2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Secure Multi-CPU Memory Access in PCIe via Tokenized Address Space Management

Sridhar Mukund, Martand Pangrekar and Sastry Puranapanda
Silicon Interfaces, Mumbai, India

Silicon Interfaces®

accellera SYSTEMS INITIATIVE

Agenda

- Abstract
- Reimagining an Existing Solution
- Multi-Core Limitations
- Race Condition Handling
- Framework Overview
- PCIe Device Identification
- Framework Implementation
- Prevent Data Loss
- Definition of Contiguous Address Space
- Conclusion

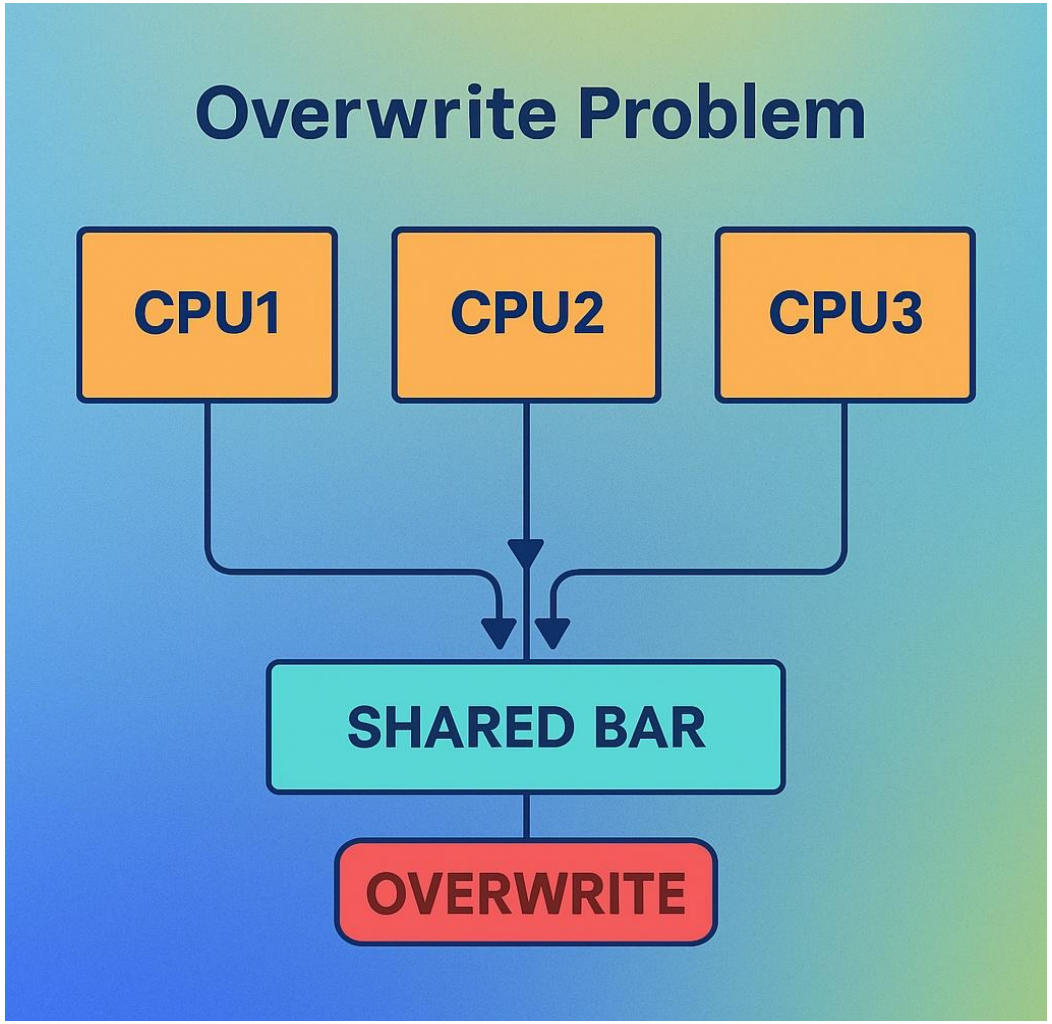
Abstract

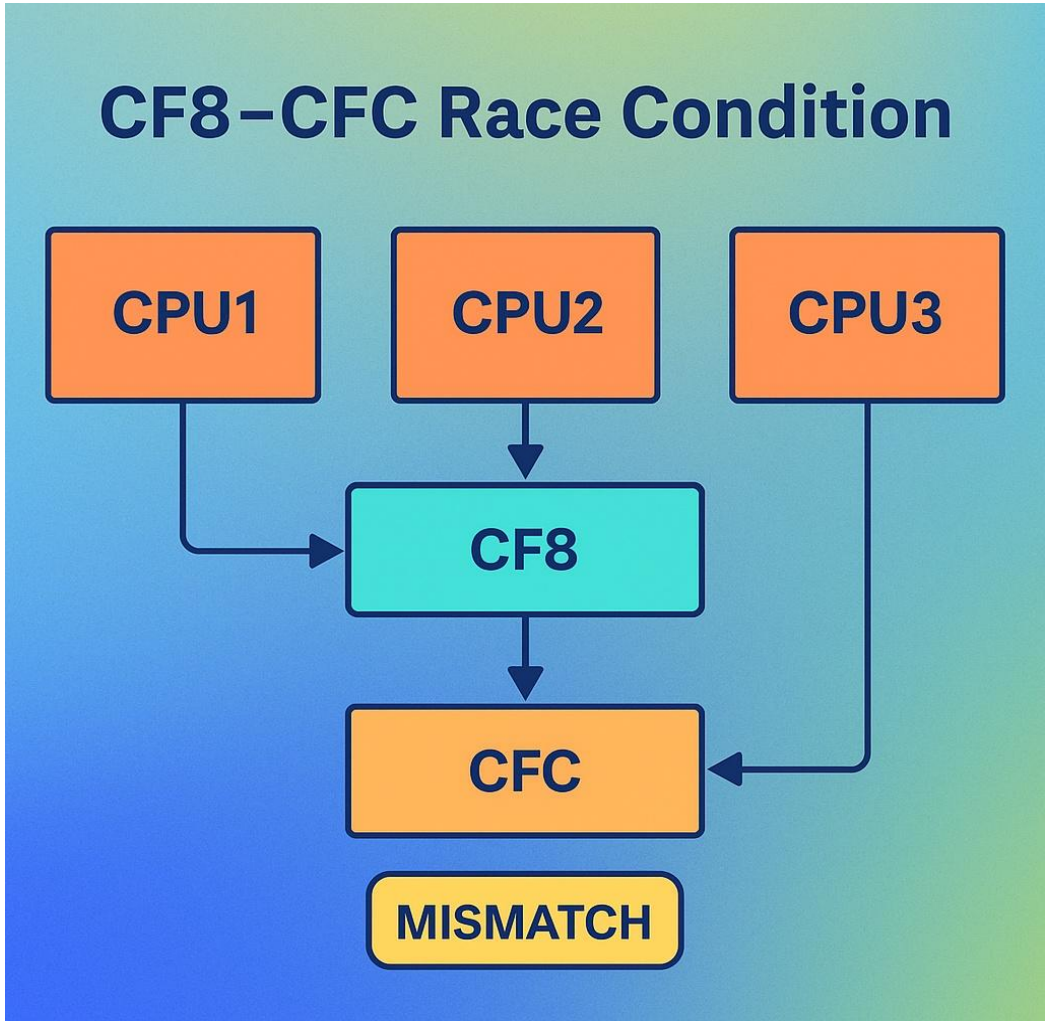
- Token-based framework to secure PCIe memory access in multi-CPU systems. By leveraging byte-addressable memory regions and hash table lookups, we enable secure device addressing in Configuration Space Registers (CF8/CFC)
- Our method enforces data integrity through synchronized access control, using inter-process communication (IPC) constraints to prevent race conditions
- The solution ensures that only authorized tokens grant memory access, mitigating security risks in shared bus architectures
- Experimental results demonstrate improved protection against unauthorized I/O operations while preventing data loss, making it viable for secure PCIe systems

Reimagining an Existing Solution with Portable Stimulus Implementation

- Race conditions in multi-CPU PCIe systems:
  - Simultaneous port addresses access (CF8/CFC) via Root Complex Hub (Bus0)
  - Example: CPU1 writes to 0xCF8, CPU2 interrupts and corrupts 0xCF8 data
- Traditional solutions:
  - Higher memory addresses: slow, static
- PSS 2.0 improvements:
  - Enhanced memory management, scalability
- Goal: Ensure data integrity, data security

Multi-Core Limitations





Race Condition Handling via Resource Locking & Sharing

To prevent race conditions during writes to CF8/CFC registers, a lock-and-share mechanism using PSS data flow resources ensures synchronized access when multiple devices thread data through CPU memory.

```
resource pcie_addr_port_r{
 bit[31:0] CF8;
}
```

```
resource pcie_data_port_r{
 bit[31:0] CFC;
}
```

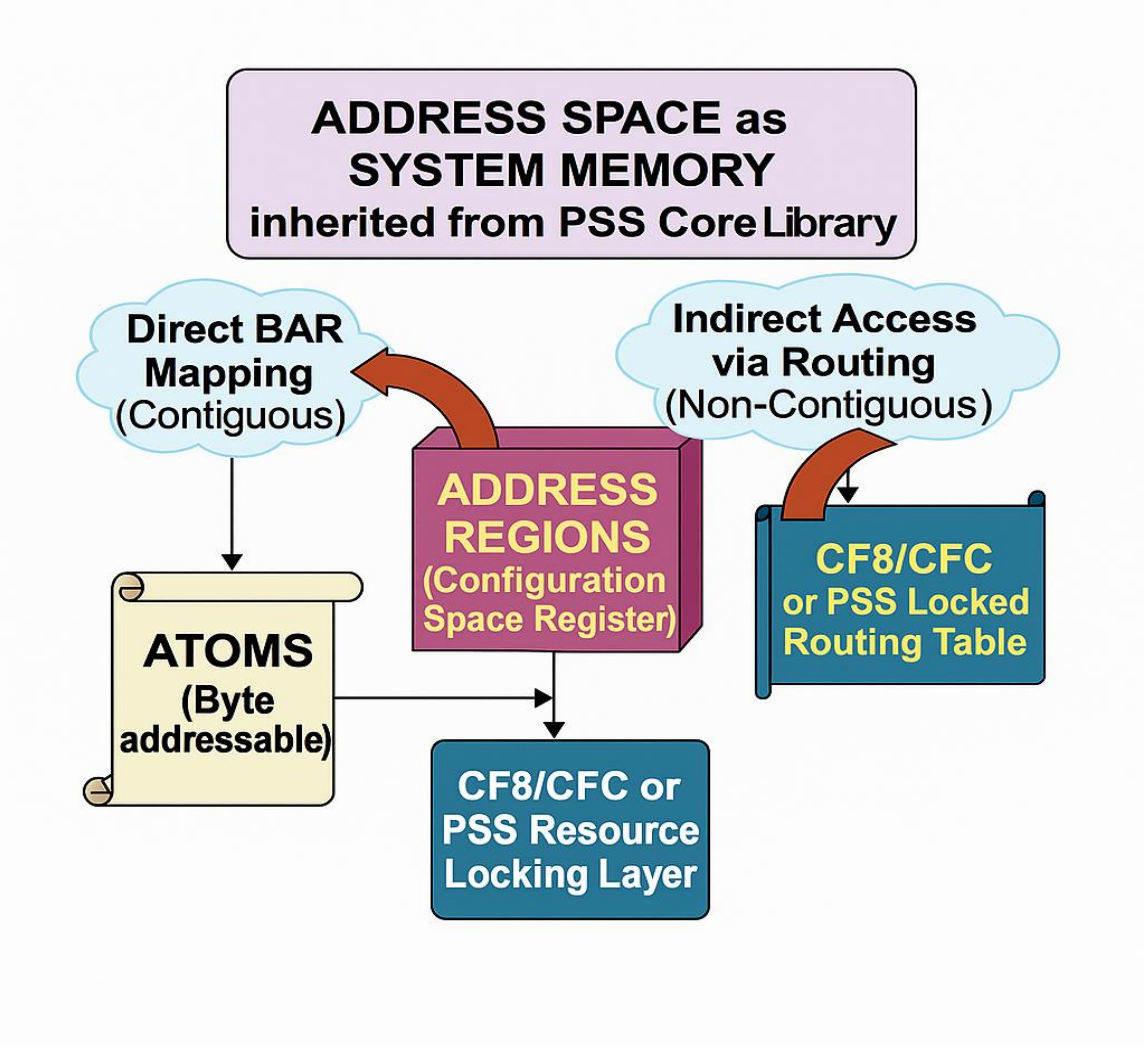
```
component pss_top {
    import ip_pkg::*;

    pool pcie_config_space_b pcs_p;
    bind pcs_p*;
    pool pcie_pss_b pss_p;
    bind pss_p*;
    pool pcie_bdf_b bdf_p;
    bind bdf_p*;
    pool pcie_di_b di_p;
    bind di_p*;
    pool  pcie_config_space_memory_region_b buff_p;
    bind buff_p*;
    pool pcie_addr_port_r p_addr_port_p;
    bind p_addr_port_p*;
```

```
    action config_sharing_a {
        share pcie_addr_port_r pcie_a_port[5];
        }
    action config_locking_a {
        lock pcie_addr_port_r pcie_a_port;
        }
    action root_a {
        claim_output_a c_op;
        activity {
            parallel {
            do config_sharing_a;
            do config_locking_a;
            }
            parallel {
            do config_sharing_a;
            do config_locking_a;
            }
            do config_sharing_a;
            c_op;
        }
    }
}
```

Token-Based Framework Overview

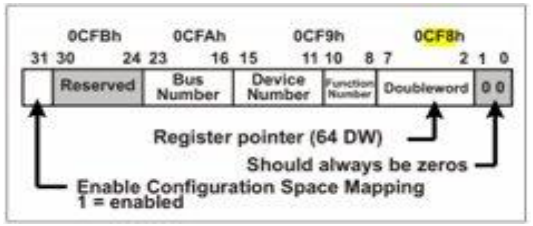
In the context of PCIe by using PSS Memory Management concepts

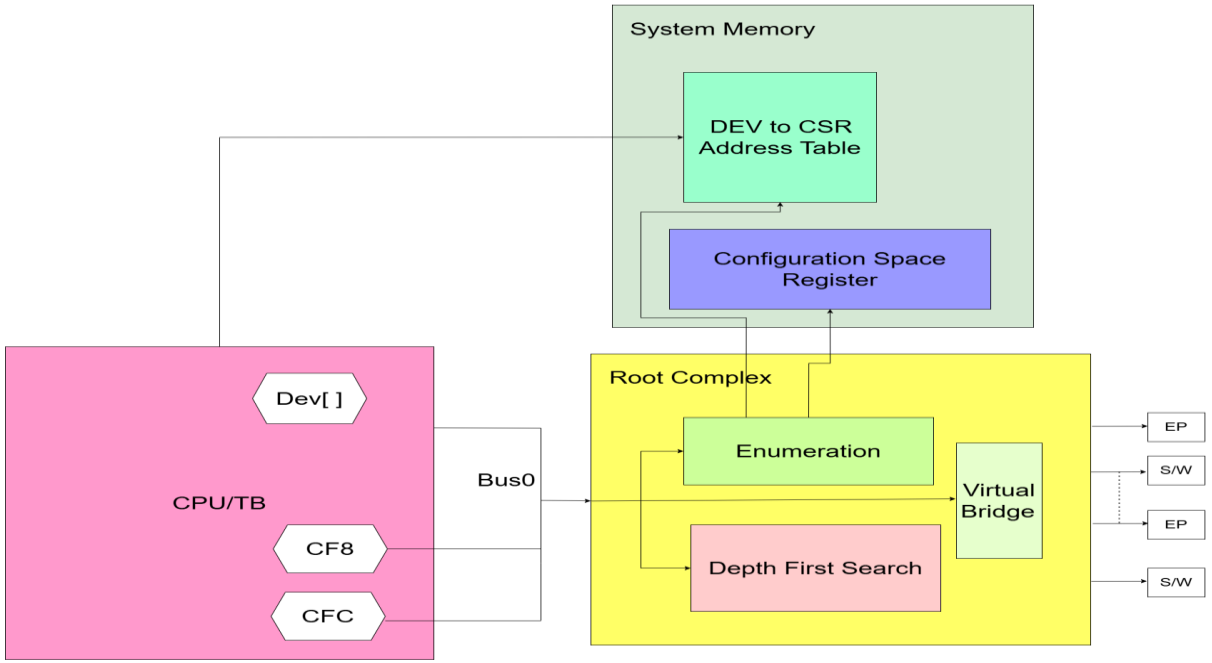
- The System Memory has been inherited as Address Space from PSS Core Library
- The Configuration Space Register is dynamically allocated based on the number of Devices present in PCIe environment as Address Regions with each Region having Atoms which are Byte Addressable

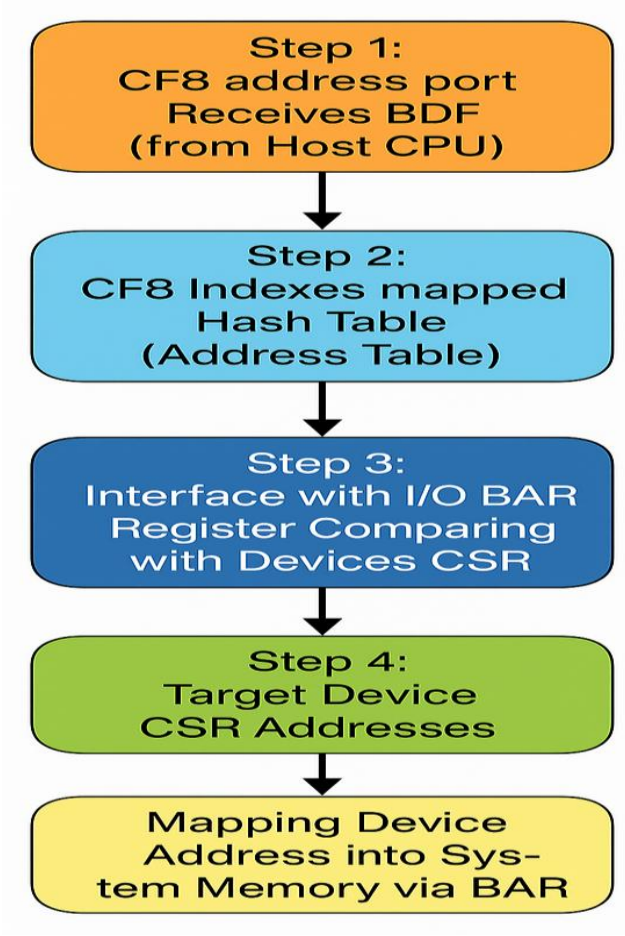


PCIe Device Identification Using Memory Mapped Address Tables

- The CPU accesses the BAR registers by Byte Addressable addresses as shown in the Address Table (hash table) to get BDF

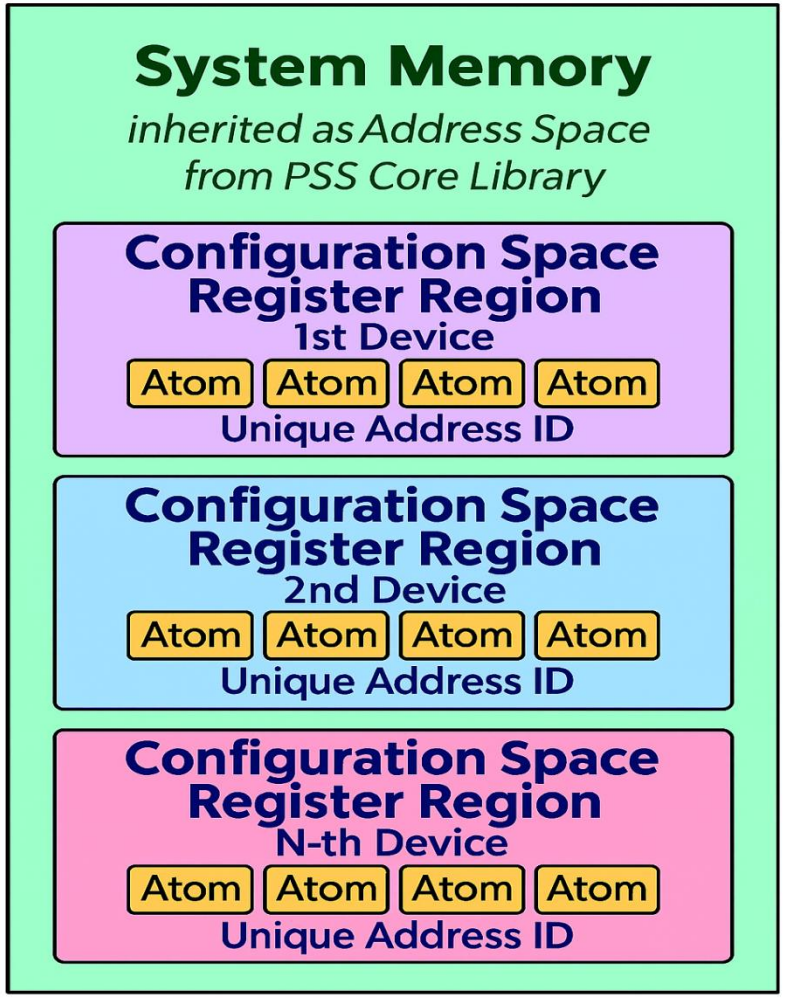






Token-Based Framework Implementation

- PSS 2.0 Address Space Management:
  - Address Spaces: pcie_sm_transparent_addr_space_c
  - Regions: add_region
  - Atoms: pcie_bdf_trait_s
- Tokenized access control:
  - Traits: pcie_bdf_trait_s, pcie_mem_trait_s
- IPC tools for synchronization:
  - Locking: config_locking_a
  - Sharing: config_sharing_a
- Byte-addressable regions:
  - Secure allocation: bool byte_addressability = True
- Address handles: make_handle_from_claim



Implementing Traits to Prevent Data Loss

```
pcie_sm_transparent_addr_region_s<pcie_mem_trait_s> sm;
 pcie_cs_transparent_addr_region_s<pcie_bdf_trait_s>csr[N];
    }

    bit[28] test_number;
    exec init_down{
        pcie_addr_space_start_addr_s passa;
        test_number=passa.csr_start_addr;
        sm.trait.kind=DRAM;
        sm.trait.ctype=WB;
        sm.trait.sec_level=level2;
        sm.size =sm.memsize;
        sm.addr = 0;
        sys_mem.add_region(sm);
        display("System_Memory = ",sm);
        display("");

        repeat(i:N) {
            csr[i].Device_No = i;
            csr[i].size = csr[i].csrsize;
            csr[i].addr = passa.csr_start_addr + (i*4096);
            csr_mem.add_region(csr[i]);
            display("CSR_Memory : ",csr[i]);
        }
```

- Address Allocation via Traits:
  PCIe devices access Configuration Space using byte-addressable regions defined by allocation traits, with each address treated as an atomic unit.
- Trait Structs for PCIe Access:
  Custom traits like pcie_bdf_trait_s and pcie_mem_trait_s, derived from addr_trait_s, identify and enable access to Configuration Spaces via defined memory regions.
- Claim and Match Mechanism:
  PSS 2.0 enables devices to claim address regions by specifying traits; matching Configuration Spaces are then accessed using address handles.

Definition of Contiguous Address Space in the PCIe Environment

- **Address Claiming via Traits and Handles:**
  Byte addressable allocation traits are using for claiming PCIe endpoint addresses, by enabling access with the handles(make_handle_from_claim).
- **Handle Creation and Data Read/Write:**
  The make_hadle_c component generates handles from claims (src_handle, dst_handle) and writes 32-bit data using write32, enabling precise PCIe register access.

```
action make_handle_a {
        pcie_addr_claim_s<pcie_bdf_trait_s> claim;
        input  pcie_config_space_memory_region_b  src_buff;
        output pcie_config_space_memory_region_b  dst_buff;
        constraint{
            src_buff.mem_seg.trait.bus_no == 8'd24;
            src_buff.mem_seg.trait.bus_no ==
sizeof_s<pcie_cs_descriptor_s>::nbytes;
            claim.trait.bus_no == 8'd24;
        }

        addr_handle_t csr_hndl, dst_hndl,my_hndl;
                exec post_solve{
            addr_handle_t csr_hndl =
make_handle_from_claim(src_buff.mem_seg,sizeof_s<pcie_cs_
descriptor_s>::nbytes);
            addr_handle_t csr_hndl =
make_handle_from_claim(src_buff.mem_seg,0);
            addr_handle_t dst_hndl =
make_handle_from_claim(dst_buff.mem_seg,0);

            addr_handle_t my_hndl =
make_handle_from_claim(claim, 5);
        }
```

Outcomes and Conclusion

- Results:
  - Eliminates race conditions
  - Maintains data security, high scalability
- Summary:
- Effective PCIe memory management
  - Robust solution for mitigating race conditions
  - Ensures data integrity and data security
- PSS 2.0 enables scalability
- Future work: Extend to other bus architectures

Questions & Answers

- Contact
  - Support, info@siliconinterfaces.com
  - Heena Bhatia, heena@siliconinterfaces.com
  - http://www.siliconinterfaces.com Agent Betsie™
  - https://www.youtube.com/@SiliconInterfaces
  - http://www.linkedin.com/company/silicon-interfaces
  - http://x.com/silicongo