

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



Constraints

 Session 09

What You Will Learn

- ▶ Algebraic Constraints
- ▶ Conditional Constraints
- ▶ Group Constraints
- ▶ Default Constraints
- ▶ Scheduling Constraints

::: PSS Constraints

- ▶ PSS scenarios are determined by solving constraints
 - ▶ Algebraic constraints on **rand action** and **struct (buffer/struct/state/resource)** fields
 - ▶ Scheduling constraints between actions
 - ▶ Defined by data flow object semantics
 - ▶ Explicit **activity** scheduling constraints (**sequence/parallel**)
- ▶ PSS constraints are composable
 - ▶ All constraints on a particular field must be satisfied
 - ▶ Constraints shared between objects/producers/consumers
- ▶ Constraints are inherited
 - ▶ Base object constraints persist to derived objects
 - ▶ Named constraints can be overridden by derived objects

Algebraic Constraint Blocks

- Define relationships between scenario attributes

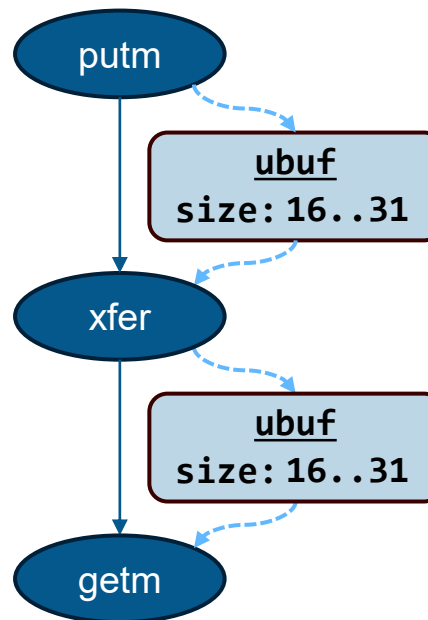
```
constraint_declaration ::= constraint constraint_set  
                        | constraint identifier { { constraint_body_item } }
```

```
action putm_a {  
  output xbuf b obuf;  
  constraint obuf.size in [8..127];  
}
```

```
action xfer_a {  
  input  xbuf b ibuf;  
  output xbuf b obuf;  
  constraint io_c {  
    ibuf.size == obuf.size;  
    ibuf.parity == obuf.parity;  
  }  
}
```

```
action getm_a {  
  input xbuf b ibuf;  
  constraint ibuf.size < 64;  
}
```

All constraints
must hold



```
action entry_a {  
  dma_c::getm_a getm;  
  dma_c::putm_a putm;  
  dma_c::xfer_a xfer;  
  
  constraint getm.ibuf.size > 15;  
  
  activity {  
    putm with {obuf.size < 32;;}  
    xfer;  
    getm;  
  }  
}
```

Optional
name

Put multiple
constraints in {}

Logical
Expression
Constraints

Implication Constraint

- Specifies a conditional constraint

`implication_constraint_item ::= expression -> constraint_set`

- Implication constraint is *bidirectional*

```
action putm_a {  
  output xbuf_b obuf;  
  
  constraint (obuf.parity == 1) -> obuf.size in [64..127];  
}
```

`!(obuf.parity == 1) || (obuf.size in [64..127])`

A	B	A -> B
T	T	T
T	F	F
F	T	T
F	F	T

`!A || B`

obuf.parity	obuf.size
0	0..127
1	64..127

if-else Constraint

- Specifies a more complex conditional constraint

`if_constraint_item ::= if ( expression ) constraint_set [ else constraint_set ]`

- **if-else** constraint is also *bidirectional*

```
action putm_a {  
  output xbuf_b obuf;
```

```
  constraint size_range_c {  
    if (obuf.parity == 1)  
      obuf.size in [64..127];  
    else  
      obuf.size in [0..63];  
  }  
}
```

`!(obuf.parity == 1) || (obuf.size in [64..127])`

`!(obuf.parity == 0) || (obuf.size in [0..63])`

if-only

obuf.parity	obuf.size
0	0..127
1	64..127

if-else

obuf.parity	obuf.size
0	0..63
1	64..127

foreach Constraint

- ▶ Apply a constraint to each member of a collection

`foreach_constraint ::= foreach ( [ iterator_identifier : ] expression [ [ index_identifier ] ] ) constraint_set`

```
action putm_a {  
  output xbuf_b obuf;  
  
  constraint foreach_c {  
    foreach (j: obuf.data[i]) {  
      j == 0xA5 + i;  
    }  
  }  
}
```

forall Constraint

- Apply a constraint to all instances of a specific type in a given **activity** scope

`forall_constraint_item ::= forall ( iterator_identifier : type_identifier [ in ref_path ] ) constraint_set`

```
action entry2_a {
```

```
  constraint forall (g_id : dma_c::getm_a) {  
    g_id.ibuf.size > 7;  
  }
```

Constraint set applies to all instances of dma_c::getm_a

```
  constraint forall (p_id : dma_c::putm_a) {  
    p_id.obuf.size < 32;  
  }
```

Also applies to any inferred instances of the specified type

```
  activity {  
    replicate(8) {  
      do dma_c::getm_a;  
    }  
  }  
}
```

::: unique Constraint

- Chooses unique values for each member of the set

```
unique_constraint_item ::= unique { hierarchical_id { , hierarchical_id } } ;
```

```
action putm_a {  
  output xbuf_b obuf0, obuf1;  
  
  constraint unique_c {  
    unique {obuf0.data.size, obuf1.data.size};  
  }  
}
```

PSS3.1 will allow specification of an array or array slice in a **unique** constraint

::: default Constraint

- Specifies a default value unless the constraint is disabled

`default_constraint ::= default hierarchical_id == constant_expression ;`

`default_disable ::= default disable hierarchical_id ;`

Only equality to a constant expression allowed

```
struct my_struct {  
  rand int in [0..3] attr1;  
  rand int in [0..3] attr2;  
  constraint default attr1 == 0;  
  constraint c1 {attr1 < attr2;}  
};
```

```
action my_action {  
  rand my_struct s1;  
  rand my_struct s2;  
  rand my_struct s3;  
  constraint default s2.attr1 == 2;  
  constraint default disable s3.attr1;  
  constraint s3.attr1 > 0;  
}
```

Higher-level default overrides lower-level

my_struct(s1)
attr1:0
attr2:1..3

my_struct(s2)
attr1:2
attr2:3

my_struct(s3)
attr1:1..2
attr2:2..3

Constraint c1 holds

Constraint c1 holds

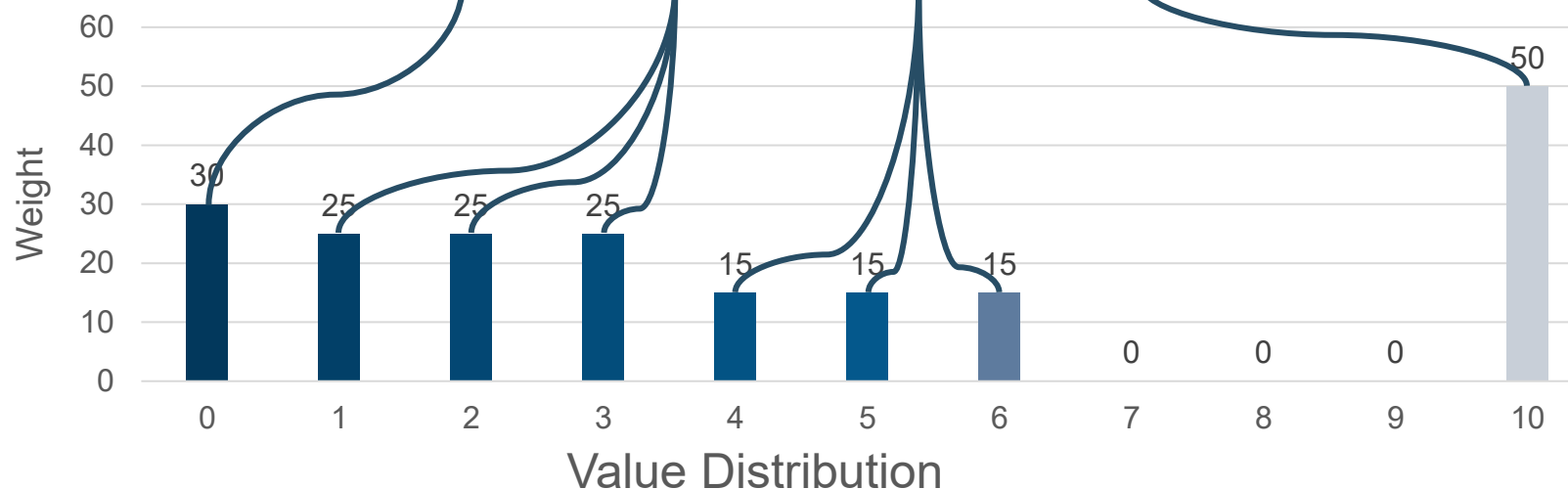
Disabled constraint no longer holds

Distribution Constraint

- Provides a value-distribution specification for a given expression

```
dist_directive ::= dist expression in [ dist_item {, dist_item} ] ;  
dist_item ::= open_range_value [ := expression | :/ expression ]
```

```
buffer xbuf_b {  
  rand bit[8] size;  
  rand bit parity;  
  rand bit[16] id;  
}  
action getm_a {  
  input xbuf_b ibuf;  
  constraint ibuf.size < 64;  
  constraint dist ibuf.id in [0 := 30, 1..3 := 25, 4..6 :/ 45, 10 := 50];  
}
```



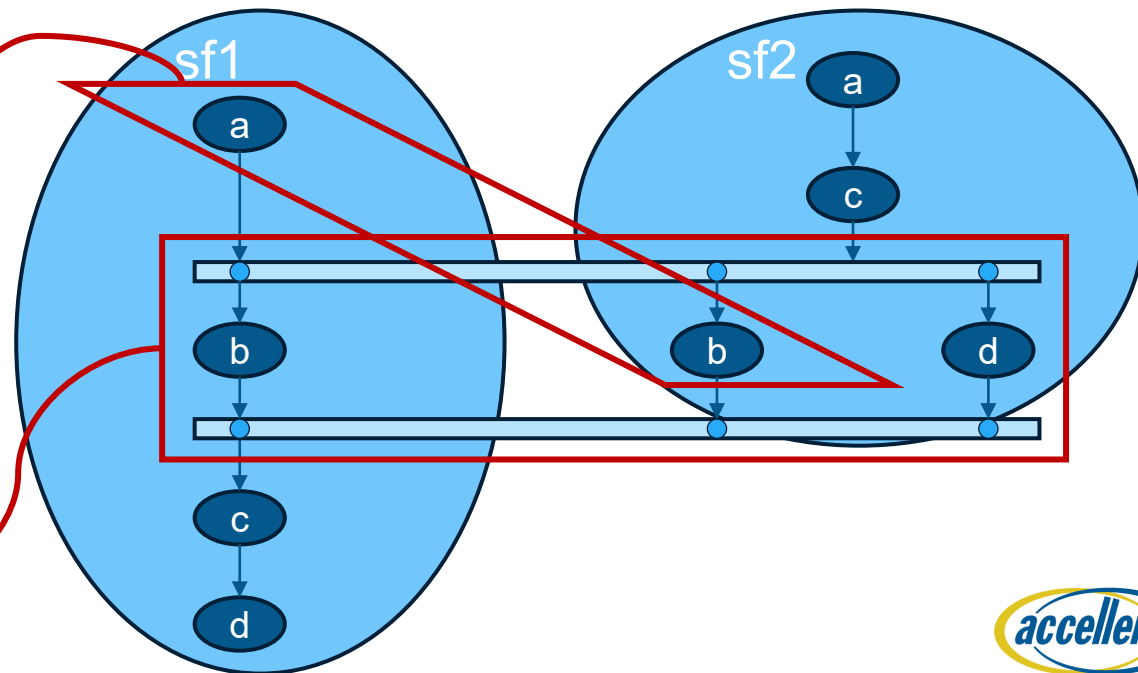
⋮ Scheduling Constraints

- Specify scheduling relationship between **scheduled** action traversals

```
activity_scheduling_constraint ::= constraint ( parallel | sequence )  
                                { hierarchical_id , hierarchical_id { , hierarchical_id } } } ;
```

Scheduling constraints apply to two or more action handles

```
action my_sub_flow {  
  A a; B b; C c; D d;  
  activity {  
    sequence {  
      a;  
      schedule {  
        b; c; d;  
      };  
    };  
  };  
};  
action my_top_flow {  
  my_sub_flow sf1, sf2;  
  activity {  
    schedule {  
      sf1;  
      sf2;  
    };  
  };  
};  
constraint sequence {sf1.a, sf2.b};  
constraint parallel {sf1.b, sf2.b, sf2.d};  
};
```





Thank You

