

Learning the Portable Stimulus Standard

Tom Fitzpatrick

CEO Big Fish EDA Consulting

On behalf of the Accellera Systems Initiative

and the Accellera Portable Stimulus Working Group



Organizing PSS Code: Packages, Inheritance, & Extension

▶▶▶ Session 08



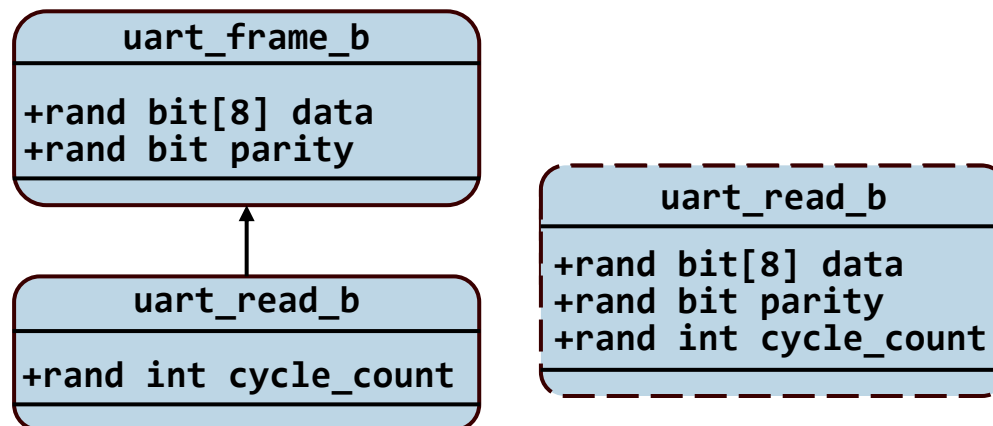
... What You Will Learn

- ▶ Object-Oriented Inheritance in PSS
- ▶ Type extension in PSS
- ▶ PSS packages
- ▶ Organizing PSS code for reuse

::: Inheritance in PSS

- ▶ Object-Oriented Inheritance lets a **new** type be based on an existing type
- ▶ Supports specialization without duplicating the original definition

Supports specialization without duplicating the original definition



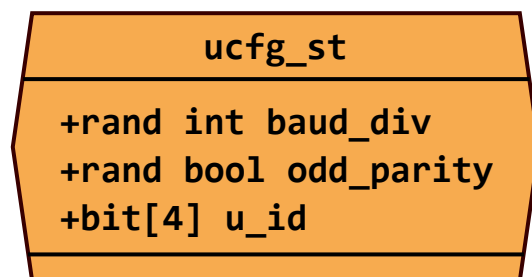
```
buffer uart_frame_b {
    rand bit[8] data;
    rand bit parity;
}
```

```
buffer uart_read_b : uart_frame_b
    rand int cycle_count;
}
```

::: Type Extension in PSS

- ▶ Type extension **adds to** an existing type
- ▶ Supports project-specific customization without modifying the original source

Customization **without**
declaring a new type



```
state ucfg_st {  
  rand int[16] in [4..256] baud_div;  
  rand bool odd_parity;  
}  
buffer uart_frame_b {  
  rand bit[8] data;  
  rand bit parity;  
}
```

```
extend state ucfg_st {  
  bit[4] u_id;  
}
```

```
buffer uart_read_b : uart_frame_b {  
  rand int cycle_count;  
}
```

⋮ PSS Packages

- ▶ Group related PSS declarations into a reusable unit
- ▶ Help organize structs, enums, and other type definitions
- ▶ Packages are also namespaces
pkg_name::type_name

```
package uart_data_pkg {  
  state ucfg_st {  
    rand int[16] in [4..256] baud_div;  
    rand bool odd_parity;  
  }  
  buffer uart_frame_b {  
    rand bit[8] data;  
    rand bit parity;  
  }  
}  
  
package uart_ext_pkg {  
  import uart_data_pkg::*;  
  extend state ucfg_st {  
    bit[4] u_id;  
  }  
}
```

```
buffer uart_read_b : uart_frame_b {  
  rand int cycle_count;  
}
```

⋮ PSS Packages

- ▶ Group related PSS declarations into a reusable unit
- ▶ Help organize actions, structs, enums, and other type definitions
- ▶ Packages are also namespaces
pkg_name::type_name
- ▶ Reduce naming conflicts in larger envs
- ▶ Make it easier to share verification and scenario content across projects
- ▶ Typical packages
 - ▶ Data types, base types
 - ▶ Type extensions

```
package uart_data_pkg {  
  state ucfg_st {  
    rand int[16] in [4..256] baud_div;  
    rand bool odd_parity;  
  }  
  buffer uart_frame_b {  
    rand bit[8] data;  
    rand bit parity;  
  }  
}  
  
package uart_ext_pkg {  
  import uart_data_pkg::*;  
  extend state ucfg_st {  
    bit[4] u_id;  
  }  
}  
  
component uart c {  
  import uart_data_pkg::*;  
  import uart_ext_pkg::*;  
  
  buffer uart_read_b : uart_frame_b {  
    rand int cycle_count;  
  }  
  
  action cfg_uart a {  
    output ucfg_st ocfg;  
    constraint ocfg.u_id == comp.u_id;  
  }  
}
```

⋮ PSS Packages

- ▶ Group related PSS declarations into a reusable unit
- ▶ Help organize actions, structs, enums, and other type definitions
- ▶ Packages are also namespaces
pkg_name::type_name
- ▶ Reduce naming conflicts in larger envs
- ▶ Make it easier to share verification and scenario content across projects
- ▶ Typical packages
 - ▶ Data types, base types
 - ▶ Type extensions

Package imports
are not transitive

```
package uart_data_pkg {  
  state ucfg_st {  
    rand int[16] in [4..256] baud_div;  
    rand bool odd_parity;  
  }  
  buffer uart_frame_b {  
    rand bit[8] data;  
    rand bit parity;  
  }  
}
```

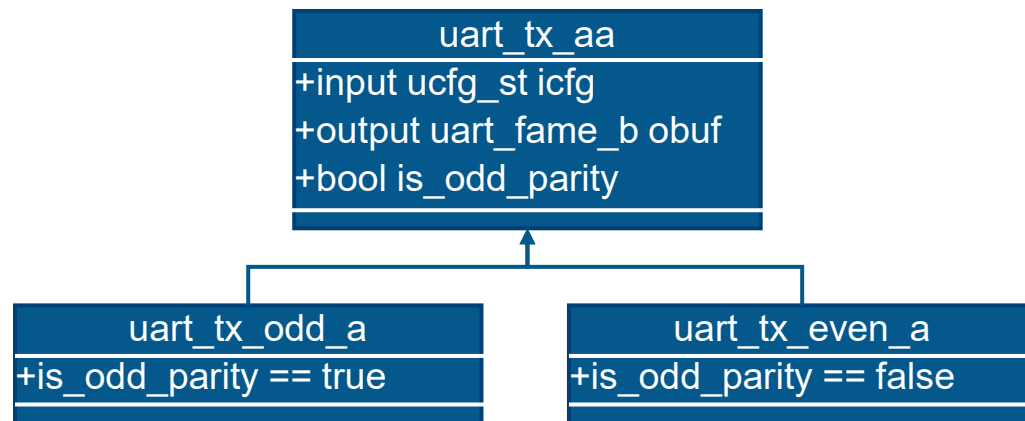
```
package uart_ext_pkg {  
  import uart_data_pkg::*;  
  extend state ucfg_st {  
    bit[4] u_id;  
  }  
}
```

```
component uart_c {  
  import uart_data_pkg::*;  
  import uart_ext_pkg::*;  
  
  buffer uart_read_b : uart_frame_b {  
    rand int cycle_count;  
  }  
  
  action cfg_uart_a {  
    output ucfg_st ocfg;  
    constraint ocfg.u_id == comp.u_id;  
  }  
}
```

Must import
both packages

⋮ Packages and Actions

- ▶ **Actions** may NOT be declared in packages
- ▶ **Abstract actions** serve as a base type
 - ▶ Abstract actions can't be instantiated directly
- ▶ Actions in components can be derived from abstract actions
 - ▶ If you **extend** an abstract action, it's still abstract



```
package uart_data_pkg {
    abstract action uart_tx_aa {
        input ucfg_st icfg;
        output uart_frame_b obuf;
        bool is_odd_parity;
    }
}
```

```
component uart_c {
    import uart_data_pkg::*;
    import uart_ext_pkg::*;

    action uart_tx_odd_a : uart_tx_aa {
        constraint is_odd_parity == true;
    }

    action uart_tx_even_a : uart_tx_aa {
        constraint is_odd_parity == false;
    }
}
```

Organizing PSS Code for Reuse

- ▶ **Packages** are namespaces to *organize* PSS content into reusable libraries
- ▶ **Inheritance** builds *specialized* types from shared base definitions
 - ▶ **Creates** a new type
- ▶ **Type extension** *customizes* existing types
 - ▶ **Modifies** an **existing** type



Thank You

