Build Your Own Traffic Generator – DPDK-in-a-Box

Table of Contents

Introduction

The purpose of this cookbook is to guide users through the steps required to build a Data Plane Development Kit (DPDK) based traffic generator.

We built a DPDK-in-a-Box using the MinnowBoard Turbot, which is a low cost, portable platform based on the Intel® Atom™ processor E3826. For the OS, we installed Ubuntu 16.04 client with DPDK. The instructions in this document are tested on our DPDK-in-a-Box, as well as on an Intel® Core i7-5960X Haswell-E desktop. You can use any Intel® Architecture (IA) platform to build your own device.

For the traffic generator, we will use the TRex* realistic traffic generator. The TRex package is self-contained and can be easily installed.

About the Author

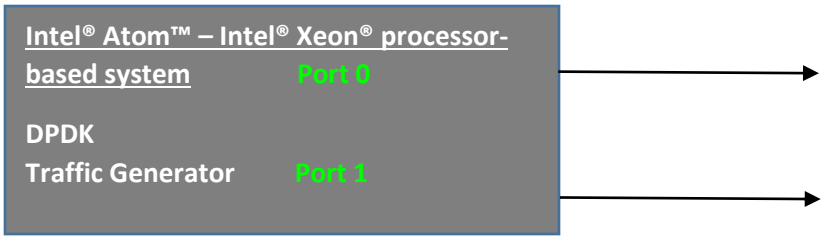
M Jay has worked with the Intel DPDK team since 2009. He joined Intel in 1991 and has worked in various divisions and roles within Intel as a 64-bit CPU front side bus architect and 64-bit HAL developer before joining the Intel DPDK team. M Jay holds 21 US patents, both individually and jointly, all issued while working at Intel. M Jay was awarded the Intel Achievement Award in 2016, Intel's highest honor based on innovation and results.

Please send your feedback to M Jay Muthurajan.Jayakumar@intel.com


Any Intel® processor-based platform will work—desktop, server, laptop or embedded system.

The DPDK Traffic Generator

Block Diagram



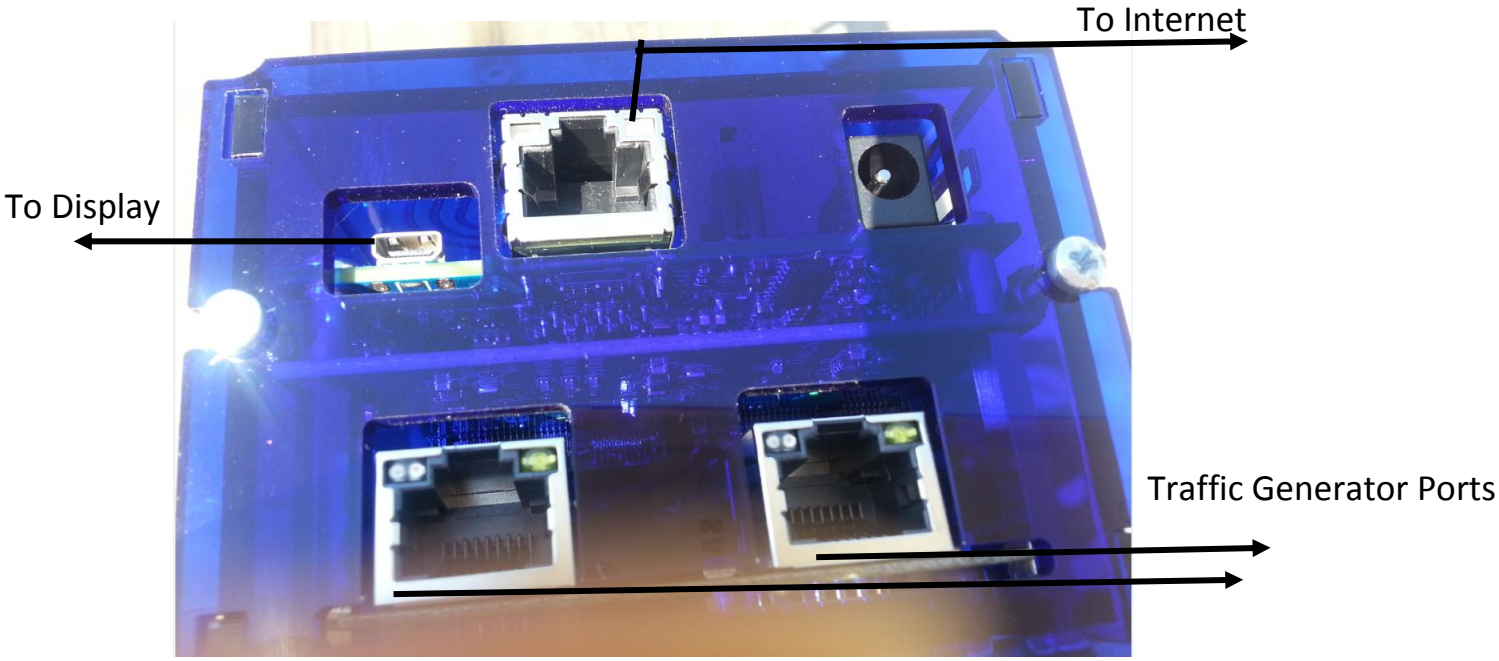
Software

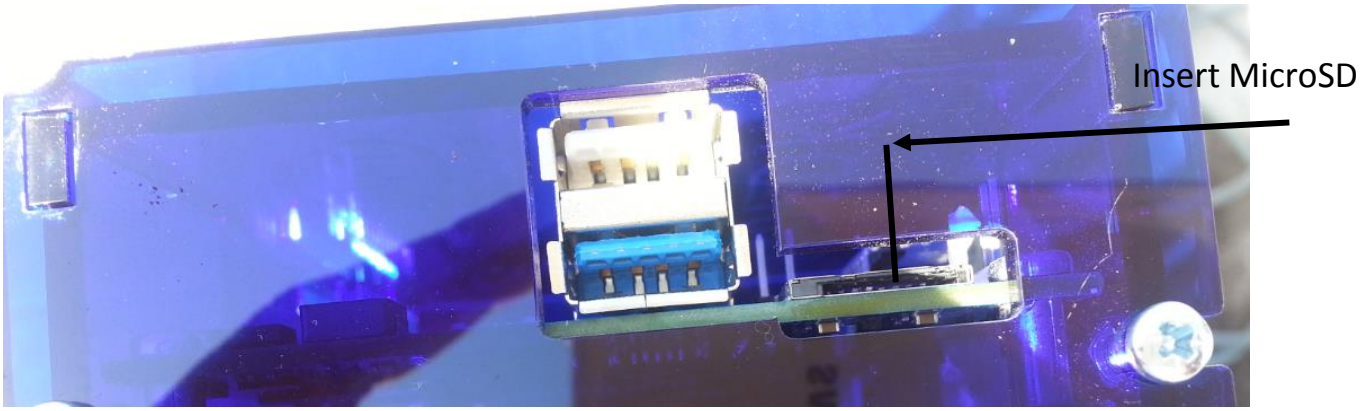
- Ubuntu 16.04 Client OS with DPDK installed
- TRex* realistic traffic generator

Hardware

Our DPDK-in-a-Box uses a MinnowBoard Turbot single board computer:

- Out of the three Ethernet ports, the two at the bottom are for the traffic generator (dual gigabit Intel® Ethernet Controller I350). Connect a loopback cable between them.
- Connect the third Ethernet port to the Internet (to download the TRex package).
- Connect the keyboard and mouse to the USB ports.
- Connect a display to the HDMI Interface.





The MinnowBoard Turbot

The MinnowBoard includes a microSD card and an SD adapter.

- Insert the microSD card into the microSD Slot. The SD adapter should be ignored and not used.
- Power on the DPDK-in-a-Box system. Ubuntu will be up and running right away.

Choose the username `test` and assign the password `tester` (or use the username and password specified by the Quick Start Guide that comes with the platform).

- Log on as root by inputting the command, and verify that you are in the /home/test directory with the following two commands:

```
# sudo su
# ls
```

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# ls
Desktop  Documents  Downloads  dpdk  examples.desktop  Music  Pictures  Public  Templates  Videos
```

Note NIC Information

The configuration file for the traffic generator needs the PCI bus-related information and the MAC address. Note this information first using Linux commands, because once the DPDK or packet generator is run, these ports are unavailable to Linux.

1. For PCI bus-related NIC information, type the following command:

   ```
   # lspci
   ```

   You will see the following output. Note down that for port 0 the information is 03:00.0 and for port 1 the information is 03:00.1.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# lspci
00:00.0 Host bridge: Intel Corporation Atom Processor Z36xxx/Z37xxx Series SoC Transaction Register (rev 11)
00:02.0 VGA compatible controller: Intel Corporation Atom Processor Z36xxx/Z37xxx Series Graphics & Display
00:14.0 USB controller: Intel Corporation Atom Processor Z36xxx/Z37xxx, Celeron N2000 Series USB xHCI (rev 1
00:1a.0 Encryption controller: Intel Corporation Atom Processor Z36xxx/Z37xxx Series Trusted Execution Engin
00:1b.0 Audio device: Intel Corporation Atom Processor Z36xxx/Z37xxx Series High Definition Audio Controller
00:1c.0 PCI bridge: Intel Corporation Atom Processor E3800 Series PCI Express Root Port 1 (rev 11)
00:1c.2 PCI bridge: Intel Corporation Atom Processor E3800 Series PCI Express Root Port 3 (rev 11)
00:1c.3 PCI bridge: Intel Corporation Atom Processor E3800 Series PCI Express Root Port 4 (rev 11)
00:1f.0 ISA bridge: Intel Corporation Atom Processor Z36xxx/Z37xxx Series Power Control Unit (rev 11)
00:1f.3 SMBus: Intel Corporation Atom Processor E3800 Series SMBus Controller (rev 11)
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet
03:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
03:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk#
```

2. Find the MAC address with this command:

   ```
   # ifconfig
   ```

   You will see the following output. Note down that for port 0 the MAC address is `00:30:18:CB:F2:70` and for port 2 the MAC address is `00:30:18:CB:F2:71`.

   Note that the first port in the screenshot below, enp2s0, is the port connected to the Internet. No need to make a note of this.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::56cd:7409:7867:9572/64 Scope:Link
          inet6 addr: 2601:647:4902:79c0:6a14:5825:3e6c:de09/64 Scope:Global
          inet6 addr: 2601:647:4902:79c0:ac82:14bd:f4da:e627/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:82453 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56424 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:60196138 (60.1 MB)  TX bytes:17006340 (17.0 MB)

enp3s0f0  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:70
          inet6 addr: fe80::ef12:bbfd:4cff:3034/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20949 (20.9 KB)  TX bytes:22055 (22.0 KB)
          Memory:90500000-9057ffff

enp3s0f1  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:71
          inet6 addr: fe80::2ad4:3549:f1fa:73f0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85 errors:0 dropped:0 overruns:0 frame:0
          TX packets:146 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16637 (16.6 KB)  TX bytes:24515 (24.5 KB)
          Memory:90600000-9067ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:10234 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1072552 (1.0 MB)  TX bytes:1072552 (1.0 MB)
```

Item	Port 0	Port 1
PCI Bus-related NIC info (from lspci)	**03:00.0**	**03:00.1**
MAC address	**00:30:18:CB:F2:70**	**00:30:18:CB:F2:71**

Fill the following table with the information you gathered from your specific platform:

Item	Port 0	Port 1
PCI Bus-related NIC info (from lspci)		

MAC address		

What if you don't see both of the ports in response to the `ifconfig` command?

One possible reason is that you might have run the DPDK based application previously, in which case the application might have claimed those ports, making them unavailable to the kernel. In that case, refer to the appendix on how to unbind the ports from DPDK so that the kernel can claim them and you can find the MAC address with the `ifconfig` command.

In the following discussion, we will assume that you successfully found the ports and have noted down the MAC addresses.

Install the TRex* Traffic Generator

Input the following commands:

```
# pwd
# mkdir trex
# cd trex
# wget –no-cache http://trex-tgn.cisco.com/trex/release/latest
```

You should see that the install is complete, and saved in /home/test/trex/latest:

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# pwd
/home/test
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# mkdir trex
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test# cd trex
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# wget --no-cache http://trex-tgn.cisco.com/trex/release/latest
--2016-08-26 01:47:22--  http://trex-tgn.cisco.com/trex/release/latest
Resolving trex-tgn.cisco.com (trex-tgn.cisco.com)... 173.39.246.118
Connecting to trex-tgn.cisco.com (trex-tgn.cisco.com)|173.39.246.118|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://trex-tgn.cisco.com/trex/release/latest [following]
--2016-08-26 01:47:22--  https://trex-tgn.cisco.com/trex/release/latest
Connecting to trex-tgn.cisco.com (trex-tgn.cisco.com)|173.39.246.118|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 146045560 (139M) [application/x-tar]
Saving to: 'latest'

latest                                              100%[=================================================================

2016-08-26 01:48:00 (3.76 MB/s) - 'latest' saved [146045560/146045560]

root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex#
```

The next step is to untar the package:

```
# tar –xzvf latest
```

Below you see that version 2.08 is the latest version at the time of this screen capture:

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# tar -xzvf latest
v2.08/
v2.08/_t-rex-64-debug
v2.08/t-rex-64-debug
v2.08/_t-rex-64
v2.08/t-rex-64
v2.08/_t-rex-64-debug-o
v2.08/t-rex-64-debug-o
v2.08/_t-rex-64-o
```

```
# ls –al
```

You will see the directory with the version installed. In this exercise, the directory is v2.08, as shown below in response to the `ls –al` command. Change directory to the version installed on your system; for example, `cd <dir name with version installed>:`

```
# cd v2.08
```

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# ls -al
total 142640
drwxr-xr-x  3 root  root        4096 Aug 26 01:48 .
drwxr-xr-x 18 test  test        4096 Aug 26 01:44 ..
-rw-r--r--  1 root  root   146045560 Aug 24 14:35 latest
drwxr-xr-x 11 33066 floppy      4096 Aug 24 14:35 v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex# cd v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

```
# ls –al
```

You will see the file t-rex-64, which is the traffic generator executable:

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# ls -al
total 176364
drwxr-xr-x 11 33066 floppy     4096 Aug 24 14:35 .
drwxr-xr-x  3 root  root       4096 Aug 26 01:48 ..
drwxr-xr-x  6 33066 floppy     4096 Aug 24 14:34 automation
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 avl
-rwxr-xr-x  1 33066 floppy 27200827 Aug 24 14:34 bp-sim-64
-rwxr-xr-x  1 33066 floppy 16798769 Aug 24 14:34 bp-sim-64-debug
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 cap2
drwxr-xr-x  2 33066 floppy     4096 Aug 24 14:34 cfg
-rwxr-xr-x  1 33066 floppy     5501 Aug 24 14:34 daemon_server
-rwxr-xr-x  1 33066 floppy     2207 Aug 24 14:34 doc_process.py
-rwxr-xr-x  1 33066 floppy    26985 Aug 24 14:34 dpdk_nic_bind.py
-rwxr-xr-x  1 33066 floppy    33325 Aug 24 14:34 dpdk_setup_ports.py
drwxr-xr-x  2 33066 floppy    16384 Aug 24 14:34 exp
drwxr-xr-x 22 33066 floppy     4096 Aug 24 14:34 external_libs
-rwxr-xr-x  1 33066 floppy     2291 Aug 24 14:34 find_python.sh
drwxr-xr-x 15 33066 floppy     4096 Aug 24 14:34 ko
-rw-r--r--  1 33066 floppy  3150071 Aug 24 14:34 libzmq.so.3
-rwxr-xr-x  1 33066 floppy    11148 Aug 24 14:34 master_daemon.py
drwxr-xr-x  3 33066 floppy     4096 Aug 24 14:34 python-lib
-rwxr-xr-x  1 33066 floppy      802 Aug 24 14:34 run_functional_tests
-rwxr-xr-x  1 33066 floppy      832 Aug 24 14:34 run_regression
drwxr-xr-x  6 33066 floppy     4096 Aug 24 14:34 stl
-rwxr-xr-x  1 33066 floppy      403 Aug 24 14:34 stl-sim
-rwxr-xr-x  1 33066 floppy 34390661 Aug 24 14:34 _t-rex-64
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64
-rwxr-xr-x  1 33066 floppy 28843174 Aug 24 14:34 _t-rex-64-debug
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-debug
-rwxr-xr-x  1 33066 floppy 28812951 Aug 24 14:34 _t-rex-64-debug-o
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-debug-o
-rwxr-xr-x  1 33066 floppy 35101658 Aug 24 14:34 _t-rex-64-o
-rwxr-xr-x  1 33066 floppy      902 Aug 24 14:34 t-rex-64-o
-rwxr-xr-x  1 33066 floppy     2086 Aug 24 14:34 trex-cfg
-rw-r--r--  1 33066 floppy  6077140 Aug 24 14:35 trex_client_v2.08.tar.gz
-rwxr-xr-x  1 33066 floppy      444 Aug 24 14:34 trex-console
-rwxr-xr-x  1 33066 floppy     5501 Aug 24 14:34 trex_daemon_server
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

Configure the Traffic Generator

The good news is that the TRex package comes with a sample config file cfg/simple_cfg.yaml. Copy that to /etc/trex_cfg.yaml and edit the file by issuing the following commands, making sure that you're in your /home/test/trex/<your version> directory:

```
# pwd
# cp cfg/simple_cfg.yaml /etc/trex_cfg.yaml
# gedit /etc/trex_cfg.yaml
```

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# cp cfg/simple_cfg.yaml /etc/trex_cfg.yaml
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# gedit /etc/trex_cfg.yaml
```

Edit the file as shown below with the applicable NIC information you gathered in previous steps:

```
- port_limit      : 2          # this option can limit the number of port of the platform
  version         : 2
  interfaces    : ["03:00.0","03:00.1"] #the interfaces using ./dpdk_setup_ports.py -s
  port_info       :  # set eh mac addr
          - dest_mac        :   [0x00,0x30,0x18,0xcb,0xf2,0x71]    # router mac addr should be taken from router
            src_mac         :   [0x00,0x30,0x18,0xcb,0xf2,0x70]  # source mac-addr - taken from ifconfig
          - dest_mac        :   [0x00,0x30,0x18,0xcb,0xf2,0x70]  # router mac addr  taken from router
            src_mac         :   [0x00,0x30,0x18,0xcb,0xf2,0x71]  #source mac-addr  taken from ifconfig
```

Below is the line-by-line description of the configuration information:

- `Port_limit` should be 2 (since DPDK-in-a-Box has two ports)
- `Version` should be 2
- Interfaces should be the PCI bus ports you gathered using `lspci`. In this exercise they are [“03:00.0”, “03:00.1”]
- `Port_information` contains a `dest_mac, src_mac` pair, which will be in the packet header of the traffic generated. The first pair is for port0. Since port0 is connected to port1, the first `dest_mac` is the MAC address of port 1. The second pair is for port1. Since port1 is connected to port0, the second `dest_mac` is the MAC address of port 0.

Please note that when you connect an appliance to which traffic must be injected, the `dest_mac` addresses will be that of the appliance.

Note Platform lcore Count

This section is for informational purposes only.

`# cat /proc/cpuinfo` will give you the lcore information as shown in the Exercises section.

Why is this information useful?

The command line below that runs the traffic generator uses the –c option to specify the number of lcores to be used for the traffic generator. You want to know how many lcores exist in the platform. Hence, issuing `cat /proc/cpuinfo` and eyeballing the number of lcores that are available in the system will be helpful.

Run the Traffic Generator

```
# sudo ./t-rex-64 –f cap2/dns.yaml –c 1 –d 100
```

What are the parameters –f, -c, and –d?

`-f` for YAML traffic configuration file

`-c` for number of cores. Monitor the CPU% of TRex – it should be ~50%. Use cores accordingly

`-d` for duration of the test (sec). Default: 0

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# sudo ./t-rex-64 -f cap2/dns.yaml -c 1 -d 100
```

Summary

Below are three output screens: 1) During the traffic run, 2) Linux `top` command output, and 3) Final output after the completion of the run.

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08


-Per port stats table
      ports |               0 |               1
 -----------------------------------------------------------------------------------------
   opackets |              15 |              15
     obytes |            1155 |            1395
   ipackets |              15 |              15
     ibytes |            1395 |            1155
    ierrors |               0 |               0
    oerrors |               0 |               0
      Tx Bw |     584.18  bps |     705.57  bps

-Global stats enabled
 Cpu Utilization : 0.0  %  0.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :       1.29 Kbps
 Total-Rx        :       1.29 Kbps
 Total-PPS       :       1.90  pps
 Total-CPS       :       0.97  cps

 Expected-PPS    :       2.00  pps
 Expected-CPS    :       1.00  cps
 Expected-BPS    :       1.30 Kbps

 Active-flows    :        0  Clients :      511   Socket-util : 0.0000 %
 Open-flows      :       15  Servers :      255   Socket :       15 Socket/Clients :  0.0
 drop-rate       :       0.00  bps
 current time    : 27.0 sec
 test duration   : 73.0 sec
```

Screen output showing traffic during run (15 packets so far Tx & Rx)

```
top - 06:21:00 up  2:05,  1 user,  load average: 1.33, 0.39, 0.18
Threads: 418 total,   2 running, 416 sleeping,   0 stopped,   0 zombie
%Cpu(s): 53.0 us,  2.0 sy,  0.0 ni, 42.0 id,  3.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1939152 total,   296884 free,  1122584 used,   519684 buff/cache
KiB Swap:  1986556 total,  1639292 free,   347264 used.   454792 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
16986 root      20   0  893584   9400   5852 R 99.3  0.5   0:45.47 lcore-slave+
 1956 test      20   0  662648  20424  11936 S  2.6  1.1   0:32.98 gnome-termi+
 1441 test      20   0 1491584  89496  23572 S  1.6  4.6   4:06.93 compiz
16983 root      20   0  893584   9400   5852 S  1.6  0.5   0:05.17 _t-rex-64-o
  775 root      20   0  379964  32892  23824 S  1.3  1.7   1:20.63 Xorg
16114 root      20   0   49268   3468   2444 R  1.0  0.2   1:07.05 top
  589 root      20   0  173360   2100   1888 S  0.3  0.1   0:00.07 thermald
16925 root      20   0       0      0      0 S  0.3  0.0   0:00.13 kworker/u8:3
    1 root      20   0  185372   3520   2192 S  0.0  0.2   0:04.57 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.43 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0.0  0.0   0:07.56 rcu_sched
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
```

Output of top –H command during the run

```
root@test-Minnowboard-Turbot-D0-PLATFORM: /home/test/trex/v2.08
 precent    : -nan %
 histogram
 -----------
 m_total_bytes                          :      15.82 Kbytes
 m_total_pkt                            :     200.00  pkt
 m_total_open_flows                     :     100.00  flows
 m_total_pkt                            : 200
 m_total_open_flows                     : 100
 m_total_close_flows                    : 100
 m_total_bytes                          : 16200
 --------------
port : 0
------------
 opackets                                : 100
 obytes                                  : 7700
 ipackets                                : 100
 ibytes                                  : 9300
 Tx :     291.61  bps
port : 1
------------
 opackets                                : 100
 obytes                                  : 9300
 ipackets                                : 100
 ibytes                                  : 7700
 Tx :     352.41  bps
 Cpu Utilization : 0.0  %  0.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :     644.02  bps
 Total-Rx        :     643.99  bps
 Total-PPS       :       0.95  pps
 Total-CPS       :       0.47  cps

 Expected-PPS    :       2.00  pps
 Expected-CPS    :       1.00  cps
 Expected-BPS    :       1.30 Kbps

 Active-flows    :        0  Clients :      511   Socket-util : 0.0000 %
 Open-flows      :      100  Servers :      255   Socket :        0 Socket/Clients :  0.0
 drop-rate       :       0.00  bps
 summary stats
 --------------
 Total-pkt-drop       : 0 pkts
 Total-tx-bytes       : 17000 bytes
 Total-tx-sw-bytes    : 0 bytes
 Total-rx-bytes       : 17000 byte

 Total-tx-pkt         : 200 pkts
 Total-rx-pkt         : 200 pkts
 Total-sw-tx-pkt      : 0 pkts
 Total-sw-err         : 0 pkts
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

Screen output after completing the run (100 packets Tx & Rx)

Next Steps

Congratulations! With the above hands-on exercise, you have successfully built your own Intel DPDK based traffic generator.

As a next step, you can connect back-to-back two DPDK-in-a-Box platforms, and use one as a traffic generator and the other as a DPDK application development and test vehicle.

Please send your feedback to M Jay Muthurajan.Jayakumar@intel.com.

Exercises

1) How would you configure the traffic generator for different packet lengths?

2) To run the traffic generator forever, what should be the value of –d?

3) How would you measure latency (assuming you have more cores)?

4) Reason out the root cause and find the solution by looking up the error, "Note that the uio or vfio kernel modules to be used should be loaded into the kernel before running the dodk-devbind.py script" in Chapter 3 of the DPDK.org document *Getting Started Guide for Linux*.

5) In the following screenshot, determine the hyperthreading state—enabled vs. disabled? (Hint: this is the Intel Atom processor platform.)

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 55
model name      : Intel(R) Atom(TM) CPU  E3826  @ 1.46GHz
stepping        : 9
cpu MHz         : 536.688
cache size      : 512 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
opl xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tr
xpriority ept vpid tsc_adjust smep erms dtherm arat
bugs            :
bogomips        : 2930.40
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 55
model name      : Intel(R) Atom(TM) CPU  E3826  @ 1.46GHz
stepping        : 9
cpu MHz         : 1394.531
cache size      : 512 KB
physical id     : 0
siblings        : 2
core id         : 2
cpu cores       : 2
apicid          : 4
initial apicid  : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
opl xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tr
xpriority ept vpid tsc_adjust smep erms dtherm arat
bugs            :
bogomips        : 2930.40
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/trex/v2.08#
```

Appendix: Unbinding from DPDK & Binding to Kernel

This section is not needed if `ifconfig` is able to find the ports you want to use for traffic generation. In that case, you can skip this section.

What is the reason `ifconfig` cannot find the two ports? If you are only interested in the solution, skip this troubleshooting section and go to the Solution section.

Root Cause

`ifconfig` is not showing the two ports below. Why?

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.6  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2601:647:4902:79c0:b98c:9a9e:55f6:8314/64 Scope:Global
          inet6 addr: 2601:647:4902:79c0:8712:fcc2:af9:a689/64 Scope:Global
          inet6 addr: fe80::3603:79d2:fe9e:8468/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:60216 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53600 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:33276201 (33.2 MB)  TX bytes:10484345 (10.4 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:15976 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15976 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1594031 (1.5 MB)  TX bytes:1594031 (1.5 MB)

root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools#
```

The reason that `ifconfig` is unable to find the two ports is possibly because the DPDK application was previously run and was aborted without releasing the ports, or it might be that a DPDK script runs automatically after boot and claims the ports. Regardless of the reason, the solution below will enable `ifconfig` to show both ports.

Solution

1. Run ./setup.sh in the directory /home/test/dpdk/tools
2. Display current Ethernet device settings
3. Unbind the first port from IGB UIO (assuming it is bound to IGB UIO)
4. Bind the port to IGB (the kernel driver)
5. Repeat steps 3-5 to unbind the second port from IGB UIO and bind to IGB.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ls
cpu_layout.py  dpdk_nic_bind.py  pmdinfo.py  setup.sh
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk/tools# ./setup.sh
```

Select “Display current Ethernet device settings” (option 23 in this case).

```
[23] Display current Ethernet device settings
[24] Bind Ethernet device to IGB UIO module
[25] Bind Ethernet device to VFIO module
[26] Setup VFIO permissions

----------------------------------------------------------
 Step 3: Run test application for linuxapp environment
----------------------------------------------------------
[27] Run test application ($RTE_TARGET/app/test)
[28] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)

----------------------------------------------------------
 Step 4: Other tools
----------------------------------------------------------
[29] List hugepage info from /proc/meminfo

----------------------------------------------------------
 Step 5: Uninstall and system cleanup
----------------------------------------------------------
[30] Unbind NICs from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings

[35] Exit Script

Option: 23
```

Status showing two ports claimed by the DPDK driver.

```
Option: 23


Network devices using DPDK-compatible driver
============================================
0000:03:00.0 'I350 Gigabit Network Connection' drv=igb_uio unused=igb,vf
0000:03:00.1 'I350 Gigabit Network Connection' drv=igb_uio unused=igb,vf

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller'
o_pci_generic *Active*

Other network devices
=====================
```

Unbind the first NIC from DPDK (specifically IGB UIO).

```
-----------------------------------------------
 Step 5: Uninstall and system cleanup
-----------------------------------------------
[30] Unbind NICs from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings
```

1. Select option 30 and then enter the PCI address of device to unbind:

```
Option: 30


Network devices using DPDK-compatible driver
============================================
0000:03:00.0 'I350 Gigabit Network Connection' drv=igb
0000:03:00.1 'I350 Gigabit Network Connection' drv=igb

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Et
o_pci_generic *Active*

Other network devices
=====================
<none>

Enter PCI address of device to unbind: 0000:03:00.0
```

2. Bind the kernel driver igb to the device:

```
Enter name of kernel driver to bind the device to: igb
```

If the inputs entered are correct, script acknowledges OK.

```
OK

Press enter to continue ...
```

3. Verify by displaying current Ethernet device settings.

4.

```
Option: 23


Network devices using DPDK-compatible driver
============================================
0000:03:00.1 'I350 Gigabit Network Connection' drv=igb_uio unused=igb,vfio-pci,uio

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller' if=enp2s0
o_pci_generic *Active*
0000:03:00.0 'I350 Gigabit Network Connection' if=enp3s0f0 drv=igb unused=igb_uio,

Other network devices
=====================
<none>
```

Success!

Above you will see the first port 0000:30:00.0 bound to the kernel.

Now on to the second port 0000:30:00.1

Success!

Below you will see both ports bound back to kernel.

```
Option: 23


Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:02:00.0 'RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller' if=enp2s0
o_pci_generic *Active*
0000:03:00.0 'I350 Gigabit Network Connection' if=enp3s0f0 drv=igb unused=igb_uio,v
0000:03:00.1 'I350 Gigabit Network Connection' if=enp3s0f1 drv=igb unused=igb_uio,v

Other network devices
=====================
<none>
```

Now that both ports are bound back to kernel, `ifconfig` will give the needed info for those ports.

```
root@test-Minnowboard-Turbot-D0-PLATFORM:/home/test/dpdk# ifconfig
enp2s0    Link encap:Ethernet  HWaddr 00:08:a2:09:f2:1d
          inet addr:192.168.0.6  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2601:647:4902:79c0:249:ce31:c570:85a/64 Scope:Global
          inet6 addr: fe80::a572:b28f:7fd6:5336/64 Scope:Link
          inet6 addr: 2601:647:4902:79c0:6cc6:fc3c:5f31:d114/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21515 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18422 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6537879 (6.5 MB)  TX bytes:5099447 (5.0 MB)

enp3s0f0  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:70
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5995 (5.9 KB)  TX bytes:21997 (21.9 KB)
          Memory:90500000-9057ffff

enp3s0f1  Link encap:Ethernet  HWaddr 00:30:18:cb:f2:71
          inet6 addr: fe80::f596:8de9:9963:4008/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:486 (486.0 B)  TX bytes:10474 (10.4 KB)
          Memory:90600000-9067ffff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:6303 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6303 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:582195 (582.1 KB)  TX bytes:582195 (582.1 KB)
```

References

- MinnowBoard Wiki Home at minnowboard.org – learn more about the MinnowBoard Turbot single board computer.
- Profiling DPDK Code with Intel® VTune™ Amplifier - Use Intel® VTune™ Amplifier to profile DPDK micro benchmarks with your application. This comprehensive reference provides guidelines and instructions.
- Intel® VTune™ and Performance Optimizations - This session recording from the July 11, 2016DPDK/NFV DevLab covers performance optimization best practices, including analysis of NUMA affinity, microarchitecture optimizations with VTune, and tips to help you identify hotspots in your own application.
- DPDK Performance Optimization Guidelines White Paper - Learn best-known methods to optimize your DPDK application's performance. Includes profiling methodology to help identify bottlenecks, then shows how to optimize BIOS settings, partition NUMA resources, optimize your Linux* configuration for DPDK, and more.

Notices