Software Driven Hardware Verification
A UVM/DPI Approach

Milan Purohit <mpurohit@solarflare.com>
Santanu Bhattacharyya <sbhattacharyya@solarflare.com>
Puneet Goel <puneet@coverify.com>
Amit Sharma <amit.sharma@synopsys.com>

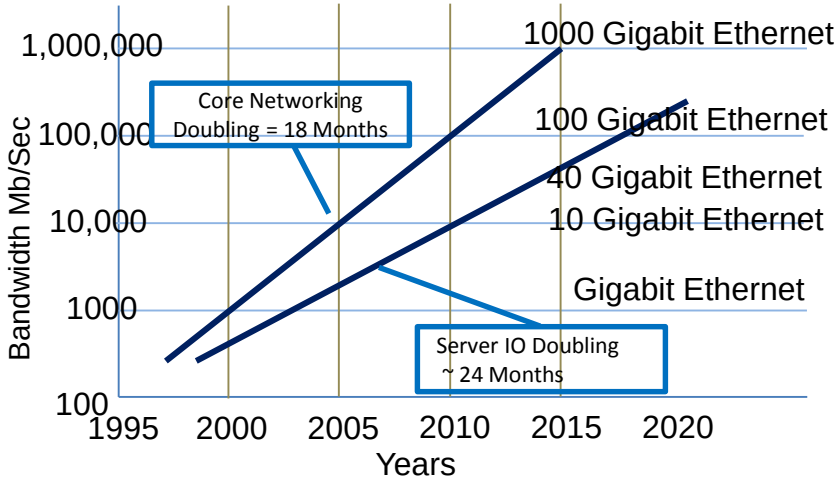
accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Agenda

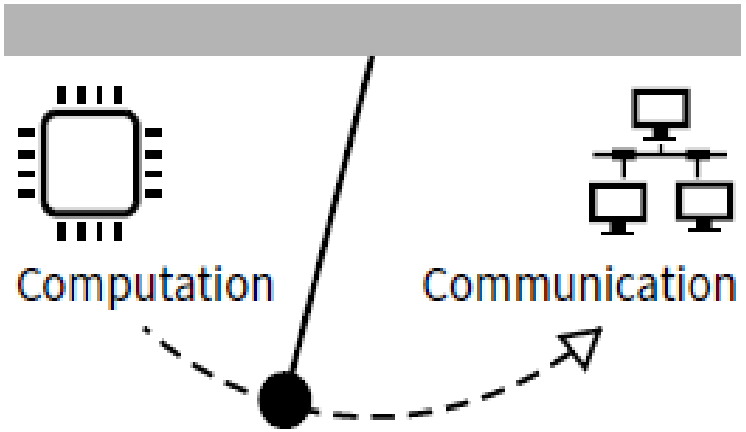
- Computation Vs Communication
  - Introduction Of The Solarflare System And The DUT
- Verification Challenges
- Software Verification Challenges
- Solution For Taking Unit Test To System Test
  - Software Test Environment
  - UVM Based Block Verification
  - System Level Software Driven Hardware Verification TE
- Unified Log Generation
- Conclusion
- Q & A

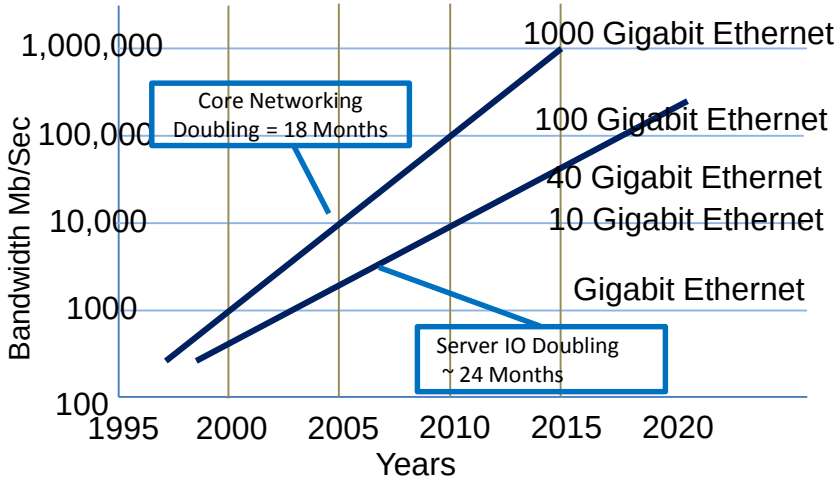
Computation Vs Communication



Computation Vs Communication

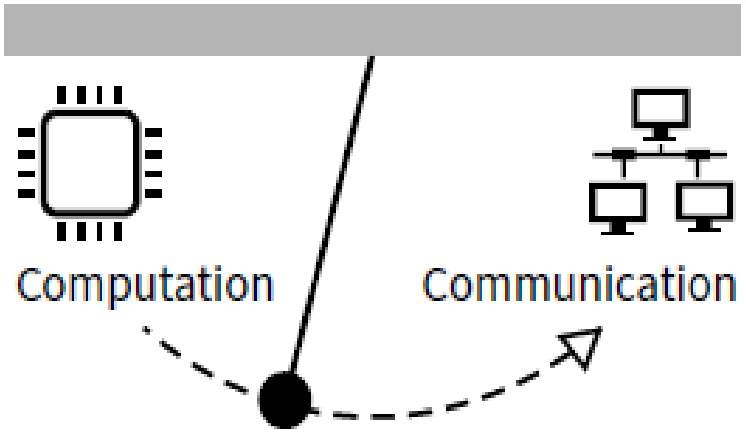
- Processor speed and network data speed have grown quite independent of each other over years

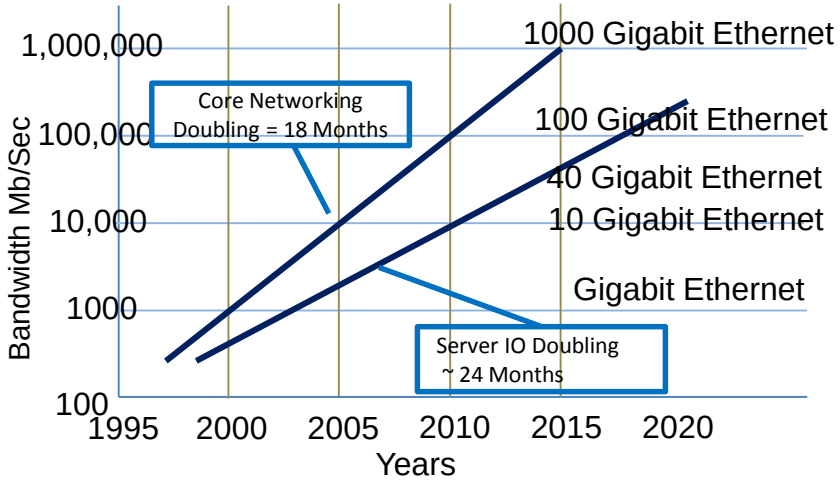




Computation Vs Communication

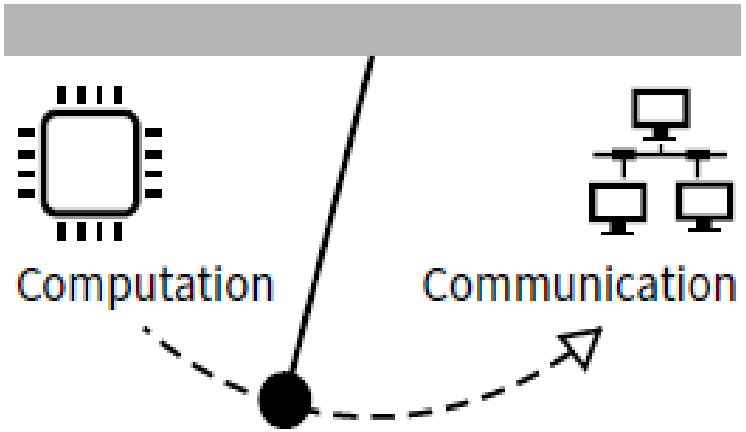
- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster

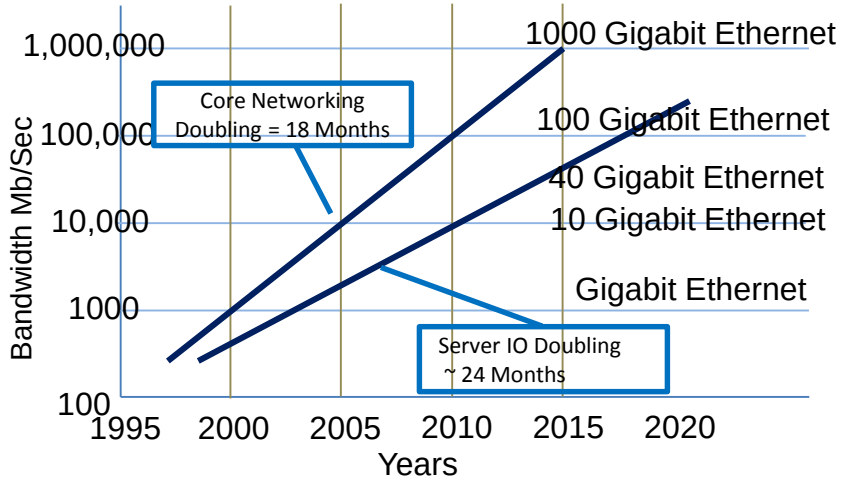




Computation Vs Communication

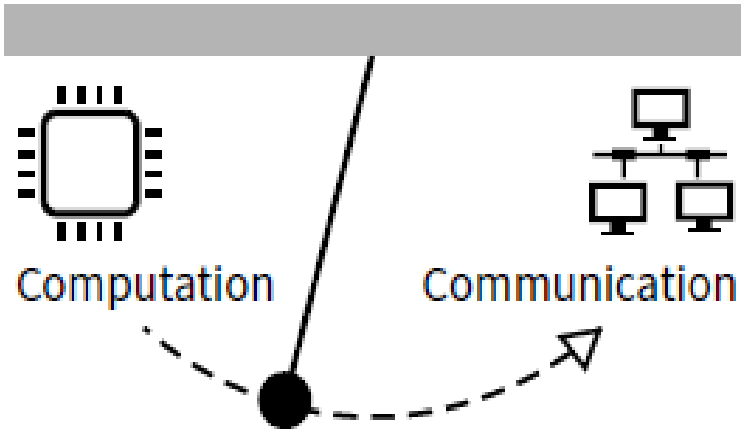
- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster
  - But network speed was usually not fully utilized by machines

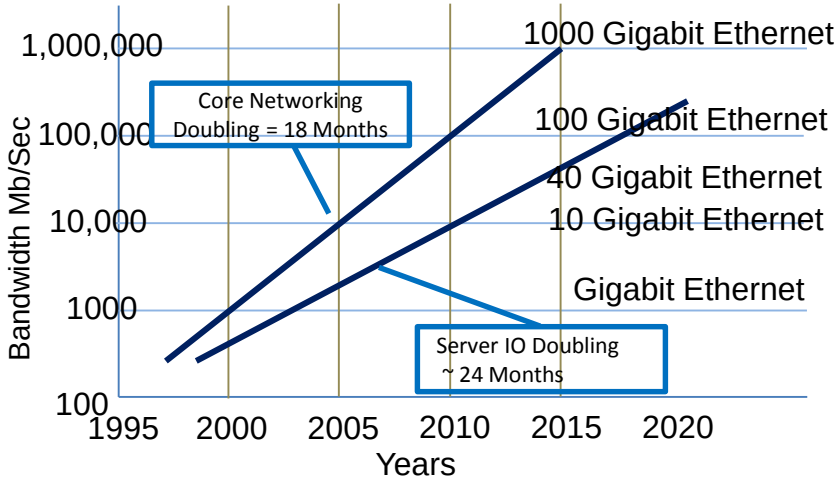




Computation Vs Communication

- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster
  - But network speed was usually not fully utilized by machines
  - Till year 2002, Ethernet ports in Linux systems had interrupt based handling

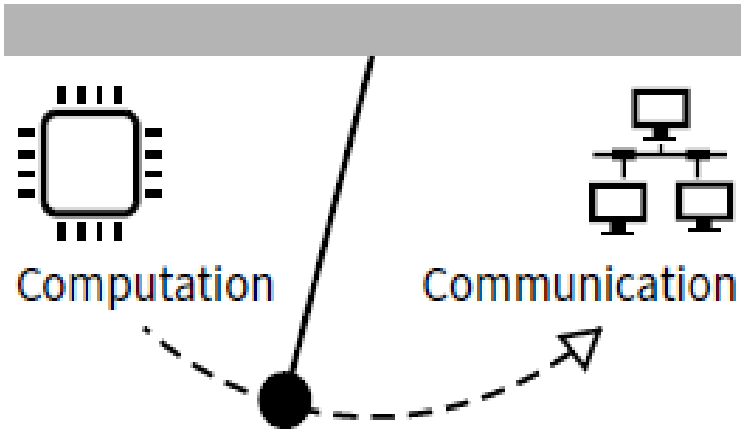


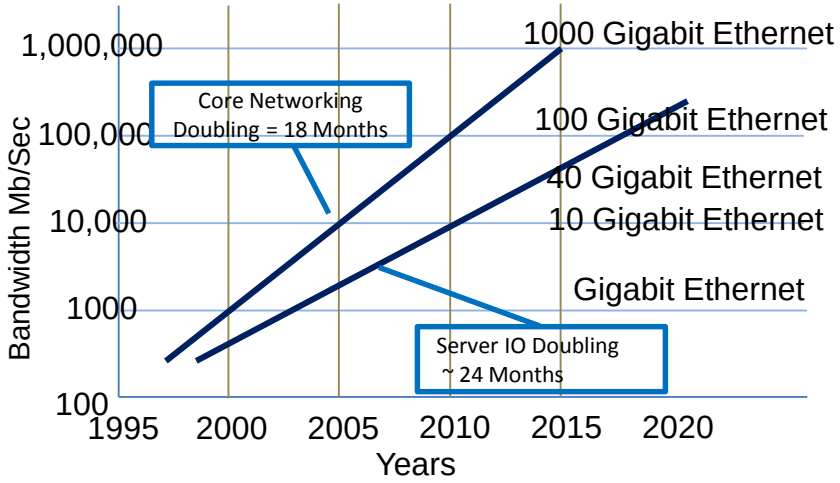


Computation Vs Communication

- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster
  - But network speed was usually not fully utilized by machines
  - Till year 2002, Ethernet ports in Linux systems had interrupt based handling

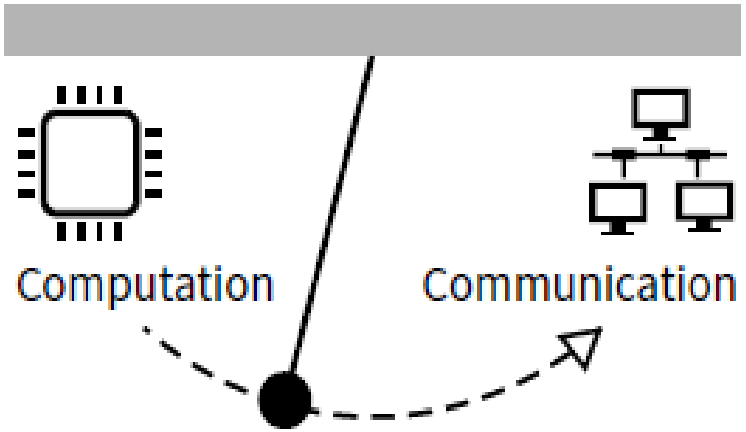
Can Processor Handle Data Now?





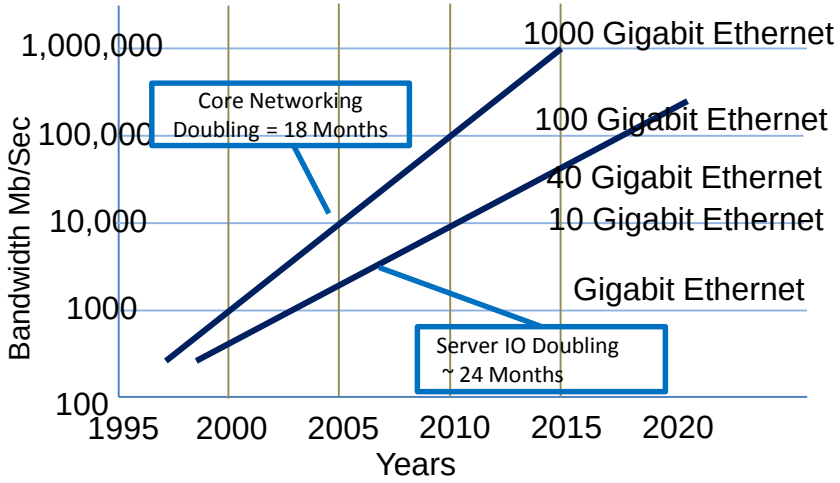
Computation Vs Communication

- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster
  - But network speed was usually not fully utilized by machines
  - Till year 2002, Ethernet ports in Linux systems had interrupt based handling



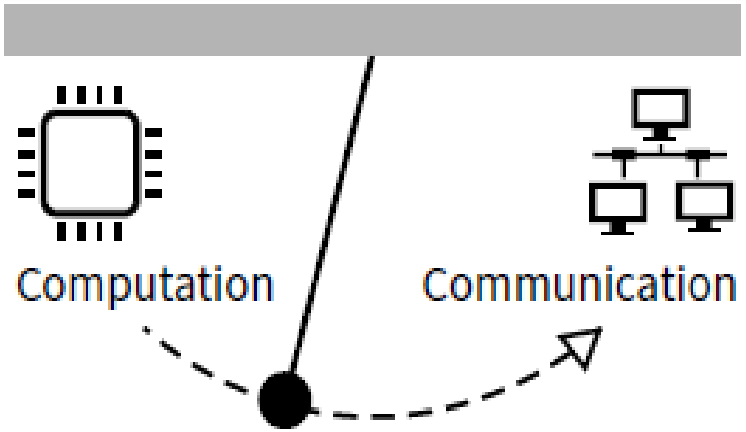
Can Processor Handle Data Now?

- Communication speed hitting 400Gbps and 1000Gbps



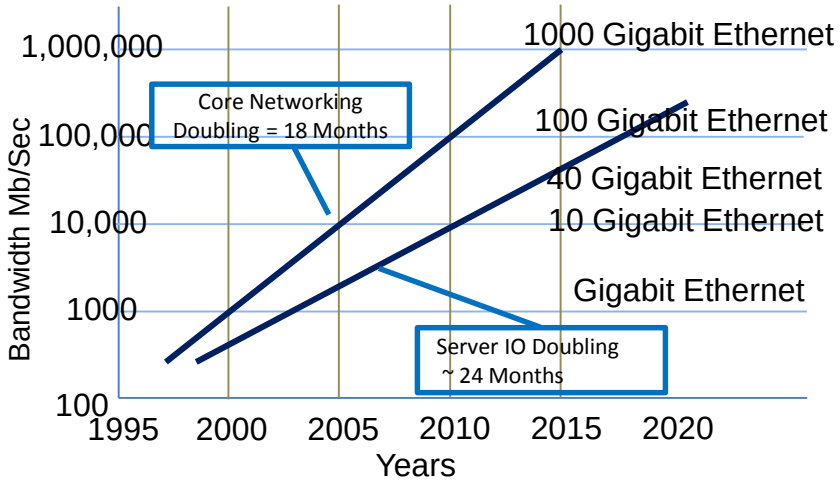
Computation Vs Communication

- Processor speed and network data speed have grown quite independent of each other over years
- With introduction of every new IEEE 803.11 standard, network became faster and faster
  - But network speed was usually not fully utilized by machines
  - Till year 2002, Ethernet ports in Linux systems had interrupt based handling



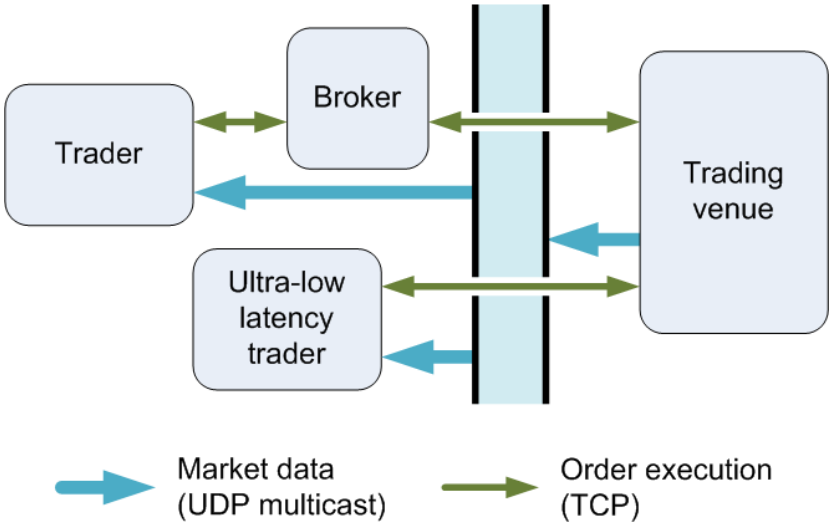
Can Processor Handle Data Now?

- Communication speed hitting 400Gbps and 1000Gbps
- Thanks to demise of Moore's Law, processor clock has stagnated at 3GHz since 2005



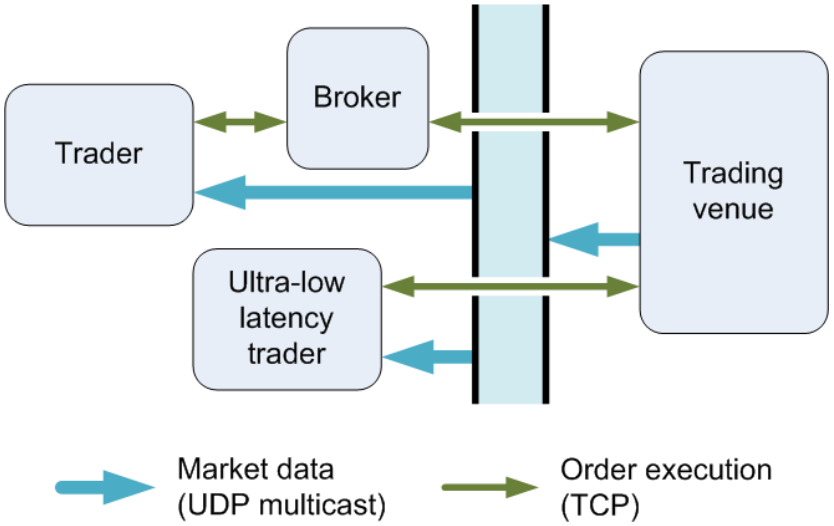
Low Latency Applications

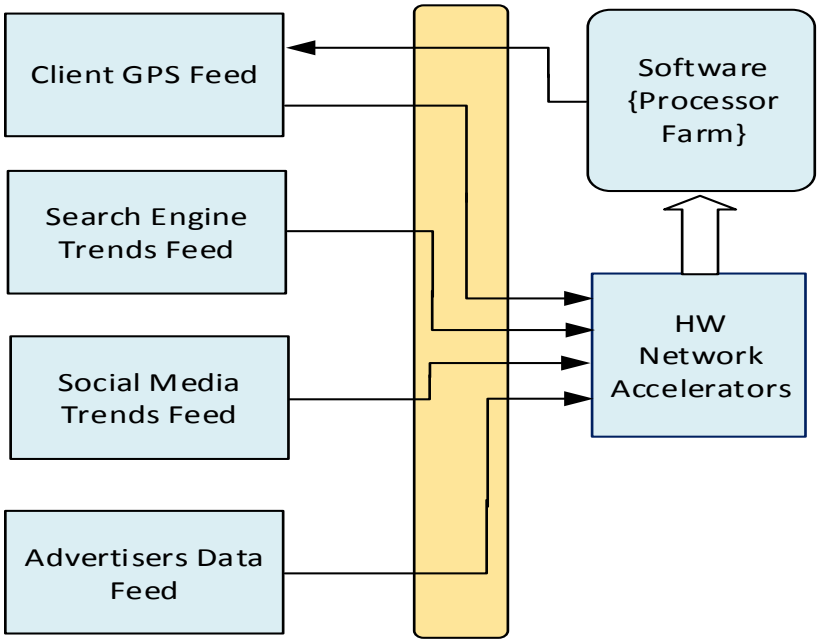
- Stock Market traders have million dollar financial incentive to reduce latency



Low Latency Applications

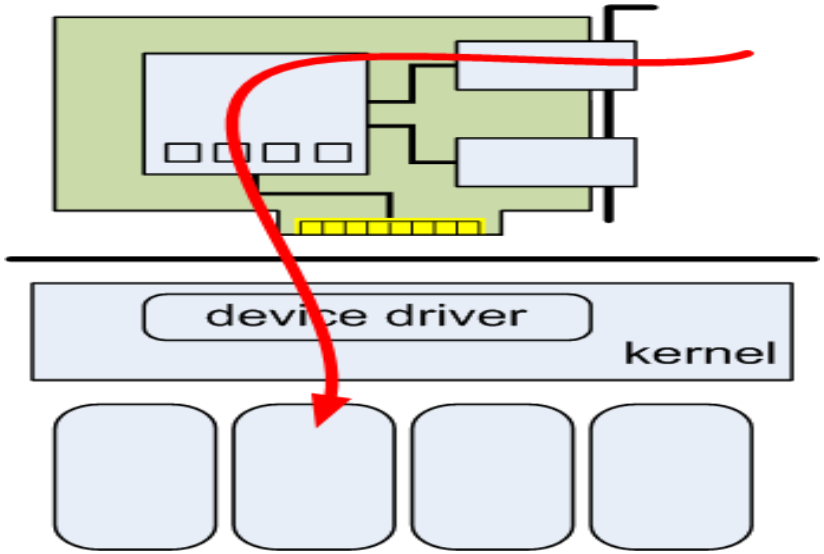
- Stock Market traders have million dollar financial incentive to reduce latency



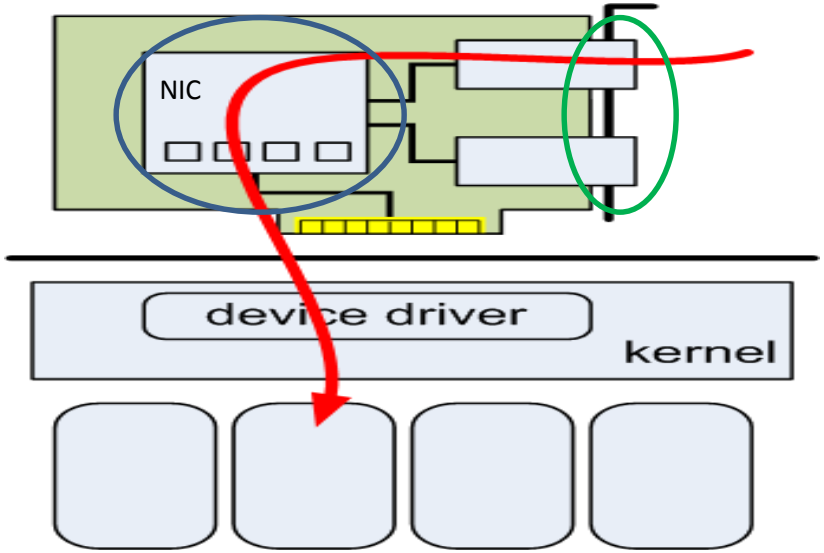


- Online advertisement agents have only a few milliseconds to decide what advertisement to upload

Traditional Network Traffic Processing

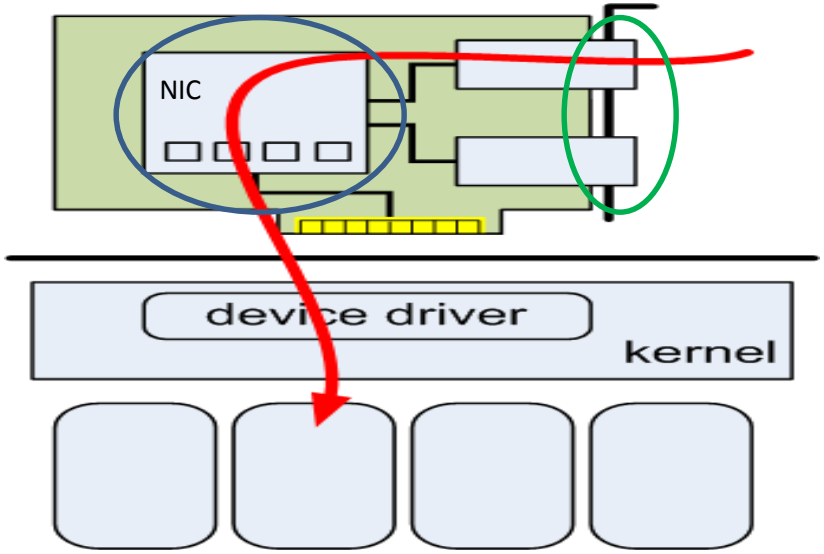


Traditional Network Traffic Processing



- Traditionally, network traffic has been processed using software the NIC card interfaces the network/wire side through Ethernet interface and software taking care of the packet processing

Traditional Network Traffic Processing



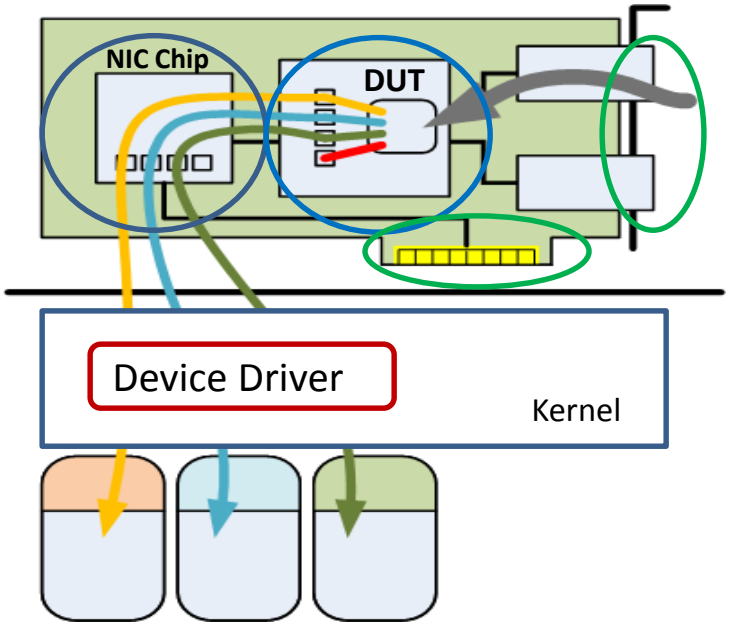
- Traditionally, network traffic has been processed using software the NIC card interfaces the network/wire side through Ethernet interface and software taking care of the packet processing

- An *Application Onload Engine* Processes Data Using Dedicated Hardware And Makes It Available To The Software Processor Farm

Application On Load Engine (AOE)

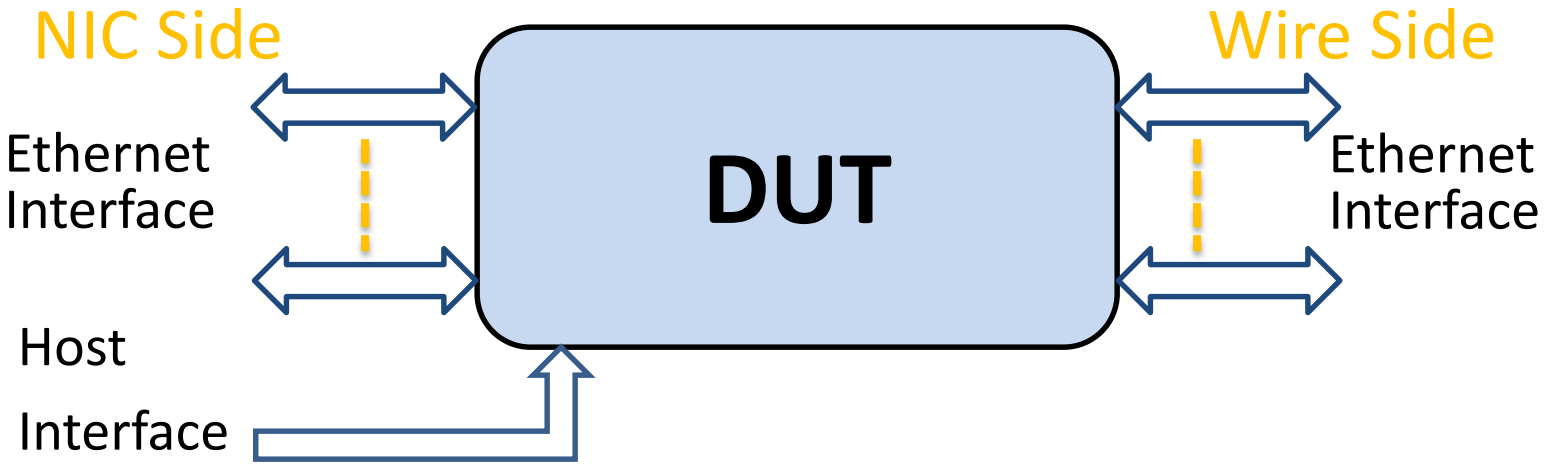
Hardware Accelerators for Data

- System consists
  - Of a NIC chip
  - An FPGA running A “user defined application or function”
- The NIC interfaces
  - Through PCIe on server host side
  - On the FPGA side it has 10G/40G Ethernet interface



- FPGA’s interface on the network side is again 10G/40G Ethernet interface
- The system performance is further increased by bypassing the kernel

The DUT And It's Interfaces



- Design under test (DUT) is an AOE application code programmed in FPGA
  - It has Ethernet interface on both input as well as output side

- Host interface for configuration and device status monitoring

Verification Challenges

Verification Challenges

Protocol Stack

- Network protocol stack has traditionally been processed by software
  - Traditionally, network hardware handled only MAC Layer protocol
  - Sufficient to send Ethernet packets with randomized payload
- With Application On-load engine, hardware processes Layer3 and Layer4 protocols as well
  - Hardware Verification requires generation of higher layer packet sequences as well

Verification Challenges

Protocol Stack

- Network protocol stack has traditionally been processed by software
  - Traditionally, network hardware handled only MAC Layer protocol
  - Sufficient to send Ethernet packets with randomized payload
- With Application On-load engine, hardware processes Layer3 and Layer4 protocols as well
  - Hardware Verification requires generation of higher layer packet sequences as well
- Software is tightly integrated with hardware accelerator
  - The application software can reconfigure the AOE at run time
  - To thoroughly test the interaction of software with hardware, it makes sense to verify the hardware and software together
  - Note that the hardware/software co-verification is not limited to HAL layer
  - Depending on application, there could be a need to drive even higher layer protocols (e.g.. session layer) as part of the stimulus

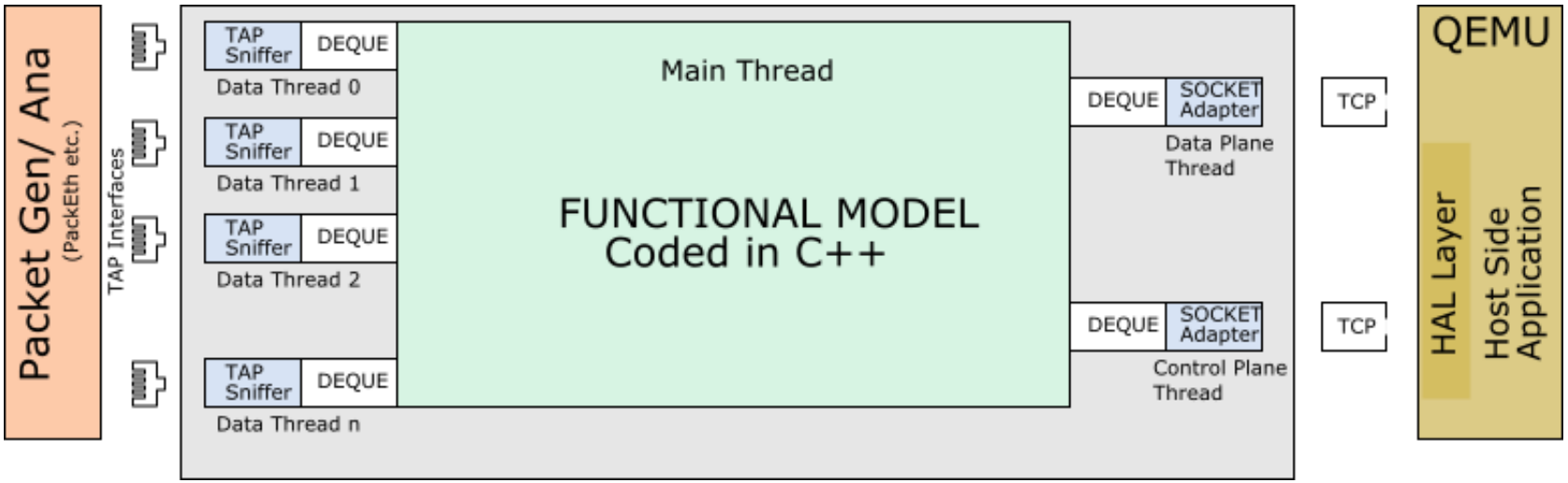
Software Verification Challenges

Software Verification Challenges

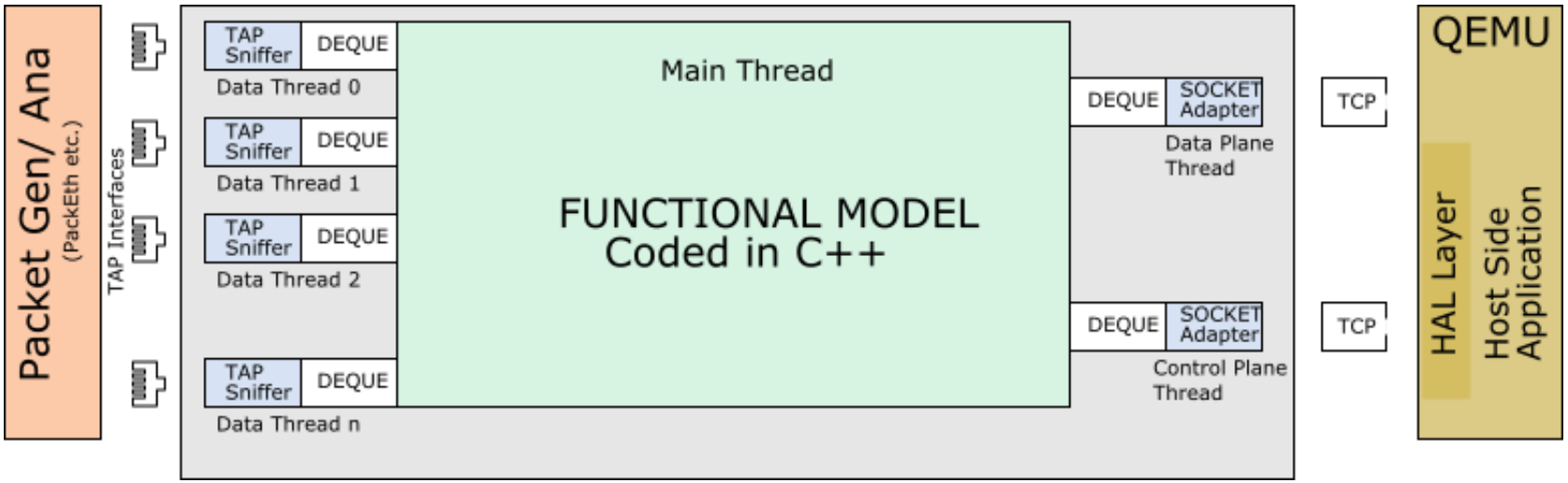
- And we think only verification team is busy working !
- Software team's start developing software drivers and in some cases, application software early in project cycle

- Since software team have "started earlier", they would require a DUT equivalent model which reacts to configuration and data applied on device interfaces
- Actual design is replaced by A software model coded in C++ which has multiple interfaces, for host access and data interface(s)
- Multi thread handling issues like race, starvation, deadlock & live-lock

Software Test Environment



Software Test Environment



- QEMU Is A Hosted Virtual Machine Monitor: Used For Emulating NIC Chip And It's MIPS Processor
- TCP Socket Is Used For Data And Host Protocol Transfers
- TAP Interface Is Virtual Interface Allows Numerous Linux Utilities To Be Hooked On To It, Like "*Packeth*", "*TCP Dump*", "*TCP Replay*" Etc.
- Since All The Interfaces Are Active Simultaneously
  - Model Is Implemented As A Threaded Code Assigning Separate Posix Thread For Each Interface
  - TAP Interfaces Have To Be Continuously Monitored For Data, Otherwise Data Will Be Lost

Thread Architecture of Software Model

Thread Architecture of Software Model

There Are Two Kinds Of Perils Of Multithread Applications: Thread Races And Thread Starvation

Avoiding Thread Races

- In Shared Memory Multithreaded Model, We Need To Protect Shared Data From Being Accessed By Two Threads Simultaneously
- We Used “Mutex” Module From C++ Boost Library
- To Make Threads Efficient, A Separate Mutex Lock Is Created For Each Shared Data Queue

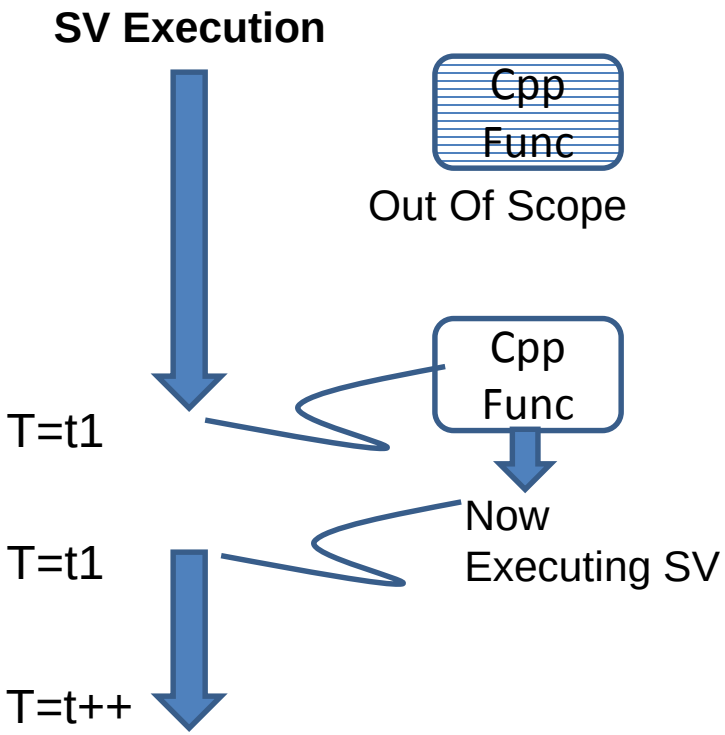
Avoiding Thread Starvation

- Each Interface Thread Leaves A Notification After Putting Data On Queue
- For Notifications, We Used Semaphores, C++ Does Not Provide A Semaphore Implementation, But It Is Easy To Code
- To Avoid Thread Starvation, The Main Thread Needs To Service The Queues In A Way That Ensures That All The Queues Get Its Attention
  - To This End, We Just Gave More Priority To Threads That Have Less Traffic Like The Control Plane Traffic

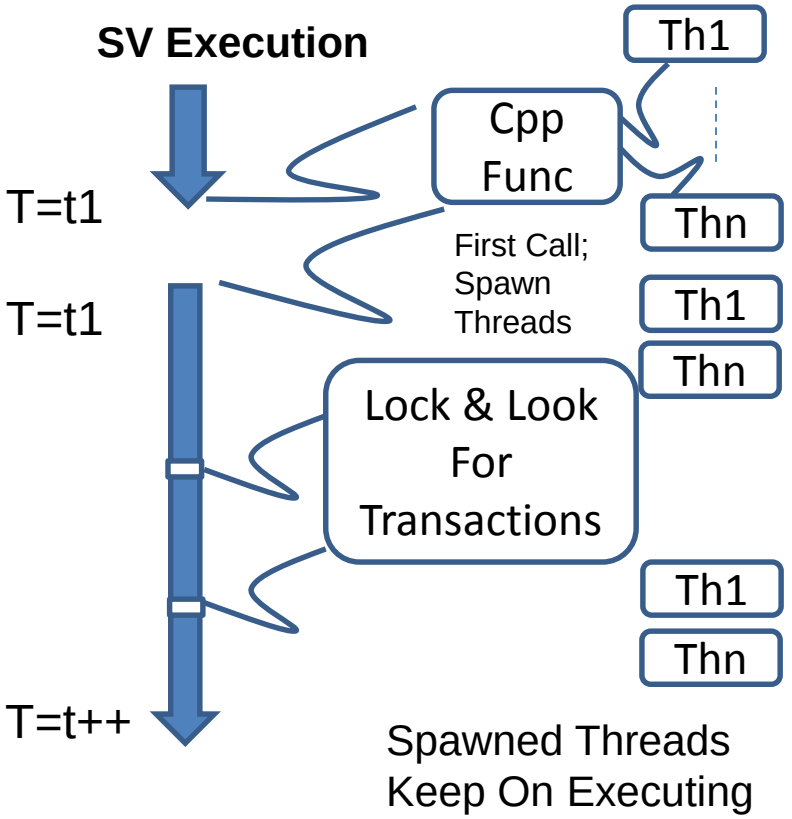
Hooking Up Simulator With Multi-Threaded Interfaces

- Simulator Runs On A Single Thread, So While It Executes Simulation Code, C World Execution Is Put On Hold
  - This Mechanism Cannot Handle Multi-Threaded Virtual Interfaces Discussed Above
- Solution Is To Create An Multi-Thread Handler Which Takes Care Of This
- Simulator Through DPI() Calls Attaches With The Main Thread Which In Turn Spawns All Other Independent Threads
- Main Thread Waits For Notification And Then Locks Each Queue One By One Looking For Transaction
- Mutex Locking : Uses Boost Mutex::scoped_lock -- C++ RAII -- Mutex Is Automatically Unlocked When The Scope Is Exited

Simulator Interfacing With Multi-Threaded Interfaces

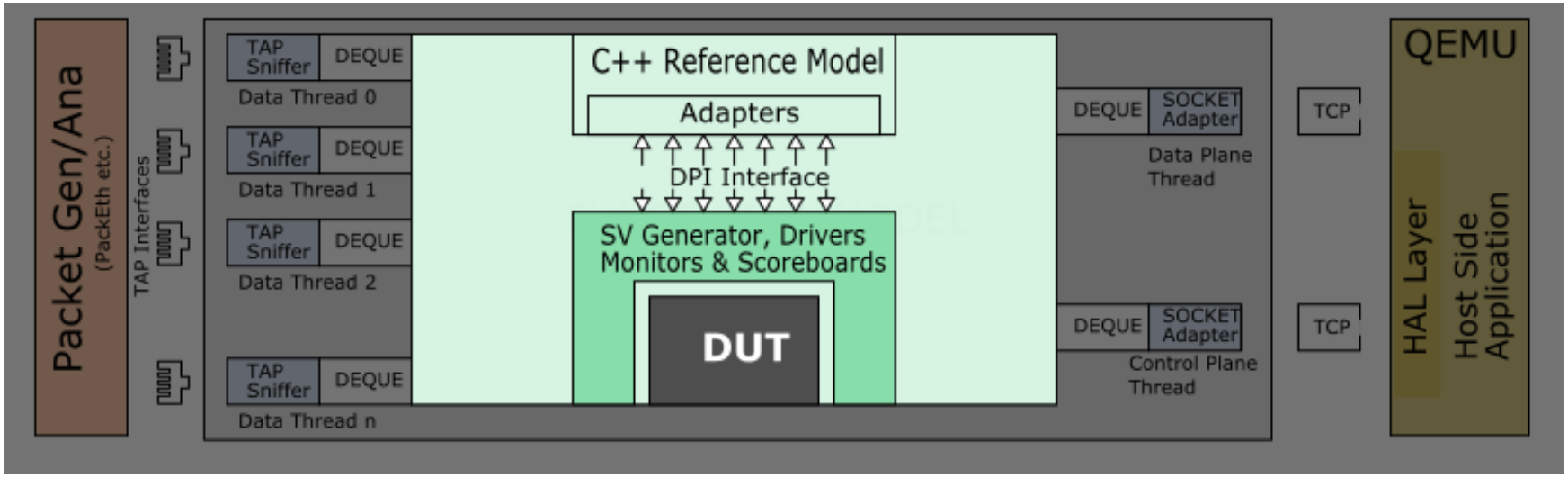


Normal DPI Based Flow

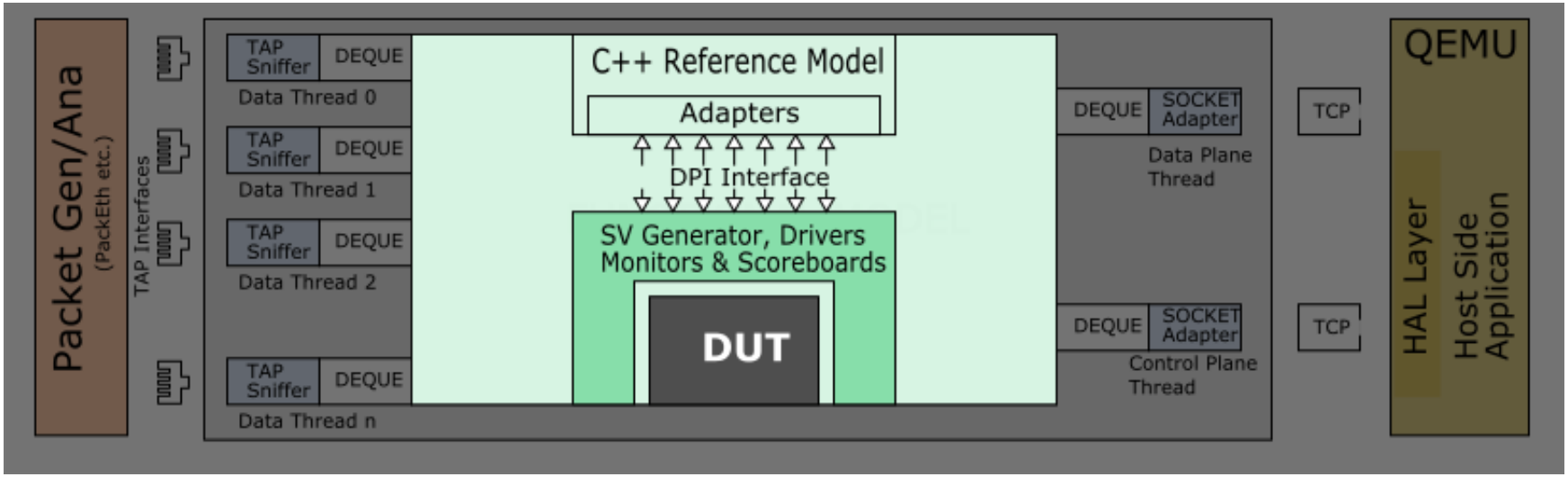


Multi Threaded Simulator Interfacing

UVM Based Block Verification

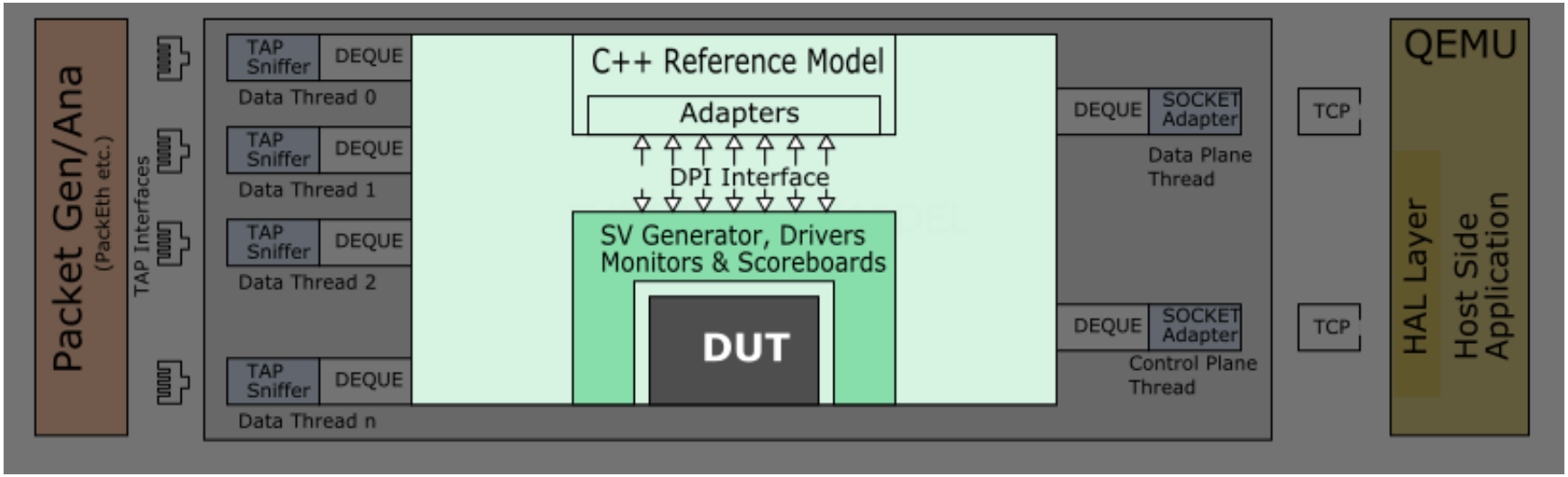


UVM Based Block Verification



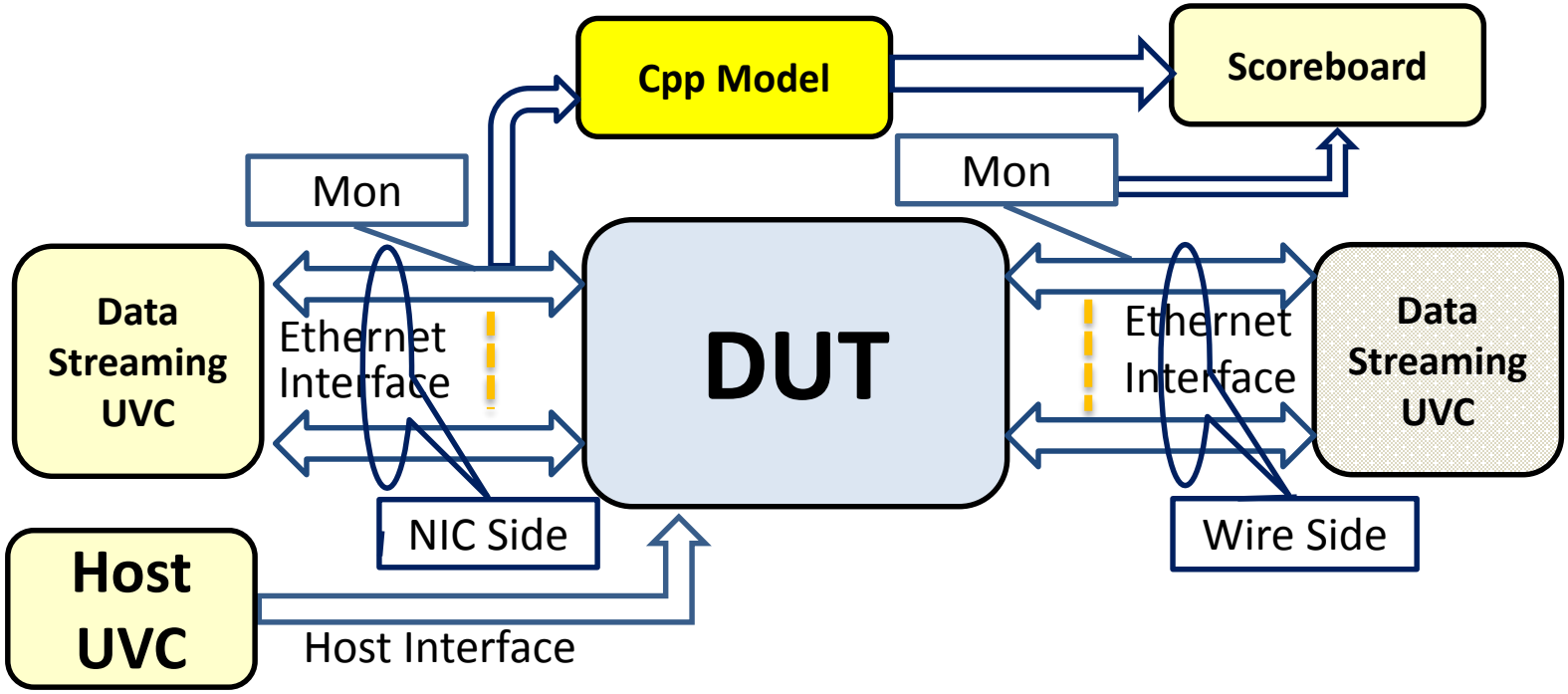
- The C++ model that we used for software testing was reused (sans the interfaces) for functional verification as A reference model

UVM Based Block Verification



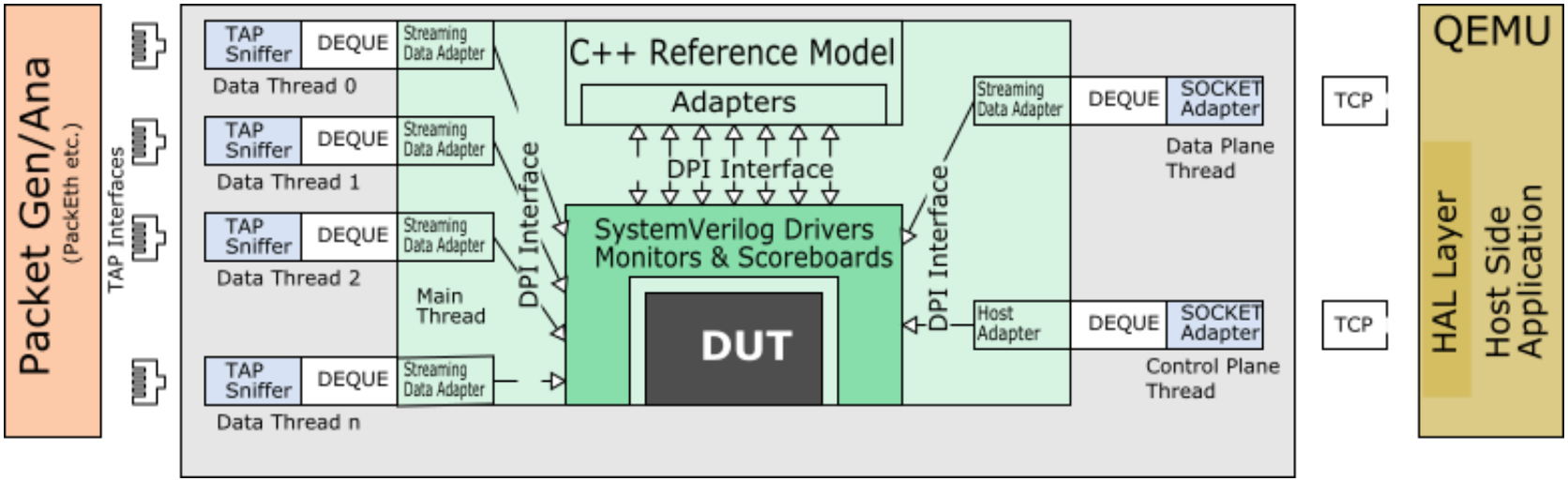
- The C++ model that we used for software testing was reused (sans the interfaces) for functional verification as A reference model
- Since the software interfaces are not involved, no concurrency is involved
- Instead the software model now deploys adapters to handle the DPI calls from UVM monitors
- The SystemVerilog simulator generates various randomized transaction sequences
- A response is created by the software model and sent to scoreboard

UVM Based Hardware Test Environment (TE)



- TE consists of packet generator(s) UVC (sequencer, BFM & monitor)
- A predictor model generating the expected data based on configurations and packet data
- Scoreboard which compares the expected data with actual data received from the DUT output monitors

System Level Software Driven Hardware Verification Test Environment



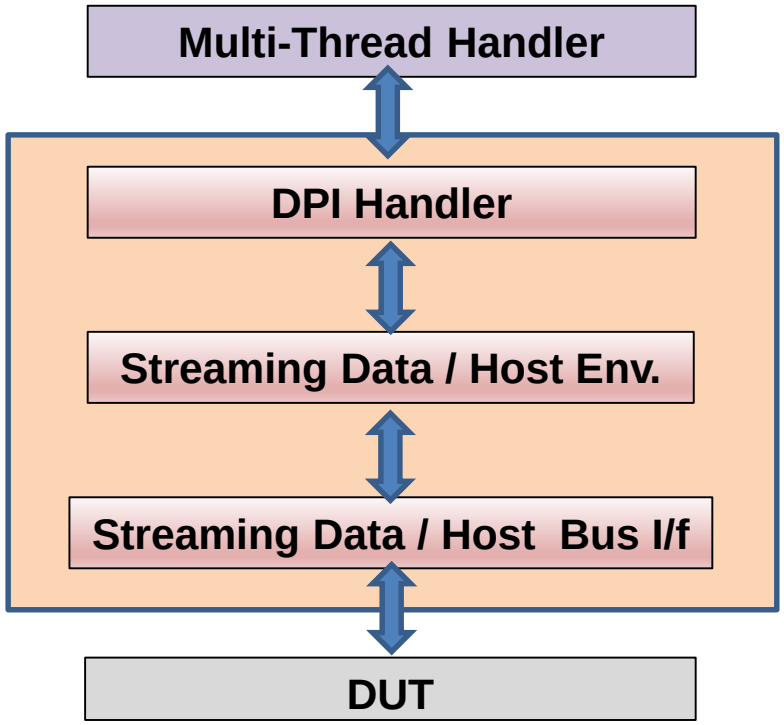
Generalized Co-verification Strategy

- Embedded software runs on QEMU
- Simulation runs on host machine
- Communication channels with packet generator/analyser is based on virtual network interface (tap socks)
- Hardware simulation interaction with tap socks based on DPI
- Hardware simulator is the master thread – pull protocol for the stimulus

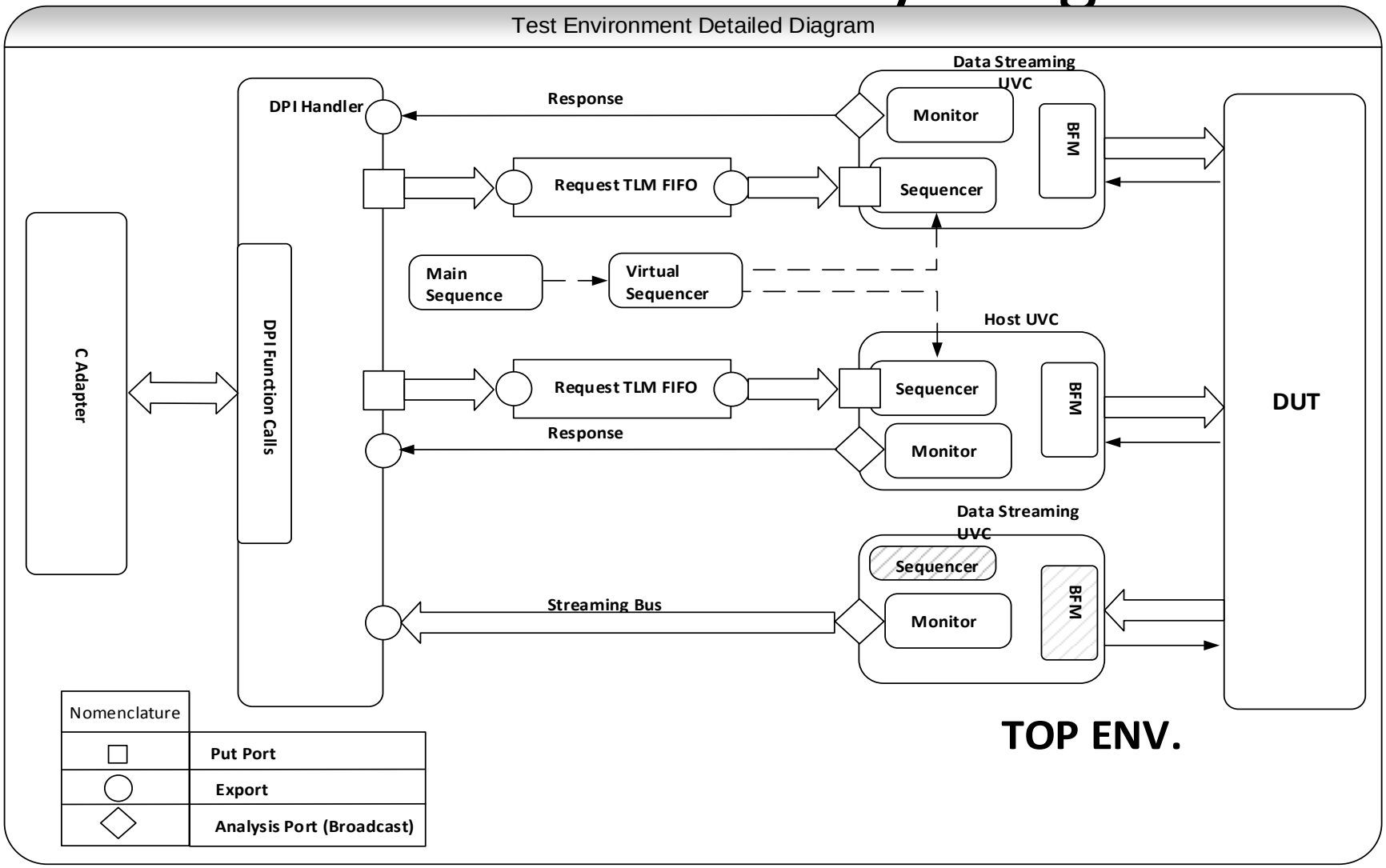
System Level Software Driven Hardware Verification Test Environment

- UVM TE Pulls Data And Configuration Transactions From Multi-Thread Handler And Drives It On To The DUT Buses
- Multi-Thread Handler Takes Care Of The Sockets And Thread Handling Mechanism

Transaction Hierarchy



TE Transaction Layering



TE Transaction Layering

- Functions of DPI handler
  - Request Received From C-Adapter via DPI calls is pushed on to Avalon sequencers via put port (*blocking put*)
  - Response From Avalon streaming And Host monitors Is broadcasted via analysis port implementations and send to C-adapter through DPI calls
- Virtual Sequencer is stitched to main sequence inside test library
  - Executing sequencer is virtual sequencer And it contains the instances of MM sequencer & Streaming Data Sequencer .
- Connections inside top Environment
  - TLM channels connects the “put ports” from dpi handler & “blocking get port” inside both MM & Streaming sequencers
  - All the sequencers from Avalon Environment are stitched to virtual sequencers
  - Monitors from MM & Streaming Environment are stitched to analysis implementations inside dpi handler

Handling The C++ Style Log Generation And Merging With Simulator Log

Handling The C++ Style Log Generation And Merging With Simulator Log

- As part of the co-verification it was required that SW messages gets dumped to regular simulation log file
- We took an approach where we implemented our own ostream along with a stringbuf implementation.
- It takes care of UVM verbosity
- The sync function of the stringbuf internally used vpi_printf to output the stream into the simulation log file
  - Details can be found in the paper

Conclusion

Conclusion

- HW – SW Co-verification Is On The Rise
- Thanks To DPI() Calls, Which Simplifies The Bridging Between Discrete Event Simulator And Multithreaded Interfaces Required For Real Time Software Verification
- Synchronization Techniques Amongst The Threads In Order To Avoid Pit Holes Of Multi-Threaded Environment
- Re-use Of Legacy C Models For Score Boarding At Multiple Levels Of Hierarchy
- Seamless Environment For Software Developer & Hardware Verification
- Novel Technique To Redirect C++ I/O Streams To The Simulation Generated Log File

Questions