2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Addressing Shared IP Instances in a Multi-CPU System Using Fabric Switch A Comprehensive Solution

Priyanka Gharat, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (*priyanka@siliconinterfaces.com*)

Avnita Pal, VLSI Design Engineer, Silicon Interfaces, Mumbai, India (*avnita@siliconinterfaces.com*)

Sastry Puranapanda, Design Engineer, Silicon Interfaces, Mumbai, India (*sastry@siliconinterfaces.com)*

***Abstract*— The objective of this paper is to utilize the capabilities of PSS-based DSL language features, such as byte addressability, resource sharing/locking, multiple component instances, and true parallel scenarios. The purpose is to address problems related to data integrity and bottlenecks in multi-core processors that communicate with multiple devices. This is achieved by implementing a learning heterogeneous switch fabric with address storage and translation. With this feature, cores can communicate with any endpoint device on one of the switch ports while leaving other ports free for communication, thus enabling parallel traversal operations. The result is a method that maintains parallelism while ensuring data integrity through resource sharing/locking.**

Keywords—Switch Fabric, bottleneck, multi-core, sharing, locking, component

I. INTRODUCTION

In modern System-on-Chip (SoC) design, data flow between various devices within the SoC occurs through the bus interface. The bi-directional channels enable data to flow between the memory and multiple instances of the same device or different devices. The ability to concurrently exchange data between multiple devices while maintaining data integrity is a crucial challenge in SoC design. To address this challenge, the Portable Stimulus Standard (PSS) introduces the powerful feature of component instances, which enables the creation of copies of IPs that can be reconfigured based on the selected component. These component instances can have exclusive or shared access to the bus interface, allowing for concurrent data exchange. The resources are mapped to agents in the UVM target test-bench, enabling effective testing of the SoC design. In this paper, the solution is implemented using component instance array, address region/space, byte addressability and discuss how they facilitate efficient data exchange and address challenges in SoC design.

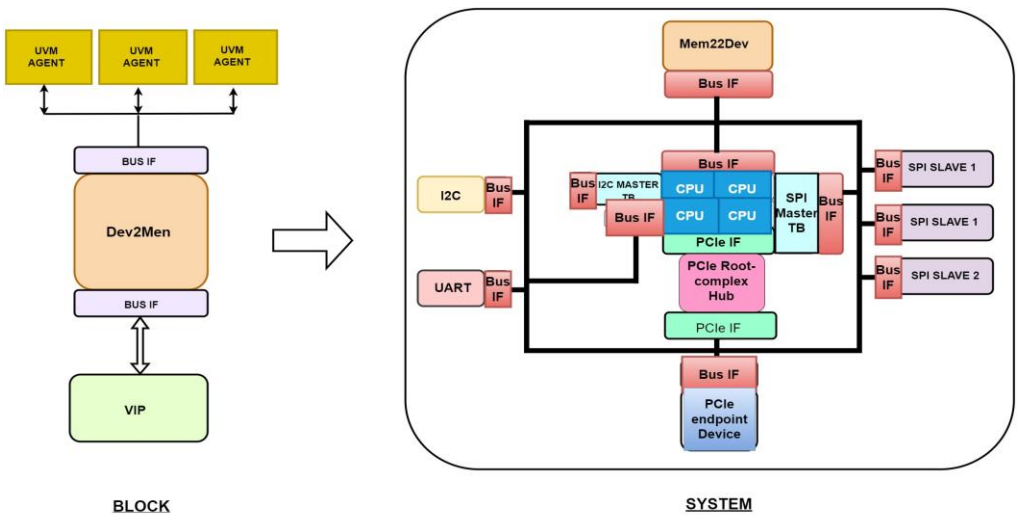

Figure. 1. Data flow between and multiple instances of samedevice or different data flow to different devices

II. WHAT IS FABRIC SWITCH?

A multi-core CPU SOC requires access to various IPs (Communications, Networking, Audio/Video, and Memory)through a common Bus Interface. When the Bus and Channels intersect, they form a matrix-like structure known as a fabric or mesh. The Fabric Switch is like a crossbar exchange wherein data may flow from any one upstream input port to any one downstream output port. The idea is that we have a matrix connectivity from upstream to downstream. The ports are bi-directional so upstream/downstream and input/output are only notional. The upstream may be connected to multi-Core and downstream ports may be connected to several different or even same sets of devices.

III. RESOURCE SHARING/LOCKING

PSS can be used to describe complex scenarios for SPI communication at the block level, which can be reused at the SoC level. With PSS, instances of SPI IP components can be defined through a component instances array, and a pool of resources can be defined and using traits a specific device is being claimed via the bus interface. Each SPI IP component's atomic action can either lock or share a resource to access the bus interface for data flow exchange

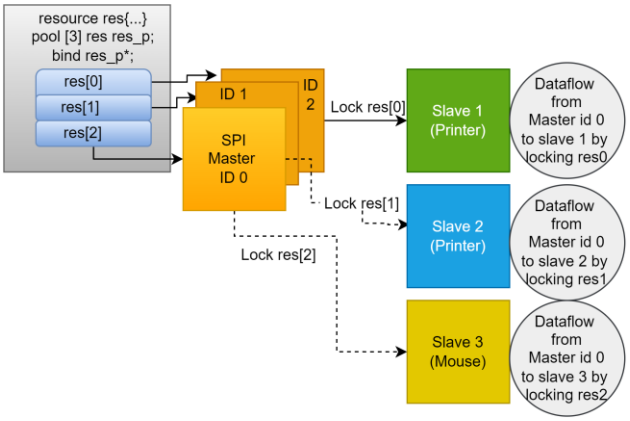

Fig. 2.a. Exclusive lock of a resource by different SPI component instances to write different data to different slaves.

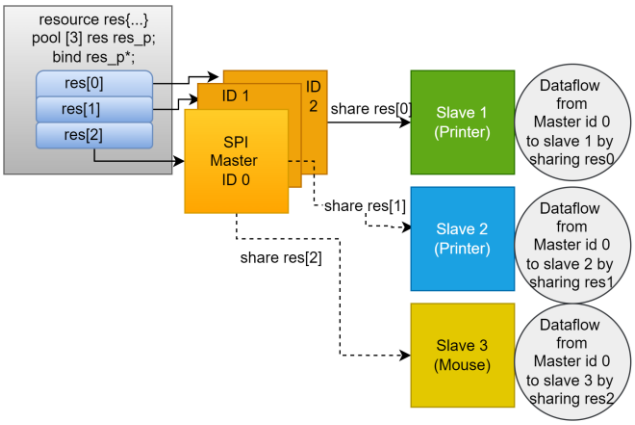

Fig. 2.b. Sharing of resource by different SPI component instances to write same data to different slaves (slave1 and slave2) and lock of resource by SPI component to write different data to slave3.

. To address this challenge, data flow is being handled through parallel channels, as depicted in Figure 2, without compromising data integrity. When the CPU wants to transfer the same data from memory to a device, it first signals to share the data between the two resource pools for the device, as shown in Figures 2. However, if the data is intended for a different device, the channel resource may be locked or Device ID may be matched by parametrized traits, and the endpoint device can claim the address space location to ensure that the data is secured for that specific device. Notably, a common BUS IF is used for several IPs through a data flow object.

IV. IMPLEMENTATION USING FABRIC SWITCH

The ability to concurrently exchange data between multiple devices while maintaining data integrity is a crucial challenge in SoC design. Without any limitations, there is a possibility of bottlenecks, data corruption, and loss.

If multi-CPU tries to write or read through the same share bus, in that case there is a potential for address and data loss. The primary goal of this paper is to leverage the features of PSS/DSL language and as an example, address the limitations of PCIe in a multi-Core environment caused by the Root Complex design.

To address this issue, a switch with multi cores on the upstream and end point devices on downstream can be embedded in the fabric, creating a Switch Fabric. This implementation is carried out with Portable Stimulus Standard (PSS) powerful features of component instances, which enables the creation of copies of IPs that can be

reconfigured based on the selected component. These component instances can have locked or shared access to the bus interface, allowing for concurrent data exchange. This is achieved through a common Bus I/F and Fabric Switch, with the presence of carrier sense logic. The resources are mapped to agents in the UVM target test-bench.

In this architecture CPU Switches connect to Fabric Switches with bidirectional port End Point Devices. These switches are intelligent devices that have a table of Device IDs and ports, enabling them to perform address translation and Network Address Translation (NAT) to send/receive packets through a single channel. The address learning for End Point Devices occurs during the Enumeration process, and Configuration Space Registers with Base Addresses are saved in Memory. This technical approach is designed to manage data transmission in a multi-CPU system.

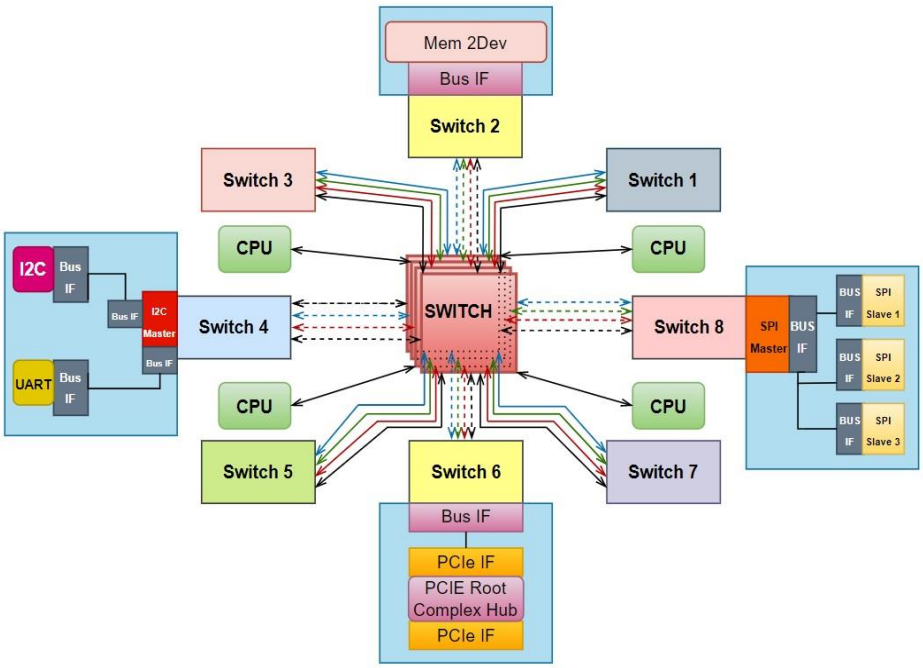

Figure. 3. Multi CPU SoC environment; the Switch 1..8 are notional ports of the Fabric Switch, shown as SWITCH in this figure

Implementation has been written for multi-Cores on with multiple SPI slaves. Each SPI master has been configured as regions and an address space and byte addressability, and memory has been allocated using PSS version 2.0 DSL language construct and have unique traits and claimed using traits. The component tree has been used to resolve the address space and along with that the scenarios such as write data and read data from memory performed parallel and then considering fabric switch the pipelining delay. This is achieved by enhancing the Root Complex and resolving the PCIe bottleneck resulting from addressing multiple instances of End Point Devices in a Multi-CPU Multi-Core system. The solution involves connecting the Multi-CPU Multi-Core CPU to a Switch and connecting the Switches to the End Point Devices.

Below is the pictorial representation of the scenario being created.

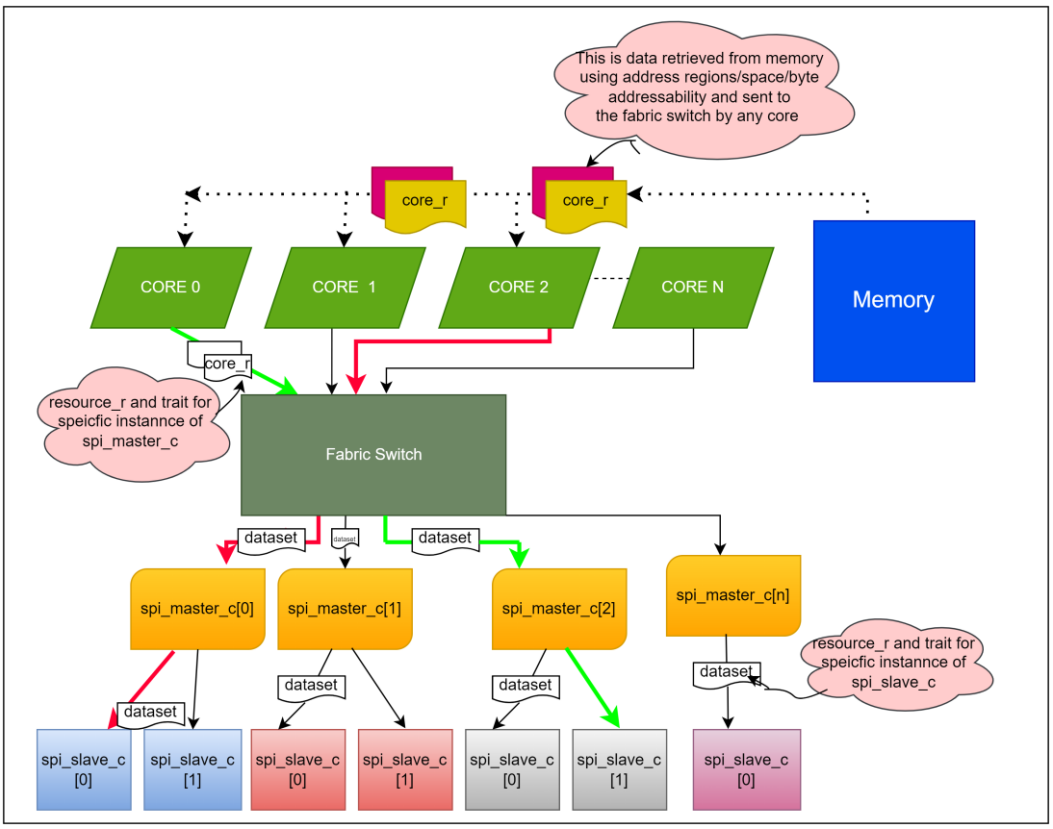

Figure. 4. Data set transferred in resources packet from the memory to bottom hierarchy through fabric switch

Using the rich constructs available within PSS/DSL, a relatively complex task has been easily implemented. The challenge of using multiple channels to access the same IP can be very complex when there are multiple CPUs present in the SOC or sub-system. The work done so far is applying the concept using PSS construct on SPI protocol as the endpoint side for the scenarios of multi slave generated using multi instance of component and several cores, say core_c[0] has communicatedwith slave1 (printer), core_c[1] with slave2 (printer), and core_c[2] with slave3 (mouse). Different packet data has been sent to the printers concurrently, the Switch Ports have locked resources from the pool to have exclusive access to the bus interface and initiate the data transfer. Meanwhile, core_c[2] has also locked a resource and performs data flow exchange. The received packets are rightly addressed and have the expected data and thereby proving data integrity even though communication is in parallel.

The below source code shows the transfer of data to different slave from master using address space, region, and byte addressability which are the construct of PSS version 2.0. This data is part of resources are being addressed from the pool created at the top-level component and the different multiple instances of master and slave get bind from those pool where they are being shared and locked during a particular transfer.

```
package pkg{
        resource core_r {rand bit [3:0] data;}
        resource spi_dataset_r{
                rand bit [31:0] data;
        }
}
//Struct definition for Trait which is the device ID.
struct TRAIT{rand int device_ID;};

enum cache_attr_e {PCI, PCIe, AXI, WB};
enum security_level_e {level0, level1, level2, level3};

struct mem_traits_s:TRAIT {
      rand cache_attr_e ctype;
      rand security_level_e sec_level;
}

struct addr_region_base_s {
        bit[64] size;
};

//Since Tools still are not supporting PSS 2.0 we
 are using global address region.
//struct addr_region_s <struct TRAIT = null_trait_s>

struct addr_region_s:addr_region_base_s
{
        mem_traits_s trait;
};

//struct transparent_addr_region_s <struct TRAIT = null_trait_s>

struct transparent_addr_region_s:addr_region_s
{
        bit[64] addr;
};
struct fb_trait_s {}
```

```
//Component definitions of address spaces
component addr_space_base_c{}

//component contiguous_addr_space_c
<struct TRAIT = null_trait_s>:addr_space_base_c
component contiguous_addr_space_c:addr_space_base_c{

   //function void add_region(addr_region_s<TRAIT> r);
   function void add_region();

   //functionvoid add_nonallocatable_region(addr_region_s<> r);
   function void add_nonallocatable_region()

   bool byte_addressable = true;
};

//component transparent_addr_space_c
<struct TRAIT = null_trait_s>:contiguous_space_c<TRAIT>
transparent_addr_space_c:contiguous_addr_space_c{

/*It is illegal to pass a non-transparent region to the
add_region() function.*/
}

component pss_top{}

component my_ip{}

extend component pss_top {
   my_ip ip;

   //transparent_addr_space_c<> sys_mem;
   transparent_addr_space_c sys_mem;

   //transparent_addr_space_c<> local_mem;
   transparent_addr_space_c local_mem;
   transparent_addr_region_s r0;
```

```
//============================
//Fabric Switch
//============================
component fabric_switch_c<struct TRAIT =null_trait_sss>{
   import pkg::spi_dataset_r;
   pool[5] spi_dataset_r dataset_p;
   bind dataset_p*;
   rand TRAIT trait_s;

   //Component Instantiated under fabric component
   spi_master_c spi_master[3];
   core_c core[3];
   action root_a{
      activity {
         schedule{
            core[0].write_a with {dataset_p.resource_id=0;
                                  dataset_p.data =4'b1011};
            spi_master[0].read_a with {dataset_p.resource_id=0;
                        dataset_p.data= core[0].write_a.dataset_p.data};
          }
         parallel{
            core[1].write_a with {dataset_p.resource_id=1;};
            spi_master[1].read_a with {dataset_p.resource_id=1;};

       }
   }
}

//============================
//SPI_MASTER
//============================
component spi_master_c <struct TRAIT = null_trait_s>
:addr_space_base_c{
   import pkg::*;
   /*Action to perform the write operation
   on the SPI slave where resources are locked */
   action write_a {
      output m_data_buff_b wr_prod_o;
      rand bit [7:0]write_data;
```

```
          assert(req.randomize() with{
              wr_data = {{write_data}};
              ctrl_reg[7] = {{wr_prod_o.ctrl_reg.spie}};
          });
          finish_item(req);
        end
        """;
    }

   action write_res_share_a:write_a{
       share core_r core_s;
    }

   action write_res_lock_a:write_a{
       //lock core_r core_1,core_10,core_11;
    }
   //Action to perform the read operation on the SPI SLAVE
   action read_a {
       lock core_r core_1;
       share core_r core_s;
       input data_buff_b  rd_cons_i;
   }

}
//============================
//SPI_SLAVE
//============================
component spi_slave_c<struct TRAIT = null_trait_s>
:addr_space_base_c {
import pkg::*;

    //Action To receive the data during write opertaion
    action rx_wr_data_a {
        lock core_r core_1;
        share core_r core_s;
        input m_data_buff_b wr_cons_i;
    }
```

```
    rand bit mode_t;
    lock spi_dataset_r core_1;
  }
  extend action write_a {
    exec run_start SV =
      """
       display("Am in run start");
      """;
    exec body SV = """
       begin
        spi_seq_item req = spi_seq_item::type_id::create("req");
        start_item(req);
```

Note: Current (Apr 2023) generation of EDA tools for PSS/DSL did not support PSS2.0 fully, so wherever necessary global variable is used to demonstrate trait passing as parameters

V. CONCLUSION

In conclusion, this research paper explains the feasibility of using specification PSSv2.0 for simultaneous upstream/downstream of byte-addressable data to multiple shared/locked devices in multi-CPU SOC environments. Additionally, the paper demonstrates that significant speed-ups can be achieved if the implementation is done in a concurrent and/or distributed environment. This research presents a promising approach to improving the performance of multi-CPU SOC systems and can be useful for future developments in this field.

REFERENCES

[1] G Portable Test and Stimulus 1.0a Language Reference Manual Accelerate Systems Initiative USA June 2018

[2] J. https://accellera.org/images/resources/videos/Accellera- PSS-Tutorial-2020.pdfPortable Stimulus: What's Coming in 1.1 and What it Means For You by Portable Stimulus Working Group (Tom Fitzpatrick, Mentor, a Siemens Business • Prabhat Gupta, AMD • Matan Vax, Cadence Design Systems • Karthick Gururaj, Vayavya Labs • Hillel Miller, Synopsys).

[3] I.S https://accellera.org/images/downloads/standards/Portable_Test_Stimulus_Standard_v20.pdf