Interoperability Validation Without Direct Integration

Nicholas Nuti (nicholas.nuti@intel.com)
Srinivasan Jambulingam (srinivasan.j@intel.com)
Intel Corporation

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

DIRECT VERSUS INDIRECT INTEGRATION

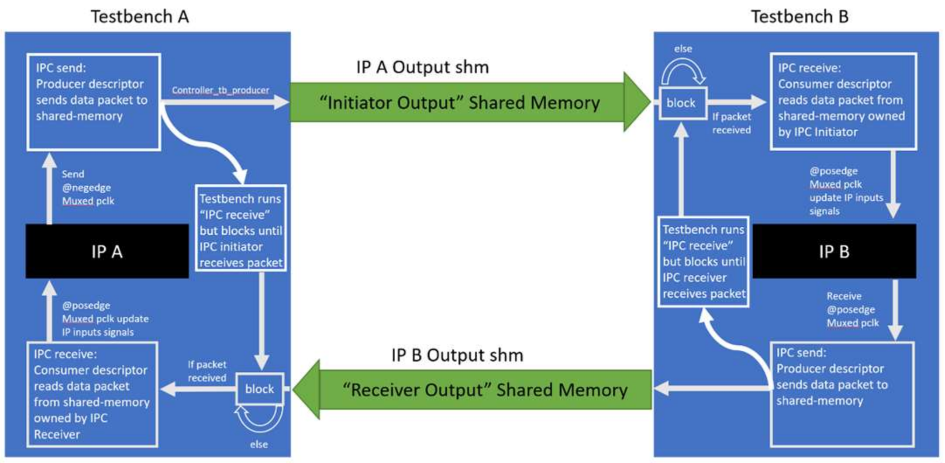
- ❑ Direct IP Integration:
  - ❑ Involves manually wiring IPs into a common simulation environment.
  - ❑ Useful for situations like integrating IPs into an SoC environment to get a full behavioral model.
- ❑ Indirect IP integration:
  - ❑ Involves two active and separate IP simulations exchanging data through software interfaces.
  - ❑ Useful for quickly verifying interaction behavior between two IPs without requiring serious alterations.

MOTIVATION

- ❑ Simplified Interoperability Validation Setup
  - ❑ Lack of common and adaptable simulation setup process
  - ❑ Overcomplicated simulations because consolidation
- ❑ Debug of Multi-IP Simulations
  - ❑ Debugging of multi-IP simulation setups is difficult for engineers inheriting new IPs because of environment and IP requirements
  - ❑ Leads for debugging focus astray from interoperability
- ❑ Simulation Environment Maintenance:
  - ❑ Requirements of IPs change for every iteration which becomes difficult to maintain when dealing with multiple IPs in one simulation environment

INDIRECT INTEGRATION METHOD

- ❑ Testbenches:
  - ❑ Signals leave and enter testbenches through DPI (Direct Programming Interface) tasks.
- ❑ DPI Tasks:
  - ❑ DPI tasks interact with C++ based IPC interface software.
- ❑ IPC Interfaces:
  - ❑ C++ IPC interfaces interact with shared memory.

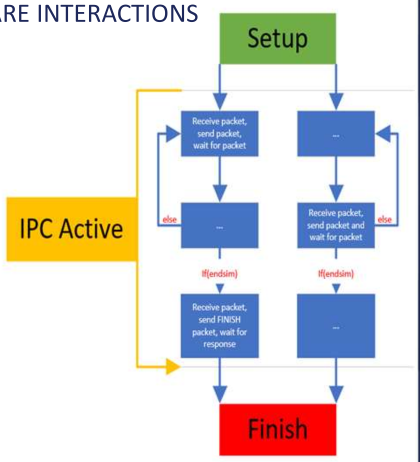

Testbench A
Testbench B
IPC send: Producer descriptor sends data packet to shared-memory
Controller_tb_producer
IP A Output shm
"Initiator Output" Shared Memory
else
block
If packet received
IPC receive: Consumer descriptor reads data packet from shared-memory owned by IPC Initiator
Send @negedge Muxed pclk
Testbench runs "IPC receive" but blocks until IPC initiator receives packet
IP A
@posedge Muxed pclk update IP inputs signals
Testbench runs "IPC receive" but blocks until IPC receiver receives packet
IP B
@posedge Muxed pclk update IP inputs signals
Receive @posedge Muxed pclk
IPC receive: Consumer descriptor reads data packet from shared-memory owned by IPC Receiver
If packet received
block
else
IP B Output shm
"Receiver Output" Shared Memory
IPC send: Producer descriptor sends data packet to shared-memory

SOFTWARE BEHAVIOR

- ❑ C++ and SystemVerilog IPC and Shared Memory Setup
  - ❑ Controller TB needs to run C++ functions to set up IPC, set up shared memory, and tell the Receiver TB where to connect to shared memory.
- ❑ SystemVerilog DPI Send and Receive Tasks
  - ❑ DPI send tasks will send signal data to C++ software to be piped into shared memory of other IP for digesting.
  - ❑ DPI receive tasks will receive data by telling C++ software to read from shared memory. These tasks halt simulation progress to provide a pseudo-synchronous behavior between IP simulations.
- ❑ C++ Send and Receive Functions
  - ❑ Functions utilized by SystemVerilog testbenches for sending signal data to shared memory of the opposite IP and for reading its own shared memory.
- ❑ C++ and SystemVerilog Simulation Cleanup
  - ❑ Controller TB needs to run a C++ function to tell the Receiver TB that the simulation is ending. Receiver destroys shared memory, sends an ACK (acknowledgement), and ends its own sim. Controller TB receives the ACK, destroys shared memory, and ends its own simulation.

SIMULATION SOFTWARE INTERACTIONS

- ❑ Intercommunication between parallel simulations happens by having one simulation send a packet while the other waits to receive a packet.
- ❑ Simulations are halted when waiting to receive a packet in shared memory.
- ❑ Testbenches switch roles after sending/receiving packets


Setup
IPC Active
Receive packet, send packet, wait for packet
...
else
...
Receive packet, send packet and wait for packet
else
If(endsim)
If(endsim)
Receive packet, send FINISH packet, wait for response
...
Finish

We would like to thank Vijayakrishnan Rousseau for his technical guidance and valuable knowledge in the fields of validation and verification.

[1] "IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," in *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)* , vol., no., pp.1-1315, 22 Feb. 2018, doi: 10.1109/IEEESTD.2018.8299595.
[2] The Open Group, "POSIX - Base Specifications, Issue 7," IEEE Std 1003.1-2017, The Open Group, 2018. [Online]. Available: https://pubs.opengroup.org/onlinepubs/9699919799/