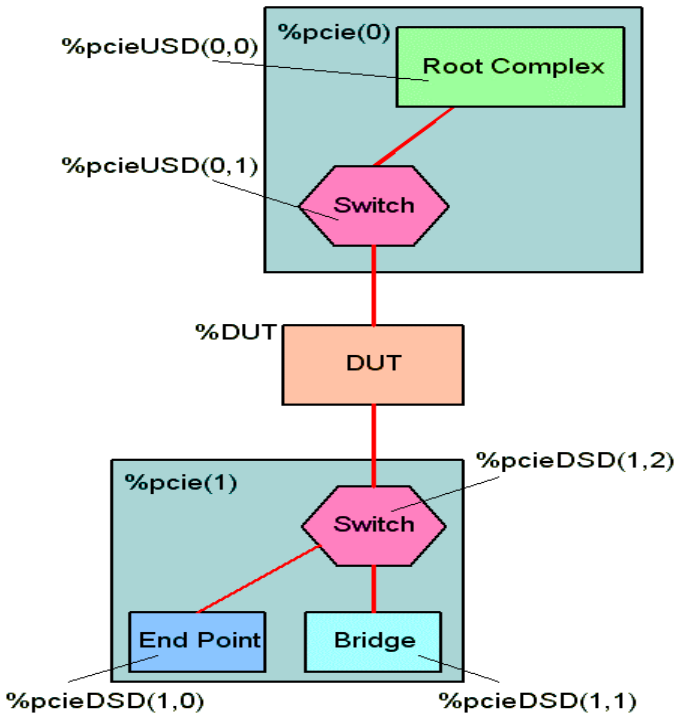
www.tarek.com

Multi-Threaded TLP Traffic Generator

Unlike previous generation of PCI bus, PCI Express truly interleaves read transactions at the hardware level. In PCI Express, a memory read that requests more than 128-byte data has multiple completions. Other transaction can happen before the memory read completes. Although a simple memory read task or function works fine in a simulation environment, it leaves a verification hole that transaction interleaving are totally unverified.

PCI Express verification IPs from other vendors leaves this task to their users to solve. PCIE-VR from Tarek covers this verification hole with a C/C++ multi-thread verification environment and ease-of-use TLP traffic generator. Moreover, you can even integrate your ASIC driver and perform hardware-software co-simulation with your ASIC design.

The following figure illustrates the verification environment with PCIE-VR running the multi-threaded traffic generator test cases. Test cases are shown on the following pages.



In the above figure, the verification environment has two PCIE-VR instances. Each instance contains multiple PCI Express devices. (Other vendor can only do one device per instance.) Because of the clustering nature of PCI Express, the traffic on a PCIE link can be truly generated only when multiple PCI Express devices are modeled together.

Not only feature-rich and ease of use, PCIE-VR also has the best performance. We have compared against one major vendor and we are about 15X faster. You can obtain an evaluation copy and see the actions by yourself. For an evaluation copy, please visit www.tarek.com/contact.htm and enter the contact information.

1.1 Multi-Threaded Traffic Object N in SDL

Although the following test case segment represents a complex multi-threaded transaction scenarios, any engineer can create such a test case with ease or it can be generated automatically with Tarek's Predicated Random Test Generator (PRTG).

```
// Write-read-verify for every transaction

// Set to 1 to enable the traffic
tlpTrafficEnable = 1

// Traffic object N, where N starts from 0 to any number
// Each traffic object is running under an independent thread
// Number of transactions
tlpTrafficCount(N) = [1000, 3000]

// Starting time of the traffic in cycle
tlpTrafficStart(N) = [0, 300]

// Starting address
// If -1, the memory block is in the system memory
tlpTrafficStartAddrHi(N) = 0x814
tlpTrafficStartAddrLo(N) = 0xc0000000

// The block that contains the traffic
// The value is 1M, 10K, or 1024 bytes, etc.
// No default, must be less than the memory size defined in RDF
tlpTrafficBlockSize(N) = 1M

// Data length
// Automatically reduced to maxPayloadLength
tlpTrafficDataLength(N) = [10,4096]

// Next address
// Automatically rewind
tlpTrafficNextAddrOffset(N) = 1
// Next data length increment
tlpTrafficDataLengthInc(N) = 1
```

1.2 A Log Example of Multi-Threaded TLP Traffic

There are six threads in this simulation. In the following log message, TLP transactions from different threads are truly interleaved.

```
TLP DMA traffic(1) write 0xa545b00, length 0x22 ---- end point thread 1
TLP DMA traffic(1) read 0xa545b00, length 0x22
TLP traffic(3) write 0x814c0060045, length 0x15 ----- root complex thread 3
TLP traffic(3) read 0x814c0060045, length 0x15
TLP traffic(0) write 0x814c0040045, length 0x38 ----- root complex thread 0
TLP traffic(0) read 0x814c0040045, length 0x38
TLP traffic(1) write 0x814c0020000, length 0xb9 ---- root complex thread 1
TLP traffic(1) read 0x814c0020000, length 0xb9
TLP traffic(2) write 0x814c0000040, length 0x6b ----- root complex thread 2
TLP traffic(2) read 0x814c0000040, length 0x6b
TLP DMA traffic(0) write 0xa5256f0, length 0x56 ---- end point thread 1
TLP DMA traffic(0) read 0xa5256f0, length 0x56
```