Ltd.

accellera SYSTEMS INITIATIVE

Agenda

- Goals for this workshop
- A brief recap of HSI in PSS1.0/PSS1.1
- Portable^2 Stimulus for system-level verification: Challenges and guidelines
  - Code walk-through: UART
- Handling complex I/O in your PSS model
  - Code walk-through: PCIe Controller

Who am I? Who are we?

- Principal Architect at Vayavya Labs
- > 20 years of experience
- Member of Accellera PSWG, past member of TLMWG
- Interests:
  - Language design
  - Technologies that use PS and HSI
  - Virtual Platforms
  - Embedded systems

- Vayavya Labs
  - 13+ old EDA/ESL company
  - “Single source” HSI specification
  - Embedded software experts
- Please check the brochure for more information!

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Goals for this workshop

- #1: Re-Use configuration programming UVM IP Bus to SOC Bus
- #2: Composing multiple Stimulus Chaining from different IP Stimulus
- #3: Developing system level patterns to enable broad users to configure system level tests

Covered in Accellera's PSS1.1 tutorial

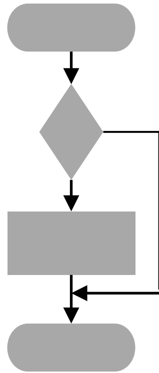
- #4: Guidelines for Portable² Stimulus
- #5: Handling complex I/O in your PSS model

This session

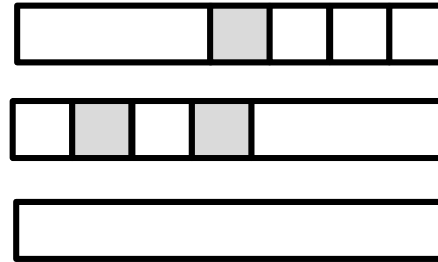
accellera SYSTEMS INITIATIVE

RECAP: RELEVANT PSS1.1 ENHANCEMENTS

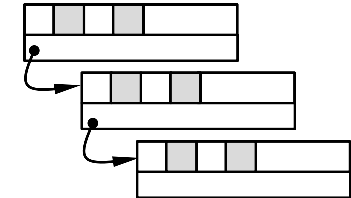
Relevant Enhancements in PSS1.1



Procedural constructs



Programmable Registers



Descriptors

Sweet spot: Capture device programming sequence

Procedural Constructs Introduction

```
function int glob_func() {
  // Define glob_func
}
component my_comp_c
{
  function void comp_func() {
    // Define comp_func
  }
  action act_a {
    exec pre_solve {
      // Expanded set of supported statements
    }
    exec body {
      // Expanded set of supported statements
    }
  };
};
```

my_model.pss

- PSS1.0
  - Foreign-language functions can be imported or specified with target-templates
  - Exec definitions are restricted
- PSS1.1 adds,
  - Support for a generic function definition
  - Many more procedural statements
    - Can be used in execs and function definition

Procedural Constructs Introduction

```
// Sum all elements of 'a' that are even, starting from a[0], except those
// that are equal to 42.  Stop summation if the value of an element is 0
function int sum(int a[100])
{
   int res;

   res = 0;

   foreach (el : a) {
      if (el == 0)
         break;
      if (el == 42)
         continue;
      if ((el % 2) == 0) {
         res = res + el;
      }
   }

   return res;
}
```

sum.pss

- Variables can be declared
- Conditional branches, Loops, Match statement supported
  - Similar to activity statement
- Break / Continue
- Return

- No randomization, constraints (algebraic, schedule)

DMAC: A Component Definition

```
buffer data_buff {
  addr_handle_t mem_seg;
};

component dma_c {
  resource channel_r {};
  pool [NUM_DMA_CHANNELS] channel_r chan_pool;
  bind chan_pool *;


  action mem2mem_xfer {
    input  data_buff src_buff;
    output data_buff dst_buff;


    addr_claim_s<> claim;
    constraint claim.size == 1024;
    lock channel_r chan;

    // contd...
```

```

    exec post_solve {
      dst_buff.mem_seg =
         make_handle_from_claim(claim);
    }

    exec body {
      comp.do_xfer(chan,
        src_buff.mem_seg,
        dst_buff.mem_seg,
        claim.size);
    }
  }; // action mem2mem_xfer

 function void do_xfer(int channel,
    addr_handle_t src,
    addr_handle_t dst,
    int length);

}; // component dma_c
```

dmac.pss

PL080: Component extensions

```
// Recommended: Extend or derive the component
// while using design-specific registers
extend component dma_c
{
  PL080_regs::Regs_c r;

  function void do_xfer(int channel,
      addr_handle_t src,
      addr_handle_t dst,
      int length) { /* Details in later slide */ }
};
```

pl080_c.pss

```
// Recommended: Keep register definitions
// in a separate file
package PL080_regs {
  struct INT_TC_CLR_s : packed<>
  {
     bit TC_CLR[8];
  };
  pure component INT_TC_CLR_c : reg_c<INT_TC_CLR_s, 32>
  { };

  // ... Etc for all registers

  pure component Regs_c : reg_group_c
  {
    INT_TC_CLR_c   INT_TC_CLR;
    INT_ERR_CLR_c  INT_ERR_CLR;
    SRC_ADDR_c     SRC_ADDR[8];
    DST_ADDR_c     DST_ADDR[8];
    // ...
  };
};
```

pl080_regs.pss

PL080: Setup Transfer

```
function void do_xfer(int channel, addr_handle_t src, addr_handle_t dst, int length)
{
  // Clear Interrupts
  comp.r.INT_TC_CLR.write_val(0xF);
  comp.r.INT_ERR_CLR.write_val(0xF);

  // Setup channel
  comp.r.SRC_ADDR[channel].write(src);
  comp.r.DST_ADDR[channel].write(dst);
  comp.r.LLI[channel].write_val(0);
  comp.r.CONTROL[channel].write_val(length);

  // Enable channel
  CONFIGURATION_s cfg;
  cfg.Enable = 1;
  comp.r.CONFIGURATION[channel].write(cfg);

  // Wait for completion
  INT_TC_STATUS_s sts;
  repeat {
    yield();
    sts = comp.r.INT_TC_STATUS.read();
  } while(sts.TC_STS[channel] == 0);
}
```

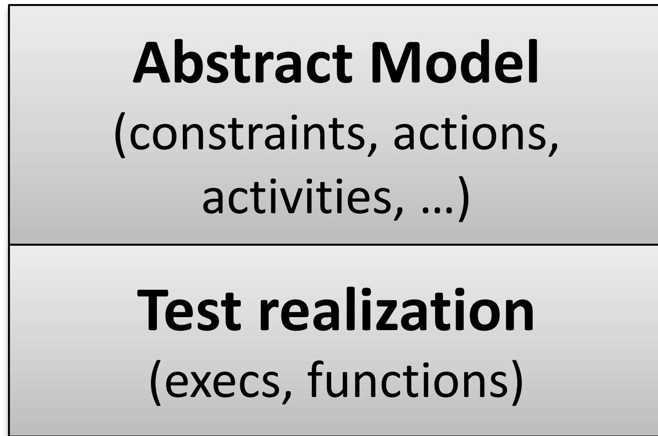
pl080_c.pss

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2020

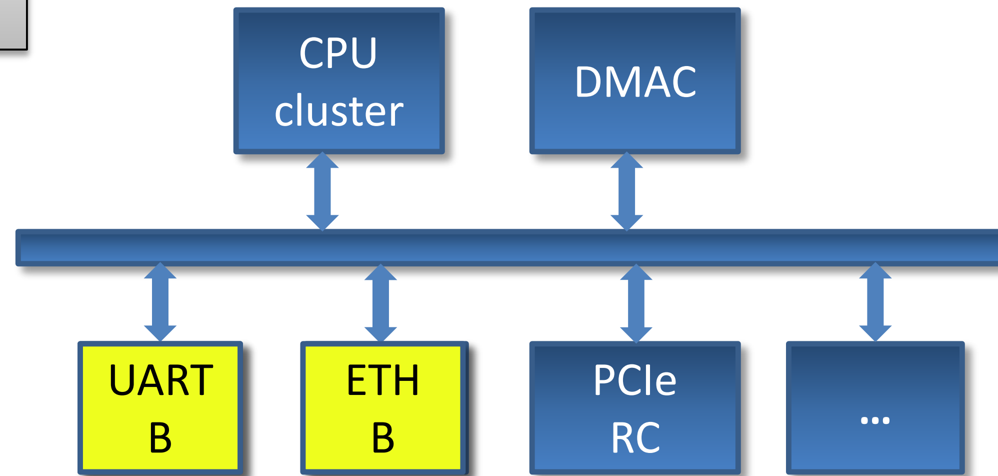
PORTABLE^2 STIMULUS FOR SYSTEM-LEVEL VERIFICATION

accellera SYSTEMS INITIATIVE

Portable² Stimulus for system-level verification

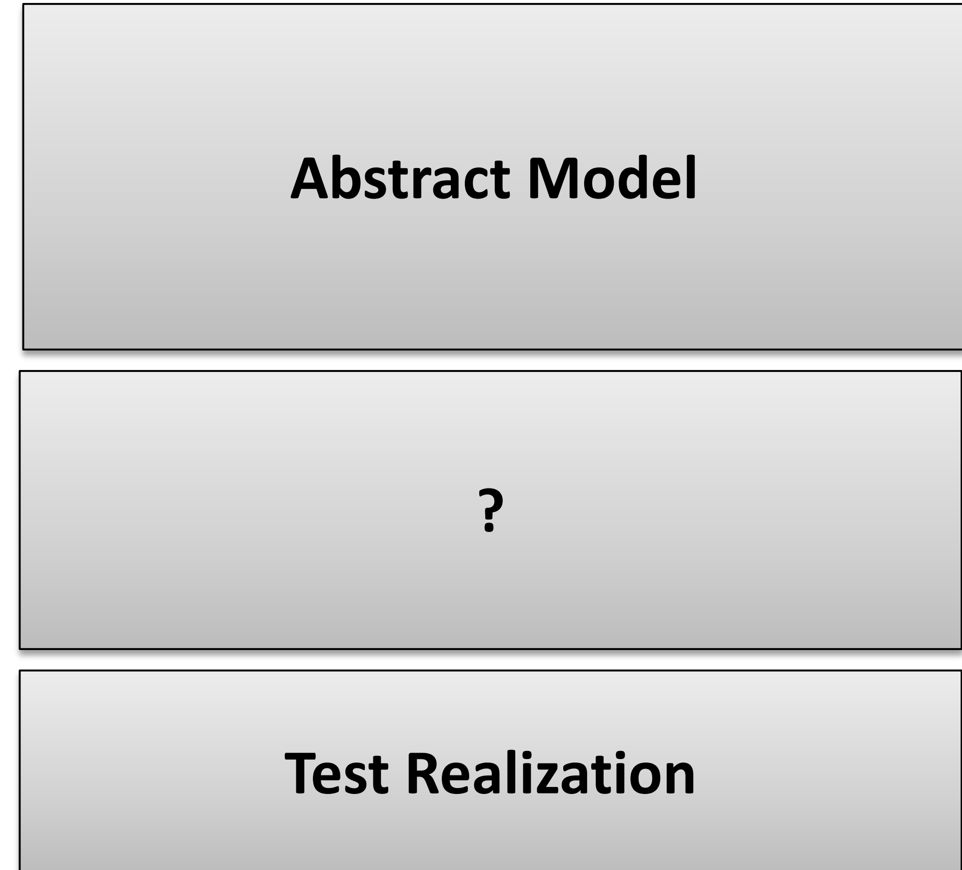


How to ensure Abstract Model is reusable across DUT mutations?



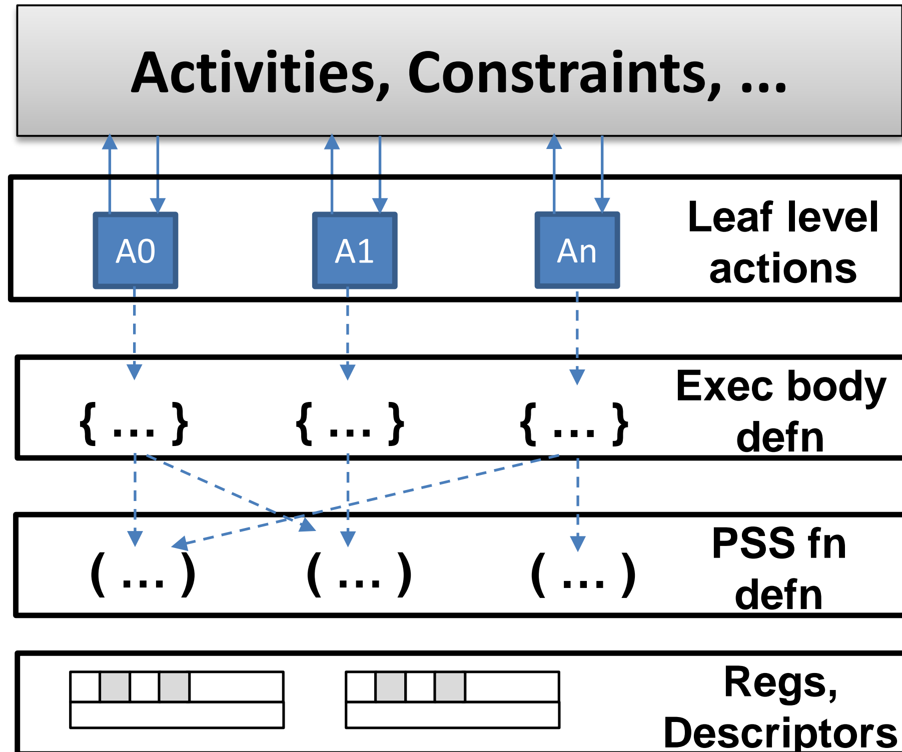
Inspirations from Software world

Android Software Stack



PSS Equivalents

Decisions to be made



- What should be the inputs/outputs of the leaf-level action?
- What goes in “exec body”?
- What should be abstracted in to a PSS function?
- What should be the API definition (function “signature”)?
- How should programmable registers be captured?

DESIGN AND VERIFICATION™ DVCON 2020 CONFERENCE AND EXHIBITION UNITED STATES

Guideline #1: Segregate Registers

- Traits
  - Register specification will be typically generated from IP-XACT / SystemRDL
- Recommendations
  - Keep register types in a separate file
  - Nest the types in a package / namespace
- Enables
  - Better round-trip engineering / configuration management

accellera SYSTEMS INITIATIVE

Guideline #2: Define PSS function API upfront

- Traits
  - Typically defined in one team/org and used in an other team/org
  - Programming sequence definition is valuable – multiple users

- Recommendations
  - Standardize the API for a class of devices (e.g., an API for Ethernet controllers)
    - Not a trivial exercise!
  - Keep the abstraction at a “device driver” level
    - E.g: Initialization, get HW capabilities, configuration, device operations (transmit / receive)
  - Keep the definitions in a separate file (separated from abstract model)

- Enables
  - Better co-ordination between Architect (defines API), IP team (implements API) and verification team (uses API)
  - Critical for ensuring abstract model reuse

Guideline #3: Keep exec body simple

- Traits
  - Code written in exec body is not amenable to reuse (in other exec or a function)
- Recommendations
  - Keep exec body simple – typically just a function call
  - Keep a straight-forward mapping between function input/outputs to action's input/output
  - Sweet spot for exec definition – act as a thin "glue" layer
- Enables
  - Better discipline in enforcing API, better reuse

HANDS ON SESSION: A SIMPLE UART EXAMPLE

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2020

HANDLING COMPLEX I/O IN YOUR PSS MODEL

accellera SYSTEMS INITIATIVE

HANDS ON SESSION: A PCIE CONTROLLER

Q&A? FEEDBACK?