Multi-Die Distributed Simulation - Next Generation Validation Framework

SNUG INDIA 2024

Ravishankar Ramaswamy
Principal Engineer
Intel

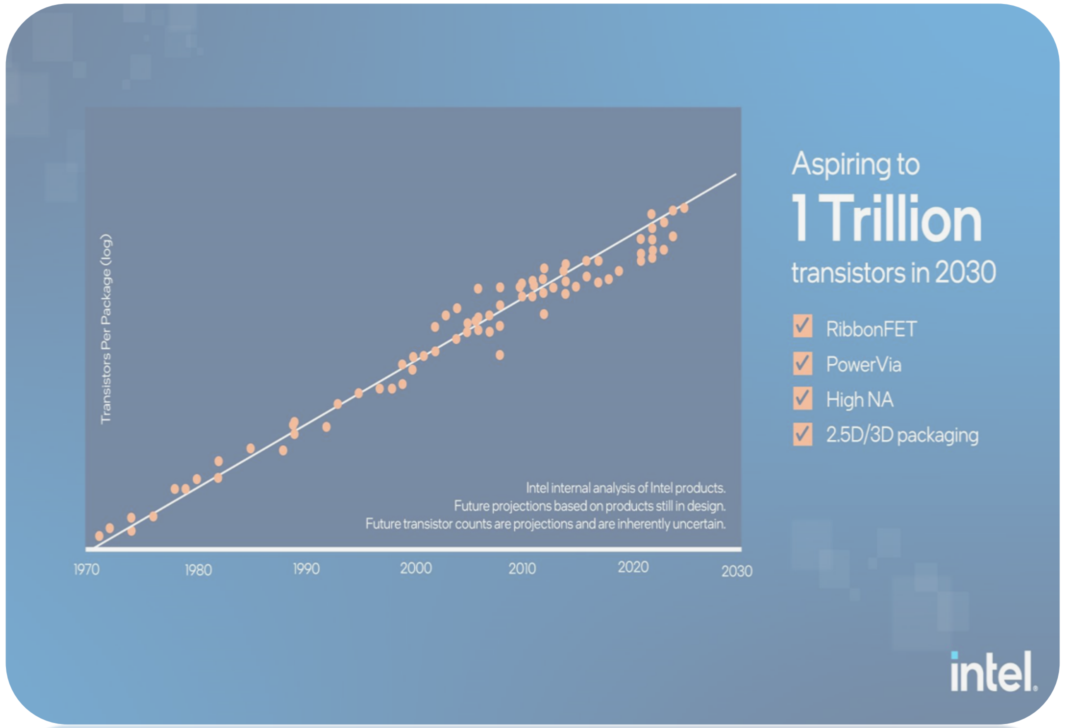
Agenda

- Problem Statement
- Requirement
- Solution - VCS Distributed Simulation
- Results & Summary
- Future enhancements

Problem Statement

Problem Statement

Evolution of Designs



- Rapid increase in design sizes and impact on PPA goals
- Reducing yield due to reticle limit of manufacturing equipment – Multi-die solutions, independent evolution of dies(process/nodes)

Validation challenges

Traditional Monolithic approach:

- Increase in compute requirements(>128G machines)
- Significant TAT for Build+Simulation(~24hrs)

- TB size explosion with newer topologies

Independent evolution of dies(process/nodes) leads to

- Name collisions – IP/TB
- Different versions - VIP, TB packages etc.

Incremental updates to design

- Reintegration at SoC

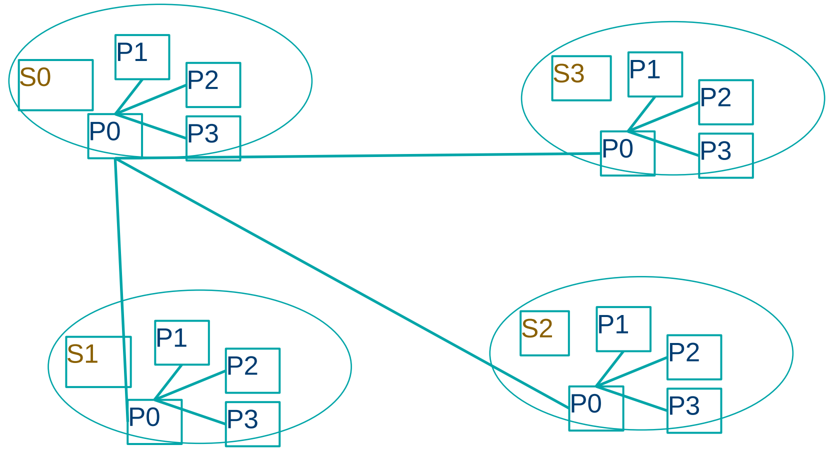
Evolve a nimble solution

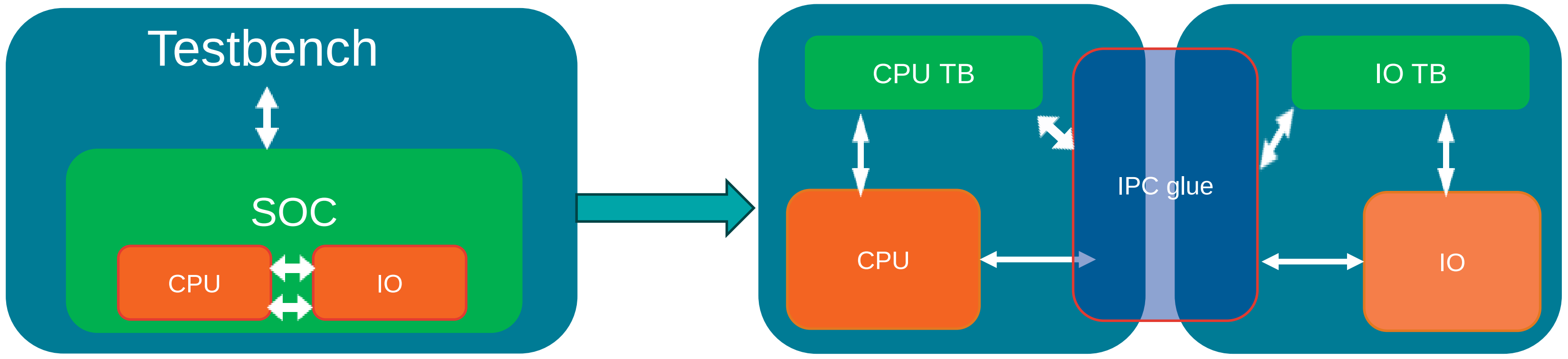
Requirement

Requirements

Disaggregation of dies

- Think design evolution - Divide monolithic setup into multiple chiplets
- Make use of existing TB at Subsystem/IP level
- Occupy less memory compute machines
- “Connect” dies through a simple IPC glue



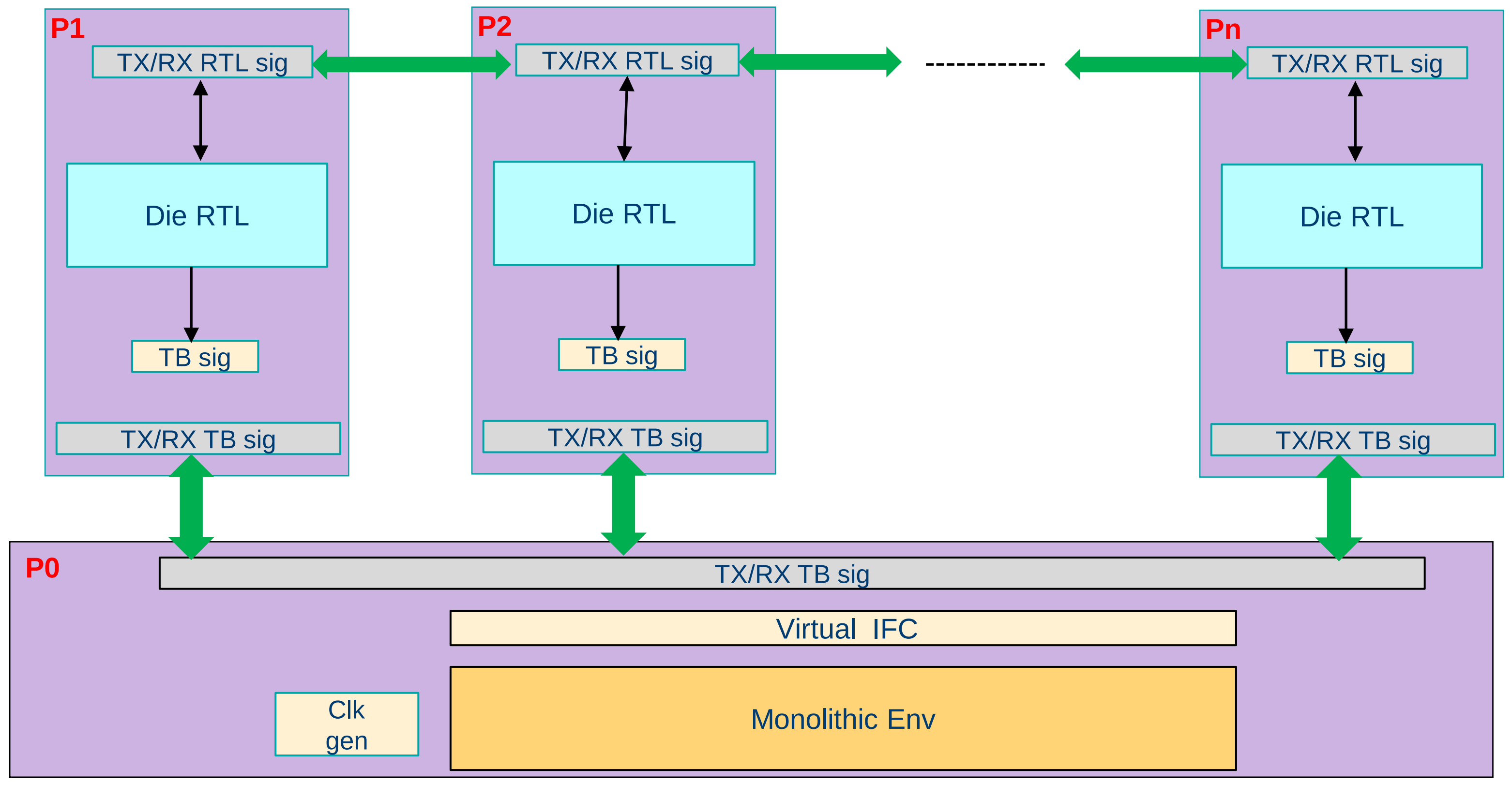


Monolithic

Disaggregated
Distributed

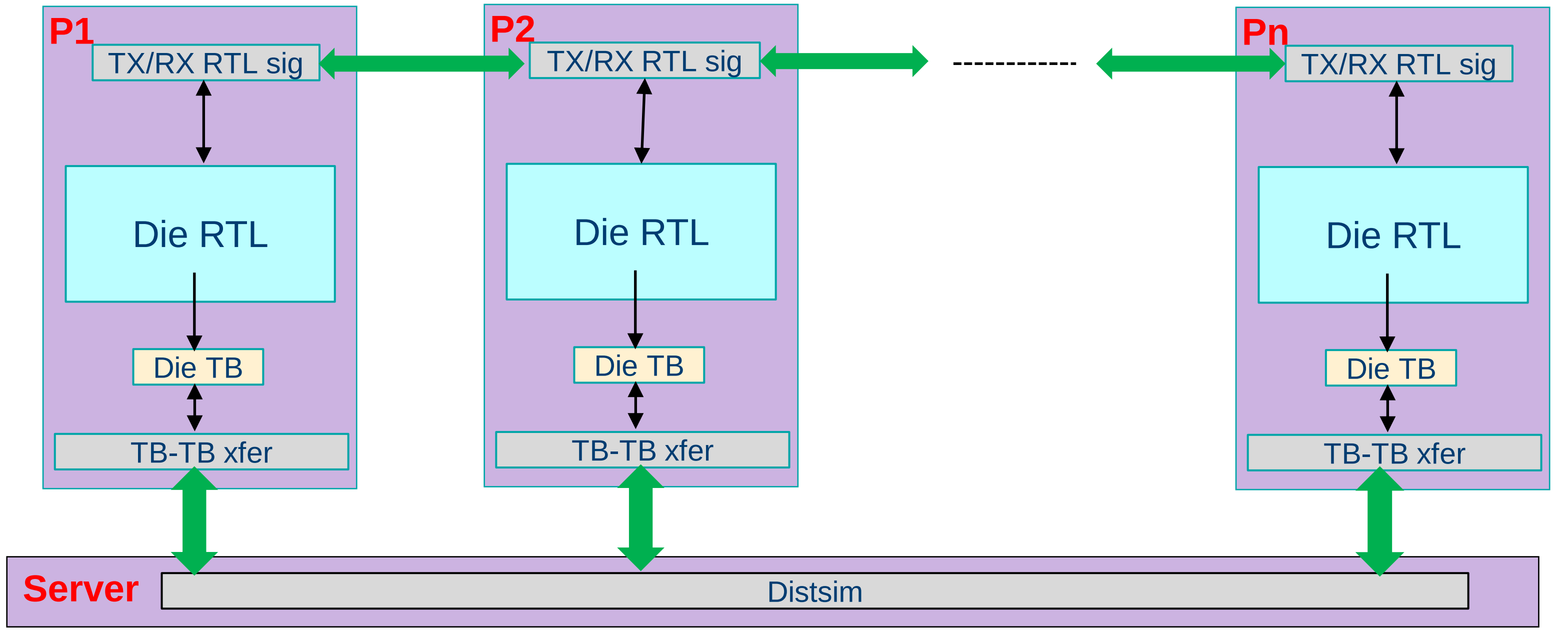
Disaggregated Simulation – Use Cases

Monolithic VAL – Disaggregated RTL



Disaggregated Simulation – Use Cases

Die VAL – Reuse at SoC (Typical Usage)



Challenges in Disaggregated Simulation

Typical challenges in custom/inhouse solution

RTL Synchronization
- Connectivity should replicate true die-to-die signal connectivity
- Support for multi clock synchronization

TB Data Transfer
- Flexible like Monolithic stimulus
- Deterministic data transfer
- Maintenance of Val code

TB Phase Synchronization
- Synchronization of UVM/User phases across dies
- Deterministic phase synchronization

Regression and Debug Productivity
- Low Impact to regression methodology
- Less debug impact
- Faster turnaround – Bug fix val

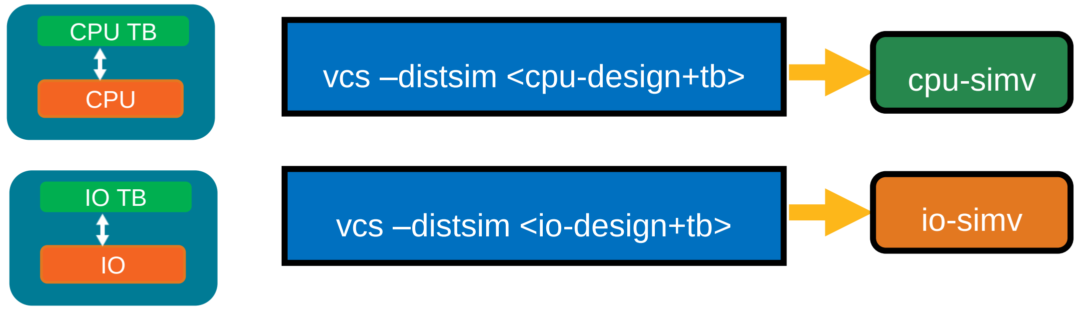
Penalty for morphing a monolithic to disaggregated simulation < 5%

Solution

VCS Distributed Simulation

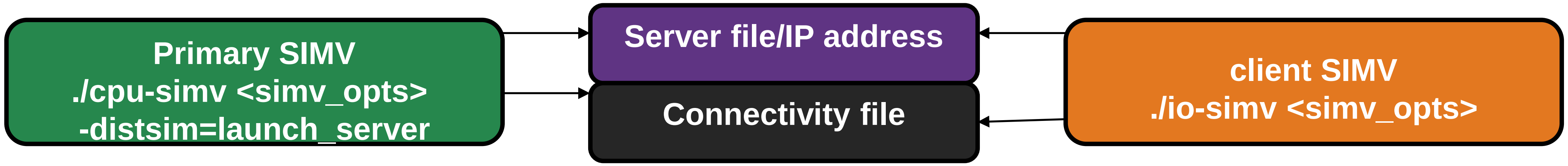
Use Model

- **Compilation**
  - -distsim needs to be used during elaboration.



✓ Benign

- **Simulation**
  - Primary simv should be launched with switch (-distsim=launch_server) which invokes separate process to control the communication



VCS Distributed Simulation RTL Synchronization

Simple RTL synchronization with connectivity file

- Connectivity file:

```
sync_interval: 100ps
s0.soc_tb.cpu.sig1 = s1:soc_tb.io.sig1
s1.soc_tb.io.sig2 = s0:soc_tb.cpu.sig2
s0.soc_tb.cpu.sig3 = s1:soc_tb.io.sig3
```

```
sync_signal: s0:soc_tb.cpu.clk
s0.soc_tb.cpu.sig1 = s1:soc_tb.io.sig1
s1.soc_tb.io.sig2 = s0:soc_tb.cpu.sig2
s0.soc_tb.cpu.sig3 = s1:soc_tb.io.sig3
```

- Synchronization can be clock based(sync_signal:<clock signal>) or time based(sync_interval)
- Communication between the simv's occurs during sample/drive phases
  - All the loads would get sampled together in Sample Phase and driven in Drive Phase
- Support of multiple clock sync signals in the connectivity file

RTL Sync protocol with Multiple Clocks

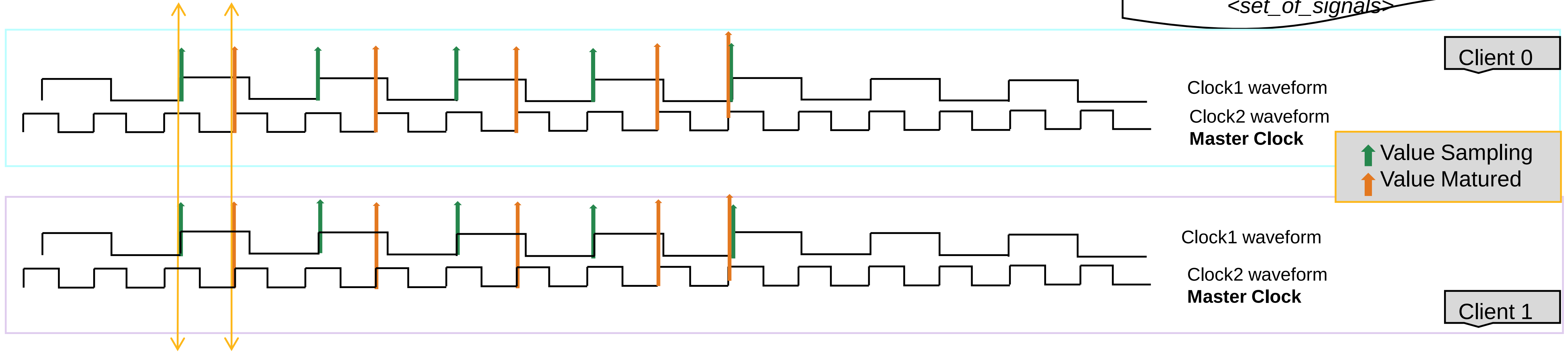
Master Clock: Fastest clock to be used for sampling

Config File

master_sync_signal: posedge Clock2
  <set_of_signals>
  <driver_sig> = <load_sig>
sync_signal: posedge Clock1
  <set_of_signals>

@Clock1: Associated signals would be sampled

@Clock2: Associated signals would be sampled
- Clock1's load signals would be sent to Client 1
- Clock2's load signals would be sent to Client 1
- Clock1's and Clock2's driver signal would get matured values from Client 1



@Clock1: Associated signals would be sampled

@Clock2: Associated signals would be sampled
- Clock1's load signals would be sent to Client 0
- Clock2's load signals would be sent to Client 0
- Clock1's and Clock2's driver signal would get matured values from Client 0

VCS Distributed Simulation TB Phase Sync

Synchronization for User defined and UVM phases

- **IPC for Test Bench Phase Sync: User defined phases**
  - Synchronization is done through code extensions by adding macro
  - Macro `VCSDISTSIM_PHASE_SYNC(PHASE_NAME) needs to be added at each sync point
  - Waits until all client simv’s reach same phase
  - TB resumes when all client simv’s are synced

- **IPC for Test Bench Phase Sync: UVM Runtime default phases**
  - Implicit UVM phase synchronization across simv’s with predefined UVM component class **dist_tbsync_comp**
  - TB phase order should be identical in client TBs
  - There can be additional Client specific Phases :-

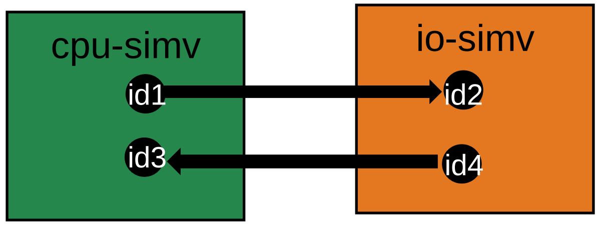
    - Sync not available for phases unavailable in other Clients

    - Sync for common phases can be achieved using runtime flag or connectivity file

VCS Distributed Simulation TB Data Transfer

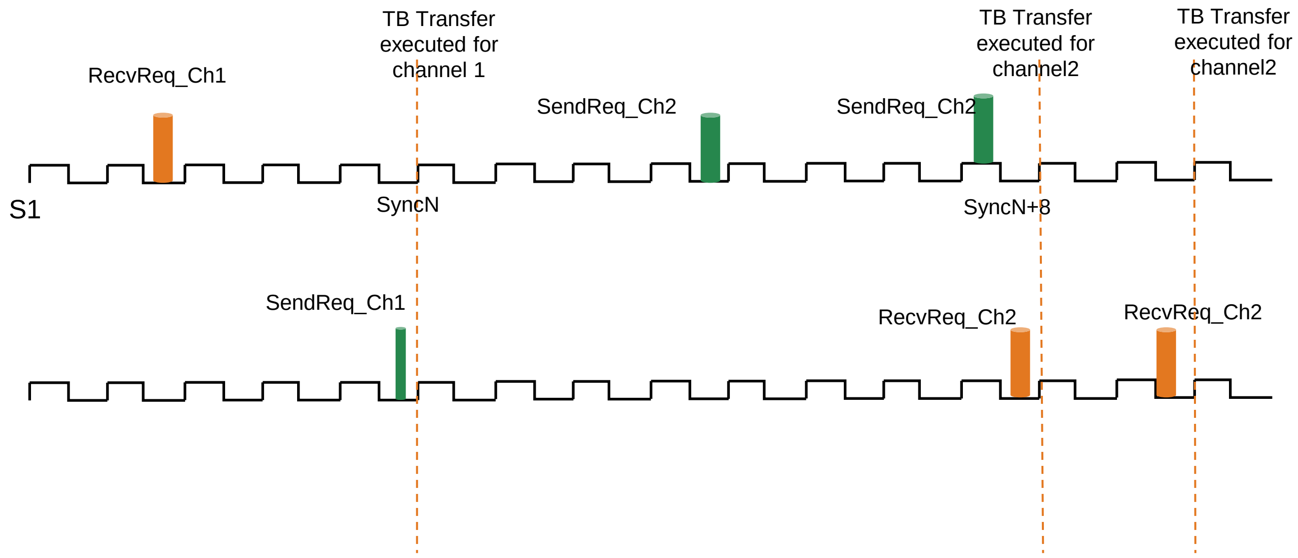
Val data transfer across clients

- Test Bench Data Transfer of Class Objects is through bit stream
- Send/receive TB data is through VCS distsim API's(`VCSDISTSIM_TB_SEND/ `VCSDISTSIM_TB_RECV)
- At sender, generate bit array from the transaction class object
- At receiver, bit array is used to fill the transaction class object



Connectivity File:

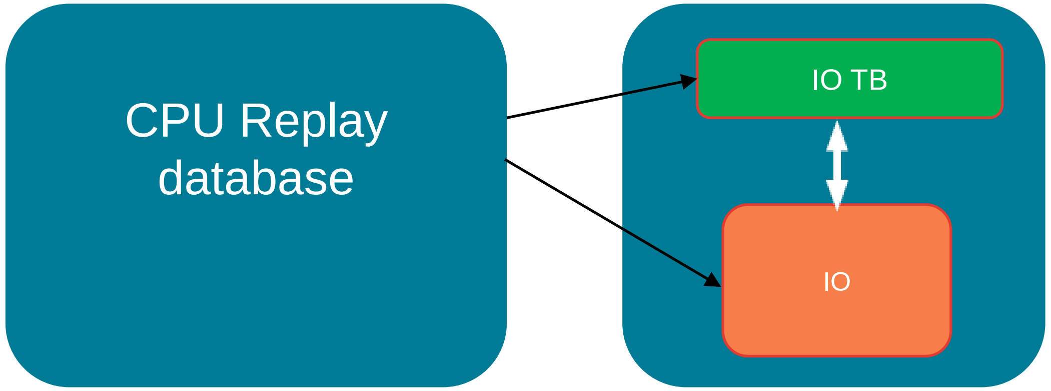
```
tb_trans=id1:s0::id2:s1
tb_trans=id4:s1::id3:s0
```


TB Transfer executed for channel 1
TB Transfer executed for channel2
TB Transfer executed for channel2
RecvReq_Ch1
SendReq_Ch2
SendReq_Ch2
S1
SyncN
SyncN+8
SendReq_Ch1
RecvReq_Ch2
RecvReq_Ch2

VCS Distributed Simulation Save Replay

Debug Productivity

- Enables simulation of single Simv with stimulus captured from other clients during save run
- Helpful for debugging if issue is present in only one die
- Save mode is used to capture all RTL/TB receive calls with respect to sync point and logged into replay database
- During Replay Mode, one can force the RTL signals which are part of connectivity file



Results & Summary

VCS Distributed Simulation

Results & Summary

- Ease of integration
  - Bringup of VCS distributed simulation in <2days
  - Complete regression results < 1week
- Performance improvement over existing solution
  - Gains for both Simulation time and memory
- Deployment
- Debug Productivity
  - 1.5X Runtime gain with replay mode.
- Scalability – Easily scalable to next projects

Future Enhancements

VCS Distributed Simulation

Challenges/Next steps

- Increase in sync time penalty with increase in number of simv’s
- Support of interface modports connections is not available; add individual signals manually
- Only bit stream is supported for TB data transfer and needs force cap on entire design

Future Enhancements

- Support of interface modport connection
- Multiple master clock synchronizations
- User assisted partitioning
- Unified Debug with Verdi
- Consolidated logs/coverage i.e merge across vdb’s
- Cloud Support

snug

THANK YOU

Our Technology, **Your Innovation**™

Sample code for TB Sync and TB data transfer

Driver: send transaction from one-die to other-die and for receiving response.

```
int dieid;
bit rsp_bit_array_d0[];
bit rsp_bit_array_d1[];
bit rsp_bit_array_d2[];
mp_tb_seq_item local_rsp;
bit rsp_bit_array[];

$value$plusargs("DIE_ID=%d",dieid);
if (dieid==0)
begin
fork
begin
forever begin
`VCSDISTSIM_TB_RECV("RECV_RSP_D0",
rsp_bit_array_d0);
end
end
begin
forever begin
`VCSDISTSIM_TB_RECV("RECV_RSP_D1",
rsp_bit_array_d1);
end
end
begin
forever begin
`VCSDISTSIM_TB_RECV("RECV_RSP",
rsp_bit_array_d2);
end
end
join_none //fork
end //dieid=0
```

Sequencer: receive transaction from driver and send response to driver

```
$value$plusargs("DIE_ID=%d",dieid);
if (dieid==1)
begin
fork
begin
forever begin
`VCSDISTSIM_TB_RECV("RECV_REQ1",
req_bit_array_n);
end
end
begin
forever begin
`VCSDISTSIM_TB_RECV("RECV_REQ2",
req_bit_array_s);
end
end
join_none //fork
end



if (dieid==1) begin
`VCSDISTSIM_TB_SEND("SEND_RSP",
req_bit_array);
end
```

Task based Phase Sync

```
virtual task body();
super.body();
……
`uvm_info(get_type_name(),">> reset distsim_phase_sync
sequence", UVM_NONE);
$display("SNPS VCSDISTSIM Phase Sync Before CP0");
`VCSDISTSIM_PHASE_SYNC(CP0);
$display("SNPS VCSDISTSIM Phase Sync After CP0");
if (get_current_test_phase() == "reset_phase")
begin
$display("SNPS VCSDISTSIM Phase Sync Before CP1");
`VCSDISTSIM_PHASE_SYNC(CP1);
$display("SNPS VCSDISTSIM Phase Sync After CP1");
……….
endtask
```