

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



Data Coverage

 Session 15

What You Will Learn

- ▶ Modeling data coverage with **covergroups**
- ▶ Specifying **coverpoints** and **cross** coverage
- ▶ Coverage sampling
- ▶ Coverage options

Modeling Data Coverage

- ▶ The coverage model is defined using **covergroups**
- ▶ A **coverpoint** defines the expression and **bins** to track values

```
component dma_c {
```

```
  action xfer_a {
```

```
    covergroup {
```

```
      size_cp : coverpoint ibuf.size {
```

```
        bins tiny = [1..7];
```

```
        bins small[] = [8..32] with ((size_cp % 4) == 0);
```

```
        bins medium[8] = [33..128];
```

```
        ignore_bins ignore_vals = [129..160];
```

```
        bins large = [197..250];
```

```
        illegal_bins illegal_vals = [251..255];
```

```
        bins others = default;
```

```
      }
```

```
    } cov_size_inline;
```

```
  }
```

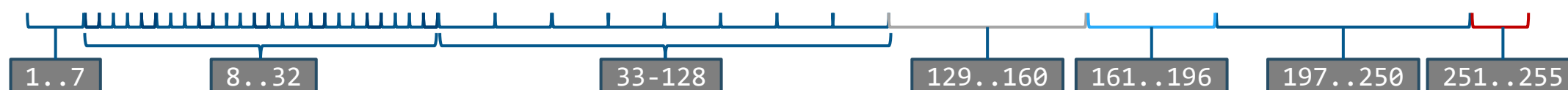
```
}
```

Single bin for all values in range

One bin per value

that match the with expression

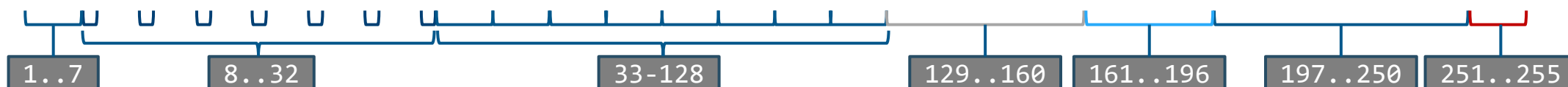
Divide range into N bins



Modeling Data Coverage

- ▶ A **covergroup** may also be a user-defined type with formal parameters

```
component dma_c {  
  covergroup cov_size_cg(bit[8] size) {  
    size_cp : coverpoint size {  
      bins tiny = [1..7];  
      bins small[] = [8..32] with ((size_cp % 4) == 0);  
      bins medium[8] = [33..128];  
      ignore_bins ignore_vals = [129..160];  
      bins large = [161..250];  
      illegal_bins illegal_vals = [251..255];  
      bins others = default;  
    }  
  }  
  action xfer_a {  
    cov_size_cg size_cov(ibuf.size);  
  }  
}
```

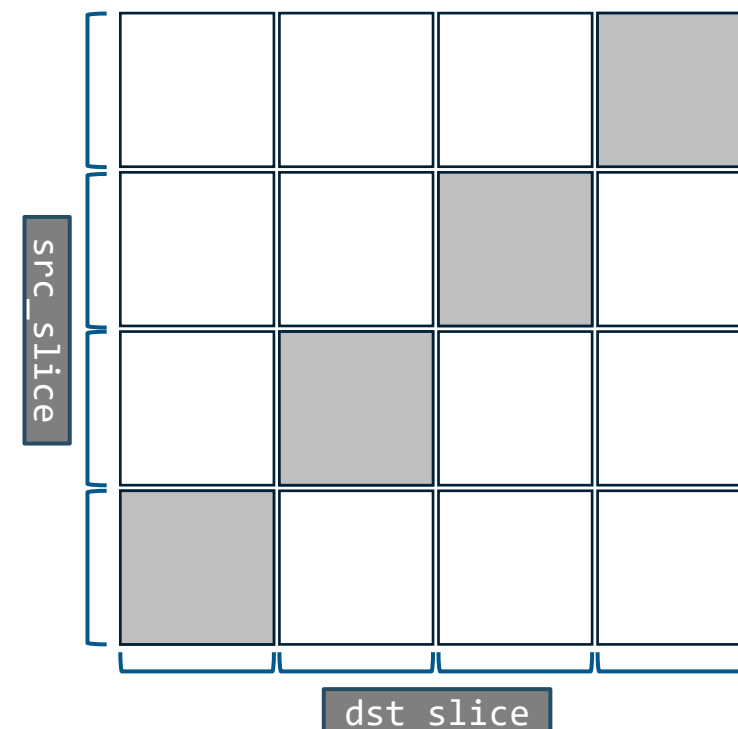


Cross Coverage

- ▶ A **cross** defines an N-way combination of bins from previous coverpoints
- ▶ Use **with** expression to group cross products into **bins**

```
component dma_c {
  covergroup cov_addr_cg(MemSliceID src_slice, MemSliceID dst_slice) {
    src_slice_cp : coverpoint src_slice;
    dst_slice_cp : coverpoint dst_slice;
    slice_x : cross src_slice_cp, dst_slice_cp {
      ignore_bins [] = ignore_vals with (src_slice_cp == dst_slice_cp);
    }
  }
}

enum MemSliceID { Slice0, Slice1, Slice2, Slice3 }
action xfer_a {
  exec body {
    src_addr = addr_value(src_hndl);
    dst_addr = addr_value(dst_hndl);
    ibuf.slice = (MemSliceID)src_addr[17:16];
    obuf.slice = (MemSliceID)dst_addr[17:16];
  }
  cov_addr_cg slice_cov(ibuf.slice, obuf.slice);
}
}
```

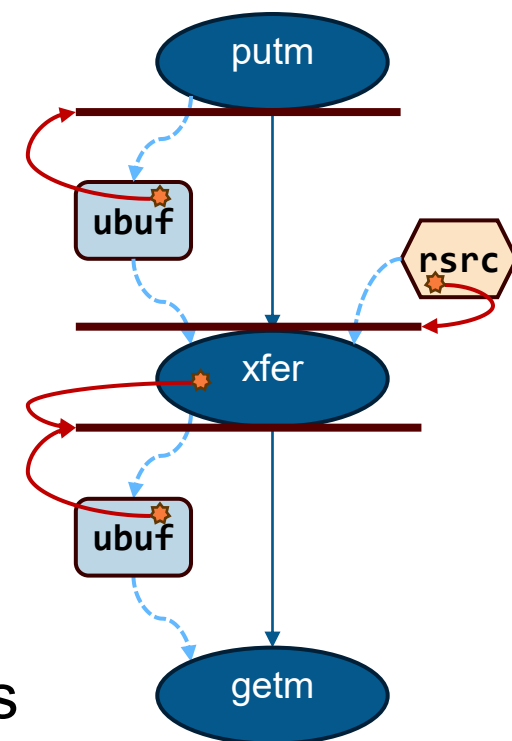


Coverage Sampling

- ▶ A **covergroup** is *sampled* during execution depending on the context
- ▶ Sampled only when the **iff** expression (if present) is true.

```
component dma_c {  
  covergroup cov_size_cg(bit[8] size, bit[32] src_addr, bit[32] dst_addr) {  
    size_cp : coverpoint size iff (size > 0) {  
      ...  
    }  
  }  
}
```

- ▶ Instantiated in **action**: sampled when action completes
- ▶ In data flow object: sampled when outputting action completes
- ▶ In **resource**: sampled before the first referencing action begins
- ▶ In **struct**: sampled in the instantiating context



Coverage Options

- ▶ A **covergroup** may include options to control behavior

```
component dma_c {  
  covergroup cov_addr_cg(MemSliceID src_slice, MemSliceID dst_slice) {  
    option.comment = "Tracking src/dst slice coverage";  
    src_slice_cp : coverpoint src_slice {  
      option.auto_bin_max = 4;  
    };  
    dst_slice_cp : coverpoint dst_slice {  
      option.weight = 2;  
    };  
    xslice_cross : cross src_slice_cp, dst_slice_cp {  
      ignore_bins ignore_vals = xslice_cross with (src_slice_cp != dst_slice_cp);  
    }  
  }  
}
```

Descriptive comment

Coverpoint-specific:Maximum # auto-created bins

Coverpoint-specific:Relative weight for coverage calculation



Thank You

