LL10GEMAC-IP with AAT Demo Instruction

Rev1.2 14-Jun-23

1 Overview

This document provides instructions for setting up the Alveo accelerator card and preparing the test environment to run the Accelerated Algorithmic Trading (AAT) demo. The default demo can be obtained from the following link provided by Xilinx.
https://www.xilinx.com/applications/data-center/financial-technology/accelerated-algorithmic-trading.html

To achieve lower latency time, the AAT demo has been modified to utilize the LL10GEMAC-IP from Design Gateway instead of the 10G/25G Ethernet subsystem. The LL10GEMAC-IP latency details can be found in the datasheet available at
https://dgway.com/products/IP/Lowlatency-IP/dg_ll10gemacip_data_sheet_xilinx_en.pdf

Running the AAT demo requires an Alveo accelerator card plugged into a compatible system. The accelerator card features a QSFP+ connector, which supports up to 4x10G Ethernet connections. For the demo, two of the 10G Ethernet connections are used: one for transferring example market data via UDP protocol and another for order transmission via FIX over TCP. Therefore, the second system needs to be prepared by integrating two channels of 10G Ethernet connection. In this document, the second system is prepared using a PC with a 10G Ethernet card. The “tcpreplay” tool must be run on the second system to send the example market data, and a TCP port must be opened to receive orders from the accelerator card via TCP. The test operation on the accelerator card is controlled via the “aat_shell_exe”.

Before running the test, please prepare the following test environment.

- Alveo accelerator card: U50 or U250 card
- Turnkey accelerator system, TKAS-D2101, compatible with the Alveo accelerator card. More information can be found at
  https://dgway.com/AcceleratorCards.html
- 10G Ethernet cable: QSFP+ to four SFP+ cable. Example cable can be found at
  https://www.sfpcables.com/5-meter-40g-qsfp-to-4-sfp-aoc-cable-om3-mmf-cisco-oem-compatible
- The PC for 10G Ethernet transferring with the Turnkey accelerator system, including
  - Ubuntu 20.04 LTS Server OS
  - Example market data for running AAT demo
  - TCPreplay package for transmitting market data
  - Two ports of 10G Ethernet connection such as Intel X710-DA2
    https://ark.intel.com/content/www/us/en/ark/products/83964/intel-ethernet-converged-network-adapter-x710da2.html

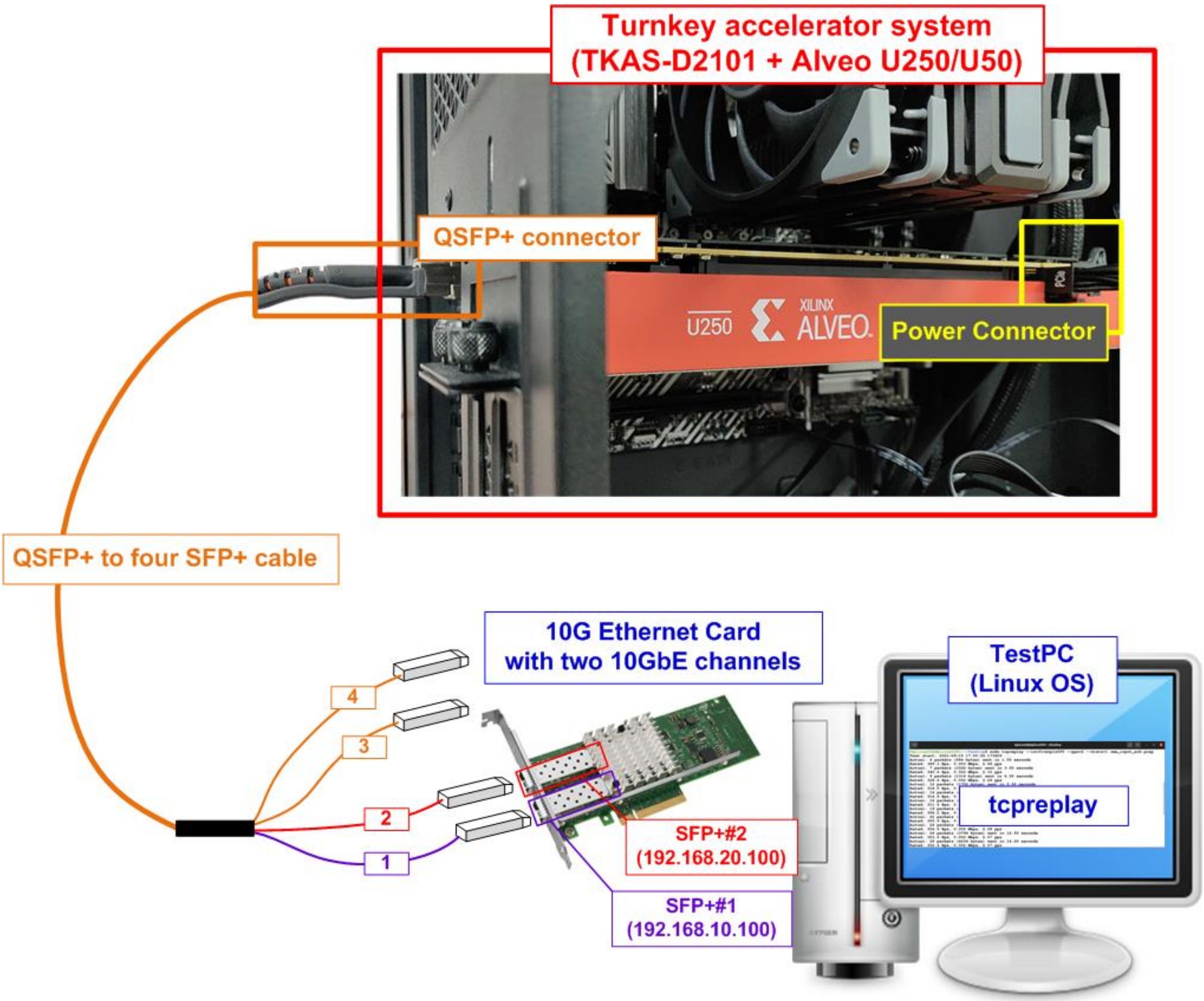


Figure 1-1 LL10GEMACIP with AAT demo (FPGA <-> PC) on Alveo U250/U50 card

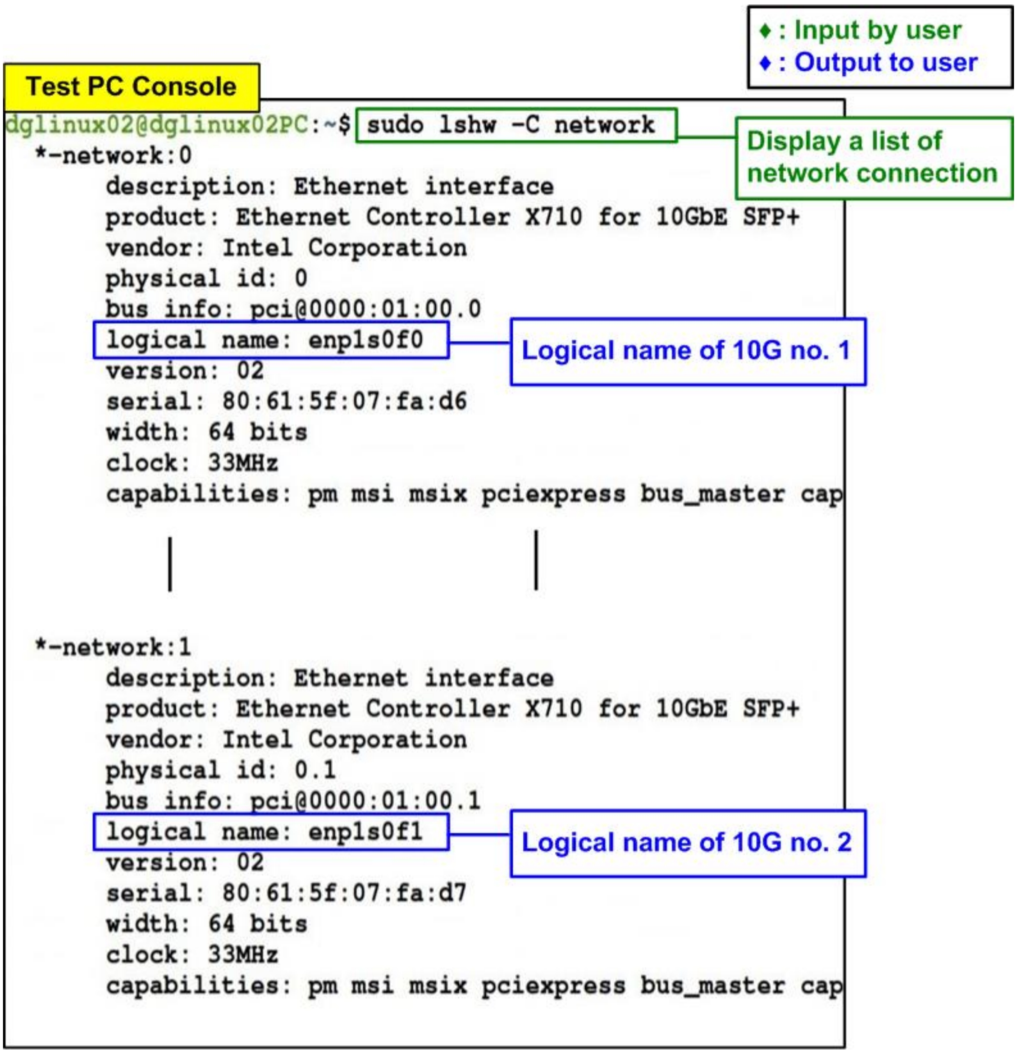
2 Test PC setup

This topic provides instructions on preparing the PC for transferring market data and order packets with the Alveo accelerator card. The example demonstrates the setup on Ubuntu 20.04 LTS Server OS.

2.1 IP Address configuration for two ports of 10G Ethernet

First, determine the logical names of the Ethernet ports that connect to the SFP+#1 and SFP+#2 cables. The logical names may vary depending on your test environment. It is important to configure the correct IP address for the SFP+#1 and SFP+#2 connections.

1) Open a Linux terminal and enter the command “lshw -C network” to list the logical names of the 10G Ethernet ports. Figure 2-1 illustrates an example result displaying the logical names for two 10G Ethernet connections. In this case, “enp1s0f0” represents the Ethernet port for SFP+#1, while “enp1s0f1” corresponds to the Ethernet port for SFP+#2.

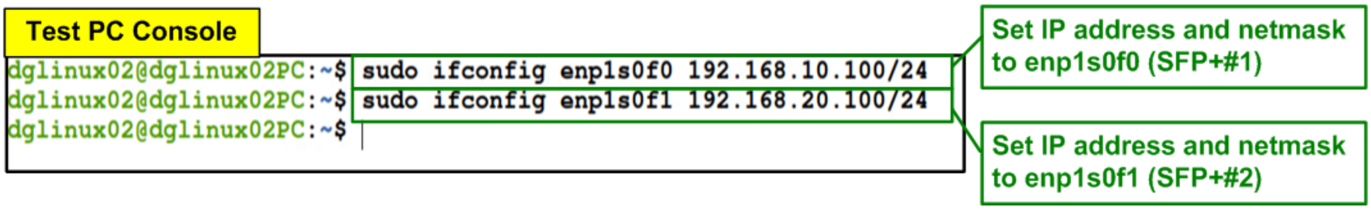


♦ : Input by user
♦ : Output to user
Test PC Console
Display a list of network connection
Logical name of 10G no. 1
Logical name of 10G no. 2

```
dglinux02@dglinux02PC:~$ sudo lshw -C network
  *-network:0
       description: Ethernet interface
       product: Ethernet Controller X710 for 10GbE SFP+
       vendor: Intel Corporation
       physical id: 0
       bus info: pci@0000:01:00.0
       logical name: enp1s0f0
       version: 02
       serial: 80:61:5f:07:fa:d6
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi msix pciexpress bus_master cap

  *-network:1
       description: Ethernet interface
       product: Ethernet Controller X710 for 10GbE SFP+
       vendor: Intel Corporation
       physical id: 0.1
       bus info: pci@0000:01:00.1
       logical name: enp1s0f1
       version: 02
       serial: 80:61:5f:07:fa:d7
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi msix pciexpress bus_master cap
```

Figure 2-1 Display logical name of 10G Ethernet port

2) Configure the IP address of SFP+#1 (enp1s0f0) as “192.168.10.100” and SFP+#2 (enp1s0f1) as “192.168.20.100” using the “ifconfig” command, as shown in Figure 2-2. Additionally, set the netmask to 24 using the same command.



Test PC Console

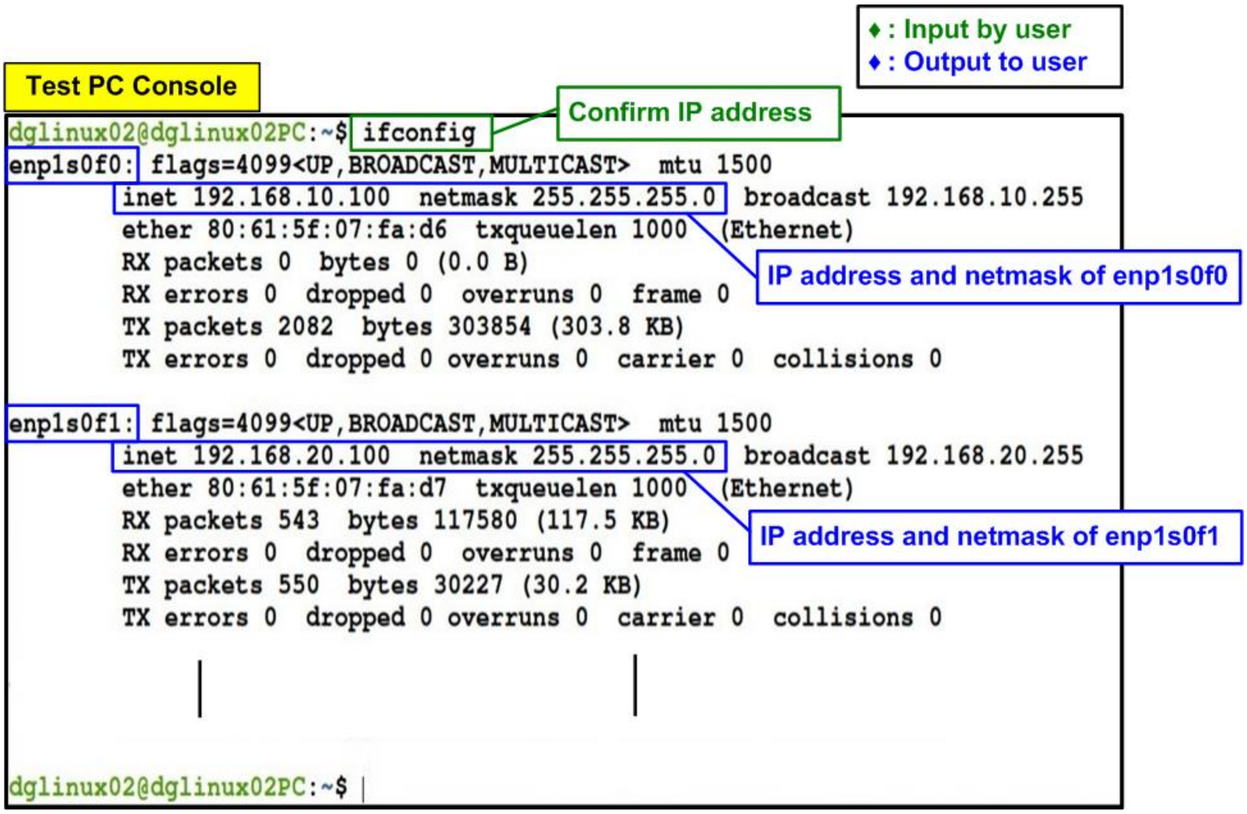
```
dglinux02@dglinux02PC:~$ sudo ifconfig enp1s0f0 192.168.10.100/24
dglinux02@dglinux02PC:~$ sudo ifconfig enp1s0f1 192.168.20.100/24
dglinux02@dglinux02PC:~$ |
```

Set IP address and netmask to enp1s0f0 (SFP+#1)

Set IP address and netmask to enp1s0f1 (SFP+#2)

Figure 2-2 Configure IP address and netmask

3) Use the “ifconfig” command to confirm the IP address and netmask after completing the configuration.



♦ : Input by user
♦ : Output to user

Test PC Console

```
dglinux02@dglinux02PC:~$ ifconfig
enp1s0f0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.10.100  netmask 255.255.255.0  broadcast 192.168.10.255
        ether 80:61:5f:07:fa:d6  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2082  bytes 303854 (303.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp1s0f1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.20.100  netmask 255.255.255.0  broadcast 192.168.20.255
        ether 80:61:5f:07:fa:d7  txqueuelen 1000  (Ethernet)
        RX packets 543  bytes 117580 (117.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 550  bytes 30227 (30.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

dglinux02@dglinux02PC:~$ |
```

Confirm IP address

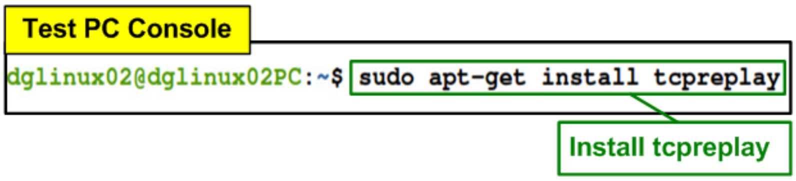
IP address and netmask of enp1s0f0

IP address and netmask of enp1s0f1

Figure 2-3 Verify IP address and netmask setting

2.2 Installation of “tcpreplay”

The Test PC needs to have “tcpreplay” installed in order to run the AAT demo. To install the package, enter the command “sudo apt-get install tcpreplay” in the terminal, as shown in Figure 2-4.



Test PC Console

```
dglinux02@dglinux02PC:~$ sudo apt-get install tcpreplay
```

Install tcpreplay

Figure 2-4 “tcpreplay” installation

3 Test environment setup

This topic outlines the steps to prepare the Turnkey acceleration system (TKAS-D2101 with U250/U50) for running the AAT demo.

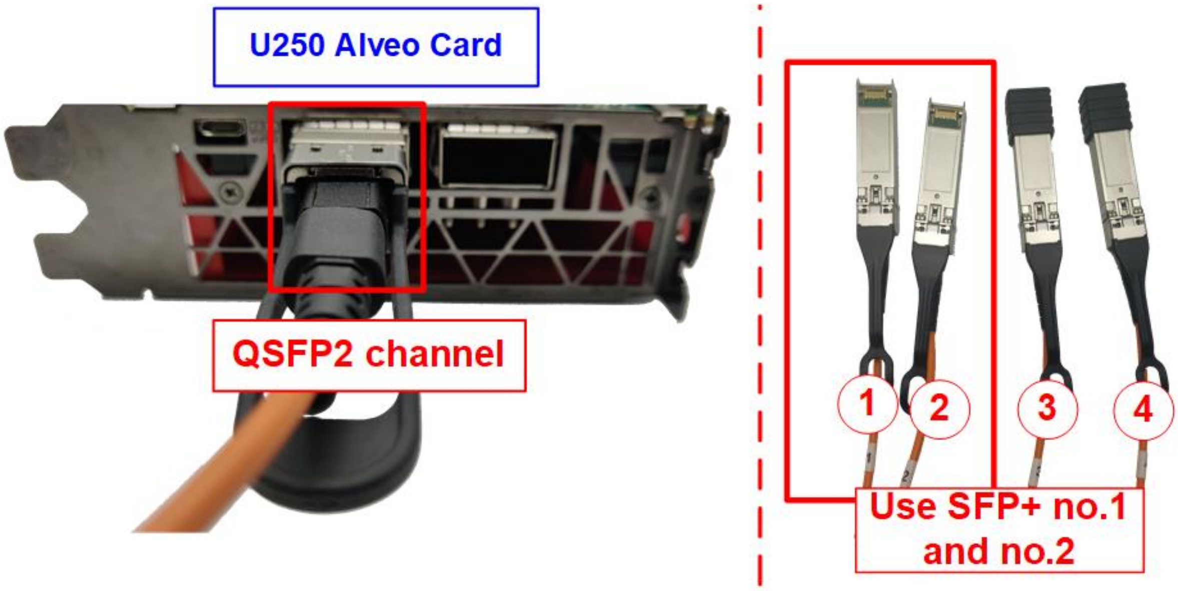
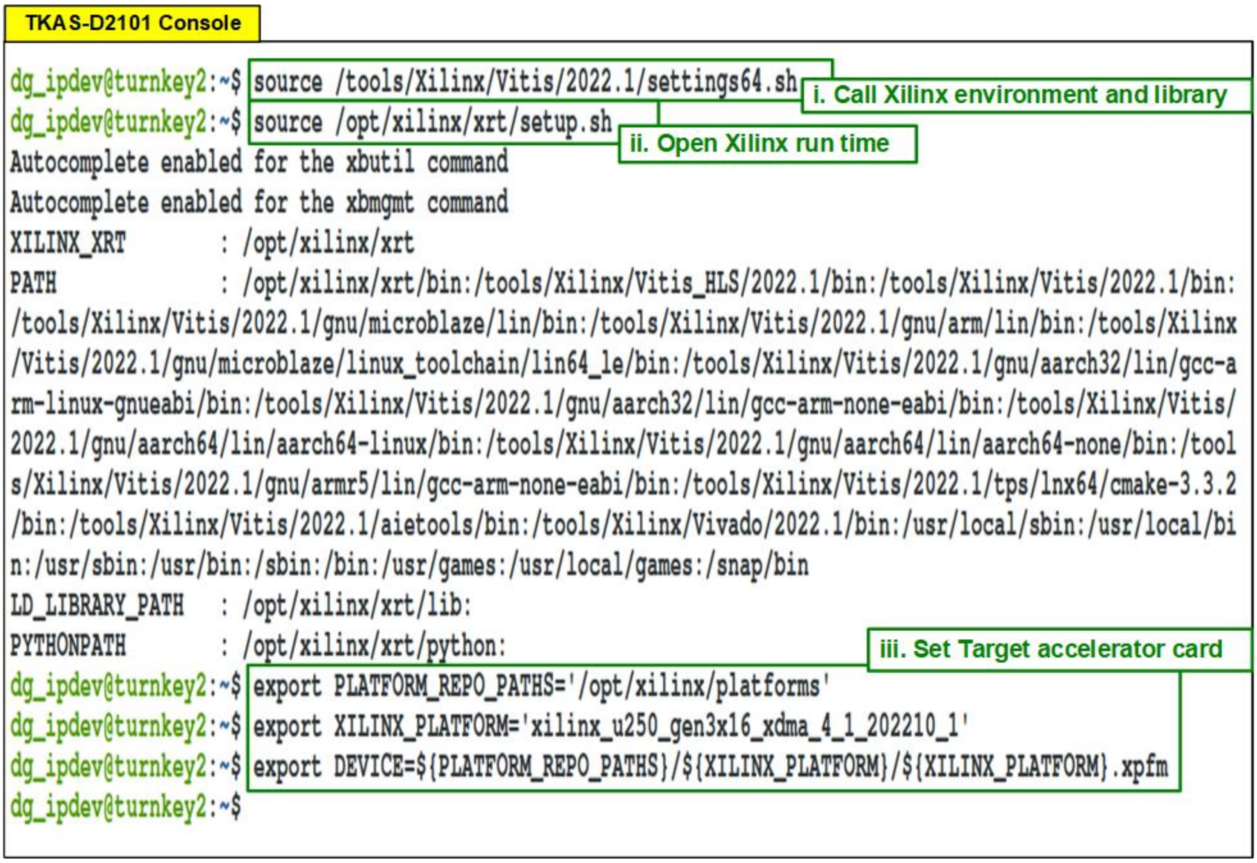


Figure 3-1 QSFP+ channel using on U250 board

1) Connect the QSFP+ to four SFP+ cable (4x10G Ethernet cable) between the Alveo accelerator card (U50/U250) and the Test PC. Use the SFP+ connectors no.1 and no.2.
    i) Insert a QSFP+ transceiver into the QSFP+ connector on the Alveo accelerator card. If you are using the U250 card, which has two QSFP+ channels, make sure to insert it into the QSFP2 channel.
    ii) Connect SFP+ no.1 (192.168.10.100) and SFP+ no.2 (192.168.20.100) to 10G Ethernet channel on Test PC.



TKAS-D2101 Console

```
dg_ipdev@turnkey2:~$ source /tools/Xilinx/Vitis/2022.1/settings64.sh
dg_ipdev@turnkey2:~$ source /opt/xilinx/xrt/setup.sh
Autocomplete enabled for the xbutil command
Autocomplete enabled for the xbmgmt command
XILINX_XRT        : /opt/xilinx/xrt
PATH              : /opt/xilinx/xrt/bin:/tools/Xilinx/Vitis_HLS/2022.1/bin:/tools/Xilinx/Vitis/2022.1/bin:
/tools/Xilinx/Vitis/2022.1/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2022.1/gnu/arm/lin/bin:/tools/Xilinx
/Vitis/2022.1/gnu/microblaze/linux_toolchain/lin64_le/bin:/tools/Xilinx/Vitis/2022.1/gnu/aarch32/lin/gcc-a
rm-linux-gnueabi/bin:/tools/Xilinx/Vitis/2022.1/gnu/aarch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/
2022.1/gnu/aarch64/lin/aarch64-linux/bin:/tools/Xilinx/Vitis/2022.1/gnu/aarch64/lin/aarch64-none/bin:/tool
s/Xilinx/Vitis/2022.1/gnu/armr5/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2022.1/tps/lnx64/cmake-3.3.2
/bin:/tools/Xilinx/Vitis/2022.1/aietools/bin:/tools/Xilinx/Vivado/2022.1/bin:/usr/local/sbin:/usr/local/bi
n:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
LD_LIBRARY_PATH   : /opt/xilinx/xrt/lib:
PYTHONPATH        : /opt/xilinx/xrt/python:
dg_ipdev@turnkey2:~$ export PLATFORM_REPO_PATHS='/opt/xilinx/platforms'
dg_ipdev@turnkey2:~$ export XILINX_PLATFORM='xilinx_u250_gen3x16_xdma_4_1_202210_1'
dg_ipdev@turnkey2:~$ export DEVICE=${PLATFORM_REPO_PATHS}/${XILINX_PLATFORM}/${XILINX_PLATFORM}.xpfm
dg_ipdev@turnkey2:~$
```

i. Call Xilinx environment and library

ii. Open Xilinx run time

iii. Set Target accelerator card

Figure 3-2 Xilinx environment setting on TKAS-D2101 for U250

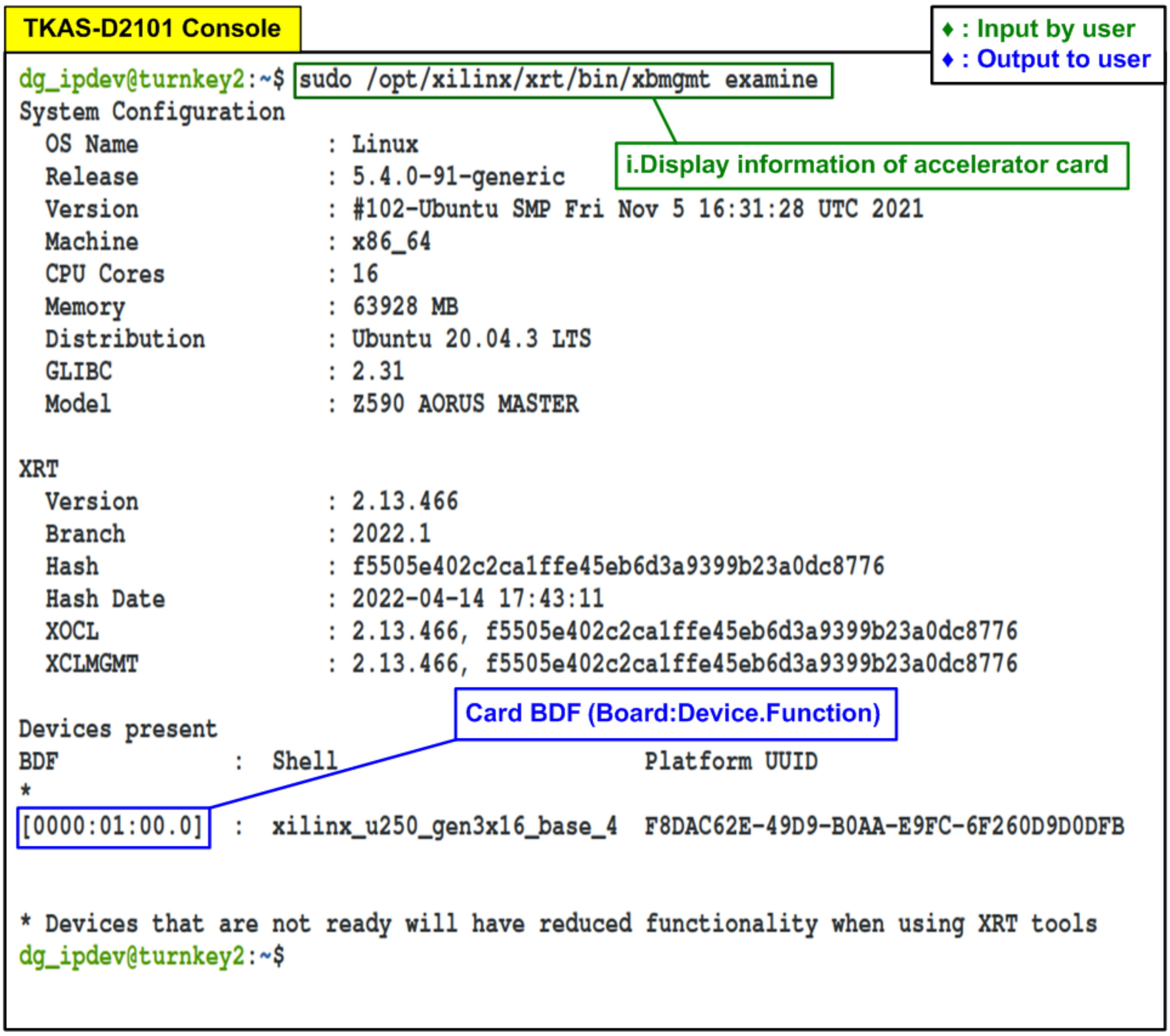
2) Prepare the Xilinx environment and library on the TKAS-D2101, which connects the U250/U50 card through the PCIe connector.
   i) Enter the command to set up the Xilinx environment and library in the terminal
      >> source <install path>/Vivado/2022.1/settings64.sh
   ii) Enter the command to open The Xilinx runtime
      >> source /opt/xilinx/xrt/setup.sh
   iii) Enter the command to set the environment for the Accelerator card (U50 or U250).
      >> export PLATFORM_REPO_PATHS='/opt/xilinx/platforms'

      For U250 card
      >> export XILINX_PLATFORM='xilinx_u250_gen3x16_xdma_4_1_202210_1'
      For U50 card
      >> export XILINX_PLATFORM='xilinx_u50_gen3x16_xdma_5_202210_1'

      >> export DEVICE=${PLATFORM_REPO_PATHS}/${XILINX_PLATFORM}/ ${XILINX_PLATFORM}.xpfm

3) For the U250 card, you need to program the shell partition to the Accelerator card once after system bootup. Refer to page 26 of UG1301 (V2.0) for detailed instructions. https://www.xilinx.com/support/documentation/boards_and_kits/accelerator-cards/2_0/ug1301-getting-started-guide-alveo-accelerator-cards.pdf

   i) User the following command to detect the Accelerator card connected to TKAS-D2101 and obtain the Card BDF ID.
   >> sudo /opt/xilinx/xrt/bin/xbmgmt examine



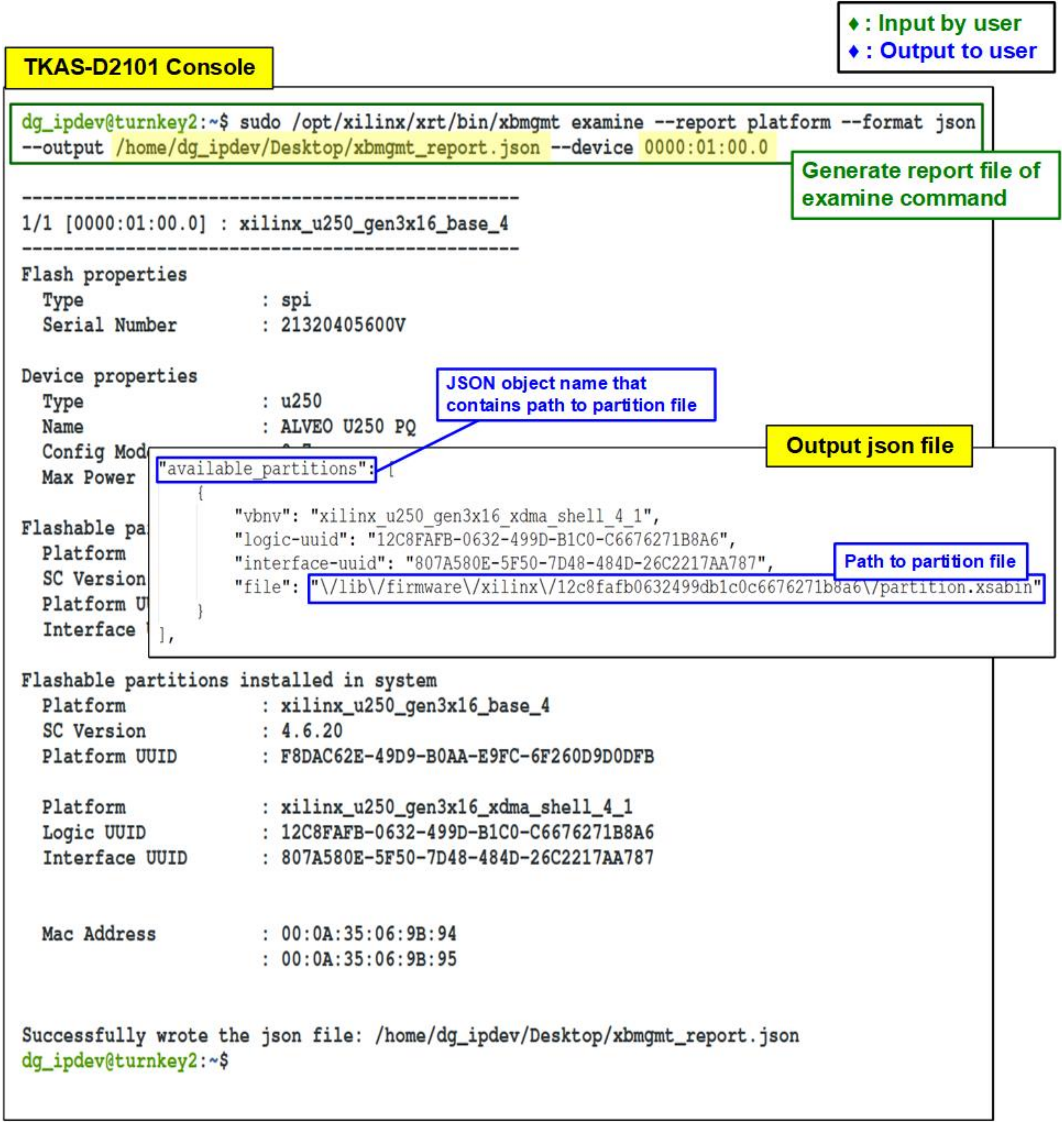
```
dg_ipdev@turnkey2:~$ sudo /opt/xilinx/xrt/bin/xbmgmt examine
System Configuration
  OS Name              : Linux
  Release              : 5.4.0-91-generic
  Version              : #102-Ubuntu SMP Fri Nov 5 16:31:28 UTC 2021
  Machine              : x86_64
  CPU Cores            : 16
  Memory               : 63928 MB
  Distribution         : Ubuntu 20.04.3 LTS
  GLIBC                : 2.31
  Model                : Z590 AORUS MASTER

XRT
  Version              : 2.13.466
  Branch               : 2022.1
  Hash                 : f5505e402c2ca1ffe45eb6d3a9399b23a0dc8776
  Hash Date            : 2022-04-14 17:43:11
  XOCL                 : 2.13.466, f5505e402c2ca1ffe45eb6d3a9399b23a0dc8776
  XCLMGMT              : 2.13.466, f5505e402c2ca1ffe45eb6d3a9399b23a0dc8776

Devices present
BDF               :  Shell                        Platform UUID
*
[0000:01:00.0]    :  xilinx_u250_gen3x16_base_4   F8DAC62E-49D9-B0AA-E9FC-6F260D9D0DFB


* Devices that are not ready will have reduced functionality when using XRT tools
dg_ipdev@turnkey2:~$
```

i.Display information of accelerator card

Card BDF (Board:Device.Function)

Figure 3-3 Examine the accelerator card

ii) Use the following command, along with the “card_BDF” (obtained from step 3i), to generate a JSON output file containing the path of the partition file.
>> sudo /opt/xilinx/xrt/bin/xbmgmt examine --report platform --format json --output <output file.json> --device <card_BDF>

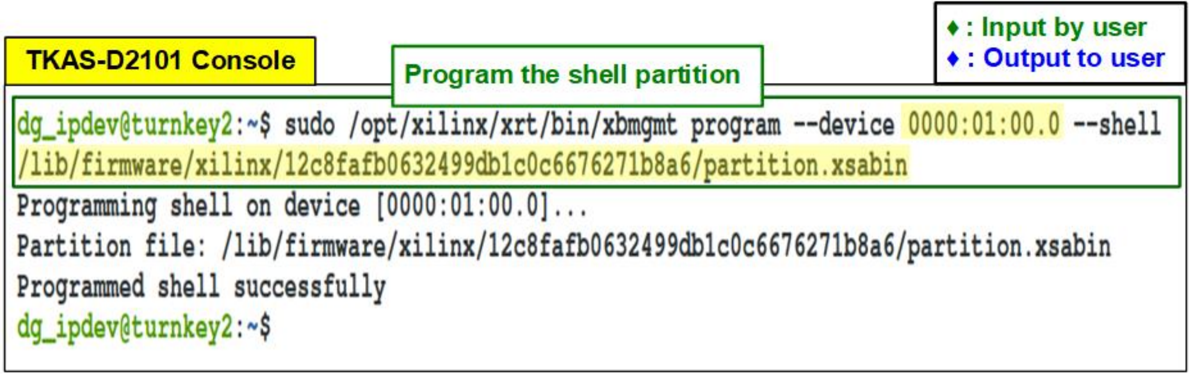

♦ : Input by user
♦ : Output to user
TKAS-D2101 Console
Generate report file of examine command
JSON object name that contains path to partition file
Output json file
Path to partition file

```
dg_ipdev@turnkey2:~$ sudo /opt/xilinx/xrt/bin/xbmgmt examine --report platform --format json
--output /home/dg_ipdev/Desktop/xbmgmt_report.json --device 0000:01:00.0

------------------------------------------------
1/1 [0000:01:00.0] : xilinx_u250_gen3x16_base_4
------------------------------------------------
Flash properties
  Type                 : spi
  Serial Number        : 21320405600V

Device properties
  Type                 : u250
  Name                 : ALVEO U250 PQ
  Config Mod
  Max Power

Flashable pa
  Platform
  SC Version
  Platform U
  Interface

Flashable partitions installed in system
  Platform             : xilinx_u250_gen3x16_base_4
  SC Version           : 4.6.20
  Platform UUID        : F8DAC62E-49D9-B0AA-E9FC-6F260D9D0DFB

  Platform             : xilinx_u250_gen3x16_xdma_shell_4_1
  Logic UUID           : 12C8FAFB-0632-499D-B1C0-C6676271B8A6
  Interface UUID       : 807A580E-5F50-7D48-484D-26C2217AA787


  Mac Address          : 00:0A:35:06:9B:94
                       : 00:0A:35:06:9B:95


Successfully wrote the json file: /home/dg_ipdev/Desktop/xbmgmt_report.json
dg_ipdev@turnkey2:~$
```

```
"available_partitions": [
    {
        "vbnv": "xilinx_u250_gen3x16_xdma_shell_4_1",
        "logic-uuid": "12C8FAFB-0632-499D-B1C0-C6676271B8A6",
        "interface-uuid": "807A580E-5F50-7D48-484D-26C2217AA787",
        "file": "\/lib\/firmware\/xilinx\/12c8fafb0632499db1c0c6676271b8a6\/partition.xsabin"
    }
],
```

Figure 3-4 Generate JSON report file of xbmgmt

iii) Program the shell partition to the target Alveo card using the following command, along with the “card_BDF” and the path of the partition file determined in step 3i and step 3ii. Remove the backslash character (‘\’) from the path to the partition file in the output JSON file before using in this command.
>> sudo /opt/xilinx/xrt/bin/xbmgmt program --device <card_BDF> --shell <path to partition file>

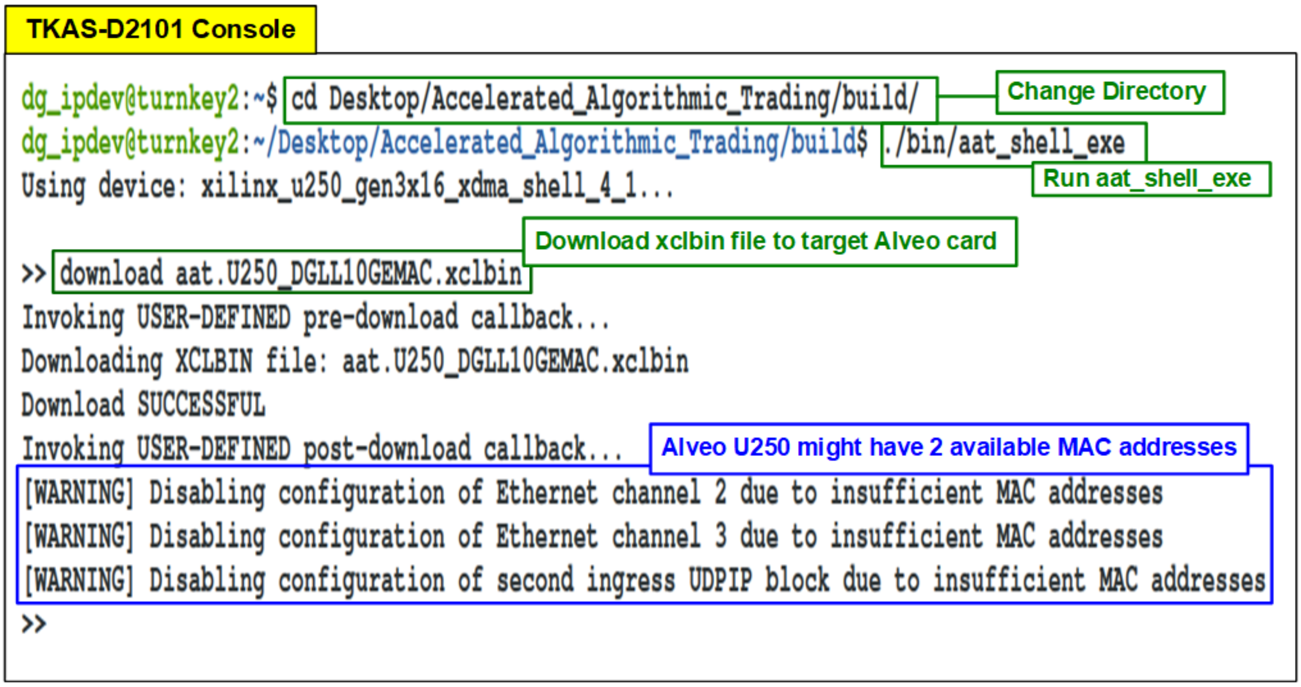


```
dg_ipdev@turnkey2:~$ sudo /opt/xilinx/xrt/bin/xbmgmt program --device 0000:01:00.0 --shell
/lib/firmware/xilinx/12c8fafb0632499db1c0c6676271b8a6/partition.xsabin
Programming shell on device [0000:01:00.0]...
Partition file: /lib/firmware/xilinx/12c8fafb0632499db1c0c6676271b8a6/partition.xsabin
Programmed shell successfully
dg_ipdev@turnkey2:~$
```

Figure 3-5 Program the shell partition

4) Copy the “aat.u250_DGLL10GEMAC.xclbin” file provided by DesignGateway to the directory “../Accelerated_Algorithmic_Trading/build”. The default path of “aat_shell_exe” in the AAT demo design is “../Accelerated_Algorithmic_Trading/build/bin”.
Note: If the build directory does not contain “aat_shell_exe”, you needs to build it following the instructions in Chapter 7 (Building and Running the AAT) of the “UG1067 Accelerated Algorithmic Trading User Guide” document.

5) Navigate to the directory that contains “./bin/aat_dgllip_shell_exe” and the xclbin file of the demo. Then, run the demo using the following command.
>> cd <directory of xclbin file >
>> ./bin/aat_shell_exe
>> download aat.u250_DGLL10GEMAC.xclbin or aat.u50_DGLL10GEMAC.xclbin


TKAS-D2101 Console
Change Directory
Run aat_shell_exe
Download xclbin file to target Alveo card
Alveo U250 might have 2 available MAC addresses

```
dg_ipdev@turnkey2:~$ cd Desktop/Accelerated_Algorithmic_Trading/build/
dg_ipdev@turnkey2:~/Desktop/Accelerated_Algorithmic_Trading/build$ ./bin/aat_shell_exe
Using device: xilinx_u250_gen3x16_xdma_shell_4_1...

>> download aat.U250_DGLL10GEMAC.xclbin
Invoking USER-DEFINED pre-download callback...
Downloading XCLBIN file: aat.U250_DGLL10GEMAC.xclbin
Download SUCCESSFUL
Invoking USER-DEFINED post-download callback...
[WARNING] Disabling configuration of Ethernet channel 2 due to insufficient MAC addresses
[WARNING] Disabling configuration of Ethernet channel 3 due to insufficient MAC addresses
[WARNING] Disabling configuration of second ingress UDPIP block due to insufficient MAC addresses
>>
```

Figure 3-6 Download xclbin file

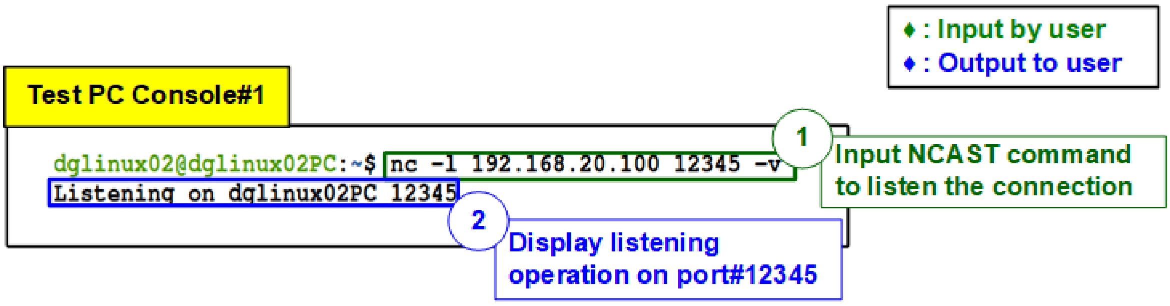
4 Run AAT Demo

To execute the demo, there are three steps for the user. Firstly, there is an initialization process to establish the connection and configure the parameters. Next, the market data transmission process begins, which involves sending market data from the Test PC. Lastly, the test status is received from the Accelerator card and displayed on the console of TKAS-D2101, providing a comprehensive result. Further information about each of these processes is provided below.

4.1 Initialization

To run the AAT demo, the user needs to enter a command on the Test PC console. This command enables the Test PC to listen on a specific port, which is designated for receiving order packets from the Accelerator card once it has completed processing the market data. Similarly, the Accelerator card requires configuration to establish parameters for receiving market data and sending the order packet. This configuration is done using "demo_setup_cfg" script file. Further information regarding the system initialization process is outlined below.

1) On the Test PC console, enter the command to listen on port 12345.
>> nc -l 192.168.20.100 12345 -v



```
dglinux02@dglinux02PC:~$ nc -l 192.168.20.100 12345 -v
Listening on dglinux02PC 12345
```

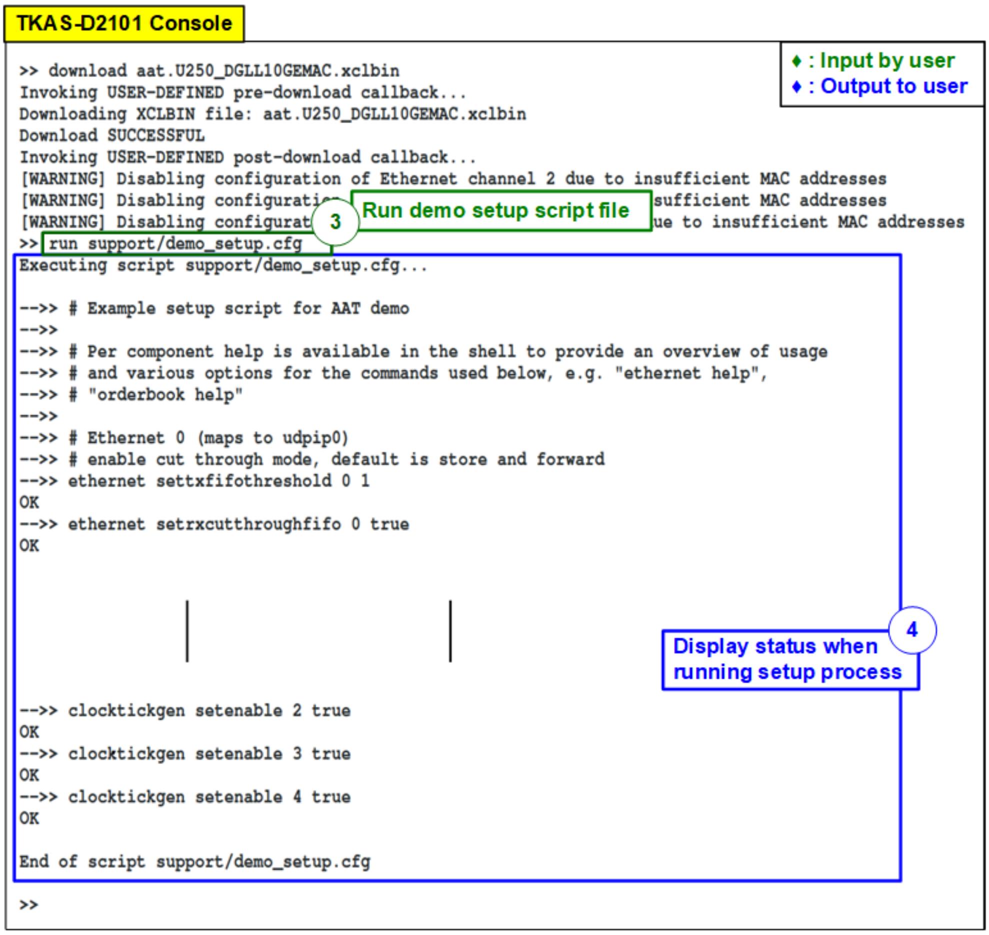
1 Input NCAST command to listen the connection

2 Display listening operation on port#12345

Figure 4-1 Listen TCP port on Test PC

2) You will see a confirmation message on the console "Listening on <Test PC name> 12345" indicating that the port is listening, as shown in Figure 4-1.

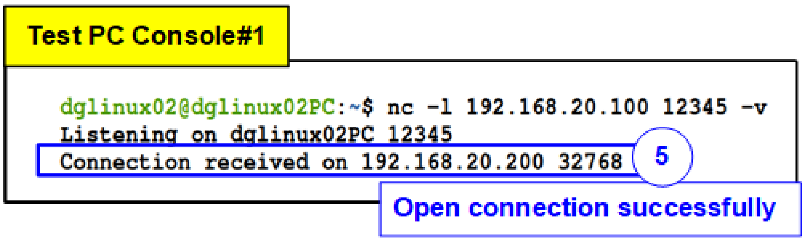
3) Run the script file “demo_setup.cfg” on TKAS-D2101 to set up the parameters for processing market data. Use the following command.
   >> run support/demo_setup.cfg
   Note: The demo_setup.cfg script file can only be run once after downloading the xclbin file. If you need to rerun the script file, you must re-download the xclbin file.


TKAS-D2101 Console
♦ : Input by user
♦ : Output to user
3
Run demo setup script file
4
Display status when running setup process

```
>> download aat.U250_DGLL10GEMAC.xclbin
Invoking USER-DEFINED pre-download callback...
Downloading XCLBIN file: aat.U250_DGLL10GEMAC.xclbin
Download SUCCESSFUL
Invoking USER-DEFINED post-download callback...
[WARNING] Disabling configuration of Ethernet channel 2 due to insufficient MAC addresses
[WARNING] Disabling configurati                            sufficient MAC addresses
[WARNING] Disabling configurat                             ue to insufficient MAC addresses
>> run support/demo_setup.cfg
Executing script support/demo_setup.cfg...

-->> # Example setup script for AAT demo
-->>
-->> # Per component help is available in the shell to provide an overview of usage
-->> # and various options for the commands used below, e.g. "ethernet help",
-->> # "orderbook help"
-->>
-->> # Ethernet 0 (maps to udpip0)
-->> # enable cut through mode, default is store and forward
-->> ethernet settxfifothreshold 0 1
OK
-->> ethernet setrxcutthroughfifo 0 true
OK

                 |                          |


-->> clocktickgen setenable 2 true
OK
-->> clocktickgen setenable 3 true
OK
-->> clocktickgen setenable 4 true
OK

End of script support/demo_setup.cfg

>>
```

Figure 4-2 Run demo setup script

4) TKAS-D2101 will display messages indicating the setup process, as shown in Figure 4-2.
5) If the parameter configuration is successful, the Test PC console will display a message indicating that the port has been opened successfully, such as “Connection received on 192.168.20.200 32768”, as shown in Figure 4-3.



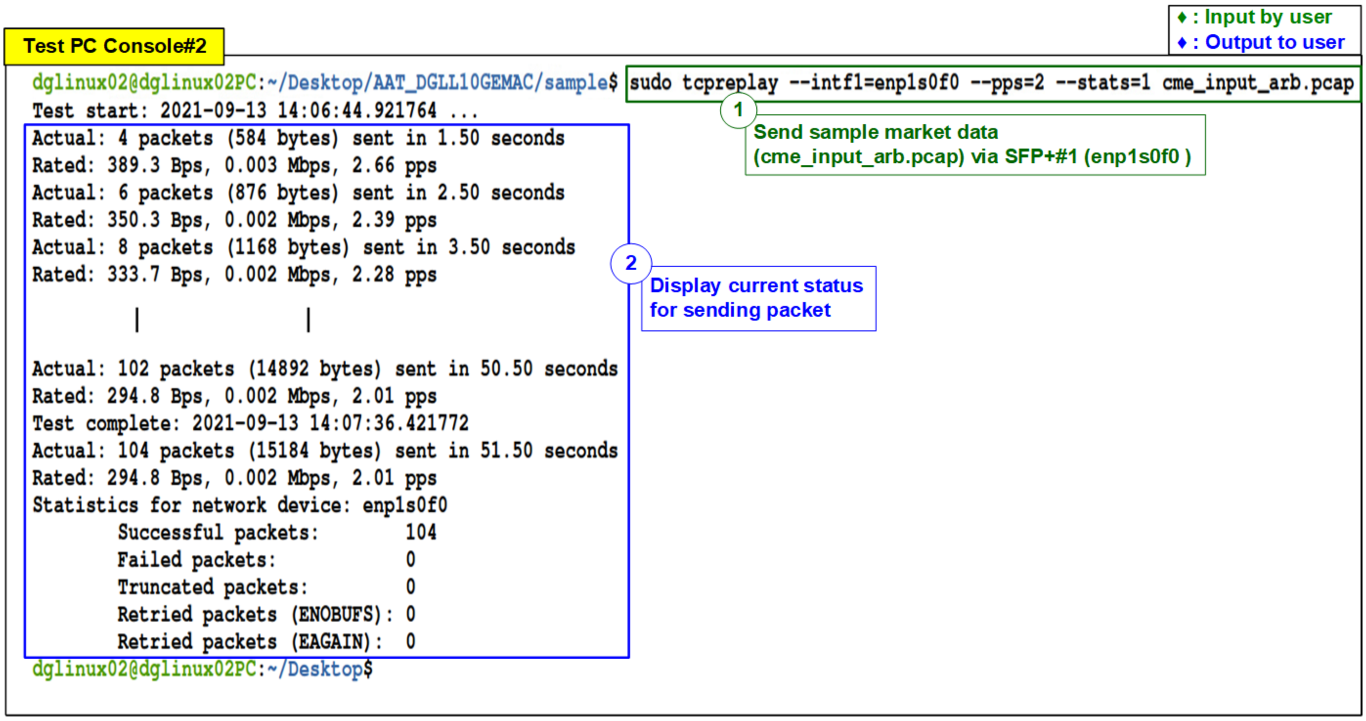
```
dglinux02@dglinux02PC:~$ nc -l 192.168.20.100 12345 -v
Listening on dglinux02PC 12345
Connection received on 192.168.20.200 32768
```

Figure 4-3 Open connection successes

4.2 Market data transmission

To transmit the sample market data, follow these steps using the “tcpreplay” on Test PC. Additionally, you will need to open two consoles on Test PC: Test PC Console#1 and Test PC Console#2. Test PC Console#1 will display details of the order packet received, while Test PC Console#2 will be used to send the sample market data. Here are the detailed instructions.

1) Send Sample market data. Use the “tcpreplay” command to send the provided sample market data file in Xilinx AAT demo (cme_input_arb.pcap). Execute the following command with four parameters.
   >> sudo tcpreplay –intf1=<eth I/F> --pps=<pac/sec> --stats=<stat period> <replay file>
   i) <eth I/F> : Ethernet interface for sending market data (SFP+#1: enp1s0f0).
   ii) <pac/sec> : Transfer speed, defined as the number of packets per second.
   iii) <stat period> : Time period in seconds to display status on the console.
   iv) <replay file> : File name of the data to transmit, which is cme_input_arb.pcap (the sample market data provided by Xilinx AAT demo).
   Note: Contact Xilinx to obtain the sample market data and AAT demo.



Test PC Console#2

♦ : Input by user
♦ : Output to user

```
dglinux02@dglinux02PC:~/Desktop/AAT_DGLL10GEMAC/sample$ sudo tcpreplay --intf1=enp1s0f0 --pps=2 --stats=1 cme_input_arb.pcap
Test start: 2021-09-13 14:06:44.921764 ...
Actual: 4 packets (584 bytes) sent in 1.50 seconds
Rated: 389.3 Bps, 0.003 Mbps, 2.66 pps
Actual: 6 packets (876 bytes) sent in 2.50 seconds
Rated: 350.3 Bps, 0.002 Mbps, 2.39 pps
Actual: 8 packets (1168 bytes) sent in 3.50 seconds
Rated: 333.7 Bps, 0.002 Mbps, 2.28 pps

          |               |

Actual: 102 packets (14892 bytes) sent in 50.50 seconds
Rated: 294.8 Bps, 0.002 Mbps, 2.01 pps
Test complete: 2021-09-13 14:07:36.421772
Actual: 104 packets (15184 bytes) sent in 51.50 seconds
Rated: 294.8 Bps, 0.002 Mbps, 2.01 pps
Statistics for network device: enp1s0f0
        Successful packets:        104
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
dglinux02@dglinux02PC:~/Desktop$
```

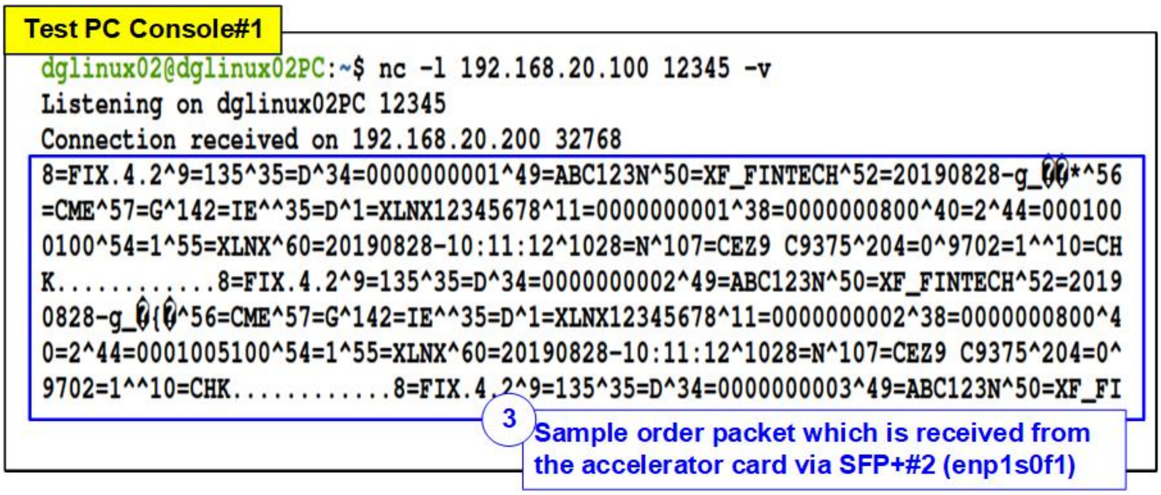
1 Send sample market data (cme_input_arb.pcap) via SFP+#1 (enp1s0f0)

2 Display current status for sending packet

Figure 4-4 Send sample market data by “tcpreplay”

2) The console will display the status, showing the total number of transmitted packets every second.

3) On Test PC console#1, which is already connected to the listening port, the console will display the received data, which represents the sample order packet returned by the Accelerator card and serves as the result of the AAT demo.



Test PC Console#1

```
dglinux02@dglinux02PC:~$ nc -l 192.168.20.100 12345 -v
Listening on dglinux02PC 12345
Connection received on 192.168.20.200 32768
8=FIX.4.2^9=135^35=D^34=0000000001^49=ABC123N^50=XF_FINTECH^52=20190828-g_��*^56
=CME^57=G^142=IE^^35=D^1=XLNX12345678^11=0000000001^38=0000000800^40=2^44=000100
0100^54=1^55=XLNX^60=20190828-10:11:12^1028=N^107=CEZ9 C9375^204=0^9702=1^^10=CH
K............8=FIX.4.2^9=135^35=D^34=0000000002^49=ABC123N^50=XF_FINTECH^52=2019
0828-g_�{�^56=CME^57=G^142=IE^^35=D^1=XLNX12345678^11=0000000002^38=0000000800^4
0=2^44=0001005100^54=1^55=XLNX^60=20190828-10:11:12^1028=N^107=CEZ9 C9375^204=0^
9702=1^^10=CHK............8=FIX.4.2^9=135^35=D^34=0000000003^49=ABC123N^50=XF_FI
```

3 Sample order packet which is received from the accelerator card via SFP+#2 (enp1s0f1)

Figure 4-5 The sample data of order packet on SFP+#2 channel

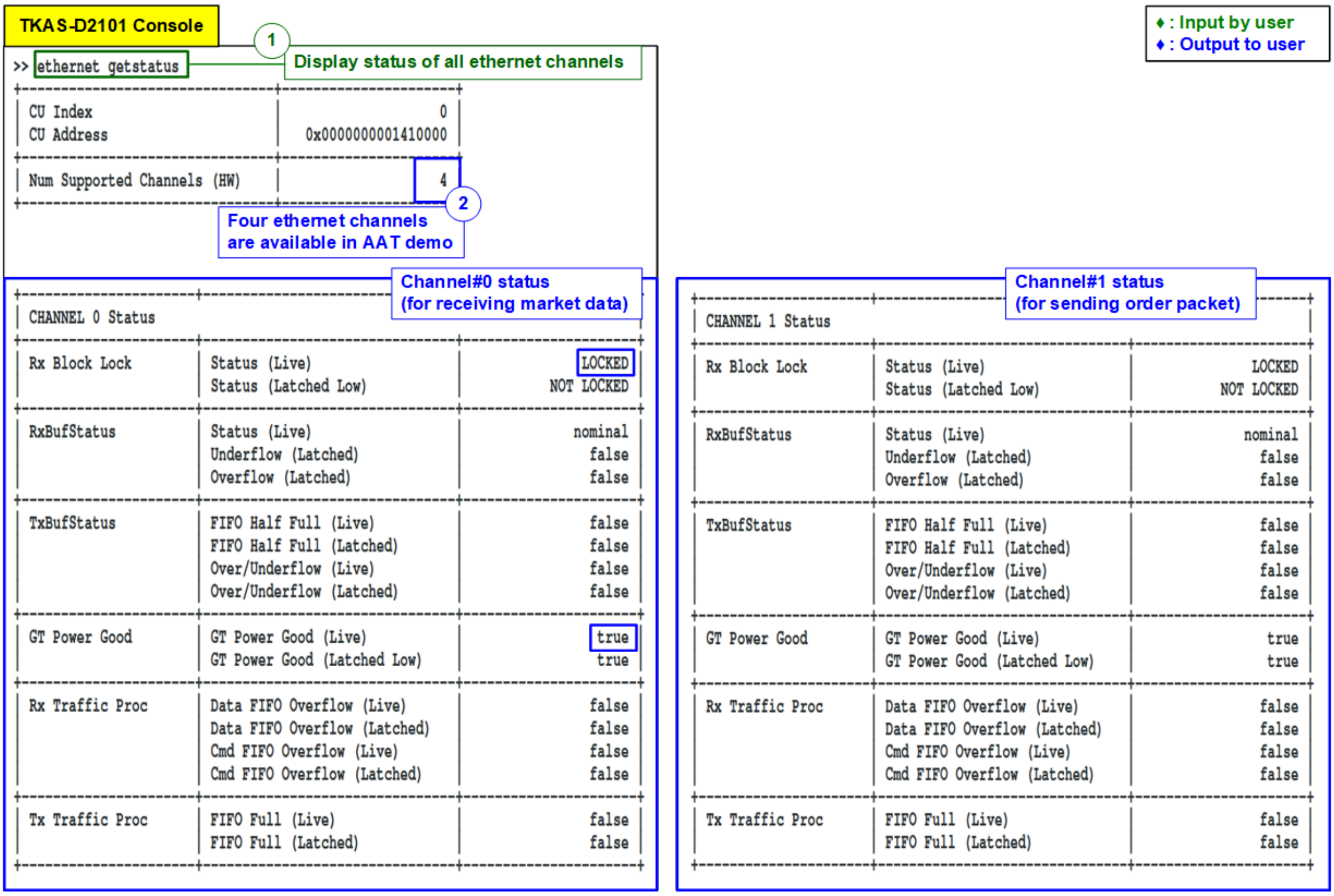
4.3 AAT demo

This topic shows the example results of market data processing on the Accelerator card. The AAT demo design comprises several kernels responsible for processing the sample market data. Specifically, we will focus on four kernels: Ethernet kernel, Feed handler kernel, Order book kernel, and Order entry kernel. The following details provide insights into the sample results obtained from these kernels within the AAT demo design.

4.3.1 Ethernet Kernel

To check the status of the Ethernet kernel in the AAT demo system, follow these steps.

1) Enter the following command to display the status of Ethernet kernel.
>> ethernet getstatus



TKAS-D2101 Console

♦ : Input by user
♦ : Output to user

1 Display status of all ethernet channels

2 Four ethernet channels are available in AAT demo

Channel#0 status (for receiving market data)

Channel#1 status (for sending order packet)

```
>> ethernet getstatus
+-----------------------------------+---------------------+
| CU Index                          |                   0 |
| CU Address                        |  0x0000000001410000 |
+-----------------------------------+---------------------+
| Num Supported Channels (HW)       |                   4 |
+-----------------------------------+---------------------+

+---------------------+-----------------------------+---------------------+
| CHANNEL 0 Status                                                        |
+---------------------+-----------------------------+---------------------+
| Rx Block Lock       | Status (Live)               |              LOCKED |
|                     | Status (Latched Low)        |          NOT LOCKED |
+---------------------+-----------------------------+---------------------+
| RxBufStatus         | Status (Live)               |             nominal |
|                     | Underflow (Latched)         |               false |
|                     | Overflow (Latched)          |               false |
+---------------------+-----------------------------+---------------------+
| TxBufStatus         | FIFO Half Full (Live)       |               false |
|                     | FIFO Half Full (Latched)    |               false |
|                     | Over/Underflow (Live)       |               false |
|                     | Over/Underflow (Latched)    |               false |
+---------------------+-----------------------------+---------------------+
| GT Power Good       | GT Power Good (Live)        |                true |
|                     | GT Power Good (Latched Low) |                true |
+---------------------+-----------------------------+---------------------+
| Rx Traffic Proc     | Data FIFO Overflow (Live)   |               false |
|                     | Data FIFO Overflow (Latched)|               false |
|                     | Cmd FIFO Overflow (Live)    |               false |
|                     | Cmd FIFO Overflow (Latched) |               false |
+---------------------+-----------------------------+---------------------+
| Tx Traffic Proc     | FIFO Full (Live)            |               false |
|                     | FIFO Full (Latched)         |               false |
+---------------------+-----------------------------+---------------------+

+---------------------+-----------------------------+---------------------+
| CHANNEL 1 Status                                                        |
+---------------------+-----------------------------+---------------------+
| Rx Block Lock       | Status (Live)               |              LOCKED |
|                     | Status (Latched Low)        |          NOT LOCKED |
+---------------------+-----------------------------+---------------------+
| RxBufStatus         | Status (Live)               |             nominal |
|                     | Underflow (Latched)         |               false |
|                     | Overflow (Latched)          |               false |
+---------------------+-----------------------------+---------------------+
| TxBufStatus         | FIFO Half Full (Live)       |               false |
|                     | FIFO Half Full (Latched)    |               false |
|                     | Over/Underflow (Live)       |               false |
|                     | Over/Underflow (Latched)    |               false |
+---------------------+-----------------------------+---------------------+
| GT Power Good       | GT Power Good (Live)        |                true |
|                     | GT Power Good (Latched Low) |                true |
+---------------------+-----------------------------+---------------------+
| Rx Traffic Proc     | Data FIFO Overflow (Live)   |               false |
|                     | Data FIFO Overflow (Latched)|               false |
|                     | Cmd FIFO Overflow (Live)    |               false |
|                     | Cmd FIFO Overflow (Latched) |               false |
+---------------------+-----------------------------+---------------------+
| Tx Traffic Proc     | FIFO Full (Live)            |               false |
|                     | FIFO Full (Latched)         |               false |
+---------------------+-----------------------------+---------------------+
```

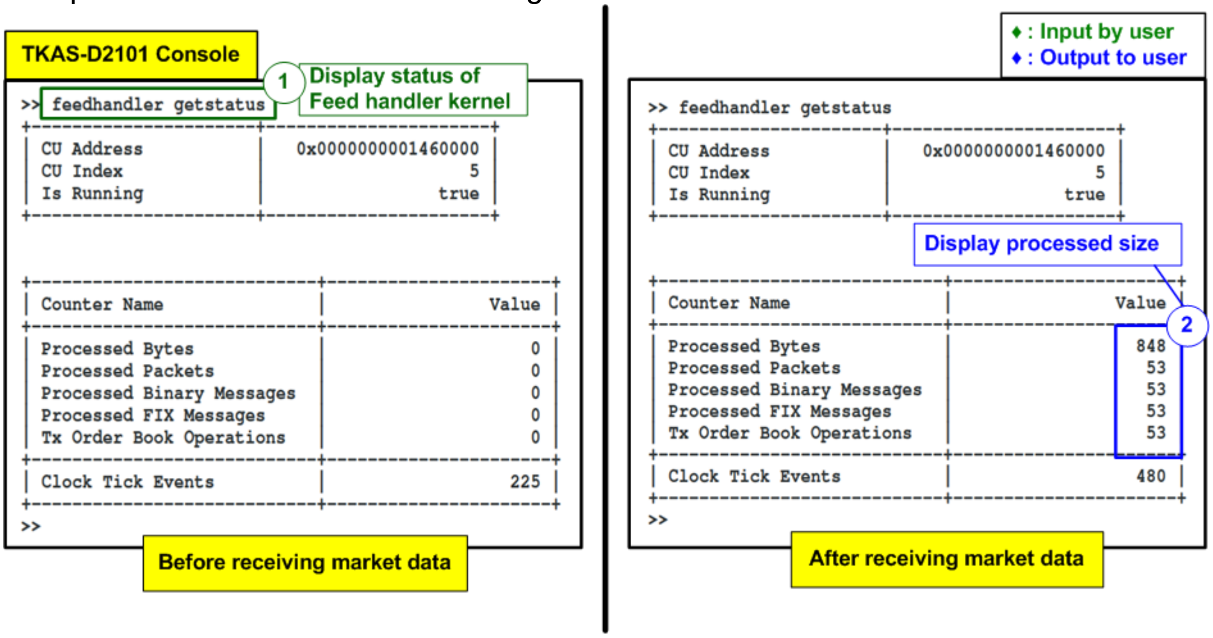
Figure 4-6: Ethernet kernel status

2) The AAT demo system has four Ethernet channels: channel0 to channel3. In this example, channel#0 is used for receiving sample market data, and channel#1 is used for returning the sample order packet. Verify that the status of channel#0 and channel#1 is in good condition by checking the following parameters.
   i) Rx Block Lock Status (Live) : LOCKED
   ii) GT Power Good (Live) : true

4.3.2 Feed Handler Kernel

To check the status of the Feed Handler kernel in the AAT demo system, follow these steps.

1) Enter the following command to display the status of the Feed handler kernel.
   >> feedhandler getstatus
2) The console will display the processed data count in various units, such as bytes, packets, and messages. Initially, before transmitting the sample market data, the processed data count will be zero. However, after transmitting the market data, the processed data count will no longer be zero.


TKAS-D2101 Console
1
Display status of Feed handler kernel
♦ : Input by user
♦ : Output to user
Display processed size
2
Before receiving market data
After receiving market data

```
>> feedhandler getstatus
+-----------------------+----------------------+
| CU Address            |  0x0000000001460000 |
| CU Index              |                    5 |
| Is Running            |                 true |
+-----------------------+----------------------+


+---------------------------+----------------------+
| Counter Name              |                Value |
+---------------------------+----------------------+
| Processed Bytes           |                    0 |
| Processed Packets         |                    0 |
| Processed Binary Messages |                    0 |
| Processed FIX Messages    |                    0 |
| Tx Order Book Operations  |                    0 |
+---------------------------+----------------------+
| Clock Tick Events         |                  225 |
+---------------------------+----------------------+
>>
```

```
>> feedhandler getstatus
+-----------------------+----------------------+
| CU Address            |  0x0000000001460000 |
| CU Index              |                    5 |
| Is Running            |                 true |
+-----------------------+----------------------+

+---------------------------+----------------------+
| Counter Name              |                Value |
+---------------------------+----------------------+
| Processed Bytes           |                  848 |
| Processed Packets         |                   53 |
| Processed Binary Messages |                   53 |
| Processed FIX Messages    |                   53 |
| Tx Order Book Operations  |                   53 |
+---------------------------+----------------------+
| Clock Tick Events         |                  480 |
+---------------------------+----------------------+
>>
```

Figure 4-7 Feed handler kernel status

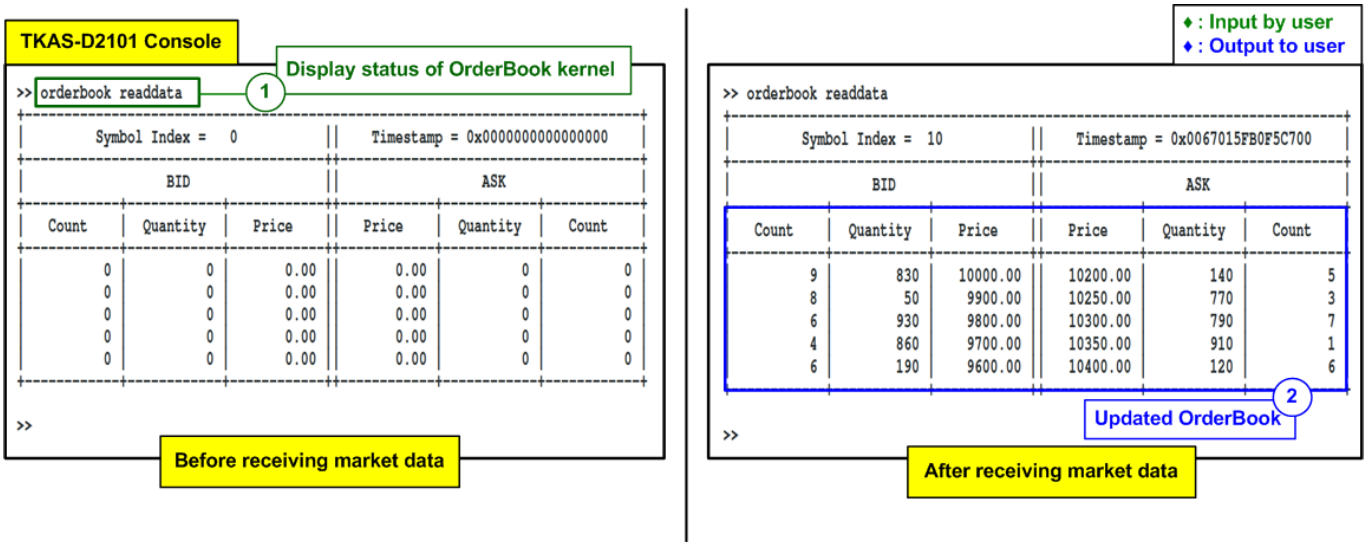
For example, in Figure 4-7, the left window shows that the processed data count is initially zero. After transmitting the sample market data, the processed data count will increase, indicating that the Feed Handler kernel is processing the data.

In Figure 4-4, it shows that there were 104 packets of sample market data sent by the Test PC. Out of these packets, 53 packets were deemed valid for Feed Handler processing, while the remaining packets were rejected by the Feed handler kernel.

4.3.3 OrderBook Kernel

To check the status of the OrderBook kernel in the AAT demo system, follow these steps.

1) Enter the following command to display the order book output from the Order Book kernel.
   >> orderbook readdata
2) The console will show the current values of the order book. Initially, before transmitting the sample market data, the order book will be a clean state. However, after transferring all the sample market data, the order book will be updated by the OrderBook kernel.



TKAS-D2101 Console
Display status of OrderBook kernel
1
Before receiving market data
♦ : Input by user
♦ : Output to user
2
Updated OrderBook
After receiving market data

```
>> orderbook readdata
+---------------------------------------------------------------+
|        Symbol Index =   0        ||  Timestamp = 0x0000000000000000  |
+---------------------------------++--------------------------------+
|               BID               ||               ASK              |
+----------+----------+-----------++----------+----------+----------+
|  Count   | Quantity |   Price   ||  Price   | Quantity |  Count   |
+----------+----------+-----------++----------+----------+----------+
|        0 |        0 |      0.00 ||     0.00 |        0 |        0 |
|        0 |        0 |      0.00 ||     0.00 |        0 |        0 |
|        0 |        0 |      0.00 ||     0.00 |        0 |        0 |
|        0 |        0 |      0.00 ||     0.00 |        0 |        0 |
|        0 |        0 |      0.00 ||     0.00 |        0 |        0 |
+----------+----------+-----------++----------+----------+----------+

>>
```

```
>> orderbook readdata
+---------------------------------------------------------------+
|        Symbol Index =  10        ||  Timestamp = 0x0067015FB0F5C700  |
+---------------------------------++--------------------------------+
|               BID               ||               ASK              |
+----------+----------+-----------++----------+----------+----------+
|  Count   | Quantity |   Price   ||  Price   | Quantity |  Count   |
+----------+----------+-----------++----------+----------+----------+
|        9 |      830 |  10000.00 || 10200.00 |      140 |        5 |
|        8 |       50 |   9900.00 || 10250.00 |      770 |        3 |
|        6 |      930 |   9800.00 || 10300.00 |      790 |        7 |
|        4 |      860 |   9700.00 || 10350.00 |      910 |        1 |
|        6 |      190 |   9600.00 || 10400.00 |      120 |        6 |
+----------+----------+-----------++----------+----------+----------+

>>
```

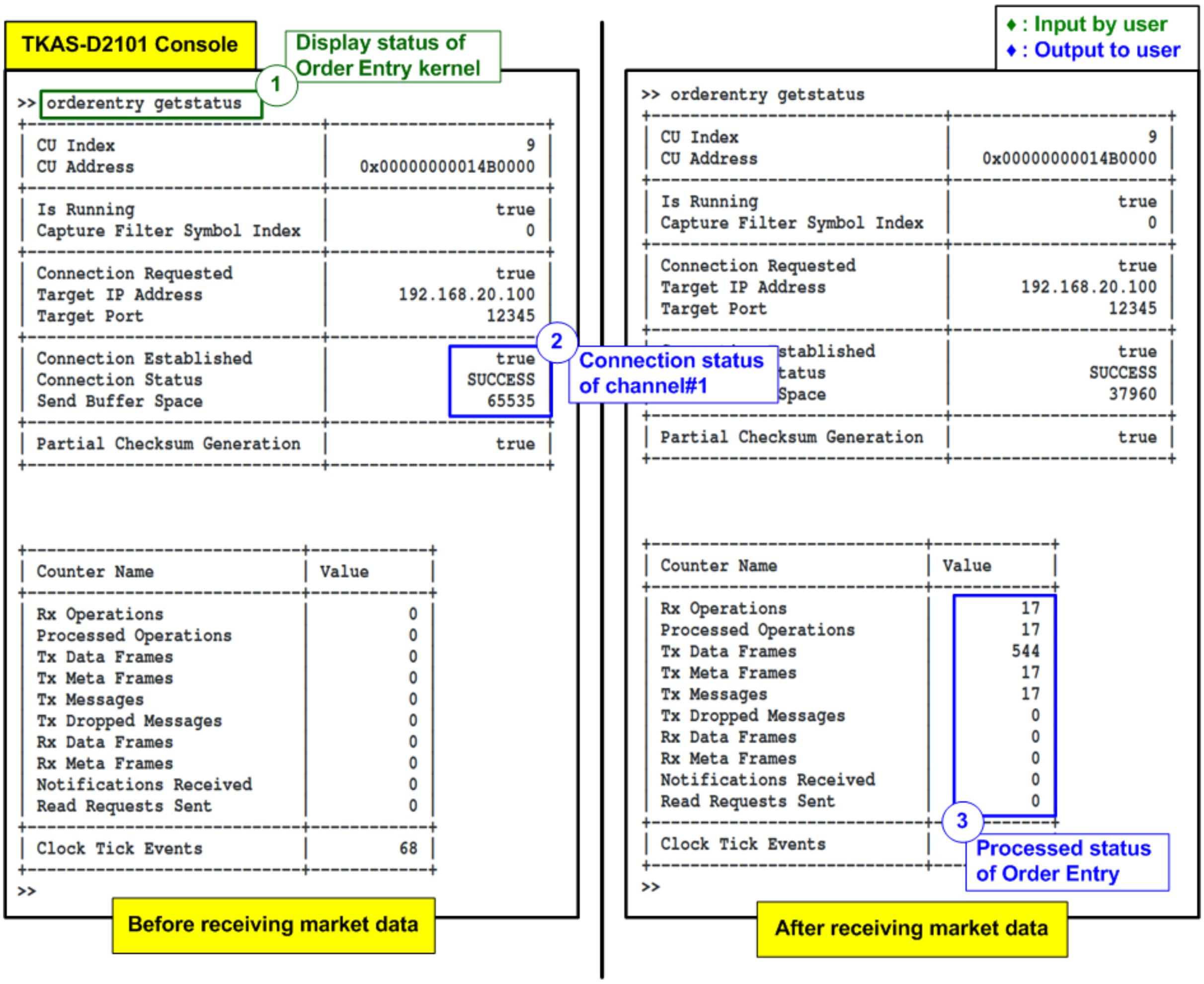
Figure 4-8 Updated OrderBook after finishing processing

For instance, in Figure 4-8, the left window displays the clean status of the order book before transmitting the sample market data. The right window shows the updated order book after transferring all the sample market data. The bid/ask quantity and bid/ask price are updated by the OrderBook kernel, reflecting the changes in the market data.

4.3.4 Order Entry Kernel

To check the current status of the Order Entry kernel in the AAT demo system, follow these steps.

1) Enter the following command to display the current status of the Order Entry kernel.
   >> orderentry status
2) Before starting the transmission of sample market data, you can check the status of Ethernet channel#1 from the Order Entry kernel status. Two key indicators to check the connection status are as follows.
   i) Connection Established : true
   ii) Connection Status : SUCCESS


TKAS-D2101 Console
Display status of Order Entry kernel
1
2
Connection status of channel#1
♦ : Input by user
♦ : Output to user
3
Processed status of Order Entry
Before receiving market data
After receiving market data

```
>> orderentry getstatus
+--------------------------------+--------------------+
| CU Index                       |                  9 |
| CU Address                     | 0x00000000014B0000 |
+--------------------------------+--------------------+
| Is Running                     |               true |
| Capture Filter Symbol Index    |                  0 |
+--------------------------------+--------------------+
| Connection Requested           |               true |
| Target IP Address              |     192.168.20.100 |
| Target Port                    |              12345 |
+--------------------------------+--------------------+
| Connection Established         |               true |
| Connection Status              |            SUCCESS |
| Send Buffer Space              |              65535 |
+--------------------------------+--------------------+
| Partial Checksum Generation    |               true |
+--------------------------------+--------------------+


+-----------------------------+------------+
| Counter Name                | Value      |
+-----------------------------+------------+
| Rx Operations               |          0 |
| Processed Operations        |          0 |
| Tx Data Frames              |          0 |
| Tx Meta Frames              |          0 |
| Tx Messages                 |          0 |
| Tx Dropped Messages         |          0 |
| Rx Data Frames              |          0 |
| Rx Meta Frames              |          0 |
| Notifications Received      |          0 |
| Read Requests Sent          |          0 |
+-----------------------------+------------+
| Clock Tick Events           |         68 |
+-----------------------------+------------+
>>
```

```
>> orderentry getstatus
+--------------------------------+--------------------+
| CU Index                       |                  9 |
| CU Address                     | 0x00000000014B0000 |
+--------------------------------+--------------------+
| Is Running                     |               true |
| Capture Filter Symbol Index    |                  0 |
+--------------------------------+--------------------+
| Connection Requested           |               true |
| Target IP Address              |     192.168.20.100 |
| Target Port                    |              12345 |
+--------------------------------+--------------------+
|                stablished      |               true |
|                tatus           |            SUCCESS |
|                Space           |              37960 |
+--------------------------------+--------------------+
| Partial Checksum Generation    |               true |
+--------------------------------+--------------------+


+-----------------------------+------------+
| Counter Name                | Value      |
+-----------------------------+------------+
| Rx Operations               |         17 |
| Processed Operations        |         17 |
| Tx Data Frames              |        544 |
| Tx Meta Frames              |         17 |
| Tx Messages                 |         17 |
| Tx Dropped Messages         |          0 |
| Rx Data Frames              |          0 |
| Rx Meta Frames              |          0 |
| Notifications Received      |          0 |
| Read Requests Sent          |          0 |
+-----------------------------+------------+
| Clock Tick Events           |
+-----------------------------+
>>
```

Figure 4-9 Order Entry kernel status

3) After transmitting the market data completely, the packet count of the Order Entry kernel will be updated from 0 to a new value. This value represents the total number of message/frames processed by the Order Entry kernel.

5 Revision History

Revision	Date	Description
1.2	26-May-23	Add table of contents and correct the website of an example cables
1.1	7-Sep-22	Support AAT2022Q1 released version
1.0	11-Oct-21	Initial version release