2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Exploring Token-Based Strategies to Enhance Data Security and Memory Management in PCIe Devices

Gopi Srinivas Deepala, Lakshya Miglani, and Sastry Puranapanda
Silicon Interfaces, Mumbai, India

Silicon Interfaces®

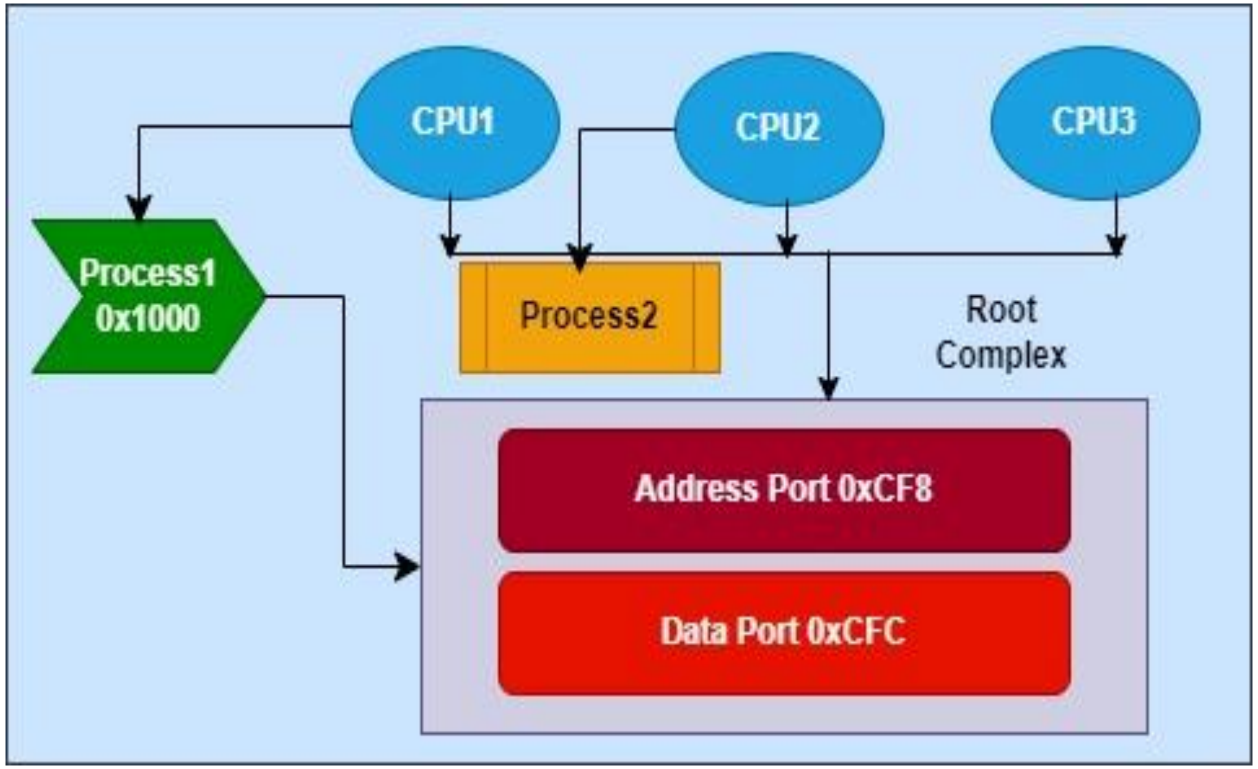
accellera SYSTEMS INITIATIVE

Abstract

- This presentation demonstrates a solution for the work-around for race condition due to call to the configuration address and data I/O ports CF8 & CFC resolved using higher memory locations in PCIe
- The issues on data integrity and security is further compounded when multiple Cores in multi-Core environment are connected to the Bus0 of the Root Complex and may attempt to access them at the same time!
- By using PSS memory management using Address Space/Regions as well as Byte Addressability, Resources and Traits, it is possible to successfully address data integrity and security issues for PCIe

Challenges

- In a multi-threading system where multiple CPUs interact with a PCIe endpoint
- The Data Send process can pose the risk of a Race Condition.
- This simultaneous access from multiple CPUs or Cores can compromise the integrity of the data being written to the configuration space, necessitating measures to prevent conflicts and ensure proper synchronization during/post the enumeration and other communication processes.



Safeguard device data

- To prevent such issues and safeguard device data, we leveraged the features of PSS v2.0
  - using Address Space, Address Regions and Traits
    - allocation of the Address Space region (Contiguous Byte Addressable) for TYPE0 or TYPE1 Configuration memory space (256 bytes to 4Kbytes extended configuration space) is achieved using Byte Addressability.
  - Like replicate and repeat, constraint for all and many more constructs it is possible to implement access to CF8 and CFC addresses and data signals when multiple devices access CPU system memory for data threading, and use parameterized traits, resource sharing and locking to ensure proper synchronization and preventing race conditions

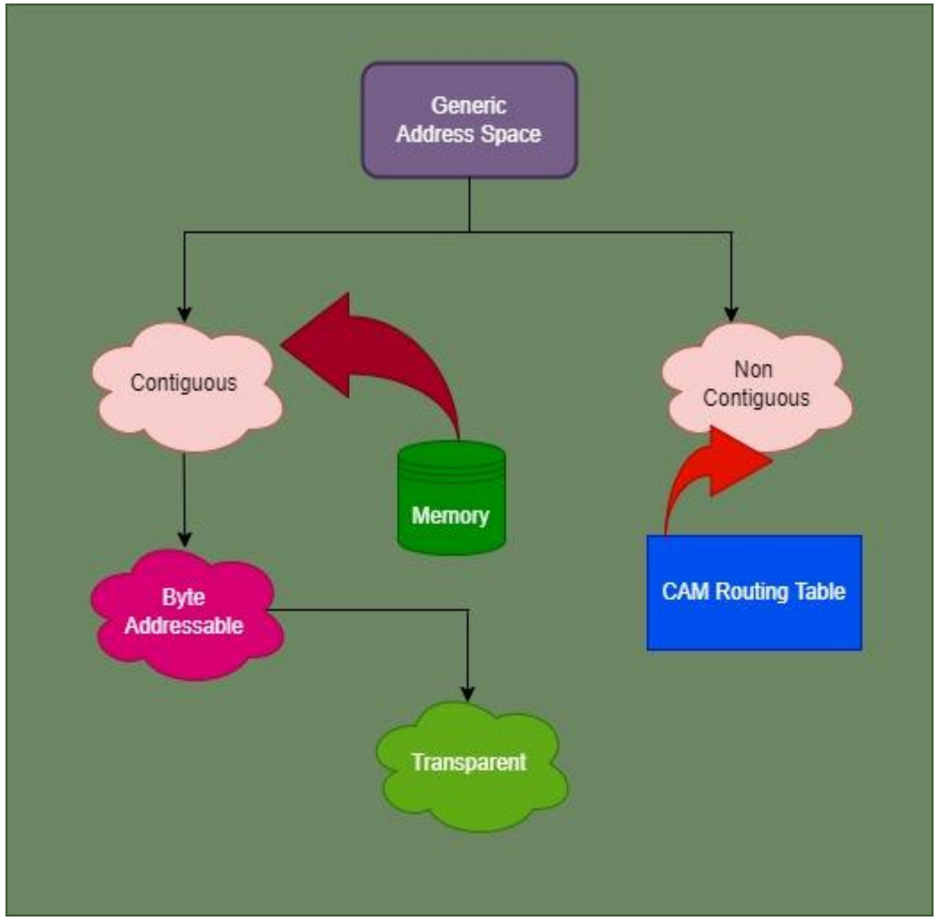
PSS 2.0 Advancements

With the release of PSS 2.0, significant advancements have been made in addressing these complexities.

- PSS 2.0 introduces enhanced mechanisms for defining test intents, addressing concerns regarding address relationships and memory overlap more effectively, like Address Space, Region and Byte Addressability.
- Additionally, PSS 2.0 leverages improved scalability, maintainability, and overall robustness in managing concurrent access to PCIe devices by using PSS 1.0 share and lock resources also helped.
- While PSS 1.0a laid the groundwork for addressing portability and reusability concerns, PSS 2.0 builds upon this foundation by offering more comprehensive solutions to challenges related to data integrity (by using parameterized Traits), conflicts, and scalability in contemporary computing environments.

Generic Address Space

- The introduction of the Address Space concepts, which represents a uniquely addressable space and comprises individual atoms.
- Address Space, Regions, and Atoms serve as the fundamental components of memory modeling within the PSS framework.
- Within these allocable address spaces, individual addresses are treated as atoms.
- These features centered on memory modeling



Address Regions

- This allocation process utilizes allocation traits, which PSS tools leverage to define regions.
- When a PCIe device aims to access and operate the traits within the Configuration Space region, it must assert a claim for allocation.
- Subsequently, data can be written to these registers using generic operations facilitated through address handles.

```
//======================

//ADD SPACE DECLARATION

//======================

//This is the base type for all address space regions

extend struct addr_region_base_s {

array<bit[31:0],16> pcie_addr_region_t0_cs_head;
array<bit[31:0],48> pcie_addr_region_t0_cs_cam;
array<bit[31:0],960> pcie_addr_region_t0_cs_ext_cam;};

//The addr_region_s type represents a region in contiguous address
space

struct empty_addr_region_s{};

//================

// ADDRESS REGION

//================

struct addr_region_s <struct TRAIT : addr_trait_s =
empty_addr_trait_s> : addr_region_base_s     {

TRAIT       pcie_cs_id_trait;          };
```

Address Space Defined

- Within the contiguous and transparent address spaces the address handles are employed to claim matching traits
- In PCIe endpoint devices, the allocation of address space for Contiguous Byte Addressable regions in TYPE0 or TYPE1 Configuration space is enabled through Byte Addressability.
- Configuration Spaces possessing traits that align with the constraints specified by the claim become eligible candidates for matching regions.

```
// Built-in library component for contiguous address space

component contiguous_addr_space_c <struct TRAIT : addr_trait_s =
empty_addr_trait_s> : addr_space_base_c {



 // add_region is the function of contiguous address space components
which is used to add allocatable address space regions to a contiguous
address space.

function addr_handle_t add_region(addr_region_s <TRAIT> r);

function addr_handle_t add_nonallocatable_region(addr_region_s <> r);

bool byte_addressable = true; };

 // Transparent address spaces are used to enable transparent claims—
constraining and otherwise operating on concrete address values on the
solve platform

component transparent_addr_space_c
contiguous_addr_space_c<TRAIT> {};

component pcie_ip_c { struct pcie_struct { rand int a; };

action pcie_op { rand pcie_struct s; };

 }
```

Address Space claim

- A claim specifies a trait, mapping it to Configuration Spaces within the address space.
- Data can be written to these registers using generic operations facilitated through address handles.
- Constraint the Address Space claims to get the Configuration Space from the Base Address claim within a Contiguous Address Space.

```
/*=======================
ADDRESS SPACE CLAIM USING TRAITS
=======================*/
struct addr_claim_s<mem_trait_s> {};
struct addr_claim_s{}

extend struct addr_claim_s{
                    TRAIT trait_claim;
}
buffer data_buff {
                    rand addr_claim_s mem_seg;
}
action pcie_op {
                    rand pcie_struct s;
                    };
 }
struct transparent_addr_claim_s <struct TRAIT : addr_trait_s = empty_addr_trait_s> :
addr_claim_s<TRAIT> {
                    }
struct transparent_addr_region_s <struct TRAIT : addr_trait_s = empty_addr_trait_s> :
addr_region_s<TRAIT> {
                    }
```

Traits for Data Integrity

- To prevent data loss and maintain data integrity we have leverage the concept of parameterized Traits.
- These traits are utilized to identify Configuration Spaces, enabling check to see the correct Dev/Function.

```
//=========================
//STRUCT TRAITS DECLARATION
//==========================
struct TRAIT{};
extend struct TRAIT
{
bit[15:0] device_id;
bit[15:0] vendor_id;
bit[15:0] subsystem_device_id;
bit[15:0] subsystem_vendor_id;
};
```

Resource Declaration

- The critical elements such as ports 0xCF8 and 0xCFC are associated with multiple devices accessing CPU memory for data threading.
- By addressing a solution, we implemented utilizing the lock and share property of the PSS data flow object as a resource.
- This ensures proper synchronization by facilitating resource sharing and locking, thereby preventing race conditions.

```
//======================
//RESOURCE DECLARATION
//======================
resource pcie_addr_port_r{
        bit[31:0] CF8;
};
resource pcie_data_port_R{
        bit[31:0] CFC;
};
```

Data Security & Memory Management

- The PCIe address space management by providing essential functions for handle creation and data writing.
- Action "create_handle_a", which initiates a handle creation process based on a claim and offset.
- By encapsulating these operations, the component simplifies PCIe device communication and configuration, boosting system efficiency and reliability

```
component make_handle_c{
        function addr_handle_t make_handle_from_claim
(addr_claim_base_s claim, bit[64] offset = 0);
        function void write32(addr_handle_t hndl, bit[32]
data);
         action create_handle_a{
           exec body {
                      bit[64] offset = 16;
                      bit[32] data = 128;

                      comp.addr_handle_t =
make_handle_from_claim(claim, offset);

                        comp.write32(hndl, data);
                    }
              }
              action trait_move:trait{}
        }
```

Conclusion

- This research presentation presents a robust solution for mitigating race conditions
- Preventing data loss (security) in the PCIe endpoint configuration space/other data transmission.
- By leveraging PSS v2.0, it harmonizes memory management across various IPs.
- Improves PCIe Configuration Space modeling using Address Space, facilitating seamless communication.

THANK YOU