

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



Actions and Activities

 Session 05

What You Will Learn

- ▶ How to encapsulate behavior with a PSS *action*
- ▶ How to schedule behaviors with an *activity* in a PSS action
- ▶ Defining control flow in an activity

Atomic Actions

- Define a behavior to be mapped to a specific target implementation

```
action_declaration ::= action action_identifier [ template_param_decl_list ] [ : type_identifier ]  
                    { { action_body_item } }
```

```
action cfg_uart_a {  
  rand int[16] baud_div;  
  rand bit[2] parity_mode;  
}
```

cfg_uart

```
action tx_uart_a <type T=bit[8] {  
  rand T tx_byte;  
}
```

tx_uart

```
action rx_uart_a {  
  rand bit[8] rx_byte;  
  
  exec body {  
    // target implementation  
  }  
}
```

rx_uart

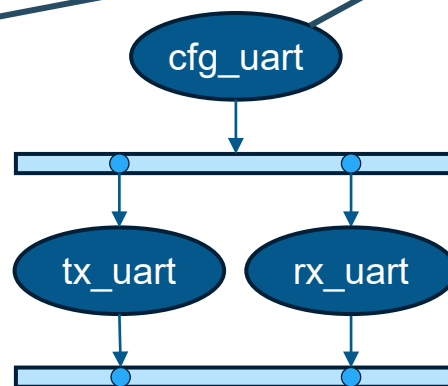
Compound Actions: Activities

- The activity defines the schedule of sub-actions to model critical intent

```
activity_declaration ::= activity { { activity_stmt } }  
activity_sequence_block_stmt ::= [ sequence ] { { activity_stmt } }  
activity_parallel_stmt ::= parallel { { activity_stmt } }
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  rx_uart_a rx;  
}
```

```
activity {  
  sequence {  
    cfg;  
    parallel {  
      tx;  
      rx;  
    }  
  }  
}
```



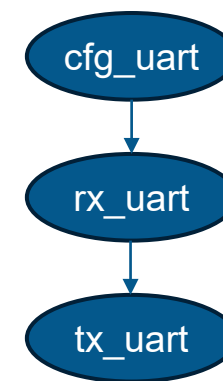
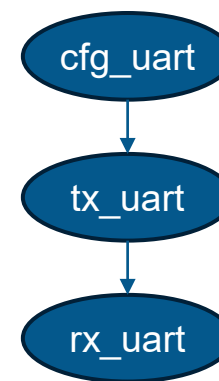
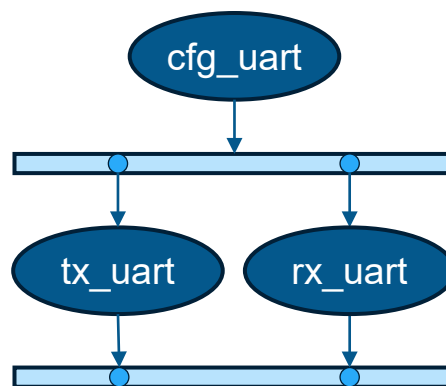
An *action traversal* statement defines the point where the action will be randomized and evaluated

Activities: schedule

- ▶ Traverse sub-actions in any legal order

```
activity_declaration ::= activity { { activity_stmt } }  
activity_schedule_stmt ::= schedule { { activity_stmt } }
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  rx_uart_a rx;  
  
  activity {  
    cfg;  
    schedule {  
      tx;  
      rx;  
    }  
  }  
}
```



Activities: select

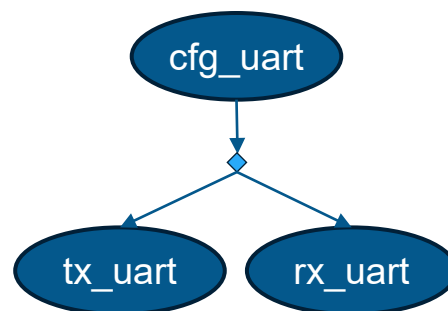
- Randomly choose a sub-action to traverse

```
activity_declaration ::= activity { { activity_stmt } }
```

```
activity_select_stmt ::= select { select_branch select_branch { select_branch } }
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  rx_uart_a rx;  
}
```

```
activity {  
  cfg;  
  select {  
    tx;  
    rx;  
  }  
}
```



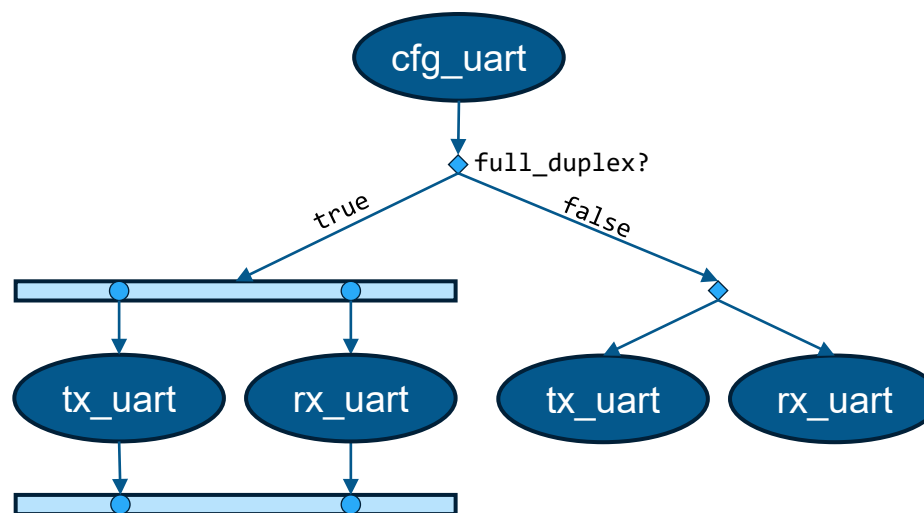
Activities: if-else

- Choose a sub-action to traverse based on a [random] field value

activity_declaration ::= **activity** { { activity_stmt } }

activity_if_else_stmt ::= **if** (expression) activity_stmt [**else** activity_stmt]

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  rx_uart_a rx;  
  rand bool full_duplex;  
  activity {  
    cfg;  
    if (full_duplex) {  
      parallel {  
        tx;  
        rx;  
      }  
    } else {  
      select {  
        tx;  
        rx;  
      }  
    }  
  }  
}
```



Activities: match

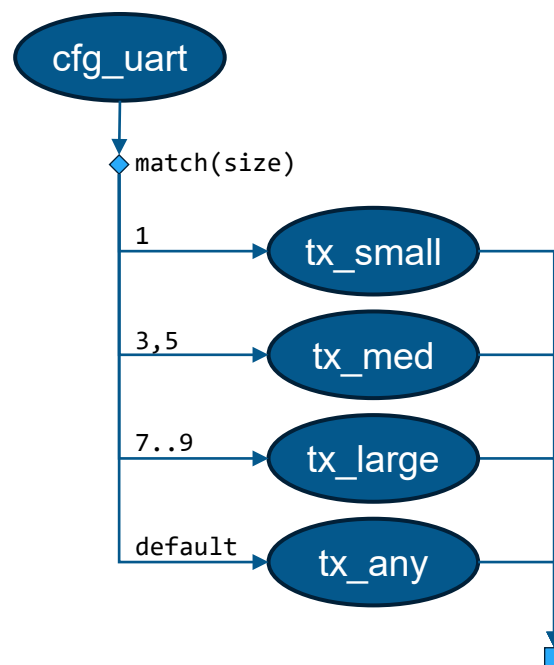
- Choose one of several sub-actions to traverse based on a field value

```
activity_declaration ::= activity { { activity_stmt } }
```

```
activity_match_stmt ::= match ( expression ) { match_choice { match_choice } }
```

```
match_choice ::= [ open_range_list ] : activity_stmt | default : activity_stmt
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx_any;  
  tx_small_a tx_small;  
  tx_med_a tx_med;  
  tx_large_a tx_large;  
  
  rand int in [1..10] size;  
  
  activity {  
    cfg;  
    match (size) {  
      [1]: { tx_small; }  
      [3,5]: { tx_med; }  
      [7..9]: { tx_large; }  
      default: { tx_any; }  
    }  
  }  
}
```



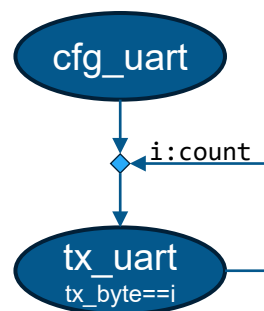
Activities: repeat

- Specify a loop of sub-action traversals

activity_declaration ::= **activity** { { activity_stmt } }

activity_repeat_stmt ::= **repeat** ([index_identifier :] expression) activity_stmt

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  
  rand bit[8] count;  
  
  activity {  
    cfg;  
    repeat (i:count) {  
      tx with {tx_byte == i};  
    }  
  }  
}
```



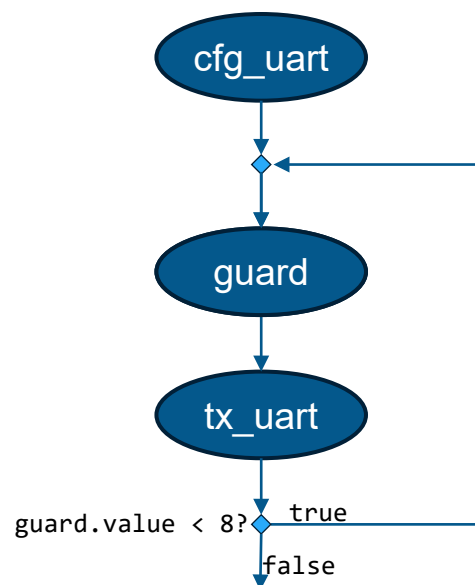
Activities: repeat-while

- ▶ Loop through sub-action traversals as long as condition is true

```
activity_declaration ::= activity { { activity_stmt } }  
activity_repeat_stmt ::= repeat ( [ index_identifier : ] expression ) activity_stmt  
                        | repeat activity_stmt while ( expression ) ;
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  guard_a guard;  
  rand bit[8] count;  
  
  activity {  
    cfg;  
    repeat {  
      guard;  
      tx;  
    } while (guard.value < 8);  
  }  
}
```

```
action guard_a {  
  rand bit[4] value;  
}
```

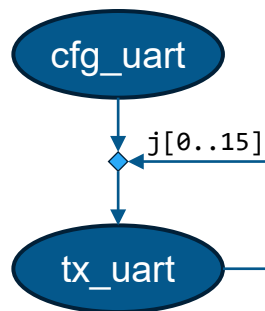


Activities: foreach

- Iterate traversals over the elements of a collection

```
activity_declaration ::= activity { { activity_stmt } }  
activity_foreach_stmt ::= foreach ( [ iterator_identifier : ] expression [ [ index_identifier ] ] ) activity_stmt
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;  
  
  rand bit[8] count[16];  
  
  activity {  
    cfg;  
    foreach (i:count[j]) {  
      tx with {tx_byte == count[j]};  
    }  
  }  
}
```



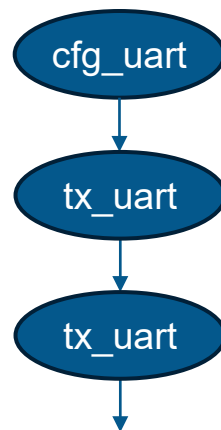
Activities: replicate

► Generative in-place expansion of sub-action traversals

```
activity_declaration ::= activity { { activity_stmt } }  
activity_replicate_stmt ::= replicate ( [ index_identifier : ] expression ) [ label_identifier [] : ]  
                                labeled_activity_stmt
```

```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;
```

```
  activity {  
    cfg;  
    sequence {  
      replicate (i:2) {  
        tx with {tx_byte == i};  
      }  
    }  
  }  
}
```



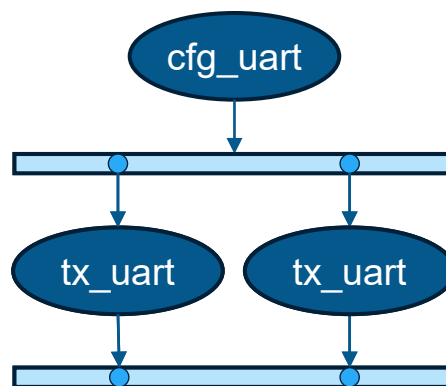
Activities: replicate

- Generative in-place expansion of sub-action traversals

```
activity_declaration ::= activity { { activity_stmt } }  
activity_replicate_stmt ::= replicate ( [ index_identifier : ] expression ) [ label_identifier [] : ]  
                             labeled_activity_stmt
```

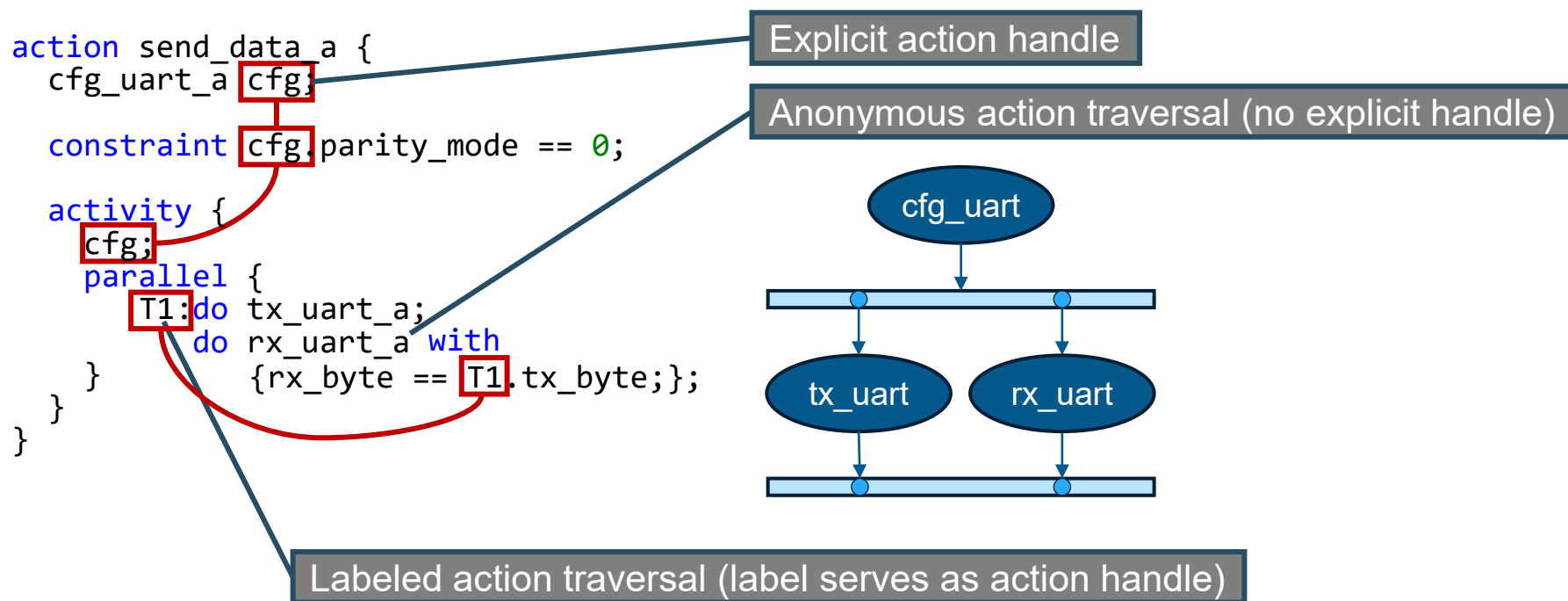
```
action send_data_a {  
  cfg_uart_a cfg;  
  tx_uart_a tx;
```

```
  activity {  
    cfg;  
    parallel {  
      replicate (i:2) {  
        tx with {tx_byte == i;};  
      }  
    }  
  }  
}
```

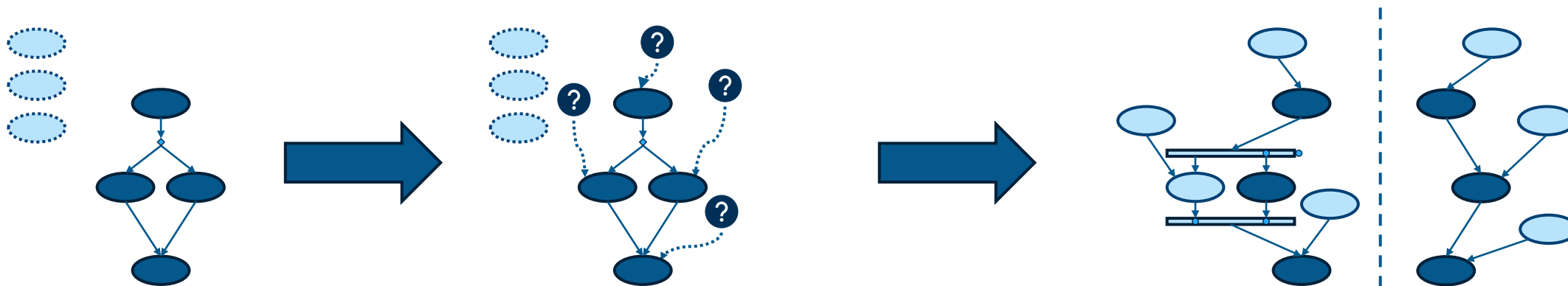
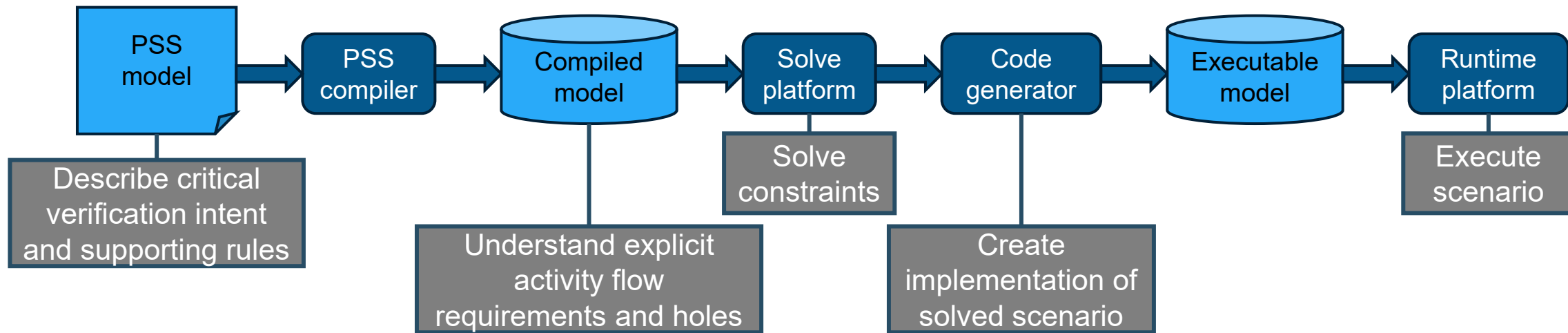


Action Handles

- ▶ Activities refer to actions as *action handles*
- ▶ All action handles reset to uninitialized upon entry to the activity



::: PSS Models are Declarative





Thank You

