

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



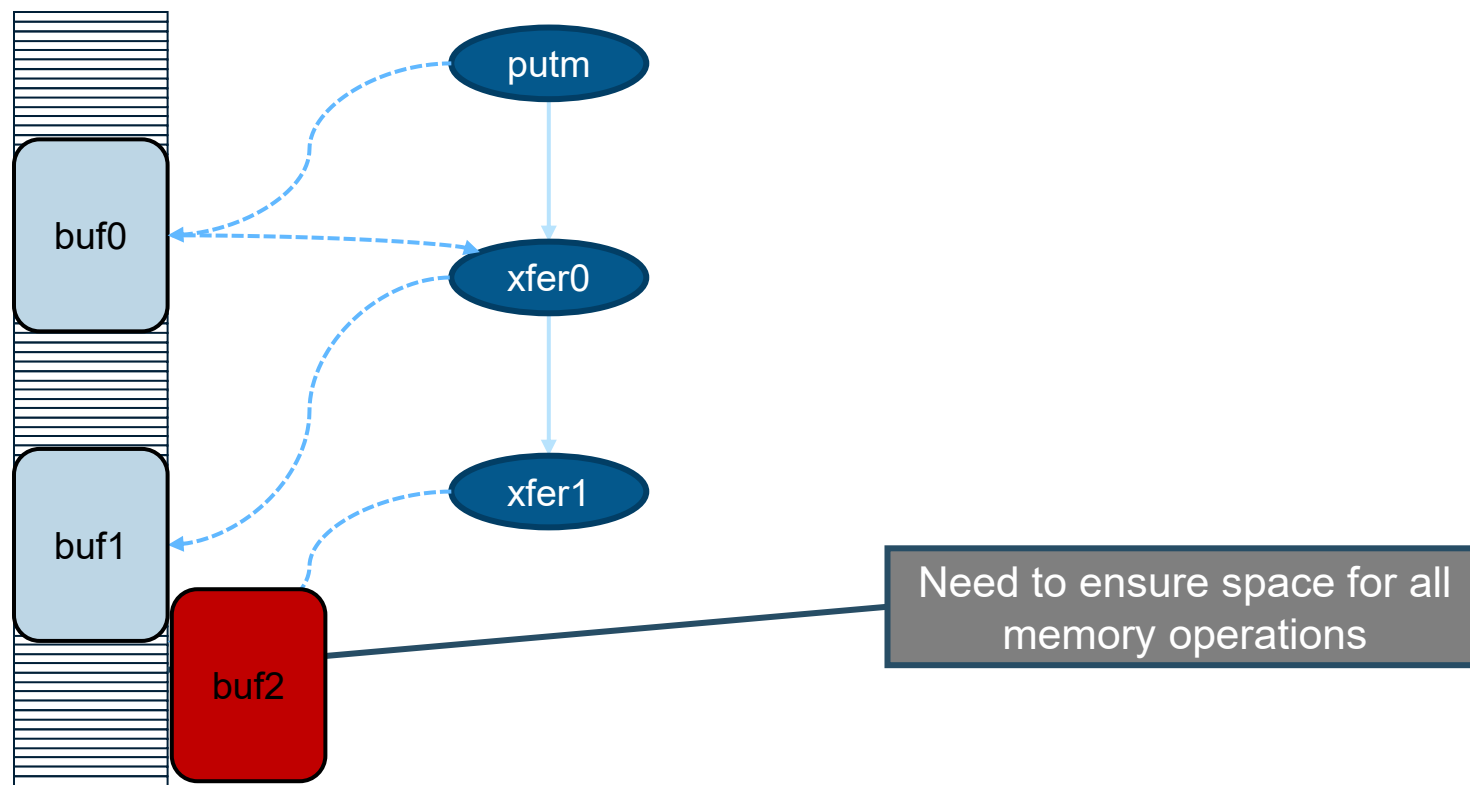
Memory Allocation in PSS

▶▶▶ Session 11

What You Will Learn

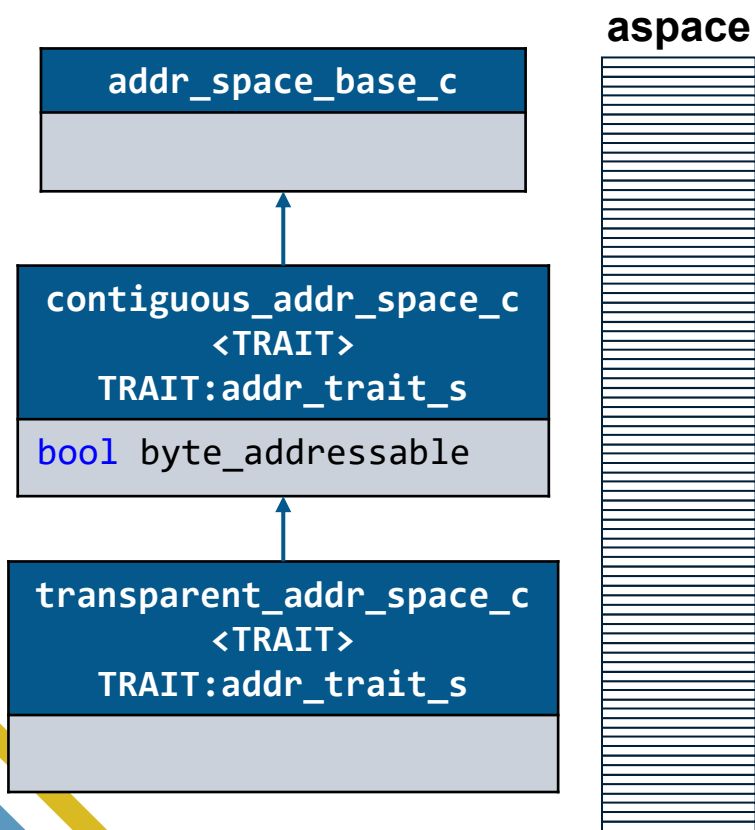
- ▶ The importance of solve-time memory allocation
- ▶ Defining address spaces
- ▶ Address regions
- ▶ Address claims

Memory Allocation at Solve-Time



Address Spaces

- ▶ Model memory and other storage accessible via unique addresses
- ▶ `addr_space` component parameterized by an `addr_trait` struct type



```
package addr_reg_pkg {  
  struct addr_trait_s {};  
  struct empty_addr_trait_s : addr_trait_s {};  
  
  component contiguous_addr_space_c  
    <struct TRAIT : addr_trait_s = empty_addr_trait_s> {  
      bool byte_addressable = true;  
    };  
  
  component transparent_addr_space_c  
    <struct TRAIT: addr_trait_s=empty_addr_trait_s>  
    : contiguous_addr_space_c<TRAIT> {};  
}  
  
component pss_top {  
  import addr_reg_pkg::*;  
  import my_addr_pkg::*;  
  
  transparent_addr_space_c<MemRegionTrait_s> aspace;  
}
```

addr TRAIT base type

default TRAIT type

TRAIT must be derived
from `addr_trait_s`

Default to empty trait

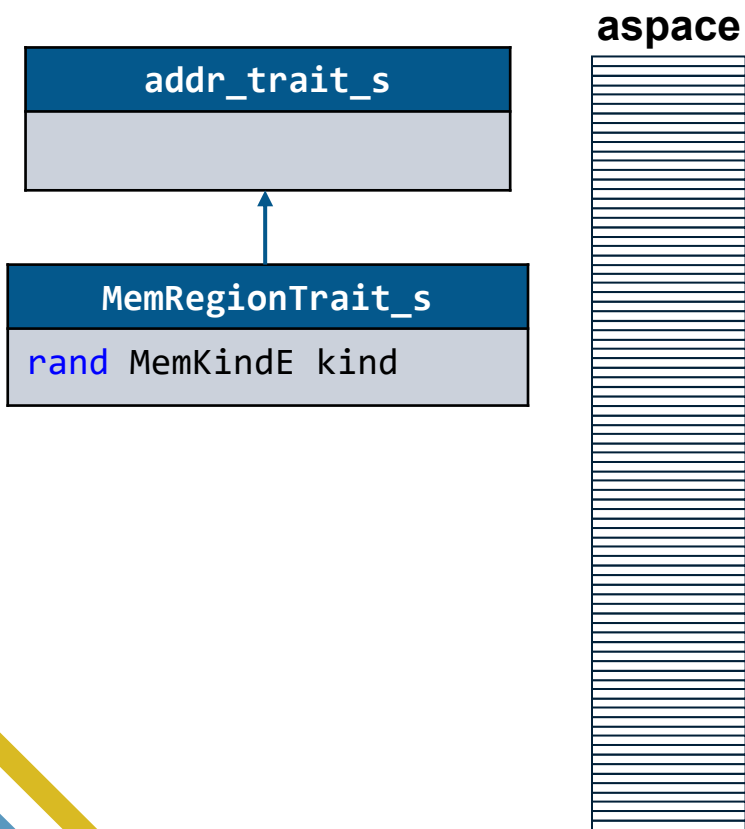
Accessible via built-in API

Use concrete addresses

Address space instantiated
as component

Address Space Traits

- ▶ A *trait* type (struct) describes properties of a contiguous address space



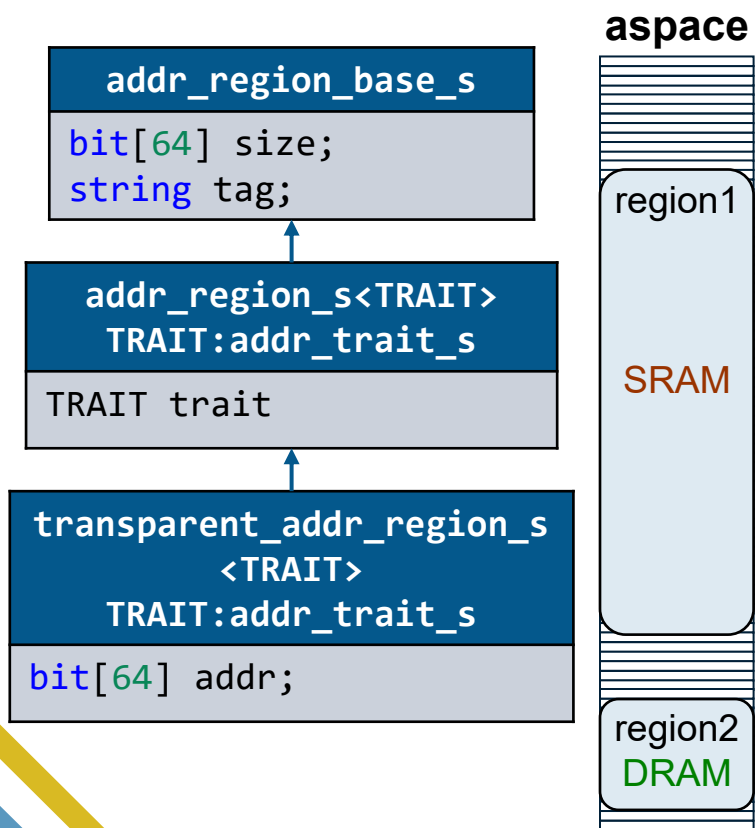
```
package my_addr_pkg {
  import addr_reg_pkg::*;
  enum MemKindE { SRAM, DRAM };
  struct MemRegionTrait_s : addr_trait_s {
    rand MemKindE kind;
  }
}

component pss_top {
  import addr_reg_pkg::*;
  import my_addr_pkg::*;

  transparent_addr_space_c<MemRegionTrait_s> aspace;
}
```

Address Space Regions

- ▶ An address space may be composed of *regions*
- ▶ Regions are characterized by *trait* types with specific values



```
component pss_top {  
  import addr_reg_pkg::*;  
  import my_addr_pkg::*;  
  
  transparent_addr_space_c <MemRegionTrait> aspace;  
  transparent_addr_region_s <MemRegionTrait> region1, region2;
```

```
  exec init_up {  
    region1.size = 0x00300000;  
    region1.addr = 0x00100000;  
    region1.trait.kind = SRAM;  
    (void)aspace.add_region(region1);  
    region2.addr = 0x00001000 ;  
    region2.size = 0x00002000;  
    region2.trait.kind = DRAM;  
    (void)aspace.add_nonallocatable_region(region2);  
  }  
}
```

Region parameterized by same TRAIT type as the `addr_space`

Address spaces set up in `init_up` or `init_down` blocks

Contiguous addr regions have a size

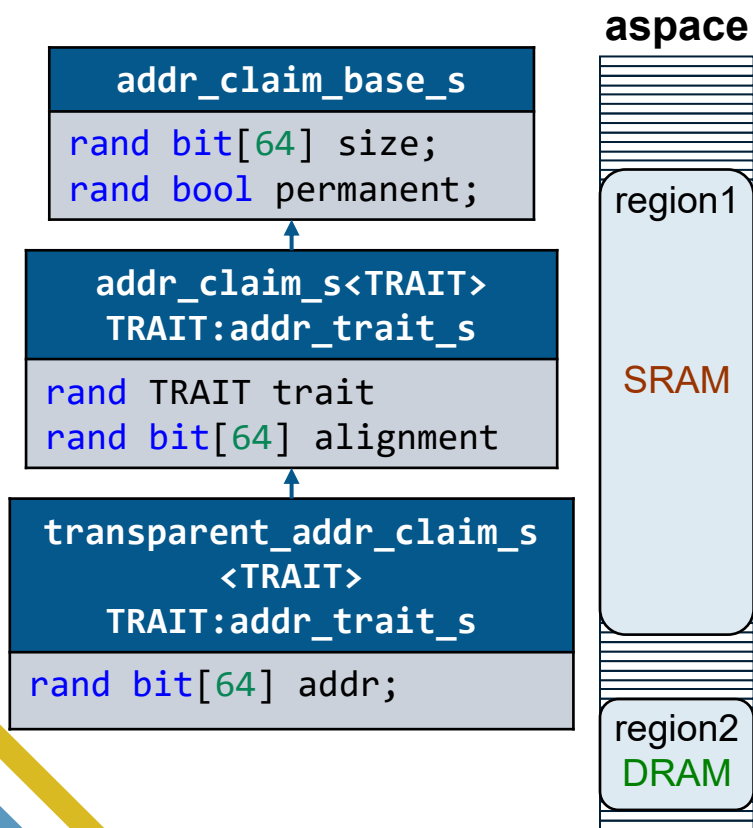
Transparent addr regions have a base addr

Region gets added to the space

Region excluded from allocation
Reserved for registers

Address Space Claims

- ▶ An address *claim* struct has a *trait* field that matches the corresponding region
- ▶ Claims can be made in **actions** or **flow/resource** objects



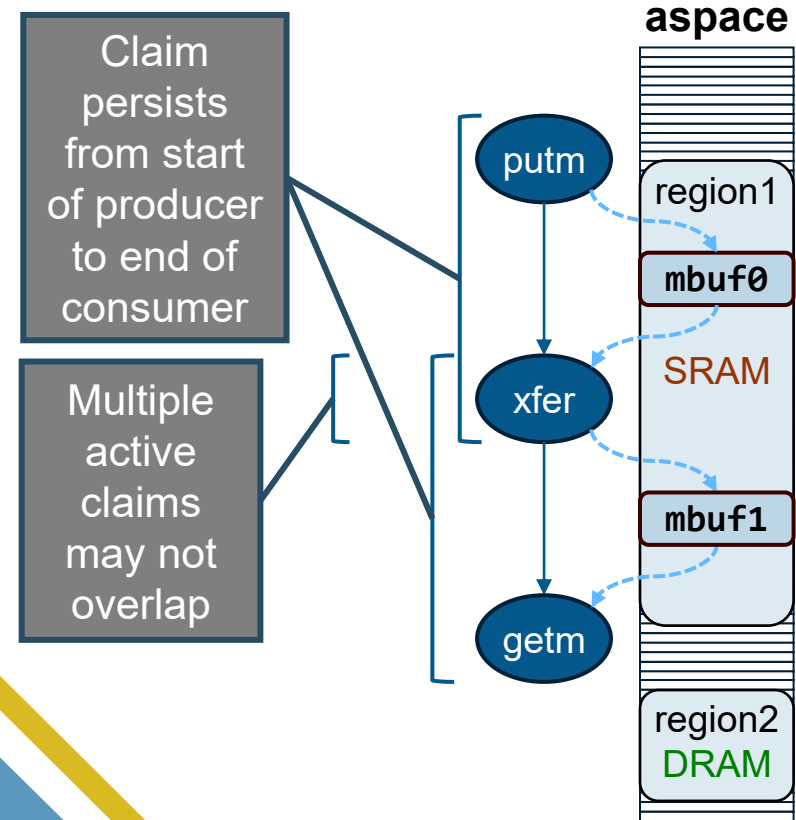
```
package my_addr_pkg {  
  import addr_reg_pkg::*;  
  struct MemRegionTrait_s : addr_trait_s {  
    rand MemKindE kind; // DRAM or SRAM  
  }  
  
  struct MemClaim : transparent_addr_claim_s<MemRegionTrait_s> {  
    constraint alignment == 64'h04;  
  }  
  
  buffer mbuf_b : xbuf_b {  
    rand bit [32] addr;  
    rand MemClaim mClaim;  
  }  
}
```

Claim must be word-aligned
addr % alignment == 0;

Claim is resolved when the
buffer is randomized

Address Space Claims

- ▶ An address *claim* struct has a *trait* field that matches the corresponding region
- ▶ Claims can be made in **actions** or **flow/resource** objects



```
component pss_top {  
  import addr_reg_pkg::*;  
  import my_addr_pkg::*;  
  pool mbuf_b mbuf_p;  
  bind mbuf_p *;
```

```
  action entry_a {  
    mem_c::getm_a getm;  
    mem_c::put_a putm;  
    dma_c::xfer_a xfer;
```

```
    activity {  
      putm;  
      xfer;  
      getm;
```

```
    }  
  }  
}
```

```
  action xfer_a {  
    input mbuf_b ibuf;  
    output mbuf_b obuf;  
    constraint io_c {  
      ibuf.size == obuf.size;  
      ibuf.parity == obuf.parity;  
      obuf.step == ibuf.step + 1;
```

```
    }  
    constraint { obuf.mClaim.trait.kind == SRAM;  
                 ibuf.mClaim.trait.kind == SRAM;
```

```
  }  
}
```

xfer_a claims only match
regions with kind==SRAM



Thank You

