Avery QEMU to CXL Type 3 VIP Co-Simulation

Volume 1: Reference Manual

Release 1.5

1. Getting Help

1.1. Support

If you need assistance with any aspect of Avery Design Systems' products, you may contact Avery Design Systems using one of the following methods.

Web site		http://www.avery-design.com
Telephone	USA	(978) 851-3627
	Taiwan	+886 2 2327-8766
Email		support_emu@avery-design.com
Bugzilla		http://www.avery-design.com.tw/bugs
Sales		http://www.avery-design.com (Click Contact)

Table 1.1 Contact Information

1.2. Documentation Differences

Revision	**History**
1.0	• Initial version of the manual
1.1	• Update Figure 2.1 and 2.2 directory structure • Redirect Chapter 3.4, 3.4.1, 3.4.2, 3.5, and 3.5.1 to AVERY_QEMU_RM • Update Figure 4.1 QEMU-based CXL type 3 device system structure • Add Chapter 4.3 introducing how to run QEMU CXL Co-simulation Across Machines
1.2	• Add Chapter 5.2.3 Avery CXL Cache Line Eviction API
1.3	• Add Chapter 4.2.2 SPDM Option for Openspdm support • Add Chapter 4.4 QEMU CPU Option • Add Chapter 4.5 QEMU Warp Function
1.4	• Remove Chapter 5.2.3 Avery CXL Cache Line Eviction API • Append Chapter 4.1 QEMU Runtime Option for SC cache related options
1.5	• Append Chapter 4.1 QEMU Runtime Option for perf option

Table 1.2 List of Documentation Differences

2. Directory Structure

The AQCXL directory contains:

aqcxl	
├── srcs_avy	Avery's source folder
├── testbench	Testbench files
├── testsuite	Testcase files
├── scripts	Simulation scripts
└── filelist	Filelist

Figure 2.1 AQCXL directory structure

The AQEMU directory contains:

aqemu	
├── bin	Contains the executable for AQEMU
├── scripts	README
└── tools	Contains co-simulation scripts
├── 3rd_party	Contains the dependencies
│ └── edk2-ovmf	
└── qemu_build_x86.tar.gz	

Figure 2.2 AQEMU directory structure

3. Installation and Setup

This section is for installing and setting up the Avery QEMU to CXL VIP co-simulation environment. When the checklists are completed, the provided testbench example will be operational and co-simulation VIP will be ready to use.

3.1. Simulator Support

Avery's QEMU to CXL VIP co-simulation environment supports the following simulators:

1. Questasim 10.6c
2. Xcelium 1903
3. VCS P-2019.06-SP1-1

3.2. License Issues

First, verify that your simulator license is installed, running properly, and indeed is Avery's license issue. If all Avery's licenses are taken, you can wait for a license by adding simulator runtime plusarg, "+licq". For other failures, please follow these steps:

Step1: Make sure your environment variable, $LM_LICENSE_FILE, points to the right server. If it is a Linux/Unix machine, use command:

```
#> echo $LM_LICENSE_FILE
```

Figure 3.1 Check LM_LICENSE_FILE path

It is possible that this variable is set by one of your simulation scripts and not displayed on to the command line. In this case, you may create a stub module, and insert following lines of code:

```
module my_stub;
    initial $system("echo $LM_LICENSE_FILE");
endmodule
```

Figure 3.2 Illustration of the stub module

Add this stub file to your compilation. Now you may observe from your simulation log file, the actual value of LM_LICENSE_FILE being used by the simulator.

Step 2: If the problem remains unresolved, type this command on the Linux machine:

```
#> $AVERY_PLI/bin.linux/lmstat -a | tee lic.log
```

Figure 3.3 Check LM_LICENSE_FILE related features

This will show all the features that your $LM_LICENSE_FILE is related to. Grep for keyword "AVERYDES" and/or the name of the other Avery product features you desired. If nothing is found, check with your system administrators to see if Avery's features are installed properly.

Step 3: Add simulator runtime plusarg "+license_debug" and send the simulation log to Avery's support.

3.3. Download and Installing Avery VIP

Step 1: Download the Avery VIP kits and Avery PLI library from:

http://www.avery-design.com.tw/products/avp	(AQCXL SIM)
http://www.avery-design.com.tw/products/apciexactor.cxl	(PCIe VIP)
http://www.avery-design.com.tw/products/pli	(Avery PLI)

Figure 3.4 List of Avery packages

Note that the PLI library is for license checking purposes. Please contact your local support for credentials before downloading.

Step 2: Extract the Emulation VIP kit and the PLI kit.

```
#> tar -zxvf apciexactor-*version number*.cxl.tar.gz
#> tar -zxvf avery_pli-*version number*.tar.gz
#> tar -zxvf aqcxl_sim-*version number*.tar.gz
```

Figure 3.5 Extract package

The tar file extracted contains the following corresponding directories:
- `apciexactor-*version number*.cxl`: contains Avery PCIe-Xactor.cxl, also referred as Avery's CXL BFM throughout this document.
- `avery_pli-*version number*`: contains Avery PLI reference library.
- `aqcxl_sim-*version number*`: contains Avery QEMU co-simulation project.

Step 3: Setup the environmental variables, each variable needed to be set to its corresponding directory on user's machine:

* `AVERY_PCIE` Avery PCIe-Xactor, directory apciexactor-*version number*.cxl.
* `AVERY_PLI` Avery PLI library, directory avery_pli-2020_*version number*.
* `AVERY_QCXL` Avery QCXL project for integrating Avery QEMU and Avery's example DUT, the directory aqcxl_sim_*version number*.

```
bash> export AVERY_PCIE=<avery_PCIe-install-dir>
bash> export AVERY_PLI=<avery_pli-install-dir>
bash> export AVERY_QCXL=<aqcxl-install-dir>
```

Figure 3.6 Setup environment variables

The license daemon is under *$AVERY_PLI/bin.linux* or *bin.sunos*.

Step 4: Add binary path to your .cshrc or .bashrc file.

```
bash> export PATH=$PATH:$AVERY_QCXL/aqcxl/scripts:$AVERY_QCXL/aqemu/scripts
```

Figure 3.7 Add path command

Step 5: Contact Avery for license file. Once the license file is available, identify which OS is to run the license server and choose the corresponding lmgrd to start the server daemon. For Linux:

```
#> $AVERY_PLI/bin.linux/lmgrd -c license.dat -l avery.lic.log
```

Figure 3.8 Add license file

3.4. Running and preparing QEMU

For running and preparing files in order to run QEMU, please refer to Avery QEMU Co-Simulation reference manual. The manual covers extensive sections on how to prepare QEMU environment and related files.

4. Running QEMU

The QEMU-CXL co-simulation executes in two domains:

1. QEMU virtualizer
2. System Verilog simulator

There are multiple debug and introspection methods into the behaviors between the user space application and DUT. This section contains the overview of the system and how to start the process invoking each domain.

Section 4.1 describes the run script for running QEMU virtualizer and the functionality of each option, the subsection 4.1.1 describes how BIOS used when installing the disk image will affect the boot up of the guest OS. Section 4.2 describes the run script for running the SystemVerilog Simulator that contains either the user DUT or Avery's CXL simulation device mode (AVIP).

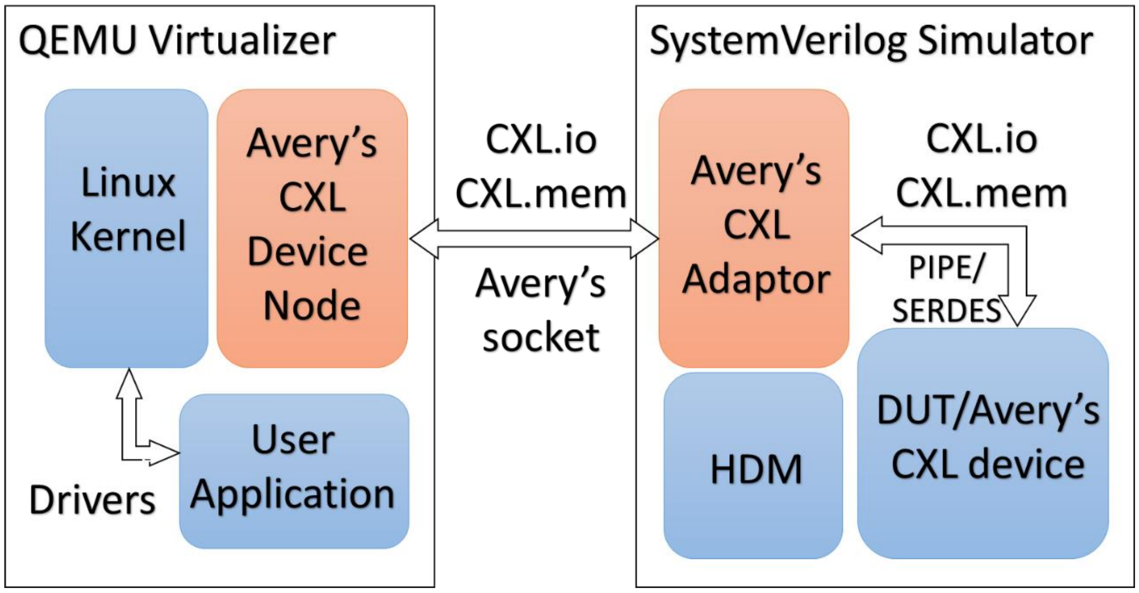

Figure 4.1 QEMU-based CXL type 3 device system structure

Running the QEMU-CXL co-simulation requires the user to start three terminals: One running the QEMU and Avery's corresponding transactor, another running the Avery VIP System Verilog simulation, and logging onto the QEMU virtual machine terminal to execute user application.

On a user's working directory, start the Avery VIP simulation process.

```
#> aqcxl_sv.pl -s (xm64|mti64|vcs64) -qemu -t apcit_qemu_basic.sv
```

Figure 4.2 Run command for starting AVIP

Avery's PCIe AVIP System Verilog simulation should be started with the selected simulator under the -s option. Tracker and simulation log files will be generated under this directory.

On the second terminal, run the Avery QEMU launch script provided:

```
#> aqcxl_qemu.pl -qc <location of QCOW image>
```

Figure 4.3 Run command for starting QEMU

QEMU will start to print boot parameters and traffic, and corresponding log file, qemu.log will be generated under the $cur_dir/log directory. Delay in the print buffer may be observed and flushed by the termination of the QEMU.

Users should observe a line containing the port number of the VNC server started while booting QEMU, port 5900 by default.

```
VNC server running on ::1:5900
```

Figure 4.4 VNC message from QEMU

On the last terminal, log in into the QEMU to start testing sequence, VNCVIEWER in used in this example:

```
#> vncviewer :5900
```

Figure 4.5 Logging into QEMU using VNCVIEWER

Users can scp to upload their software tools and applications to the virtual machine once QEMU has successfully boot-up. These will be persistent on the QEMU disk image.

4.1. QEMU Runtime Option

Here is a list of run times options users can enable for their QEMU co-simulation:

aqcxl_qemu.pl		
[options]	function	Default
-qc <path>	Specify path of QCOW image	$AVERY_QEMU
-dbg <level>	Turn on debug level 0x3 for HDM message level 0x1 for QEMU only debug	0x3
-ip <address>:<port>	Specify ip address and port number for socket	127.0.0.1:9210
-kvm	Enable Kernel-based Virtual Machine feature.	
-kgdb	Set port 4321 for kgdb connection	
-os_img <path>	Path to OS installation image	
-ovmf	Specify BIOS to boot	$AQEMU/tools/3rd_party/edk2-ovmf
-io_warp	Turn on warp mechanism, refer to section 4.5 QEMU warp Function	
-cpu	Specify CPU to model, refer to section 4.4 QEMU CPU Model	Broadwell
-cache_cap	Specify SC cache total capacity	16MB
-Nway	Specify SC cache Numer of ways	16
-line_size	Specify SC cache line size	64Bytes
-perf <offset>	Specify Avery Performance Measurement DVSEC Register offset, the DVSEC register length is currently 48 bytes	

Figure 4.6 List of QEMU runtime options

4.2. CXL AVIP Runtime Option

Here is a list of run times options users can enable for their QEMU co-simulation:

aqcxl_sv.pl	[options]	Function
	-s mti64|xm64|vcs64	Specify simulator
	-t apcit_qemu_basic.sv	Default test for QEMU
	-qemu	
	-C +define+*defines*	Specify additional defines
	-spdm	Enable the Openspdm support

Figure 4.7 List of AVIP simulation runtime options

4.2.1. OVMF Option

Please be noted that the BIOS installed on the provided QCOW image should be consistent with the BIOS used to boot QEMU machine, SeaBIOS is used by QEMU as default. However, newer BIOS version UEFI is downloaded from tianocore's git repository and used to boot Avery's QEMU co-simulation, the UEFI files are under $AVERY_QEMU/tools/3rd_party/edk2-ovmf.
Link here: https://github.com/tianocore/edk2.git
Users can specify UEFI used by invoking -ovmf option with their own UEFI built when launching the aqcxl_qemu.pl script.

4.2.2. SPDM Option

Openspdm is an open source project which emulates the requester and responder of DMTF SPDM specification.
Source: https://github.com/jyao1/openspdm.git
Avery has created an interface to communicate with Openspdm's responder, so users can have the DMTF SPDM functionality in the SytemVerilog simulation.
Users can gain Openspdm support by invoking -spdm option when launching the aqcxl_sv.pl script.

4.3. QEMU CXL Co-simulation Across Machines

The SytemVerilog simulation process and QEMU process can be run across different machines, simply providing the IP address and port for QEMU to connect to after enabling the firewall.

Please be aware that enabling port on firewall requires root access, please enquire your local IT authorities for using such feature. For example, to enable port 9210 on firewall:

```
#> firewall-cmd --zone=public --add-port=9210/tcp
```

Figure 4.8 Command for enabling the port on firewall

After enabling the port that you are hosting the SystemVerilog, launch the process with port using the -C +define+AVY_PORT=<port number> option:

```
#> aqcxl_sv.pl -qemu -s (nc64|mti64|vcs64) -C +define+AVY_PORT=9210 \
  -t apcit_qemu_basic.sv
```

Figure 4.9 Running SystemVerilog Simulation with custom port

Now the SystemVerilog process is up running for example on machine ip "192.168.1.21", connect QEMU process from a different machine using option -ip <ip>:<port>

```
#> aqcxl_qemu.pl -qc <location of QCOW image> -bios <location of BIOS image> \
-ip 192.168.1.21:9210
```

Figure 4.10 Running QEMU with target machine IP and port

On SystemVerilog process, you should see IP of QEMU process dynamically connected.

```
SimCluster: For instance dynamically connected, channel with data socket 11 at
192.168.1.60, t has different design time scale s, please change it to fs
SimCluster: Info, Child registration completed. Start simulation.
```

Figure 4.11 SystemVerilog connects to QEMU socket

To list and remove port:

```
#> firewall-cmd --list-port
9210/tcp
#> firewall-cmd --remove-port=9210/tcp
Success
```

Figure 4.12 Commands for listing and remove port on firewall

4.3.1. QEMU CXL Co-simulation on Different Port

Please note that user can use the same mechanism as 4.3 QEMU CXL Co-simulation Across Machine to run multiple co-simulations on the same machine with alternating ports, default IP address is 127.0.0.1 and default port number is 9210, port number 12321 is used as example:

```
#> aqcxl_sv.pl -qemu -s (nc64|mti64|vcs64) -C +define+AVY_PORT=12321 \
-t apcit_qemu_basic.sv
#> aqcxl_qemu.pl -qc <location of QCOW image> -bios <location of BIOS image> \
-ip 127.0.0.1:12321
```

Figure 4.13 Commands for running on same machine but different port

4.4. QEMU CPU Option

QEMU allows users to select the CPU model they are expecting to run with. Broadwell is selected by Avery by default and users can choose the CPU model accordingly using the -cpu option. Note this feature is developed by QEMU, any warnings or change in behaviors should be supported by QEMU development group. A list of supported CPUs can be found under QEMU office documentations: https://qemu.readthedocs.io/en/latest/system/qemu-cpu-models.html

4.5. QEMU Warp Function

To reduce time synchronization issues between the RTL simulation and QEMU, Avery implemented a warp mechanism enabled by default.

KVM (Kernel Virtual Machine) is a full virtualization solution of Linux and supported by x86 hardware that enables speed up of the virtual CPUs and related feature models for QEMU simulation. When hardware supports KVM, CPU instructions can be passed from the virtual machine to the host CPU resulting in CPU instruction speed-up to near native speed. CPU instances in the QEMU user mode are accelerated by KVM and viewed as vCPUs (virtual CPUs) by the QEMU middleware. Clock of the vCPUs advances according to wall clock of the host machine and is updated via a specific register controlled by KVM using QEMU event-based semantics.

- warp Mechanism Explained

Considering RTL simulation is relatively much slower than QEMU running KVM acceleration and setting up QEMU to be cycle accurate will add more cost to the already slow process. Instead, Avery has enhanced the KVM lock mechanism to stop the KVM from executing CPU instructions while allowing traffic to finish up the traffic directed to the hardware running in the RTL simulation. From the perspective of the QEMU guest OS, the traffic accesses would finish in zero time, thus, eliminating the possibility of OS timeout or panics greatly. Currently warp feature is enabled by default, which might slow down the QEMU co-simulation, users can disable it using -nowarp option according to their performance requirement.

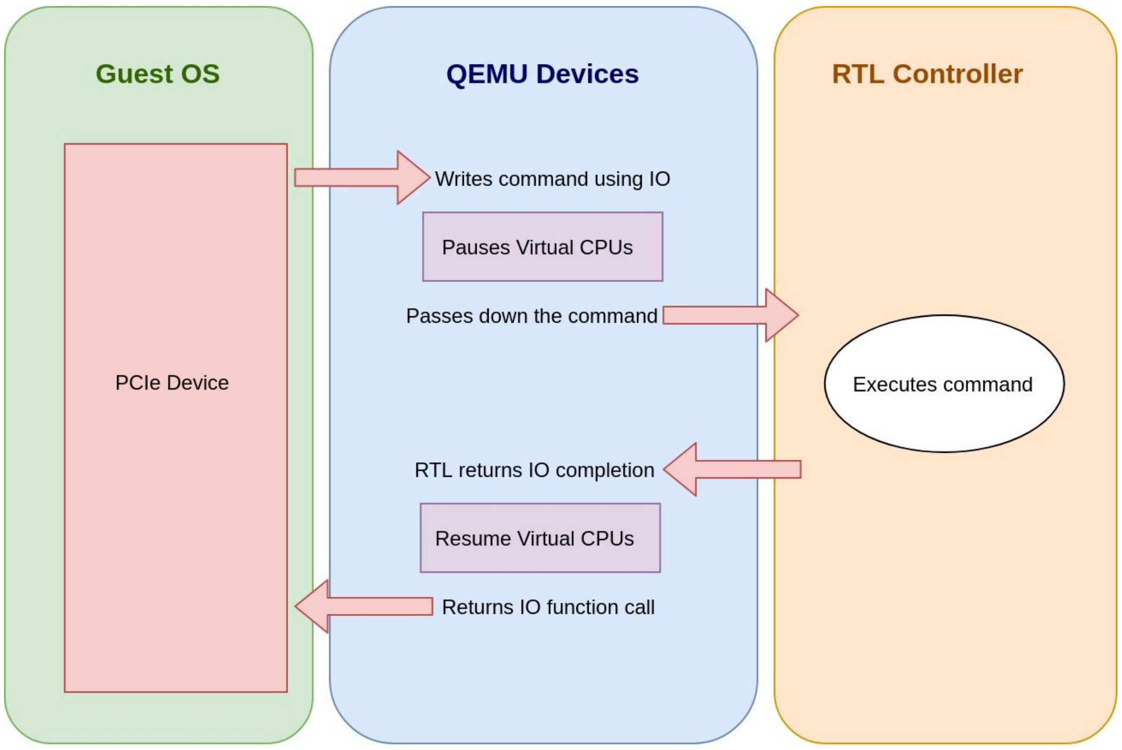

Figure 4.14 warp mechanism overview

5. Debugging and Developing on QEMU

When developing on QEMU guest OS, any Linux built-in tools or custom application should behave the same as targeting a real CXL device.

Built-in applications like devmem2 can be used to access PCIe BAR and HDM address of the CXL device, meanwhile lspci and setpci can be used to access the config space of the device. User can use it to aid custom application development.

5.1. QEMU Log

QEMU-based CXL simulation sets up device nodes to communicate with System Verilog counterparts. Avery provides an optional argument in the run script to gain behavioral transparency of the co-simulation.

To enable the debug message, adds '-dbg' after the aqcxl_qemu.pl run script as follows:

```
$AVERY_QCXL/aqemu/scripts/aqcxl_qemu.pl -dbg -qc $QCOW/ubuntu.img
```

Figure 5.1 QEMU run command with debug on

PCIe device configuration reads and writes will be displayed as follows:

```
[QEMU] pci_config_read: addr: e data: 0 size: 1
[QEMU] pci_config_read: addr: 0 data: abcd size: 2
[QEMU] pci_config_read: addr: 0 data: 1234abcd size: 4
[QEMU] pci_config_read: addr: 8 data: 1080272 size: 4
```

Figure 5.2 PCIe config read example

While the BAR offset address is dynamic allocated by Linux driver, PCIe device memory reads and writes from host to DUT will be displayed using size of bytes as follows:

```
[QEMU] pci_io_read: addr: fe600000 data: 10720fd size: 4
[QEMU] pci_io_read: addr: fe600004 data: 3c00030 size: 4
[QEMU] pci_io_read: addr: fe600008 data: 10400 size: 4
[QEMU] pci_io_read: addr: fe60003c data: 0 size: 4
```

Figure 5.3 PCIe IO read example

When debug verbosity 0x3 is specified to launch QEMU, HDM reads and writes access will also be printed on the QEMU log:

```
[QEMU] cxl_hdm_access write addr: 0 data: 00003421 size: 4 bytes
[QEMU] cxl_hdm_access write addr: 4 data: 00000000 size: 4 bytes
[QEMU] cxl_hdm_access read addr: 0 data: 00003421 size: 4 bytes
```

```
[QEMU] cxl_hdm_access read addr: 4 data: 00000000 size: 4 bytes
[QEMU] cxl_hdm_access read addr: 0 data: 00003421 size: 4 bytes
[QEMU] cxl_hdm_access read addr: 4 data: 00000000 size: 4 bytes
```

Figure 5.4 CXL mem log example

5.2. Debugging on Avery CXL VIP

Since Avery's CXL QEMU co-simulation runs on simulated CXL model on the SystemVerilog side, verification IP traffic log in both PCIe and CXL layer is visible and will greatly aid Kernel and user application development.

5.2.1. Avery CXL VIP Tracker

Details of CXL.mem traffic generated from QEMU will be displayed under cxl_tl_rc.txt log generated under the run directory, please refer to Avery's CXL document for more details.

```
>>> @210448.599ns  M2S_REQ_MemRd#3aac0 (tag 1, addr 4c0000000,
D2H_REQ_RdOwn#3aabf)
   |__ opcode 1 | metaField 0=Meta0 | metaValue 2=A | SnpInv | addr51_5 26000000 | tag
1 | tc 0 |
   |__ (speculative read)

           <<< @210540.498ns  S2M_NDR_CmpE#3ab25 (tag 1, M2S_REQ_MemRd#3aac0)
              |__ opcode 2 | metaField 0 | metaValue 2 | tag 1 |

           <<< @210544.498ns  S2M_DRS_MemData#3ab29 (tag 1,
M2S_REQ_MemRd#3aac0)
              |__ opcode 0 | metaField 0 | metaValue 2 | tag 1 | poison 0 |
              |__ be ffffffffffffffff
              |__ ee ee ee ee ee ee ee ee  ee ee ee ee ee ee ee ee
              |__ ee ee ee ee ee ee ee ee  ee ee ee ee ee ee ee ee
              |__ ee ee ee ee ee ee ee ee  ee ee ee ee ee ee ee ee
              |__ ee ee ee ee ee ee ee ee  ee ee ee ee ee ee ee ee
```

Figure 5.5 CXL Transaction Layer Root Complex VIP Tracker

5.2.2. Avery PCIe VIP Tracker

Transactions coming from QEMU are received and passed down to Design Under Test from Avery's Root Complex PCIe device via PIPE interface, the device is configured as a medium to only pass down the transaction without generating any extra behavior. Users can use the tracker generated by Avery PCIe VIP on Transaction Layer to debug their RTL design.

Configuration reads and writes originated from QEMU will be displayed as follows:

```
==> @29662.453ns  CFGRD0#472d (bdf_1_0_0, offset c)(req_id 0000, tag 005)
(APCI_TRANS_cfg#472c)
   |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|             length          |
   | 000b|  00100b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001h _____|
   |___________ req_id: 0000 __________|___ tag: 05 ___|lbe: 0 |fbe: f |
   |       bdf.bus    |bdf.dev |bdf.func|rsvd20|reg_no.ext|reg_no.low|rsv|
   |_____ 01 ___|__ 00 __|___ 0 __|__ 0 __|___ 0 __ __|____ 03 ___| 0 |
   04000001 0000050f 0100000c

           <== @29826.391ns  CPLD#473b (CFGRD0#472d, SUCCESS)(req_id 0000, tag 005)
              |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|          length        |
              | 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_____ 001h __  _|
              |_____ cpl_id: 0000 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 004 __|
              |_____ req_id: 0000 ___|___ tag: 05 ___|rsvd20: 0| low_addr: 00 |
              4a000001 00000004 00000500
               00000000
```

Figure 5.6 PCIe config read on PCIe VIP tracker

Memory reads and writes originated from QEMU will be displayed as follows:

```
==> @38242.800ns  MRD#4b3e (addr fe600008, 'h1 dwords)(req_id 0000, tag 019)
(APCI_TRANS_mem#4b3d)
   |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|             length          |
   | 000b|  00000b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001h _____|
   |___________ req_id: 0000 __________|___ tag: 19 ___|lbe: 0 |fbe: f |
   |________________________ addr: fe600008 ______ _______________|
   00000001 0000190f fe600008

           <== @38406.391ns  CPLD#4b4e (MRD#4b3e, SUCCESS)(req_id 0000, tag 019)
              |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|          length        |
              | 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_____ 001h ____|
              |_____ cpl_id: 0100 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 004 __|
              |_____ req_id: 0000 ___|___ tag: 19 ___|rsvd20: 0| low_addr: 08 |
              4a000001 01000004 00001908
               00010400
```

Figure 5.7 PCIe memory read on PCIe VIP tracker

Memory reads and writes originated from design are displayed as MRD or MWR with an arrow pointing at the opposite direction. This transaction will be displayed as a DMA transaction on QEMU. Because of the limitation of System Verilog's event driven property, the completion of MRD or MWR transaction from System Verilog to QEMU will be implemented as a dropped packet at Transaction Layer, and then injected back to the PCIe device with the corresponding tag. '"tx_pkt_exit_tl", CPLD#tag' can be observed and used to track down the actual injected packet of the dropped dummy packet.

```
          <== @42246.391ns  MRD#4cd7 (addr 7fffb000, 'h10 dwords)(req_id 0100, tag 000)
             |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|             length          |
             | 000b|  00000b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 010h _____ |
             |___________ req_id: 0100 _________|____ tag: 00 ___| lbe: f | fbe: f |
             |_______________________ addr: 7fffb000 _________________________|
             00000010 010000ff 7fffb000

rc@42246.391ns  AVY_INF: apci_test_log@42246.391ns: RC received read_cb#1: ADDR
'h7fffb000, NDW 'h10, FBE 'hf, LBE 'hf
==> @42246.392ns  CPLD#4cda (MRD#4cd7, SUCCESS)(req_id 0100, tag 000) (MRD#4cd7)
   |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|             length          |
   | 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 010h _____|
   |_____ cpl_id: 0000 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 040 __|
   |_____ req_id: 0100 ___|___ tag: 00 ___|rsvd20: 0| low_addr: 00 |
   4a000010 00000040 01000000
   (Dropped, inject at CPLD#4cdc)
    eeeeeeee eeeeeeee eeeeeeee eeeeeeee eeeeeeee eeeeeeee eeeeeeee eeeeeeee

rc@42250.417ns  AVY_API: rc.inject_pkt("tx_pkt_exit_tl", CPLD#4cdc, 0)
==> @42250.417ns  CPLD#4cdc (SUCCESS)(req_id 0100, tag 000) (MRD#4cd7)
   |  fmt |       typ     |t|   tc   |t |a |l |t |t |e|att|at|             length          |
   | 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 010h _____|
   |_____ cpl_id: 0000 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 040 __|
   |_____ req_id: 0100 ___|___ tag: 00 ___|rsvd20: 0| low_addr: 00 |
   4a000010 00000040 01000000
   (From CPLD#4cda addr 'h1000000 length 'h10)(inject_pkt at tx_pkt_exit_tl)
    00000006 00000000 00000000 00000000 00000000 00000000 7ff9f000 00000000
```

Figure 5.8 Controller read on PCIe VIP tracker

Since SystemVerilog side runs Avery's CXL simulated device, users can observe the Endpoint configuration space setting set by the SystemVerilog Testbench. Under the generated tracker_tl_ep0.txt tracker file, the configuration space offset of the PCIe extended configuration space list is displayed. Users can modify the testbench file for specific settings to match and cross reference with their application design.

```
======= PCIe Extended Config Space [100 : FFFh] =====
  --ep0.rciep.f0.device3  [func_no 0, offset 100, speed_sup 5, APCI_PORT_rc_ie, 10 bytes]
  --ep0.rciep.f0.aer  [func_no 0, offset 3fc, speed_sup 5, APCI_PORT_rc_ie, 48 bytes]
  --ep0.rciep.f0.vc  [func_no 0, offset 620, speed_sup 5, APCI_PORT_rc_ie, 1c bytes]
  --ep0.rciep.f0.acs  [func_no 0, offset 884, speed_sup 5, APCI_PORT_rc_ie, 24 bytes]
  --ep0.rciep.f0.mc  [func_no 0, offset 8d0, speed_sup 5, APCI_PORT_rc_ie, 30 bytes]
  --ep0.rciep.f0.pri  [func_no 0, offset a58, speed_sup 5, APCI_PORT_rc_ie, 10 bytes]
  --ep0.rciep.f0.sriov  [func_no 0, offset b10, speed_sup 5, APCI_PORT_rc_ie, 40 bytes]
  --ep0.rciep.f0.cxl_device  [func_no 0, offset bec, speed_sup 5, APCI_PORT_rc_ie, 38 bytes]
  --ep0.rciep.f0.cxl_device_test  [func_no 0, offset ca0, speed_sup 5, APCI_PORT_rc_ie, 2c
bytes]
  --ep0.rciep.f0.doe_array[0]  [func_no 0, offset e3c, speed_sup 5, APCI_PORT_rc_ie, 18
bytes]
```

Figure 5.9 PCIe Transaction Layer Endpoint VIP tracker

6. Current Known Limitation

Note that for the current release, PCIe® TLP completion status UR, Completion Abort are not supported, but will be supported in the future release.